c0t0d0s0: Less known Solaris features

Joerg Moellenkamp

February 21th, 2010

Contents

Part I.

Introduction

1. The genesis of LKSF

1.1. How it started

In January 2008 I felt like writing short tutorials on less-known features in Solaris.

It turns out that although many of Solaris' features are known to customers, there are few customers aware of all of Solaris features.

An second insight was a little bit different: Sun doesn't have a "not enough documentation" problem, it has a problem with "too much documentation". Solaris is a tool with extreme capabilities – all well documented at docs.sun.com or in the man pages. The problem is (vastly exaggerated): it's like having the Encyclopedia Britannica, sufficient tools, skilled engineers and mechanics and the material, leaving you with the job of building an Airbus 380. While that may be possible, it is difficult to find the right starting point.

"The less known Solaris Features" (a.k.a LKSF) wants to give you this start, and I started writing these tutorials on my blog `http://www.c0t0d0s0.org` in February 2008.

1.2. The scope

The tutorials in this book don't attempt to explain a feature in its entirety, as that would be redundant to the man pages and docs.sun.com. The examples in this book take some typical use cases and describe the configuration of a feature to match. At the end of each feature you will find some links to more information and the documentation about the feature.

1.3. The disclaimer

- Please don't try the stuff in this tutorial on your production systems until you are familiar with the steps.
- Please obey the licensing note in the Appendix.

1.4. Credits and Kudos

1.5. Credits

Credits and kudos for providing helping hands (in alphabetic order of the surname):

- **Ceri Davies** (`http://typo.submonkey.net/`) for proof reading the document.
- **Jan-Piet Mens** (`http://blog.fupps.com/`) for proof reading for the document and suggestions to the LaTeXsource.
- **Marina Sum** for the help with the `pfexec` chapter.

2. The guide to LKSF

2.1. Solaris Administration

2.1.1. Liveupgrade

From time to time you have to update or patch your system. How do you patch the system without long service interruptions? How do you keep a running version of your operating environment in case something goes wrong? Solaris can help you cope with these situations using its LiveUpgrade feature – patching and Updating while the system is still operational.

You will find the Liveupgrade tutorial in section 3 on page 27

2.1.2. Boot environments based on ZFS snapshots

Live Upgrade was introduced for multiple disks or multiple partitions several years ago. How would such a functionality look like on modern file systems with ubiquitous snapshots? Well, ZFS boot and boot environments on ZFS give you such functionality today.

The introduction to the ZFS based boot environments is located in section 4 on page 35

2.1.3. Working with the Service Management Facility

`init.d` was a venerable concept for many years to start and stop services. But it had its shortfalls. Sun therefore introduced the service management facility in Solaris 10, to offer functionalities like service dependencies, service instances and a central point of service management.

Section 5 on page 40 will give you insight to this interesting feature of Solaris.

2.1.4. Solaris Resource Manager

You can run a multitude of services and applications on a single big system. But how can you ensure, that every application gets its share of performance? Solaris Resource Manager can help you to control a process's CPU, memory and other resources.

In section 6 on page 61 you can learn how use this feature.

2.1.5. /home? /export/home? AutoFS?

Many people wonder about the different location for the user home directories on a Solaris system. Why are the home directories located in `/export/home` and not in `home`?

The history of these two directories and some insight into AutoFS will be described in section 7 on page 80

2.1.6. lockfs

Sometimes you have to ensure that a file system doesn't change while you're working on it. To avoid that, use the `lockfs` command.

You will learn more about this function in section 8 on page 87.

2.2. Solaris Security

2.2.1. Role Based Access Control and Least Privileges

`root` can do everything on a system, but is it a wise choice to give everyone the `root` password and thus the key for all rooms in the kingdom? Does a process need all the privileges of `root` just to bind to a privileged port? How do you configure the least set of privileges to a process?

Section 12 at page 115 answers this questions.

2.2.2. The Solaris Security Toolkit

The Solaris Security Toolkit is designed to automate the hardening and minimization of a Solaris system. The toolkit contains the knowledge even to harden a tough target like a Sun Cluster installation and simplifies the necessary steps.

A tutorial on the usage of the Solaris Security Toolkit is located in section 13 on page 135.

2.2.3. Auditing

What happens on your system? When did a user use which command? When did a user delete a particular file? You need log files to answers this question. The auditing functionality in Solaris generates these and reports on a vast amount of actions happening on your system.

The configuration of this feature is explained in 14 on page 145.

2.2.4. Basic Audit Reporting Tool

Sometimes you need to know, what has changed on a system since you installed it. For example when all your fellow admins tell you after a system crash. The Basic Audit Reporting Tool can answer this question by comparing different states of you system.

The usage of BART is explained in section 15 on page 151.

2.2.5. IPsec

Secure communication between hosts gets more and more important. Secure communication does not only mean encrypted traffic. It also includes the authentication of your communication partner. Solaris has had an IPsec implementation since a number of versions.

The configuration of IPsec is described in 16 on page 153.

2.2.6. On Passwords

It is important to have good and secure passwords. All other security systems are rendered worthless without good keys to the systems. Solaris has some features to help the administrator to enforce good passwords.

Section ?? on page ?? describes this feature.

2.2.7. Signed binaries

There are literally thousands of binaries on your system, but are they really all supplied by Sun? Every binary in Solaris is digitally signed by Sun.

The section 17 on page 162 explains, how you verify these signatures.

2.3. Networking

2.3.1. Crossbow

Project "Crossbow" resulted in a new IP stack for Solaris. It solves challenges like the question how do load network interfaces in the 10GBe age and introduces an integrated layer for network virtualization.

Some interesting features of Crossbow and their configuration is described in section 20 on page 177.

2.3.2. IPMP

Solaris provides an matured mechanism to ensure the availability of the network connection. This feature is called IP Multipathing (or short: IPMP). It's in Solaris for several versions now and it's easy to use.

An description of the configuration of new and classic IPMP is available in section 21 on page 197.

2.3.3. kssl

In Solaris 10 got an interesting feature to enable SSL for any service by adding a transparent SSL proxy in front of its . This proxy runs completely in kernel-space and yields better performance compared to a solution in the user-space.

The section 23 on page 237 explains, how you enable kssl.

2.4. Storage

2.4.1. fssnap - snapshots for UFS

File system backups can be faulty. When they take longer, the file system has a different content at the beginning at the end of the backup, thus they are consistent. A solution to this problem is freezing the file system. `fssnap` delivers this capability to UFS.

Section 24 describes this feature. The tutorial starts on page 242.

2.4.2. iSCSI

With increasing transfer speed of Ethernet it gets more and more feasible to use this media to connect block devices such as disks to a server. Since Update 4 Solaris 10 has a built-in functionality to act as an iSCSI initiator and target.

The configuration of iSCSI is the topic of section 26 on page 258.

2.4.3. Remote Mirroring with the Availability Suite

You have two data centers with similar hardware and same software on both. The second data center is a perfect replica of the first. Good basics for disaster recovery. The remaining issue: How to get the data to your backup data center? And how to get the data back after the you've recovered your primary data center. The Remote Mirroring facility of the Availability Suite was designed for such situations.

Section 27 on page 270 explains the configuration of a remote mirror.

2.4.4. Point-in-Time Copy with the Availability Suite

Sometimes you need a copy of your data to test applications or to keep a working copy when you upgrade them. Copying the whole data is possible, but inefficient. Generating a frozen snapshot is easier. The Availability Suite enables the admin to freeze the state of a disk with much lower overhead in a short time.

Point-in-Time copies will be explained in 28 on page 288.

2.4.5. SamFS - the Storage Archive Manager File System

Typically documents and data are put away for years without them being accessed, but you cannot delete the data – possibly due to law. So your data rotates on rotating rust for years on power-hungry disks. It would be nice to archive all this data on tape without thinking about it. SamFS is a tool for this task.

The section 29 on page 312 will give you a basic overview of the configuration and use of SamFS, the hierarchical storage management system from Sun.

2.5. Solaris Administrators Toolbox

2.5.1. fuser

This short tutorial will show you how to find out which processes are using files on a file system. It's located in section 30 on page 341.

2.5.2. pfiles

Many people install `lsof` on their system as they know it from Linux. But you have an similar tool in Solaris. In section 31 on page 344 you will find a short tip for its usage.

2.5.3. Installing Solaris Packages directly via Web

This trick isn't widely known. You can install a package directly from a HTTP source. Look in this section 32 on page 346 for a description.

2.5.4. About crashes and cores

There is no bug-free code, thus from time to time an operating system has to react: it crashes and dumps core to protect itself. Learn to control the core-dumps and how you can do some basic analysis in section 33 on page 347.

2.6. Nontechnical feature

2.6.1. Long support cycles

Solaris has a long life time for a single Solaris release with a defined time line that governs the life cycle. Get some insight to the life of a release in section 36 on page 396.

Part II.

Solaris Administration

3. Liveupgrade

Solaris 10/Opensolaris

3.1. How to change the world

Once in a while root saw some imperfections in his world. He had to change some things. But root couldn't stop the turning of the world for hours as people lived in this world. Because of root's special powers, root was able to create a second world without people. Thus root created a second world as an exact copy of the old world. And now root was able to work on the imperfections of his world as long as root wanted. Then he behold and all was good. A magic chant late at night when all people slept and the people woke up in the new world.

3.2. What's Live Upgrade

Okay, Live Upgrade isn't really a "less known feature", but in the time working at the keyboard at several customer sites, I've got aware of a fact: One of the most simple, but brilliant feature of Solaris is a somewhat unused feature. The feature is called Live Upgrade.

We've got painfully aware of two facts in the past: At first ... yes, we know of our somewhat suboptimal patch process. And: You can't expect updates of the operating environment when you have to bring down the machine for some time. Thus Sun introduced a feature called Live Upgrade.

Okay, Live Upgrade is so easy that nobody has an excuse not to use it. And with 6 GB size of the SOE and 73 GB boot disks minimum "no space" isn't an excuse too ;)

3.3. The concept behind Live Upgrade

The basic concept behind Live Upgrade is pretty simple. All mechanisms are grouped around the concept of alternate boot environments. At first you have your running boot

environment and a empty slice or disk (the symbol with the thick lines is the active boot environment).

Figure 3.1.: Live Upgrade: Situation before start

Now you create an alternate boot environment. It's a copy of the actual boot environment. The system still runs on this environment.

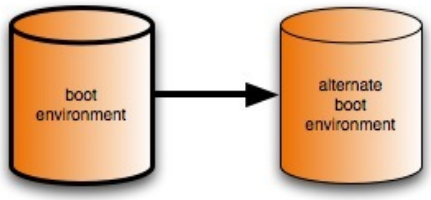

Figure 3.2.: Live Upgrade: Creating the alternate boot environment

The trick is: The update/patch processes doesn't work on the actual boot environment, they use this alternate but inactive boot environment. The running boot environment isn't touched at all.

After the completion of the updating you have an still running boot environment and a fully patched and updated alternate boot environment. Now the boot environments swap their roles with a single command and a single reboot.

After the role swap the old system stays untouched. So, whatever happens with your new installation, you can fall back to you old system. In case you see problems with your new configuration, you switch back the but environments and you run with your old operating environment.

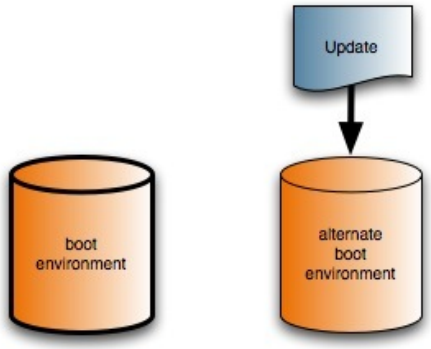

Figure 3.3.: Live Upgrade: Patching/Upgrading the alternate boot environment

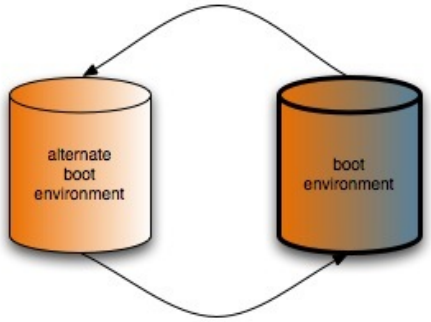

Figure 3.4.: Live Upgrade: Switching alternate and actual boot environment

3.4. A hint for testing this

Use some spare equipment. I've used my MacBook Pro for the first try, and it took forever. Do yourself a favor and don't use a virtual environment. At least use a real DVD and not an ISO to prevent the harddisk from jumping around.

3.5. Using Live Upgrade without Updating

Okay, I will present two usecases to you: The first one doesn't exactly match to the "upgrade" moniker. It's a solution for a small problem: You've installed your system and after a few weeks you realize, that you filesystem model wasn't the best choice. /export/home to large, / to small. and a separated /var would be nice. Okay, how you

separate those filesystems without a longer service interruption. Bringing the system down, moving files around, booting it up is not an option for productive system. Moving while running isn't a good idea.

Live Update is a nice, but simple solution for this problem: Live Upgrade replicates the boot environments by doing a file system copy. The filesystem layout of the old boot environment and the new environment doesn't have to be the same. Thus you can create a filesystem layout with a bigger /, a smaller /export/home and a separate /var. And the best is: The system runs while doing this steps.

In my example I will start with an operating system on a single partition. The partition is located on `/dev/dsk/c0d0s0` and has the size of 15 GB.

```
/ on /dev/dsk/c0d0s0 read/write/setuid/devices/intr/largefiles/
   logging/xattr/onerror=panic/dev=1980000 on Mon Feb 11
   21:06:02 2008
```

At the installation time I've created some additional slices. `c0d0s3` to `c0d0s6`. Each of the slices has the size of 10 GB.

Separating the single slice install to multiple slices is nothing more than using Live Upgrade without upgrading. At first I create the alternate boot environment:

```
# lucreate -c "sx78" -m /:/dev/dsk/c0d0s3:ufs -m /usr:/dev/dsk/
   c0d0s4:ufs -m /var:/dev/dsk/c0d0s5:ufs -n "sx78_restructured
   "
Discovering physical storage devices
[..]
Populating contents of mount point </>.
Populating contents of mount point </usr>.
Populating contents of mount point </var>.
[..]
Creation of boot environment <sx78_restructured> successful.
```

We've successfully created a copy of the actual boot environment. But we told the mechanism to put / on c0d0s3, /usr on c0d0s4 and /var on c0d0s5. As this was the first run of Live Upgrade on this system the naming of the environment is more important than on later runs. Before this first run, the boot environment has no name. But you need it to tell the process, which environment should be activated, patched or updated. Okay, my actual environment runs with Solaris Express CE build 78, thus I've called it "sx78". The lucreate command set this name to the actual environment. My new boot environment has the name `"sx78_restructured"` for obvious reasons.

Okay, now you have to activate the alternate boot environment.

```
# luactivate sx78_restructured
Saving latest GRUB loader.
Generating partition and slice information for ABE <
   sx78_restructured>
Boot menu exists.
Generating direct boot menu entries for ABE.
Generating direct boot menu entries for PBE.
[..]
Modifying boot archive service
GRUB menu is on device: </dev/dsk/c0d0s0>.
Filesystem type for menu device: <ufs>.
Activation of boot environment <sx78_restructured> successful.
```

Now we have to reboot the system. Just use init or shutdown. If you use any other command to reboot the system, Live Upgrade will not switch to new environment:

```
# init 6
```

Okay, this takes a minute. But let's have a look on the mount table after the boot.

```
# mount
/ on /dev/dsk/c0d0s3 read/write/setuid/devices/intr/largefiles/
   logging/xattr/onerror=panic/dev=1980003 on Tue Feb 12
   05:52:50 2008
[...]
/usr on /dev/dsk/c0d0s4 read/write/setuid/devices/intr/
   largefiles/logging/xattr/onerror=panic/dev=1980004 on Tue
   Feb 12 05:52:50 2008
[...]
/var on /dev/dsk/c0d0s5 read/write/setuid/devices/intr/
   largefiles/logging/xattr/onerror=panic/dev=1980005 on Tue
   Feb 12 05:53:12 2008
```

Mission accomplished. Okay, but we want to use LiveUpgrading for upgrading, later. Switch back to your old environment:

```
# luactivate sx78
```

Boot the system. And your are back on your old single-slice installation on `c0d0s0`:

```
/ on /dev/dsk/c0d0s0 read/write/setuid/devices/intr/largefiles/
   logging/xattr/onerror=panic/dev=1980000 on Mon Feb 12
   06:06:02 2008
```

3.6. Using Live Upgrade for upgrading Solaris Express

With a new Solaris Express Community Edition every week a Live Upgrade procedure is a good practice to update your system to a new release of the operating system.

Okay, I've burned a DVD with the Solaris Express Community Edition Build 81. I want to upgrade the existing boot environment on the three slices. Just to keep the naming in line, I rename it so sx81.

```
# lurename -e sx78_restructured -n sx81
Renaming boot environment <sx78_restructured> to <sx81>.

Changing the name of BE in the BE definition file.
Changing the name of BE in configuration file.
Updating compare databases on boot environment <sx81>.
Changing the name of BE in Internal Configuration Files.
Propagating the boot environment name change to all BEs.
Boot environment <sx78_restructured> renamed to <sx81>.
```

You don't have to rename it, you just could use the old name. But why should you confuse your fellow admins by calling your Build 81 boot environment `sx78_restructured`.

Okay, now start the upgrade. My installation DVD was mounted under `cdrom/sol_11_x86` by Solaris and I want to upgrade the `sx81` boot environment. This will take a while. Do this overnight or go shopping or play with your children. Your system is still running and the process will not touch your running installation:

```
# luupgrade -u -n sx81 -s /cdrom/sol_11_x86

Copying failsafe kernel from media.
Uncompressing miniroot
[...]
The Solaris upgrade of the boot environment <sx81> is complete.
Installing failsafe
Failsafe install is complete.
```

Okay. Let's check the `/etc/release` before booting into the new environment:

```
# cat /etc/release
                    Solaris Express Community Edition snv_78 X86
           Copyright 2007 Sun Microsystems, Inc.  All Rights
              Reserved.
                        Use is subject to license terms.
                           Assembled 20 November 2007
```

Activate the new boot environment:

```
# luactivate sx81

Saving latest GRUB loader.
Generating partition and slice information for ABE <sx81>
Boot menu exists.
Generating direct boot menu entries for ABE.
Generating direct boot menu entries for PBE.
[...]
Modifying boot archive service
GRUB menu is on device: </dev/dsk/c0d0s0>.
Filesystem type for menu device: <ufs>.
Activation of boot environment <sx81> successful.
```

Eject the installation DVD and reboot the system:

```
# eject
cdrom /dev/dsk/c1t0d0s2 ejected
# init 6
```

Wait a minute, login to the system and let's have a look at `/etc/release` again:

```
bash-3.2$ cat /etc/release
                    Solaris Express Community Edition snv_81 X86
            Copyright 2008 Sun Microsystems, Inc.  All Rights
               Reserved.
                         Use is subject to license terms.
                             Assembled 15 January 2008
```

By the way, the system runs on the three seperated slices now:

```
/ on /dev/dsk/c0d0s3 read/write/setuid/devices/intr/largefiles/
   logging/xattr/onerror=panic/dev=1980003 on Tue Feb 12
   07:22:32 2008
[..]
/usr on /dev/dsk/c0d0s4 read/write/setuid/devices/intr/
   largefiles/logging/xattr/onerror=panic/dev=1980004 on Tue
   Feb 12 07:22:32 2008
[..]
/var on /dev/dsk/c0d0s5 read/write/setuid/devices/intr/
   largefiles/logging/xattr/onerror=panic/dev=1980005 on Tue
   Feb 12 07:22:54 2008
```

Neat, isn't it ?

3.7. Do you want to learn more ?

Documentation

Solaris 10 8/07 Installation Guide: Solaris Live Upgrade and Upgrade Planning ¿¿ Upgrading With Solaris Live Upgrade

Others

Solaris Live Upgrade Software: Minimum Patch Requirements (the infodoc formerly known as 72099

4. Boot environments based on ZFS snapshots

Opensolaris

4.1. Using snapshots for boot environments

One of the nice features of ZFS is the fact that you get snapshots for free. The reason lies in the copy-on-write nature of ZFS. You can freeze the filesystem by simply not freeing the old blocks. As new data is written to new blocks, you don't even have to copy the blocks (in this sense the COW of ZFS is more like a ROW, a "redirect on write").

ZFS boot enables the system to work with such snapshots, as you can use one to boot from. You can establish multiple boot environments just by snapshotting the boot filesystems, cloning them and promoting them to real filesystems. These are features inherent to ZFS.

4.2. A practical example

A warning at first: Don't try this example without a good backup of your system. Failing that, use a test system or a test VM. We will fsck up the system during this example. Okay...

I've updated my system, so I have already two boot environments:

```
jmoekamp@glamdring:~# beadm list

BE             Active Active on Mountpoint Space
Name                  reboot               Used
----           ------ --------- ---------- -----
opensolaris-1  yes    yes       legacy     2.31G
opensolaris    no     no        -          62.72M
```

This mirrors the actual state in your ZFS pools. You will find filesystems named accordingly.

```
NAME
   USED  AVAIL  REFER  MOUNTPOINT
rpool                                                          2.39
   G   142G  56.5K  /rpool
rpool@install                                                  18.5
   K      -    55K  -
rpool/ROOT                                                     2.37
   G   142G    18K  /rpool/ROOT
rpool/ROOT@install
   0      -    18K  -
rpool/ROOT/opensolaris                                         62.7
   M   142G  2.23G  legacy
rpool/ROOT/opensolaris-1                                       2.31
   G   142G  2.24G  legacy
rpool/ROOT/opensolaris-1@install                               4.66
   M      -  2.22G  -
rpool/ROOT/opensolaris-1@static:-:2008-04-29-17:59:13          5.49
   M      -  2.23G  -
rpool/ROOT/opensolaris-1/opt                                   3.60
   M   142G  3.60M  /opt
rpool/ROOT/opensolaris-1/opt@install
   0      -  3.60M  -
rpool/ROOT/opensolaris-1/opt@static:-:2008-04-29-17:59:13
   0      -  3.60M  -
rpool/ROOT/opensolaris/opt
   0   142G  3.60M  /opt
rpool/export                                                   18.9
   M   142G    19K  /export
rpool/export@install                                             15
   K      -    19K  -
rpool/export/home                                              18.9
   M   142G  18.9M  /export/home
rpool/export/home@install                                        18
   K      -    21K  -
```

After doing some configuration, you can create an boot environment called `opensolaris-baseline` : It's really easy. You just have to create a new boot environment:

```
# beadm create -e opensolaris-1 opensolaris-baseline
```

We will not work with this environment. We use it as a baseline, as a last resort when we destroy our running environment. To run the system we will create another snapshot:

```
# beadm create -e opensolaris-1 opensolaris-work
```

Now let's look into the list of our boot environments.

```
jmoekamp@glamdring:~# beadm list

BE                   Active Active on Mountpoint Space
Name                        reboot               Used
----                 ------ --------- ---------- -----
opensolaris-baseline no     no        -          53.5K
opensolaris-1        yes    yes       legacy     2.31G
opensolaris          no     no        -          62.72M
opensolaris-work     no     no        -          53.5K
```

We activate the `opensolaris-work` boot environment:

```
jmoekamp@glamdring:~# beadm activate opensolaris-work
```

Okay, let's look at the list of boot environments again.

```
jmoekamp@glamdring:~# beadm list

BE                   Active Active on Mountpoint Space
Name                        reboot               Used
----                 ------ --------- ---------- -----
opensolaris-baseline no     no        -          53.5K
opensolaris-1        yes    no        legacy     24.5K
opensolaris          no     no        -          62.72M
opensolaris-work     no     yes       -          2.31G
jmoekamp@glamdring:~#
```

You will see that the `opensolaris-1` snapshot is still active, but that the `opensolaris-work` boot environment will be active at the next reboot. Okay, now reboot:

```
jmoekamp@glamdring:~# beadm list

BE                   Active Active on Mountpoint Space
Name                        reboot               Used
----                 ------ --------- ---------- -----
opensolaris-baseline no     no        -          53.5K
opensolaris-1        no     no        -          54.39M
opensolaris          no     no        -          62.72M
opensolaris-work     yes    yes       legacy     2.36G
```

Okay, you see that the boot environment `opensolaris-work` is now active and it's activated for the next reboot (until you activate another boot environment).

Now we can reboot the system again. GRUB comes up and it will default to the `opensolaris-work` environment. Please remember on which position you find `opensolaris-baseline`

in the boot menu. You need this position in a few moments. After a few seconds, you can log into the system and work with it.

Now let's drop the atomic bomb of administrative mishaps to your system. Log in to your system, assume the root role and do the following stuff:

```
# cd /
# rm -rf *
```

You know what happens. Depending on how fast you are able to interrupt this run, you will end up somewhere between a slightly damaged system and a system fscked up beyond any recognition. Normally the system would send you to the tapes now. But remember - you have some alternate boot environments.

Reboot the system, wait for GRUB. You may have garbled output, so it's hard to read the output from GRUB. Choose `opensolaris-baseline`. The system will boot up quite normally.

You need a terminal window now. How you get such a terminal window depends on the damage incurred. The boot environment snapshots don't cover the home directories, so you may not have a home directory any more. I will assume this for this example: you can get a terminal window by clicking on "Options", then "Change Session" and choose "Failsafe Terminal" there.

Okay, log in via the graphical login manager; a xterm will appear. At first we delete the defunct boot environment:

```
# beadm destroy opensolaris-work
Are you sure you want to destroy opensolaris-work? This action
   cannot be undone (y/[n]):
y
```

Okay, now we clone the `opensolaris-baseline` environment to form a new `opensolaris-work` environment.

```
# beadm create -e opensolaris-baseline opensolaris-work
```

We reactivate the `opensolaris-work` boot environment:

```
# beadm activate opensolaris-work
```

Now, check if you still have a home directory for your user:

```
# ls -l /export/home/jmoekamp
/export/home/jmoekamp: No such file or directory
```

If your home directory no longer exists, create a new one:

```
# mkdir -p /export/home/jmoekamp
# chown jmoekamp:staff /export/home/jmoekamp
```

Now reboot the system:

```
# reboot
```

Wait a few moments. The system starts up. GRUB defaults to `opensolaris-work` and the system starts up normally, without any problems, in the condition that the system had when you created the `opensolaris-baseline` boot environment.

```
# beadm list

BE                     Active Active on Mountpoint Space
Name                          reboot               Used
----                   ------ --------- ---------- -----
opensolaris-baseline   no     no        -          3.18M
opensolaris-1          no     no        -          54.42M
opensolaris            no     no        -          62.72M
opensolaris-work       yes    yes       legacy     2.36G
```

Obviously you may have to recover your directories holding your own data. It's best practice to make snapshots of these directories on a regular schedule, so that you can simply promote a snapshot to recover a good version of the directory.

4.3. Conclusion

You see, recovering from a disaster in a minute or two is a really neat feature. Snapshotting opens a completely new way to recover from errors. Unlike with LiveUpgrade, you don't need extra disks or extra partitions and, as ZFS snapshots are really fast, creating alternate boot environments on ZFS is extremely fast as well.

At the moment this feature is available on Opensolaris 2008.05 only. With future updates it will find its way into Solaris as well.

5. Working with the Service Management Facility

Solaris10/Opensolaris

5.1. Introduction

The Service Management Facility is a quite new feature. But sometimes I have the impression that the most used feature is the capability to use old legacy init.d scripts. But once you use SMF with all its capabilities, you see an extremely powerful concept.

5.1.1. init.d

For a long time, the de-facto standard of starting up services was the init.d construct. This concept is based of startup scripts. Depending from their parametrisation they start, stop or restart a service. The definition of runlevels (what has to be started at a certain stage of booting) and the sequencing is done by linking this startup scripts in a certain directory and the naming of link.

This mechanism worked quite good, but has some disadvantages. You can't define dependencies between the services. You emulate the dependencies by sorting the links, but that's more of a kludge as a solution. Furthermore the init.d scripts run only once. When the service stops, there are no means to start it again by the system (you have to login to the system and restart it by using the init.d script directly or using other automatic methods)

With init.d a service (like httpd on port 80) is just a consequence of running scripts, not a configurable entity in itself.

5.1.2. Service Management Facility

SMF was invented to solve many of the problems of the old startup mechanism. Most problems from resulting from the init.d framework result from a lack of knowledge of the system about services it's running. What do I need to run my service? Is this services

needed for other services? What is the status of a service? Should I restart another service (e.g. database) to circumvent problems in another service (an old web application for example)? Okay, an expert has the knowledge to do such tasks manually ... but do you want to wake up at night, just to restart this fscking old application?

The concepts of SMF enables the admin to put this knowledge into a machine readable format, thus the machine can act accordingly. This knowledge about services makes the SMF a powerful tool to manage services at your system. SMF enables the system to:

- starting, restarting and stopping services according to their dependencies
- resulting from this the system startup is much faster, as services are started in a parallel fashion when possible
- When a service fails, SMF restarts this service
- the delegation of tasks like starting, stopping and configuration of services to non-root users
- and much more

The following tutorial wants to give you some insights to SMF. Have fun!

5.2. The foundations of SMF

The additional capabilities of the SMF comes at a price. SMF has to know more about your services. Most of the new components of the SMF has to do with this capabilities. So we have to define some foundations before doing some practical stuff.

5.2.1. Service and Service Instance

At first we start with the service and the service instance. This difference is important. The service is the generic definition how a service is started. The service instance is the exact configuration of a service. A good example is a webserver. The service defines the basic methods how to start or stop an apache daemon, the service instance contains the information, how an specific configuration should work (which port to listen on, the position of the config file). A service can define to allow just one instance, as well you can define, you can have multiple instances of a certain service.

But: A service doesn't have to be a long running process. Even a script that simply exits after executing (e.g. for some commands to do network tuning) is a special kind of a service in the sense of SMF.

5.2.2. Milestone

A milestone is somehow similar to the old notion of runlevel. With milestones you can group certain services. Thus you don't have to define each service when configuring the dependencies, you can use a matching milestones containing all the needed services.

Furthermore you can force the system to boot to a certain milestone. For example: Booting a system into the single user mode is implemented by defining a single user milestone. When booting into single user mode, the system just starts the services of this milestone.

The milestone itself is implemented as a special kind of service. It's an anchor point for dependencies and a simplification for the admin. Furthermore some of the milestones including single-user, multi-user and multi-user-server contain methods to execute the legacy scripts in rc*.d

5.2.3. Fault Manager Resource Identifier

Every service instance and service instance has a unique name in the system do designate it precisely. This name is called Fault Management Resource Identifier. For example the SSH server is called: `svc:/network/ssh:default` .The FMRI is divided by the `:` into three parts. The first part designates the resource as an service. The second parts designates the service. The last part designates the service instance. Into natural language: It's the default configuration of the ssh daemon.

But why is this identifier called Fault Manager Resource Identifier? Fault Management is another important feature in Solaris. The FM has the job to react automatically to failure events. The failure of a service isn't much different to the failure of a hard disk or a memory chip. You can detect it and you can react to it. So the Fault Management is tightly integrated into the service management.

5.2.4. Service Model

As I mentioned before, not at all services are equal and they have different requirements to starting them. Thus the System Management Facility knows different service models:

5.2.5. Transient service

The simplest service model is `transient`. You can view it as a script that gets executed while starting the system without leaving a long-lived server process. You use it for scripts to tune or config things on your system. A good example is the script to configure the core dumping via coreadm.

A recommendation at this place: Don't use the `transient` model to transform your old startup scripts. Albeit possible, you loose all the advantages of SMF. In this case it would be easier to use the integrated methods to use legacy init.d scripts.

5.2.6. Standalone model

The third service model is the `standalone` model. The inner workings of this model are really simple. Whenever the forked process exits, SMF will start it again.

5.2.7. Contract service

The standard model for services is `contract`. This model uses a special facility of the Solaris Operating Environment to monitor the processes

5.2.8. A short digression: Contracts

Did you ever wondered about the `/system/contract` filesystems. It's the most obvious sign of the contracts. The contract model is based on a kernel level construct to manage the relationships between a process and other kernel managed resources. Such resources are processor sets, memory, devices and most important for SMF other processes. Process contracts describe the relation between a process and its child process. The contract subsystem generates events available to other processes via listeners. Possible events are:

Your system already use this contracts. Let's have a look at sendmail.

```
# ptree -c `pgrep sendmail`
[process contract 1]
  1     /sbin/init
    [process contract 4]
      7     /lib/svc/bin/svc.startd
        [process contract 107]
          792   /usr/lib/sendmail -bd -q15m -C /etc/mail/local.
             cf
```

Table 5.1.: Events of the contract subsystem

Event	Description
empty	the last process in the contract has exited
process exit	a process in the process contract has exited
core	a member process dumped core
signal	a member process received a fatal signal from outside the
contract	a member process received a fatal signal from outside the contract
hwerr	a member process has a fatal hardware error

```
794   /usr/lib/sendmail -Ac -q15m
```

With the `-c` option ptree prints the contract IDs of the processes. In our example, the sendmail processes run under the contract ID 107. With ctstat we can lookup the contents of this contract:

```
# ctstat -vi 107
CTID    ZONEID  TYPE    STATE   HOLDER  EVENTS  QTIME   NTIME
107     0       process owned   7       0       -       -
        cookie:                 0x20
        informative event set:  none
        critical event set:     hwerr empty
        fatal event set:        none
        parameter set:          inherit regent
        member processes:       792 794
        inherited contracts:    none
```

Contract 107 runs in the global zone. It's an process id and it was created by process number 7 (the svc.startd). There wasn't any events so far. The contract subsystem should only throw critical evens when the processes terminate due hardware errors and when no processes are left. At the moment there are two processes under the control of the contract subsystem (the both processes of the sendmail daemon)

Let's play around with the contracts:

```
# ptree -c `pgrep sendmail`
[process contract 1]
```

```
1     /sbin/init
  [process contract 4]
    7     /lib/svc/bin/svc.startd
      [process contract 99]
        705   /usr/lib/sendmail -bd -q15m -C /etc/mail/local.
           cf
        707   /usr/lib/sendmail -Ac -q15m
```

You can listen to the events with the `ctwatch`:

```
# ctwatch 99
CTID    EVID    CRIT ACK CTTYPE   SUMMARY
```

Okay, open a second terminal window to your system and kill the both sendmail processes:

```
# kill 705 707
```

After we submitted the kill, the contract subsystem reacts and sends an event, that there are no processes left in the contract.

```
# ctwatch 99
CTID    EVID    CRIT ACK CTTYPE   SUMMARY
99      25      crit no  process  contract empty
```

Besides of ctwatch the event there was another listener to the event: SMF. Let's look for the sendmail processes again.

```
# ptree -c `pgrep sendmail`
[process contract 1]
  1     /sbin/init
    [process contract 4]
      7     /lib/svc/bin/svc.startd
        [process contract 103]
          776   /usr/lib/sendmail -bd -q15m -C /etc/mail/local.
             cf
          777   /usr/lib/sendmail -Ac -q15m
```

Et voila, two new sendmail processes with a different process id and a different process contract ID. SMF has done its job by restarting sendmail.

To summarize things: The SMF uses the contracts to monitor the processes of a service. Based on this events SMF can take action to react on this events. Per default, SMF stops and restart a service, when any member of the contract dumps core, gets a signal or dies due a hardware failure. Additionally the SMF does the same, when theres no member process left in the contract.

5.2.9. Service State

Fault management brings us to the next important definition. Every service instance has a service state. This state describes a point in the lifecycle of the process:

Table 5.2.: Description of service states

Service state	Description
degraded	The service runs, but somehow the startup didn't fully succeeded and thus the service has only limited capabilities
disabled	¿The service was enabled by the admin, and thus SMF doesn't attempt to start it
online	The services is enabled and the bring-up of the service was successful
offline	The service is enabled, but the service hasn't been started so far, as dependencies are not fulfilled.
maintenance	The service didn't started properly and exited with an error code other than 0. For example because of typos in config files
legacy_run	This is an special service state. It's used by the SMF for services under the control of the restarter for legacy init.d scripts

Each service under the control of the SMF has an service state throughout it whole lifetime on the system.

5.2.10. Service Configuration Repository

All the configurations about the services in the Service Configuration Repository. It's the central database regarding the services. This database is backed up and snapshotted in a regular manner. So it's easy to fall back to a known running state of the repository (after you or a fellow admin FOOBARed the service configuration)

5.2.11. Dependencies

The most important feature of SMF is the knowledge about dependencies. In SMF you can define two kinds of dependency in a services:

- which services this service depends on
- the services that depend on this service

This second way to define a dependency has an big advantage. Let's assume, you have a new service. You want to start it before an other service. But you don't want to change the object itself (perhaps, you need this service only in one special configuration and the normal installation doesn't need your new service ... perhaps it's the authentication daemon for a hyper-special networking connection ;)). By defining, that another service depends on your service, you don't have to change the other one.

I will show you how to look up the dependencies in the practical part of this tutorial.

5.2.12. Master Restarter Daemon and Delegated Restarter

Okay, now you have all the data. But you need someone to do something: For this task you have the SMF Master Restarter Daemon. This daemon reads the Service Configuration Repository and acts accordingly. It starts a services when all its dependencies are fulfilled. By this simple rule all services will be started in the process of booting until there are no enabled services left in the offline state.

But not all processes are controlled by the Master Restarter. The Master Restarted can delegate this task to other restarters, thus the are called SMF Delegated Restarter Daemons.

5.2.13. Delegated Restarter for inetd services

The most obvious example for such an delegated restarter is inetd, the daemon to start network demons only on demand. One important effect of this is a change in behavior of the inetd. ¡code¿inetd.conf¡/code¿ isn't used to control inetd anymore. The Solaris services which were formerly configured using this file are now configured via SMF. So you don't edit the inetd.conf to disable or enable an inetd service. You use the same commands like for all other services.

5.2.14. Enough theory

Enough theory, let's do some practical stuff ...

5.3. Working with SMF

After so much theory SMF may look a little bit complex but it isn't. For the admin it's really simple. You can control the complete startup of the system with just a few commands.

5.3.1. What's running on the system

At first let's have a look on all services running on the system:

```
# svcs
legacy_run      10:04:44 lrc:/etc/rc3_d/S84appserv
disabled        10:04:22 svc:/system/xvm/domains:default
online          10:03:48 svc:/system/svc/restarter:default
offline         10:03:54 svc:/network/smb/server:default
```

This is only a short snippet of the configuration. The output of this command is 105 lines long on my system. But you services in several service states in it. For example I hadn't enabled xvm on my system (makes no sense, as this Solaris is already virtualized, and the smb server is still online.

Let's look after a certain service

```
 #  svcs name-service-cache
STATE          STIME    FMRI
online         10:08:01 svc:/system/name-service-cache:default
```

The output is separated into three columns. The first shows the service state, the second the time of the last start of the service. The last one shows the exact name of the service.

5.3.2. Starting and stopping a service

Okay, but how do I start the service, how do I use all this stuff:let's assume, you want to disable sendmail. At first we check the current state:

```
# svcs sendmail
STATE          STIME    FMRI
online         10:23:19 svc:/network/smtp:sendmail
```

Now we disable the service. It's really straight forward:

```
# svcadm disable sendmail
```

Let's check the state again.

```
# svcs sendmail
STATE          STIME    FMRI
disabled       10:23:58 svc:/network/smtp:sendmail
```

Okay, a few days later we realize that we need the sendmail service on the system. No problem we enable it again:

```
# svcadm enable  sendmail
# svcs sendmail
STATE          STIME    FMRI
online         10:25:30 svc:/network/smtp:sendmail
```

The service runs again. Okay, we want to restart the service. This is quite simple, too

```
# svcadm restart sendmail
# svcs sendmail
STATE          STIME    FMRI
online*        10:25:55 svc:/network/smtp:sendmail
# svcs sendmail
STATE          STIME    FMRI
online         10:26:04 svc:/network/smtp:sendmail
```

Did you notice the change in the STIME column. The service has restarted. By the way: STIME doesn't stand for "start time". It's a short form for "State Time". It shows, when the actual state of the services was entered.

Okay, now let's do some damage to the system. We move the config file for sendmail, the glorious sendmail.cf. The source of many major depressions under sys admins.

```
# mv /etc/mail/sendmail.cf /etc/mail/sendmail.cf.old
# svcadm restart sendmail
# svcs sendmail
STATE          STIME    FMRI
offline        10:27:09 svc:/network/smtp:sendmail
```

Okay, the service went in the offline state. Offline? At first, the maintenance state would look more sensible. But let's have a look in some diagnostic informations. With `svcs -x` you can print out fault messages regarding services.

```
# svcs -x
svc:/network/smtp:sendmail (sendmail SMTP mail transfer agent)
 State: offline since Sun Feb 24 10:27:09 2008
Reason: Dependency file://localhost/etc/mail/sendmail.cf is
   absent.
   See: http://sun.com/msg/SMF-8000-E2
   See: sendmail(1M)
   See: /var/svc/log/network-smtp:sendmail.log
Impact: This service is not running.
```

The SMF didn't even try to start the service. There is an dependency implicit to the service.

```
 svcprop sendmail
config/value_authorization astring solaris.smf.value.sendmail
config/local_only boolean true
config-file/entities fmri file://localhost/etc/mail/sendmail.cf
config-file/grouping astring require_all
config-file/restart_on astring refresh
config-file/type astring path
[..]
```

The service configuration for sendmail defines a dependency to the config-file /etc/-mail/sendmail.cf. Do you remember the definition of the service states? A service stays in offline mode until all dependencies are fulfilled. We renamed the file, the dependencies isn't fulfilled. The restart leads correctly to the "offline state"

Okay, we repair the damage:

```
# mv /etc/mail/sendmail.cf.old /etc/mail/sendmail.cf
```

And now we restart the service

```
# svcadm refresh sendmail
# svcs sendmail
STATE          STIME    FMRI
online         10:33:54 svc:/network/smtp:sendmail
```

All is well, the service in online again

5.3.3. Automatic restarting of a service

Okay, let's test another capability of the SMF. The kill the sendmail daemon.

```
# svcs sendmail
STATE          STIME    FMRI
online         10:33:54 svc:/network/smtp:sendmail
# pkill "sendmail"
# svcs sendmail
STATE          STIME    FMRI
online         10:38:24 svc:/network/smtp:sendmail
```

The SMF restarted the daemon automatically as you can see from the stime-column

5.3.4. Obtaining the configuration of a service

Okay, as I wrote before every service has some configuration in the SMF Service Configuration Repository. You can dump this configuration with the `svcprop` command. Let's print out the configuration for the name service cache:

```
# svcprop svc:/system/name-service-cache:default
general/enabled boolean false
general/action_authorization astring solaris.smf.manage.name-
   service-cache
[..]
restarter/state astring online
restarter/state_timestamp time 1203844081.231748000
general_ovr/enabled boolean true
```

5.3.5. Dependencies

But how do I find out the dependencies between services. The `svcadm` commands comes to help:

The `-d` switch shows you all services, on which the service depends. In this example we check this for the ssh daemon.

```
# svcs -d ssh
STATE          STIME    FMRI
disabled        8:58:07 svc:/network/physical:nwam
online          8:58:14 svc:/network/loopback:default
online          8:58:25 svc:/network/physical:default
online          8:59:32 svc:/system/cryptosvc:default
online          8:59:55 svc:/system/filesystem/local:default
online          9:00:12 svc:/system/utmp:default
online          9:00:12 svc:/system/filesystem/autofs:default</
   code></
```

To check, what services depend on ssh, you can use the `-D` switch:

```
# svcs -D ssh
STATE          STIME    FMRI
online          9:00:22 svc:/milestone/multi-user-server:
   default
```

There is no further service depending on ssh. But the milestone ¡code¿multi-user-server¡/code¿ depends on ssh. As long the ssh couldn't started successfully, the `multi-user-server` milestone can't be reached.

5.4. Developing for SMF

Okay, now you know how to do basic tasks. But how to use SMF for your own applications. I will use OpenVPN as an example.

5.4.1. Prerequisites

A good source for this program is Blastwave[1]. Please install the package `tun` and `openvpn`

I want to show a running example, thus we have to do some work. It will be just a simple static shared key configuration, as this is a SMF tutorial, not one for OpenVPN. We will use theoden and gandalf again. gandalf will be the server. theoden the client.

```
10.211.55.201 gandalf
10.211.55.200 theoden
```

5.4.2. Preparing the server

Okay ... this is just a short configuration for an working OpenVPN server.

```
# mkdir /etc/openvpn
# cd /etc/openvpn
# openvpn --genkey --secret static.key
# openvpn --dev tun --ifconfig 192.16.1.1 172.16.1.2 --secret
   static.key --daemon
# scp static.key jmoekamp@10.211.55.200:/tmp/static.key
```

Now just leave this terminal this way.

[1] `http://www.blastwave.org`

5.4.3. Preparing the client

Okay, we need some basic configurations to get the client side of OpenVPN working, even when it's under control of the SMF ;)

```
# mkdir /etc/openvpn
# mv /tmp/static.key /etc/openvpn
# cd /etc/openvpn/
# ls -l
total 2
-rw-------   1 jmoekamp other        636 Feb 27 16:11 static.
   key
# chown -R root:root /etc/openvpn
# chmod 600 static.key
```

5.4.4. Before working with SMF itself

At first I remove this stinking init.d links. We don't need them anymore:

```
# rm /etc/rc0.d/K16openvpn
# rm /etc/rc1.d/K16openvpn
# rm /etc/rc2.d/S60openvpn
# rm /etc/rcS.d/K16openvpn
```

Okay, and now let's hack the startup script....? Wrong! SMF can do many task for you, but this needs careful planing. You should answer yourself some questions:

1. What variables make a generic description of a service to a specific server?
2. How do I start the process? How do I stop them? How can I force the process to reload its config?
3. Which services are my dependencies? Which services depend on my new service?
4. How should the service react in the case of a failed dependency?
5. What should happen in the case of a failure in the new service.

Okay, let's answer this questions for our OpenVPN service. The variables for our OpenVPN client are the hostname of the remote hosts and the local and remote IP of the VPN tunnel . Besides of this the filename of the secret key and the tunnel device may differ, thus it would be nice to keep them configurable.

Starting openvpn is easy. We have just to start the openvpn daemon with some command line parameters. We stop the service by killing the process. And a refresh is done via stopping and starting the service.

We clearly need the networking to use a VPN service. But networking isn't just bringing up the networking cards. You need the name services for example. So make things easier, the service don't check for every networking service to be up and running. We just define an dependency for the "network" milestone.

As it make no sense to connect to a server without a network it looks like a sensible choice to stop the service in case of a failed networking. Furthermore it seems a good choice to restart the service when the networking configuration has changed. Perhaps we modified the configuration of the name services and the name of the OpenVPN server resolves to a different IP.

What should happen in the case of a exiting OpenVPN daemon? Of course it should started again.

Okay, now we can start with coding the scripts and xml files.

5.4.5. The Manifest

Okay, as I wrote before, the manifest is the source of the configuration of a service. Thus we have to write such a manifest. The manifest is a XML file. Okay, at first we obey the gods of XML and do some definitions:

```
# cat openvpn.xml
<?xml version="1.0"?>
<!DOCTYPE service_bundle SYSTEM "/usr/share/lib/xml/dtd/
   service_bundle.dtd.1">
<service_bundle type='manifest' name='openvpn'>
```

Okay, at first we have to name the service:

```
name='application/network/openvpn'
type='service'
version='1'>
```

In this example, we define some simple dependencies. As I wrote before: Without networking a VPN is quite useless, thus the OpenVPN service depends on the reached `network milestone`.

```
<dependency
name='network'
grouping='require_all'
restart_on='none'
type='service'>
<service_fmri value='svc:/milestone/network:default'>
<dependency>
```

In this part of the manifest we define the exec method to start the service. We use a script to start the service. The `%m` is a variable. It will be substituted with the name of the called action. In this example it would be expanded to =verb=/lib/svc/method/openvpn start=.

```
<exec_method
type='method'
name='start'
exec='/lib/svc/method/openvpn %m'
timeout_seconds='2' />
```

Okay, we can stop OpenVPN simply by sending a SIGTERM signal to it. Thus we can use a automagical exec method. In case you use the `:kill` SMF will kill all processes in the actual contract of the service.

```
<exec_method
type='method'
name='stop'
exec=':kill'
timeout_seconds='2'>
</exec_method>
```

Okay, thus far we've only define the service. Let's define a service. We call the instance `theoden2gandalf` for obvious names. The service should run with root privileges. After this we define the properties of this service instance like the remote host or the file with the secret keys.

```
<instance name='theoden2gandalf' enabled='false'>
<method_context>
<method_credential user='root' group='root'>
</method_context>
<property_group name='openvpn' type='application'>
<propval name='remotehost' type='astring' value='gandalf'>
<propval name='secret' type='astring' value='/etc/openvpn/
   static.key' />
<propval name='tunnel_local_ip' type='astring' value
   ='172.16.1.2'>
<propval name='tunnel_remote_ip' type='astring' value
   ='172.16.1.1' />
<propval name='tunneldevice' type='astring' value='tun'>
</property_group>
</instance>
```

At the end we add some further metadata to this service:

```
<stability value='Evolving' />
<template>
<common_name>
<loctext xml:lang='C'>OpenVPN</loctext>
</common_name>
<documentation>
<manpage title='openvpn' section='1'>
<doc_link name='openvpn.org' uri='http://openvpn.org'>
</documentation>
</template>
</service>
</service_bundle>
```

I saved this xml file to my home directory under `/export/home/jmoekamp/openvpn.xml`

5.4.6. The exec methods script - general considerations

Okay, we referenced a script in the exec method. This script is really similar to a normal init.d script. But there are some important differences. As theres no parallel startup of services in init.d most scripts for this system bring-up method tend to return as quickly as possible. We have to change this behavior.

Scripts for transient or standalone services should only return in the case of the successful execution of the complete script or when we've terminated the process. For services under control of the contract mechanism the script should at least wait until the processes of the service generate some meaningful error messages, but they have to exit, as SMF would consider the service startup as failed, when the script doesn't return after

There are some general tips:

- When you write your own `stop` method don't implement it in a way that simply kills all processes with a certain name (e.g. `pkill "openvpn"`) this was and is really bad style, as there may be several instances of service. Just using the name to stop the processes would cause unneeded collateral damage.
- It's a good practice to include the `/lib/svc/share/smf_include.sh`. It defines some variables for errorcodes to ease the development of the method scripts.

5.4.7. Implementing a exec method script

We store configuration properties in the Service Component repository. It would be nice to use them for configuration. Thus we have to access them.

Here comes the svcprop command to help:

```
# svcprop -p openvpn/remotehost svc:/application/network/
   openvpn:theoden2gandalf
gandalf
```

With a little bit of shell scripting we can use this properties to use them for starting our processes.

```
#!/bin/sh

. /lib/svc/share/smf_include.sh

getproparg() {
val=`svcprop -p $1 $SMF_FMRI`
[ -n "$val" ] && echo $val
}

if [ -z "$SMF_FMRI" ]; then
echo "SMF framework variables are not initialized."
exit $SMF_EXIT_ERR
fi

OPENVPNBIN='/opt/csw/sbin/openvpn'
REMOTEHOST=`getproparg openvpn/remotehost`
SECRET=`getproparg openvpn/secret`
TUN_LOCAL=`getproparg openvpn/tunnel_local_ip`
TUN_REMOTE=`getproparg openvpn/tunnel_remote_ip`
DEVICETYPE=`getproparg openvpn/tunneldevice`

if [ -z "$REMOTEHOST" ]; then
echo "openvpn/remotehost property not set"
exit $SMF_EXIT_ERR_CONFIG
fi

if [ -z "$SECRET" ]; then
echo "openvpn/secret property not set"
exit $SMF_EXIT_ERR_CONFIG
fi

if [ -z "$TUN_LOCAL" ]; then
echo "openvpn/tunnel_local_ip property not set"
exit $SMF_EXIT_ERR_CONFIG
fi

if [ -z "$TUN_REMOTE" ]; then
```

```
echo "openvpn/tunnel_remote_ip property not set"
exit $SMF_EXIT_ERR_CONFIG
fi

if [ -z "$DEVICETYPE" ]; then
echo "openvpn/tunneldevice property not set"
exit $SMF_EXIT_ERR_CONFIG
fi

case "$1" in
'start')
$OPENVPNBIN --daemon --remote $REMOTEHOST --secret $SECRET --
   ifconfig $TUN_LOCAL $TUN_REMOTE --dev $DEVICETYPE
;;

'stop')
echo "not implemented"
;;

'refresh')
echo "not implemented"
;;

*)
echo $"Usage: $0 {start|refresh}"
exit 1
;;

esac
exit $SMF_EXIT_OK
```

I saved this script to my home directory under `/export/home/jmoekamp/openvpn`.

5.4.8. Installation of the new service

Okay, copy the script to `/lib/svc/method/`:

```
# cp openvpn /lib/svc/method
# chmod +x /lib/svc/method/openvpn
```

After this step you have to import the manifest into the Service Configuration Repository:

```
# svccfg validate /export/home/jmoekamp/openvpn.xml
# svccfg import /home/jmoekamp/openvpn.xml
```

5.4.9. Testing it

Let's test our brand new service:

```
# ping 172.16.1.2
^C
```

The OpenVPN service isn't enabled. Thus there is no tunnel. The ping doesn't get through. Now we enable the service and test it again.

```
# svcadm enable  openvpn:theoden2gandalf
# ping 172.16.1.2
172.16.1.2 is alive
```

Voila ... SMF has started our brand new service. When we look into the list of services, we will find it:

```
# svcs openvpn:theoden2gandalf
STATE          STIME    FMRI
online         18:39:15 svc:/application/network/openvpn:
   theoden2gandalf
```

When we look into the process table, we will find the according process:

```
# /usr/ucb/ps -auxwww | grep "openvpn" | grep -v "grep"
root      1588  0.0  0.5 4488 1488 ?        S 18:39:15  0:00 /
   opt/csw/sbin/openvpn --daemon --remote gandalf --secret /etc
   /openvpn/static.key --ifconfig 172.16.1.2 172.16.1.1 --dev
   tun
```

Okay, we doesn't need the tunnel any longer after a few day,thus we disable it:

```
# svcadm disable openvpn:theoden2gandalf
# /usr/ucb/ps -auxwww | grep "openvpn" | grep -v "grep"
#
```

No process left.

5.5. Conclusion

Okay, I hope I was able to give you some insights into the Service Management Framework. It's a mighty tool and this article only scratched on the surface of the topic. But there are several excellent resources out there.

5.6. Do you want to learn more

Documentation

Solaris 10 System Administrator Collection - Basic Administration - Managing Services

man page - smf(5)

FAQ

opensolaris.org: SMF(5) FAQ

Other resources

Bigadmin - Solaris Service Management Facility - Quickstart Guide

Bigadmin - Solaris Service Management Facility - Service Developer Introduction

SMF shortcuts on wikis.sun.com

cuddletech.com: An SMF Manifest Cheatsheet

6. Solaris Resource Manager

Solaris10/Opensolaris

Resource Management is an rather old feature in Solaris, albeit it got more mind share since it got a really important part of the Solaris Zones. But this tutorial doesn't focus on the usage of Resource Management in conjunction with the zone configuration. I want to go to the basics, because at the end the zone configuration just uses this facilities to control the resource consumption of zones. Secondly you can use this knowledge to limit resource usage in a zone itself.

Resource Management was introduced to solve one important question. You can run multiple programs and services at once in a single instance of the operating system, but how do I limit the consumption of resources of a single application? How do I prevent a single program from consuming all the resources leaving nothing to others? Resource Management in Solaris solves this class of problems.

6.1. Why do you need Resource Management?

Unix systems were capable to execute a multitude of independent services on one system since its introduction many years ago. But there was a problem: What happens in case of a process running amok? A user eating away resources of the system? You have to control the allocation of this resources. How many CPU does a process get? How many file descriptors are allocated to a user? The Solaris Resource Management is capable to put other processes under the control of an user configurable ruleset. Thus you can prevent a situation where your system is down, because a single process takes all resources.

With such an ability, the usage of a single system for a multitude of services becomes more manageable and thus more feasible.

6.2. Definitions

Okay, as usual, this technology has its own jargon. So I have to define some of it at first:

- *Tasks*: A task is a group of processes. For example when you log into a system and do some work all the steps you've done are an task until you logout or open a new task. Another example would be a webserver. It consists out of a multitude of processes, but they are all part of the same task. The database server on the same machine may have a completely different task id.
- *Projects*: A project is a group of tasks. For example you have a webserver. It consists out of the processes of the database task and the webserver task.
- *Zones*: From the perspective of the resource management, a Solaris Zone is just a group of one or more projects.

6.3. The basic idea of Solaris Resource Management

The basic idea of resource management is to use the grouping of processes and to impose limits on a group on processes defined by one of the above entities. A limit on the zone would limit all projects in a zone. A limit on the project would limit all tasks grouped by the project. A limit on a task would limit all processes grouped by the tasks.

Let's take an example: There is a resource control called `max-lwps`. It controls the amount of processes. This control exists as `task.max-lwps`, `project.max-lwps` and `zone.max-lwps`. With this control you can define limits for the number of processes in a group of processes defined by one of the entities mentioned above.

You want to ensure, that you have 1000 processes at maximum in your zone `webserver`. You can define it by using `zone.max-lwps` for the zone `webserver`. You want to ensure, that the www.c0t0d0s0.org webserver project consisting out of mysql and apache processes uses 250 processes at maximum. You use the `project.max-lwps` for the project `webserver-c0t0d0s0`. You want to limit the httpd-server itself to 100 processes. You impose this limit by using the `task.max-lwps` control.

How do you configure this? This will be the topic of this tutorial. It sounds complex, but it isn't, once you've understand the entities.

6.4. How to work with projects and tasks

Even when you never heard of constructs like tasks and projects you already use them, as there are a few projects installed per default. For example whenever you log into your system as a regular user, all you do is part of the project `default` because your user is member of this project:

```
# su jmoekamp
# id -p
uid=100(jmoekamp) gid=1(other) projid=3(default)
```

When you assume the privileges of the root user you work as a member of the user.root project.

```
$ su root
Password:
# id -p
uid=0(root) gid=0(root) projid=1(user.root)
```

Let's have another look at the alread running processes of your system.

```
# ps -ef -o pid,user,zone,project,taskid,args
  PID      USER      ZONE  PROJECT TASKID COMMAND
    0      root    global   system      0 sched
[...]
  126    daemon    global   system     22 /usr/lib/crypto/kcfd
  646 jmoekamp    global  default     73 /usr/lib/ssh/sshd
[...]
  413      root    global user.root     72 -sh
  647 jmoekamp    global  default     73 -sh
[...]
  655      root    global user.root     74 ps -ef -o pid,user,zone
     ,project,taskid,args
  651      root    global user.root     74 sh
```

As you see from the output of `ps` you already use some projects, that are default on every Solaris system. And even the concept of task is in use right now. Whenever a user log into a Solaris system, a new task is opened. Furthermore you will see, that all your services started by the Service management facility have their own task id. The SMF starts every service as a new task.

```
# ps -ef -o pid,user,zone,project,taskid,args | grep " 74 "
  653      root    global user.root     74 bash
  656      root    global user.root     74 ps -ef -o pid,user,zone
     ,project,taskid,args
  657      root    global user.root     74 bash
  651      root    global user.root     74 sh
```

The projects are stored in a file per default, but you can use LDAP or NIS for a common project database on all your systems:

```
# cat /etc/project
system:0::::
```

```
user.root:1::::
noproject:2::::
default:3::::
group.staff:10::::
```

A freshly installed system has already this pre-defined projects:

Table 6.1.: Factory-configured project in Solaris

Project	Description
system	The system project is used for all system processes and daemons.
user.root	All root processes run in the user.root project.
no.project	The noproject project is a special project for IP Quality of Service. You can savely ignore it for this tutorial
default	When there isn't a matching group, this is the catch-all. A user without an explicitly defined project is member of this project
group.staff	The group.staff project is used for all users in the group staff

Okay, but how do we create our own projects? It's really easy:

```
[root@theoden:~]$ projadd -p 1000 testproject
[root@theoden:~]$ projmod -c "Testserver project" testproject
[root@theoden:~]$ projdel testproject
```

We created the project `testproject` with the project id `1000`. Then we modified it by adding informations to the comment field. After this, we've deleted the project.

6.5. A practical example

Okay, I will use a practical example, now. We working as an administrator at the Unseen University. We've got a new system and want to create the users.

```
# useradd alice
# useradd bob
# useradd mike
```

```
# useradd laura
```

Looks familiar? I've omitted to set the password here. But of course you have to set one. Now we create some projects for two classes:

```
# projadd class2005
# projadd class2006
```

Okay. This projects have no users. We assign the project to our users.

```
# usermod -K project=class2005 alice
# usermod -K project=class2005 bob
# usermod -K project=class2006 mike
# usermod -K project=class2006 laura
```

Like roles this assignments are stored in the file `/etc/user_attr`:

```
alice::::type=normal;project=class2005
bob::::type=normal;project=class2005
mike::::type=normal;project=class2006
laura::::type=normal;project=class2006
```

Okay, let's `su` to the user alice and check for the project assignment:

```
bash-3.2$ su alice
Password:
$ id -p
uid=2005(alice) gid=1(other) projid=100(class2005)
```

As configured the user `alice` is assigned to the project `class2005`. When we look into the process table, we can check the forth columns. All processes owned by alice are assigned to the correct project.

```
# ps -ef -o pid,user,zone,project,taskid,args | grep "class2005
   "
  752    alice   global class2005    76 sh
  758    alice   global class2005    76 sleep 10
```

Okay, obviously our teachers want their own projects: You can configure this on two ways. You can configure it by hand, or create a project beginning with `user.` and ending with the username. We use the second method in this example.

```
# useradd einstein
# projadd user.einstein
# passwd einstein
New Password:
Re-enter new Password:
passwd: password successfully changed for einstein
```

Now we check for the project:

```
$ su einstein
Password:
$ id -p
uid=2009(einstein) gid=1(other) projid=102(user.einstein)
$ sleep 100 &
$ ps -ef -o pid,user,zone,project,taskid,args | grep "user.
   einstein"
  777 einstein   global user.einstein    78 sh
  782 einstein   global user.einstein    78 sleep 100
  788 einstein   global user.einstein    78 grep user.einstein
  789 einstein   global user.einstein    78 ps -ef -o pid,user,
     zone,project,taskid,args
```

Et voila! We don't have to assign the project explicitly. It's done automagically by the system.

The logic behind the automatic assigning of the project to a user is simple:

- If the name of the project is defined by adding the `project` user attribute, use the assigned project as the default for the user.
- If it's not defined with the user, look for a project beginning with `user.` and ending with the name of the user and use it as default project. For example: `user.root` or `user.jmoekamp`
- If there no such project, search for a project beginning with `group.` and ending with the name of the group of a user and use it as default project. For example: `group.staff`
- If theres no group with this name, use the project "default" as the default project.

But how do you create a task? You don't have to configure tasks! A task is created automatically by the system in certain events. Those events are:

- login
- cron
- newtask
- setproject
- su

Okay, let's start a sleep in a new task:

```
$ newtask sleep 10 &
761
```

Okay, when you look at the process table now, you will see the different taskid in the fifth column:

```
# ps -ef -o pid,user,zone,project,taskid,args | grep "class2005
   "
  752    alice   global class2005    76 sh
  761    alice   global class2005    77 sleep 10
```

But you can use the newtask command to assign a different project, too. At first we create another project. We work at the Large Hadron Collider project, thus we call it `lhcproject`

```
$ newtask -p lhcproject sleep 100 &
802
$ newtask: user "einstein" is not a member of project "
   lhcproject"
```

Hey, not this fast! You have to be a member of the project to add a task to a project. Without this hurdle it would be too easy to use the resources of different project ;) Okay, let's add the user `einstein` to the group lhcproject again.

```
# projmod -U einstein lhcproject
```

Now we can try it again:

```
$ newtask -p lhcproject sleep 1000&
817
$ ps -ef -o pid,user,zone,project,taskid,args | grep "einstein"
  777 einstein   global user.einstein    78 sh
  817 einstein   global lhcproject    80 sleep 1000
  819 einstein   global user.einstein    78 ps -ef -o pid,user,
     zone,project,taskid,args
  818 einstein   global user.einstein    78 grep einstein
$
```

Voila, the sleep runs within a different project.

6.6. Why do I need all this stuff?

But why is all this stuff important for Resource Management? While it's possible to put resource management onto a single process, this isn't a sensible choice most of the time. Your workload consists out of multiple services out of multiple processes and it would be more complex to configure resource management to each of this processes. With the concepts of projects and tasks you can assign those limits to a group of processes.

6.7. Limiting operating environment resources

The kernel of an operating system provides a huge amount of resources to the processes running on it. Such resources are file descriptors, shared memory segments or the process tables. Albeit hard to fill up this resources with modern operating systems it's not impossible. When a resource is consumed by a single malicious or erroneous application, all other can't run as well when they need more resources from the operating system.

Let's assume this scenario: There is an course "Perl Scripting for beginners" at the Unseen University and in the last year the lesson "About fork" ended in chaos as some of your students coded forkbombs as they though this would be funny (as the class before, and the one before ...)

```
#!/usr/bin/perl
fork while 1
```

I've stored this little script at `/opt/bombs/forkbomb.pl`. A few seconds after starting such a script, the system is toast because of the hundreds of forked processes. Don't try this without resource management. Okay, but this year, you've migrated to Solaris. You can impose resource management.

Okay, we have to modify our project configuration:

```
# projmod -K "task.max-lwps=(privileged,10,deny)" class2005
```

Now we have configured a resource limit. A single task in the `class2005` cant have more than 9 processes. The tenth attempt to fork will be denied. Okay, do you remember the reasons, why the system starts a new task? One of it is "login". Thus every login of a user gives him 10 threads to work with. And this is exactly the behavior we want.

Let's assume Alice starts her forkbomb:

```
# ps -ef | grep "alice"
   alice   685   682   0 14:58:12 ?           0:00 /usr/lib/ssh
      /sshd
   alice   693   686  14 14:58:42 pts/1       0:38 /usr/bin/
      perl /opt/bombs/forkbomb.pl
   alice   686   685   0 14:58:12 pts/1       0:00 -sh
   alice   694   693  15 14:58:42 pts/1       0:38 /usr/bin/
      perl /opt/bombs/forkbomb.pl
   alice   695   694  14 14:58:42 pts/1       0:37 /usr/bin/
      perl /opt/bombs/forkbomb.pl
   alice   696   695  14 14:58:42 pts/1       0:37 /usr/bin/
      perl /opt/bombs/forkbomb.pl
   alice   697   696  14 14:58:42 pts/1       0:37 /usr/bin/
      perl /opt/bombs/forkbomb.pl
```

```
    alice   698   697  14 14:58:42 pts/1       0:39 /usr/bin/
       perl /opt/bombs/forkbomb.pl
    alice   699   698  14 14:58:42 pts/1       0:38 /usr/bin/
       perl /opt/bombs/forkbomb.pl
 # ps -ef | grep "alice" | wc -l
        9
```

After forking away 7 forkbomb.pl processes, any further fork is denied by the system. The load of the system goes up (as there are hundreds of denied forks) but the system stays usable.

Alice sends her script to Bob. He tries it, too:

```
 # ps -ef | grep "alice"
    alice   685   682   0 14:58:12 ?           0:00 /usr/lib/ssh
       /sshd
    alice 28520 28519   6 15:15:08 pts/1       0:03 /usr/bin/
       perl /opt/bombs/forkbomb.pl
    alice   686   685   0 14:58:12 pts/1       0:00 -sh
    alice 28521 28520   6 15:15:08 pts/1       0:03 /usr/bin/
       perl /opt/bombs/forkbomb.pl
    alice 28519   686   6 15:15:08 pts/1       0:02 /usr/bin/
       perl /opt/bombs/forkbomb.pl
    alice 28522 28521   6 15:15:08 pts/1       0:03 /usr/bin/
       perl /opt/bombs/forkbomb.pl
    alice 28524 28523   6 15:15:08 pts/1       0:03 /usr/bin/
       perl /opt/bombs/forkbomb.pl
    alice 28523 28522   6 15:15:08 pts/1       0:03 /usr/bin/
       perl /opt/bombs/forkbomb.pl
    alice 28525 28524   6 15:15:08 pts/1       0:02 /usr/bin/
       perl /opt/bombs/forkbomb.pl
 # ps -ef | grep "bob"
      bob 28514 28511   0 15:14:47 ?           0:00 /usr/lib/ssh
         /sshd
      bob 28515 28514   0 15:14:47 pts/3       0:00 -sh
      bob  2789  2502   6 15:15:10 pts/3       0:03 /usr/bin/
         perl /opt/bombs/forkbomb.pl
      bob  2791  2790   6 15:15:10 pts/3       0:03 /usr/bin/
         perl /opt/bombs/forkbomb.pl
      bob  2502 28515   6 15:15:10 pts/3       0:03 /usr/bin/
         perl /opt/bombs/forkbomb.pl
      bob  2790  2789   6 15:15:10 pts/3       0:03 /usr/bin/
         perl /opt/bombs/forkbomb.pl
      bob  2792  2791   6 15:15:10 pts/3       0:03 /usr/bin/
         perl /opt/bombs/forkbomb.pl
```

```
bob   2793   2792    6 15:15:10 pts/3       0:02 /usr/bin/
   perl /opt/bombs/forkbomb.pl
bob   2794   2793    6 15:15:10 pts/3       0:03 /usr/bin/
   perl /opt/bombs/forkbomb.pl
```

This is still no problem for the system. After a few forks of the forkbomb, the system denies further forks. And the system stays usable. The limitation of the number of processes is only one example. You can limit other resources. You can find a list of all controls at the man page of `resource_controls`

6.8. Limiting CPU resources

Okay, but your system wasn't purchased for students at night. You've bought it for computational science at night. The Large Hadron Collider project spend 75% of the costs, and the Small Hadron Collider project spend 25%. Thus the compute power should be distributed in a similar manner.

6.8.1. Without Resource Management

But when you start two compute intensive processes, both get roundabout 50 percent of processor. Okay, we have an super-duper simulation script called `/opt/bombs/cpuhog.pl`:

```
#! /usr/bin/perl
while (1) { my $res = ( 3.3333 / 3.14 ) }
```

Let's login as user `einstein`:

```
# su einstein
Password:
$ /opt/bombs/cpuhog.pl &
$ /opt/bombs/cpuhog.pl &
```

After a few moments the system will stabilize at aprox. 50% CPU resources for both processes. Just look at the first column:

```
bash-3.2$ ps -o pcpu,project,args
%CPU  PROJECT COMMAND
 0.0 user.einstein -sh
 0.3 user.einstein bash
47.3 user.einstein /usr/bin/perl /opt/bombs/cpuhog.pl
48.0 user.einstein /usr/bin/perl /opt/bombs/cpuhog.pl
 0.2 user.einstein ps -o pcpu,project,args
```

This isn't the intended behavior.

6.8.2. Using the Fair Share Scheduler

Solaris can solve this problem. There is something called Fair Share Scheduler in Solaris. This scheduler is capable to distribute the compute power of the system based on a configuration. The FFS isn't activated by default. You have to login as root to activate it:

```
# dispadmin -d FSS
```

There are ways to enable this scheduler with a running system, but it's easier to reboot the system now.

When the system has started, we get root privileges by using `su`. At first we create an additional project for the SHC project. We created the other project (`lhcproject`) before:

```
# projadd shcproject
#projmod -U einstein shcproject
```

Now we configure an resource limit on the projects.

```
# projmod -K "project.cpu-shares=(privileged,150,none)"
   lhcproject
# projmod -K "project.cpu-shares=(privileged,50,none)"
   shcproject
```

We've used the resource control `project.cpu-shares`. With this control we can assign an amount of CPU power to an project. We've defined an privileged limit, thus only root can change this limit later on.

6.8.3. Shares

Okay, what is the meaning of these numbers 150 and 50 in the last commands ? Where are the 25% and the 75%? Well, the resource management isn't configured with percentages, it's configured in a unit called shares.

It's like with stocks. The person with the most stocks owns most of the company. The project with the most shares owns most of the CPU. In our example we divided the CPU in 200 shares. Every share represents 1/200 of the CPU. Project shcproject owns 50 shares. Project lhcproject owns 150. I think, you already saw it: 150 is 75% of 200 and 50 is 25% of 200. Here we find our planed partitioning of the CPU we've planed before.

By the way: I deliberately choose 150/50 instead of 75/25 to show you that these share definitions are not scaled in percent.

Okay, but what happens when you add a third project and you give this project 200 shares (For example because a new project gave money for buying another processor board). Then the percentages are different. In total you have 400 shares on the system. The 200 shares of the new project are 50%, thus the project gets 50% of the compute power. The 150 shares of the `lhcproject` are 37.5 percent. This project gets 37.5 of the computing power. And the 50 shares of the `shcproject` are now 12.5 percent and thus the project get this part of the CPU power.

6.8.4. Behavior of processes with Resource Management

Okay, now let's start a process under the control of the new resource limitation. Login as user `einstein` and type the following command:

```
$ newtask -p shcproject /opt/bombs/cpuhog.pl &
```

We start at first only one of the `cpuhog.pl` processes .

```
bash-3.2$ ps -o pcpu,project,args
%CPU  PROJECT COMMAND^
 0.0 user.einstein -sh
 0.3 user.einstein bash
 0.2 user.einstein ps -o pcpu,project,args
95.9 shcproject /usr/bin/perl /opt/bombs/cpuhog.pl
```

Wait ... the process gets 95.9 percent? An error ? No. It makes no sense to slow down the process, when there is no other process needing the compute power.

Now we start the second process, this time as a task in the lhcproject:

```
bash-3.2$ newtask -p lhcproject /opt/bombs/cpuhog.pl &
[2] 784
```

We look in the process table again.

```
bash-3.2$ ps -o pcpu,project,args
%CPU  PROJECT COMMAND
 0.0 user.einstein -sh
 0.1 user.einstein bash
72.5 lhcproject /usr/bin/perl /opt/bombs/cpuhog.pl
25.6 shcproject /usr/bin/perl /opt/bombs/cpuhog.pl
 0.2 user.einstein ps -o pcpu,project,args
```

Voila, each of our compute processes get their configured part of the compute power. It isn't exactly 75%/25% all the time but in average the distribution will be this way.

A few days later, the Dean of the department comes into the office and tells you that we need the results of the SHC project earlier, as important persons want to see them soon to spend more money. So you have to change the ratio of the shares. We can do this without restarting the processes at runtime. But as we've defined the limits as privileged before, we have to login as root:

```
# prctl -n project.cpu-shares -r -v 150 -i project shcproject
# prctl -n project.cpu-shares -r -v 50 -i project lhcproject
```

Let's look after the processes again after a few moments:

```
bash-3.2$ ps -o pcpu,project,args
%CPU  PROJECT COMMAND
 0.0 user.einstein -sh
 0.1 user.einstein bash
25.7 lhcproject /usr/bin/perl /opt/bombs/cpuhog.pl
72.9 shcproject /usr/bin/perl /opt/bombs/cpuhog.pl
 0.2 user.einstein ps -o pcpu,project,args
```

The ratio has changed to new settings. It's important to know that only the settings in `/etc/projects` is boot-persistent. Everything you set via `prctl` is lost at the boot.

6.9. Limiting memory resources

So far we configured limits on the usage of operating system resources and the CPU. Let's get back to the class room example used before. A different course at the unseen university is "Memory Management for beginners". And at start our students have problem with severe memory leaks. It would be useful to impose resource management to the memory management.

6.9.1. Without memory resource management

Bob wrote a little script within Perl that eats a large amount of the available memory.

```
#!/usr/bin/perl
for ($i=0;$i<10;$i++) {
 push @_,"x"x(1*1024*1024);
 sleep(5);
}
```

```
sleep(30);
for ($i=0;$i<10;$i++) {
 push @_,"x"x(1*1024*1024);
 sleep(5);
}
```

When we start this script, it will allocate memory by pushing blocks of 1 Megabyte of x chars onto a stack.

```
# ps -ef -o pid,user,vsz,rss,project,args | grep "bob" | grep -
   v "grep"
 1015       bob 8728  892 class2005 /usr/lib/ssh/sshd
 1362       bob 26148 24256 class2005 /usr/bin/perl ./memoryhog.
    pl
 1016       bob 1624  184 class2005 -sh
 1031       bob 3260 1336 class2005 bash
```

When you look in the forth column you see the resident set size. The resident set is the amount of data of a process in the real memory. After a short moment the process `./memoryhog.pl` uses 24 MB of our precious memory (20 times 1 MB plus the perl interpreter minus some shared stuff).

6.9.2. With memory resource management

It would be nice to control the amount of data, that resides in the real memory. Solaris has such a capability. At first we have to activate the resource cap enforcement daemon. You have to login as root to do so:

```
# rcapadm -E
```

The rcapd daemon enforces resource caps on a group of processes. The rcapd supports caps on projects and zones at the moment. When the resident set size of a group of processes exceeds the defined cap. To reduce the resource consumption of a process group, the daemon can page out infrequently uses pages of memory to swap.

For testing purposes we configure the daemon in such a way, that it scans every second for new processes in projects with a resource cap. Furthermore we configure it to sample the resident set size every 1 second, too. Additionally the pageout statistics of the rcapd will be updated every second, too.

```
# rcapadm -i scan=1,sample=1,report=1
```

For testing purposes we define a new project called mmgntcourse and add the user bob to this project:

```
# projadd mmgntcourse
# projmod -U bob mmgntcourse
```

Now we set an resource cap for the resident set size of 5 Megabytes for this project:

```
# projmod -K rcap.max-rss=5242880 mmgntcourse
```

Okay, now get back to the window with the login of user ¡code¿bob¡/code¿. Let's start the `memoryhog.pl` in the `mmgntcourse`:

```
$ newtask -p mmgntcourse ./memoryhog.pl
```

Okay, get back to a different window and login as root. With the `rcapstat` you can observe the activities of the rcapd. In our example we tell `rcapstat` to print the statistics every 5 seconds:

```
# rcapstat 5
   id project         nproc    vm   rss   cap    at avgat
      pg avgpg
  105 mmgntcourse         - 2352K 3876K 5120K    0K    0K    0
     K    0K
  105 mmgntcourse         - 3376K 4900K 5120K    0K    0K    0
     K    0K
  105 mmgntcourse         1 4400K 5924K 5120K  812K  812K  804
     K  804K
  105 mmgntcourse         1 5424K 6408K 5120K 3380K 1126K 3380
     K 1126K
  105 mmgntcourse         - 6448K 4856K 5120K    0K    0K    0
     K    0K
  105 mmgntcourse         1 7472K 5880K 5120K  760K  760K  760
     K  760K
  105 mmgntcourse         1 8496K 5384K 5120K 1024K  512K 1024
     K  512K
  105 mmgntcourse         1 9520K 6144K 5120K 1024K 1024K 1024
     K 1024K
  105 mmgntcourse         1   10M 5120K 5120K 1024K 1024K 1024
     K 1024K
  105 mmgntcourse         1   11M 6144K 5120K 1024K 1024K 1024
     K 1024K
  105 mmgntcourse         1   11M 4096K 5120K 1024K 1024K 1024
     K 1024K
  105 mmgntcourse         -   11M 4096K 5120K    0K    0K    0
     K    0K
[...]
  105 mmgntcourse         -   11M 4096K 5120K    0K    0K    0
     K    0K
```

```
105 mmgntcourse          1   14M 5892K 5120K 2820K  940K 2820
   K  940K
105 mmgntcourse          1   15M 5628K 5120K 1280K  640K 1280
   K  640K
105 mmgntcourse          1   16M 6144K 5120K 1024K 1024K 1024
   K 1024K
105 mmgntcourse          1   17M 6144K 5120K 1024K 1024K 1024
   K 1024K
105 mmgntcourse          1   18M 5120K 5120K 1024K 1024K 1024
   K 1024K
105 mmgntcourse          -   18M 5120K 5120K    0K    0K    0
   K    0K
105 mmgntcourse          1   20M 5120K 5120K 2048K 1024K 2048
   K 1024K
105 mmgntcourse          -   20M 5120K 5120K    0K    0K    0
   K    0K
105 mmgntcourse          1   22M 5120K 5120K 2048K 1024K 2048
   K 1024K
```

As you see, the resident set size stays at approx. 5 Megabyte. The RSS may increase above the configured size, as the applications may allocate memory between the RSS sampling intervals. But at the next sampling size the `rcap` starts to force the page out of this exceeding pages. After a few seconds the resident set size is enforced, again. You can observe this behaviour in column 5 of the rcapstat printout.

6.10. Resource Management and SMF

A few days ago, I wrote about the Service Management Facility, a framework to manage services in Solaris. This framework enables Solaris to start services in a more intelligent manner. Albeit almost all services started by SMF are member of the project ¡code¿system¡/code¿ it doesn't have to be this way. You can specify the project used for a service.

6.10.1. Assigning a project to an already running service

This method is described in the SMF FAQ[1]. I will use an practical example. You run a server in the internet, but it's under high load from spam bot. You've already created a project `sendmail` and configured the resource management.

[1] `http://www.opensolaris.org/os/community/smf/faq#33`

At the moment, the sendmail runs in the `system` project:

```
# ps -o pid,project,args  -ef | grep "sendmail" | grep -v "grep
   "
  648   system /usr/lib/sendmail -Ac -q15m
  647   system /usr/lib/sendmail -bd -q15m -C /etc/mail/local.
     cf
```

How do you start it as a part of a different project? Okay, check for an already configured project.

```
#  svcprop -p start/project smtp:sendmail
svcprop: Couldn't find property `start/project' for instance `
   svc:/network/smtp:sendmail'.
```

Okay, nothing defined ... this makes it a little bit harder, because we can't set the project only at the moment, as there is a bug in the restarter daemon. You need a fully populated start method. If the `svcprop` run delivers you the name of a project, you can ignore the next block of commands:

```
svccfg -s sendmail setprop start/user = astring: root
svccfg -s sendmail setprop start/group = astring: :default
svccfg -s sendmail setprop start/working_directory = astring: :
   default
svccfg -s sendmail setprop start/resource_pool = astring: :
   default
svccfg -s sendmail setprop start/supp_groups = astring: :
   default
svccfg -s sendmail setprop start/privileges = astring: :default
svccfg -s sendmail setprop start/limit_privileges = astring: :
   default
svccfg -s sendmail setprop start/use_profile = boolean: false
```

Okay, now we can set the project property of the start method:

```
svccfg -s smtp/sendmail setprop start/project = astring:
   sendmail
\end{lstlsting}
Now we have to refresh the configuration of the service. After
   refreshing the service we can check the property:
\begin{lstlisting}
# svcadm refresh sendmail
#  svcprop -p start/project sendmail
sendmail
```

Okay, the new properties are active. Now restart the service:

```
# svcadm restart sendmail
```

Let's check for the result:

```
# ps -o pid,project,args  -ef | grep "sendmail" | grep -v "grep
   "
 1539 sendmail /usr/lib/sendmail -bd -q15m -C /etc/mail/local.
    cf
 1540 sendmail /usr/lib/sendmail -Ac -q15m
```

6.10.2. Configuring the project in a SMF manifest

Okay, you have written your own application and want to run it under the control of the SMF. You can define the project for the service in the manifest. You have to define the project in the start method of the manifest:

```
<exec_method type='method' name='start' exec='/path/to/method
   start' timeout_seconds='0' >
        <method_context project=projectname'>
        <method_credential user='someuser'>
        </method_context>
           </exec_method>
```

That's all. The important part is the second row of the fragment. We define the project as a part of the startup method context. The rest is done as described in the SMF tutorial[2]

6.11. Conclusion

The Solaris Resource Management is an powerful tool to partition the resources of a single instance of an operating system. By using Solaris Resource Management you can use a single operating system for a multitude of services but still ensuring the availability of resources to them. I hope, this tutorial gave you an good insights into the basics of resource management.

[2] http://www.c0t0d0s0.org/archives/4147-Solaris-Features-Service-Management-Facility-Part-4-Develop
html

6.12. Do you want to learn more?

Documentation

System Administration Guide: Solaris Containers-Resource Management and Solaris Zones

man page for resource_controls(5)

man page for the Fair Share Scheduler

Misc. links

Zones and Containers FAQ - Section 3: Resource Management, Performance

Solaris 10+ Resource Management

7. /home? /export/home? AutoFS?

Solaris 10/Opensolaris

7.1. History

The ever recurring question to me at customer sites relatively new to Solaris is: "Okay, on Linux I had my home directories at `/home`. Why are they at `/export/home` on Solaris?" This is old hat for seasoned admins, but I get this question quite often. Well, the answer is relatively simple and it comes from the time when we started to use NIS and NFS and it had something to do with our slogan "The network is the computer", because it involves directories distributed in the network. Okay, we have to go back 20 years in the past.

There was a time, long, long ago, when you worked at your workstation. The harddisk in your workstation was big and it was a time when you didn't need 200 megabytes for your office package alone. So you and your working group used workstations for storing their data, but there were several workstations and even some big servers for big computational tasks. The users wanted to share the data, so Sun invented NFS to share the files between the systems. As it was a tedious task to distribute all of the user accounts on all of the systems, Sun invented NIS (later NIS+, but this is another story).

But the users didn't want to mount their home directories manually on every system. They wanted to log in to a system and work with their home directory on every system. They didn't want to search in separate places depending on whether they were using their own machine or a different one.

So Sun invented the automounter - it found its way into SunOS 4.0 in 1988. The automounter mounts directories on a system based upon a ruleset. In Solaris 2.0 and later the automounter was implemented as a pseudo filesystem called autofs. autofs was developed to mount directories based on rules defined in so-called maps.

There are two of them provided by default. At first there is the `/etc/auto_master`. To cite the manual:

> The `auto_master` map associates a directory with a map. The map is a master list that specifies all the maps that autofs should check

On a freshly installed system the file looks like this:

```
[root@gandalf:/net/theoden/tools/solaris]$ cat /etc/auto_master
+auto_master
/net            -hosts          -nosuid,nobrowse
/home           auto_home       -nobrowse
```

The file `/etc/auto_home` is such a map referenced by the master map. To cite the manual again:

> An indirect map uses a substitution value of a key to establish the association between a mount point on the client and a directory on the server. Indirect maps are useful for accessing specific file systems, such as home directories. The auto_home map is an example of an indirect map.

We will use this map later in this article.

7.2. The use case

Okay, an example. gandalf is the workstation of Waldorf and Statler. theoden is the workstation of Gonzo and Scooter. They each have their home directories on their own workstation. Sometimes a team uses the workstations of the other teams and they have a gentleman's agreement allowing each other to do so. But they want to use their home directories on the system of the other team.

7.3. Prerequisites

At first we have to export the directories which store the real home directories on both hosts via NFS.

At first on gandalf:

```
[root@gandalf:/etc]$ echo "share -F nfs -d \"Home Directories\"
    /export/home" >> /etc/dfs/dfstab
[root@gandalf:/etc]$ shareall
[root@gandalf:/etc]$ exportfs
-               /export/home   rw   "Home Directories"
```

Now we repeat these steps on theoden:

```
[root@theoden:/export/home]$ echo "share -F nfs -d \"Home
   Directories\" /export/home" >> /etc/dfs/dfstab
[root@theoden:/export/home]$ shareall
[root@theoden:/export/home]$ exportfs
-               /export/home   rw   "Home Directories"
```

Okay, it's important that both hosts can resolve the hostname of the other system. I've added some lines to ¡code¿/etc/hosts¡/code¿ in my test installation:

```
10.211.55.201 gandalf
10.211.55.200 theoden
```

7.4. Creating users and home directories

Normally you wouldn't create the users this way. You would use a centralised user repository with LDAP. But that is another very long tutorial.

The userids and user names of the users have to be the same on both systems. At first I create the local users. I use the `-m` switch for creating the home directory at the same time as the user.

```
[root@gandalf:~]$ useradd -u 2000 -m -d /export/home/waldorf
   waldorf
64 blocks
[root@gandalf:~]$ useradd -u 2001 -m -d /export/home/statler
   statler
64 blocks
```

Now I set the home directory of both users to the `/home` under the control of autofs:

```
[root@gandalf:~]$ usermod -d /home/statler statler
[root@gandalf:~]$ usermod -d /home/waldorf waldorf
```

Now I create the users for the other team, without the `-m` switch and directly with the correct home directory. The home directories come from the other system, so we don't have to create them:

```
[root@gandalf:~]$ useradd -u 2002 -d /home/gonzo gonzo
[root@gandalf:~]$ useradd -u 2003 -d /home/scooter scooter
```

Now we switch to Theoden. We do almost the same on this system. We create the accounts for Waldorf and Statler without creating a home directory. After this we create the local users together with their home directories, which we then set to be autofs controlled:

```
[root@theoden:~]$ useradd -u 2001 -d /home/statler statler
[root@theoden:~]$ useradd -u 2000 -d /home/waldorf waldorf
[root@theoden:~]$ useradd -u 2002 -d /export/home/gonzo -m
   gonzo
64 blocks
[root@theoden:~]$ useradd -u 2003 -d /export/home/gonzo -m
   scooter
64 blocks
[root@theoden:~]$ usermod -d /home/gonzo gonzo
[root@theoden:~]$ usermod -d /home/scooter scooter
```

7.5. Configuring the automounter

Execute the following four commands on both hosts:

```
echo "waldorf gandalf:/export/home/&" >> /etc/auto_home
echo "scooter theoden:/export/home/&" >> /etc/auto_home
echo "gonzo theoden:/export/home/&" >> /etc/auto_home
```

Here, the ampersand is a variable. It stands for the key in the table. So `gonzo theoden:/export/home/&` translates to `theoden:/export/home/gonzo`. Now start the autofs on both hosts:

```
[root@theoden:~]$svcadm enable autofs
```

and

```
[root@gandalf:~]$svcadm enable autofs
```

7.6. Testing the configuration

Okay, let's log in to theoden as user gonzo. Gonzo is a user with a home directory local to theoden:

```
$ ssh gonzo@10.211.55.200
Password:
Last login: Sun Feb 17 14:16:41 2008 from 10.211.55.2
Sun Microsystems Inc.   SunOS 5.11      snv_78  October 2007
$ /usr/sbin/mount
[...]
/home/gonzo on /export/home/gonzo read/write/setuid/devices/dev
   =1980000 on Sun Feb 17 14:13:35 2008
```

Now we try waldorf on theoden. Waldorf doesn't have its home directory on theoden, it's on gandalf.

```
$ ssh waldorf@10.211.55.200
Password:
Last login: Sun Feb 17 14:17:47 2008 from 10.211.55.2
Sun Microsystems Inc.   SunOS 5.11      snv_78  October 2007
$ /usr/sbin/mount
[...]
/home/waldorf on gandalf:/export/home/waldorf remote/read/write
   /setuid/devices/xattr/dev=4dc0001 on Sun Feb 17 14:17:48
   2008
```

autofs has mounted the `/export/home/waldorf` automatically to `/home/waldorf`, the directory we used when we created the user.

Let's crosscheck. We log into gandalf with the user waldorf. Now this user has a local home directory. It's a local mount again.

```
$ ssh waldorf@10.211.55.201
Password:
Last login: Sat Feb 16 09:12:47 2008 from 10.211.55.2
Sun Microsystems Inc.   SunOS 5.11      snv_78  October 2007
$ /usr/sbin/mount
[...]
/home/waldorf on /export/home/waldorf read/write/setuid/devices
   /dev=1980000 on Sat Feb 16 09:12:47 2008
```

7.7. Explanation for the seperated /home and /export/home

The explanation for the existence of `/home` and `/export/home` is really simple. I think you got it already. `export/home` is the directory where all the local directories are located.`/home` is the playground for autofs to unify all home directories at a central place, whereever they are located.

7.8. The /net directory

Did you ever wonder about the `/net` in the root directory and its job? It's an autofs controlled directory, too. Let's assume you have an `/tools/solaris` directory at theoden:

```
[root@theoden:/tools/solaris]$ ls -l /tools/solaris
total 0
-rw-r--r--   1 root     root           0 Feb 17 15:21 tool1
-rw-r--r--   1 root     root           0 Feb 17 15:21 tool2
-rw-r--r--   1 root     root           0 Feb 17 15:21 tool3
```

Share it via NFS

```
[root@theoden:/tools/solaris]$ share -F nfs -d "Tools" /tools/
   solaris
[root@theoden:/tools/solaris]$ share -F nfs
-               /export/home   rw   "Home Directories"
-               /tools/solaris   rw   "Tools"
[root@theoden:/tools/solaris]$
```

Now change to the other workstation. Look into the directory `/net/theoden`:

```
[root@gandalf:/]$ cd /net/theoden
[root@gandalf:/net/theoden]$ ls
export  tools
```

You will notice all of the directories shared by theoden. Change into the `tools/solaris` directory:

```
[root@gandalf:/net/theoden]$ cd tools
[root@gandalf:/net/theoden/tools]$ ls
solaris
[root@gandalf:/net/theoden/tools]$ cd solaris
[root@gandalf:/net/theoden/tools/solaris]$ ls -l
total 0\
-rw-r--r--   1 root     root           0 Feb 17  2008 tool1
-rw-r--r--   1 root     root           0 Feb 17  2008 tool2
-rw-r--r--   1 root     root           0 Feb 17  2008 tool3
[root@gandalf:/net/theoden/tools/solaris]$
[root@gandalf:/net/theoden/tools/solaris]$ mount
[..]
/net/theoden/tools/solaris on theoden:/tools/solaris remote/
   read/write/nosetuid/nodevices/xattr/dev=4dc0002 on Sat Feb
   16 10:23:01 2008
```

Neat isn't it... it's configured by default, when you start the autofs.

7.9. Do you want to learn more?

Manuals

How Autofs Works

Task Overview for Autofs Administration

8. lockfs

Solaris 10/Opensolaris

Quite often the configuration of a feature or application mandates that the data on the disk doesn't change while you activate it. An easy way to achieve this would be simply un-mounting the disk - that's possible but then you can't access the data on the disk at all. You can't even read from the filesystem, even though this doesn't change anything on the disk (okay, as long you've mounted the disk with noatime).

So: How else can you ensure that the content of a filesystem doesn't change while you work with the disk? ufs has an interesting feature. It's called `lockfs` and with it, you can lock the filesystem. You can lock it to an extent that you can only unmount and remount it to gather access to the data, but you can also lock out a subset of the many ways in which one might try to access it.

8.1. Types of Locks

The `lockfs` command can establish a number of different locks on an UFS filesystem:

- *Delete Lock* `-d`: suspends all access that could remove directory entries
- *Hard Lock* `-h`: suspends all access to the filesystem. It can't be unlocked. You have to unmount and remount it
- *Name Lock* `-n`: suspends all access to the filesystem that could remove or change directory entries
- *Error Lock* `-e`: this lock is normally established when UFS detects an internal inconsistency. It's released by the usage of `fsck`. It suspends all access to the filesystem.
- *write lock* `-w`: this lock suspends all accesses that could modify the filesystem
- *flush log* `-f`: this isn't really a lock - this option forces a synchronous flush of the named filesystem. It returns when all data has been written to disk and the log has been rolled

8.2. Write Lock

Okay, let's assume we've mounted a UFS filesystem at `/mnt`.

```
# cd /mnt

# echo "test" > testfile1

# ls

lost+found  testfile1
```

No problem. Our testfile found its way into the file system. Now we establish a write lock on our file system.

```
# lockfs -w /mnt
```

You set the lock with the `lockfs` command, and the switch `-w` tells `lockfs` to set a write lock. With a write lock, you can read a filesystem, but you can't write to it.

Okay, let's check the existing locks. You use the `lockfs` command without any further options.

```
# lockfs

Filesystem            Locktype   Comment
/mnt                  write
```

When we try to add an additional file, the write system call simply blocks.

```
# echo "test" > testfile2

^Cbash: testfile2: Interrupted system call
```

We have to break the echo command with CTRL-C. Okay, now let's release the lock.

```
# lockfs -u /mnt
```

The `-u` commands lockfs to release the lock. When you list the existing locks, the lock on `/mnt` is gone.

```
# lockfs
```

And just to check it, we try to write "test" to `testfile2` again.

```
# echo "test" > testfile2
```

The command returns instantly. When you check the filesystem, you will see both files.

```
# ls -l

total 20

drwx------   2 root     root         8192 Apr 25 18:10 lost+
   found

-rw-r--r--   1 root     root            5 Apr 27 03:49 testfile1

-rw-r--r--   1 root     root            5 Apr 27 03:50 testfile2
```

8.3. Delete lock

A rather strange kind of lock is the delete lock. It blocks all delete operations. This is actually of less practical use than you might think at first. You can't delete a file, but you can stil zero it or fill it with other content. Let's use the testfile from our last example. At first we try to delete the first file:

```
# rm testfile1
```

No problem. Now we establish the delete lock. This time we also add a comment. You can use this command to tell other administrators why you have established the lock.

```
# lockfs -c "no deletes today" -d /mnt
```

When you check for existing locks, you will see the delete lock on `/mnt` and the comment:

```
# lockfs

Filesystem            Locktype    Comment

/mnt                  delete      no deletes today
```

When you try to delete the file, the `rm`just blocks and you have to break it with CTRL-C again:

```
# rm testfile2

^C
```

When you've delete-locked an filesystem, you can create new files, you can append data to existing files and you can overwrite them:

```
# echo "test" > testfile3

# echo "test" >> testfile3

# echo "test" > testfile3
```

There is only one thing you can't do with this new file: delete it.

```
# rm testfile3

^C
```

Next we release the lock

```
# lockfs -u /mnt
```

Now you can clean up your test directory `/mnt`again.

```
# rm testfile2

# rm testfile3
```

8.4. Conclusion

The `lockfs` is a really neat feature to deny certain accesses to your filesystem without un-mounting it completely. Some locks are more useful for general use than others. For example, the write lock is really useful when you want to freeze the content of the filesystem while working with tools like AVS. Delete locks or name locks are useful when you need a stable directory, which is less of a day-to-day problem for administrators.

8.5. Do you want to learn more?

Documentation docs.sun.com: man page lockfs[1]

[1] `http://docs.sun.com/app/docs/doc/816-5166/6mbb1kq6a?l=en&a=view`

9. CacheFS

Solaris10

9.1. Introduction

There is a hidden gem in the Solaris Operating Environment, solving a task that many admins solve with scripts. Imagine the following situation. You have a central fileserver, and let's say 40 webservers. All of these webservers deliver static content and this content is stored on the harddisk of the fileserver. Later you recognize that your fileserver is really loaded by the webservers. Harddisks are cheap, thus most admins will start to use a recursive rcp or an rsync to put a copy of the data to the webserver disks.

Well... Solaris gives you a tool to solve this problem without scripting, without cron based jobs, just by using NFSv3 and this hidden gem: CacheFS. CacheFS is a really nifty tool. It does exactly what the name says. It's a filesystem that caches data of another filesystem. You have to think about it like a layered cake. You mount a CacheFS filesystem with a parameter that tells CacheFS to mount another one in the background.

9.2. History of the feature

Sun didn't introduce this feature for webservers. Long ago, admins didn't want to manage dozens of operating system installations. Instead of this they wanted to store all this data on a central fileserver (you know... the network is the computer). Thus net-booting Solaris and SunOS was invented. But there was a problem: Swap via the network was a really bad idea in those days (it was a bad idea in 10 MBit/s times and it's still a bad idea in 10 GBit/s times). Thus the diskless systems got a disk for local swap. But there was another problem. All the users started to work at 9 o'clock... they switched on their workstations... and the load on the fileserver and the network got higher and higher. They had a local disk... local installation again? No... the central installation had its advantages. Thus the idea of CacheFS was born.

CacheFS is a really old feature of Solaris/SunOS. Its first implementation dates back to the year 1991. I really think you can call this feature mature ;)

9.3. CacheFS in theory

The mechanism of CacheFS is pretty simple. As I told you before, CacheFS is somewhat similar to a caching web proxy. The CacheFS is a proxy to the original filesystem and caches files on their way through CacheFS. The basic idea is to cache remote files locally on a harddisk, so you can deliver them without using the network when you access them the second time.

Of course, the CacheFS has to handle changes to the original files. So CacheFS checks the metadata of the file before delivering the copy. If the metadata have changed, the CacheFS loads the original file from the server. When the metadata hasn't changed it delivers the copy from the cache.

The CacheFS isn't just usable for NFS, you could use it for caching optical media like CD or DVD as well.

9.4. A basic example

Okay... using CacheFS is really easy. Let's assume that you have an fileserver called `theoden`. We use the directory `/export/files` as the directory shared by NFS. The client in our example is `gandalf`.

9.4.1. Preparations

Let's create an NFS server at first. This is easy. Just share a directory on a Solaris Server. We log in to `theoden` and execute the following commands with root privileges.

```
[root@theoden:/]# mkdir /export/files
[root@theoden:/]# share -o rw /export/files
# share
-               /export/files   rw   ""
```

Okay, of course it would be nice to have some files to play around with in this directory. I will use some files of the Solaris Environment.

```
[root@theoden:/]# cd /export/files
[root@theoden:/export/files]# cp -R /usr/share/doc/pcre/html/*
   .
```

Let's do a quick test and see if we can mount the directory:

```
[root@gandalf:/]# mkdir /files
[root@gandalf:/]# mount theoden:/export/files /files
[root@gandalf:/]# unmount /files
```

Now you should be able to access the `/export/files` directory on theoden by accessing `/files` on gandalf. There should be no error messages.

Okay, firstly we have to create the location for our caching directories. Let's assume we want to place our cache at `/var/cachefs/caches/cache1`. At first we create the directories above the cache directory. You don't create the last part of the directory structure manually.

```
[root@gandalf:/]# mkdir -p /var/cachefs/caches
```

This directory will be the place where we store our caches for CacheFS. After this step we have to create the cache for the CacheFS.

```
[root@gandalf:/files]# cfsadmin -c -o maxblocks=60,minblocks
   =40,threshblocks=50 /var/cachefs/caches/cache1
```

The directory `cache1` is created automatically by the command. In the case where the directory already exists, the command will quit and do nothing.

Additionally you have created the cache and you specified some basic parameters to control the behavior of the cache. Citing the manpage of cfsadmin:

maxblocks: Maximum amount of storage space that CacheFS can use, expressed as a percentage of the total number of blocks in the front file system.

minblocks: Minimum amount of storage space, expressed as a percentage of the total number of blocks in the front file system, that CacheFS is always allowed to use without limitation by its internal control mechanisms.

threshblocks: A percentage of the total blocks in the front file system beyond which CacheFS cannot claim resources once its block usage has reached the level specified by `minblocks`.

Each of these parameters can be tuned to prevent CacheFS from eating away all of the storage available in a filesystem, a behavior that was quite common to early versions of this feature.

9.4.2. Mounting a filesystem via CacheFS

We have to mount the original filesystem now.

```
[root@gandalf:/files]# mkdir -p /var/cachefs/backpaths/files
[root@gandalf:/files]# mount -o vers=3 theoden:/export/files /
   var/cachefs/backpaths/files
```

You may notice the parameter that sets the NFS version to 3. This is necessary as CacheFS isn't supported with NFSv4. Thus you can only use it with NFSv3 and below. The reason for this limitation has its foundation in the different way NFSv4 handles inodes.

Okay, now we mount the cache filesystem at the old location:

```
[root@gandalf:/files]# mount -F cachefs -o backfstype=nfs,
   backpath=/var/cachefs/backpaths/files,cachedir=/var/cachefs/
   caches/cache1 theoden:/export/files /files
```

The options of the mount command control some basic parameters of the mount:

backfstype specifies what type of filesystem is proxied by the CacheFS filesystem

backpath specifies where this proxied filesystem is currently mounted

cachedir specifies the cache directory for this instance of the cache. Multiple CacheFS mounts can use the same cache.

From now on every access to the `/files` directory will be cached by CacheFS. Let's have a quick look into the `/etc/mnttab`. There are two important mounts for us:

```
[root@gandalf:/etc]# cat mnttab
[...]
theoden:/export/files   /var/cachefs/backpaths/files    nfs
       vers=3,xattr,dev=4f80001        1219049560
/var/cachefs/backpaths/files    /files  cachefs backfstype=nfs,
   backpath=/var/cachefs/backpaths/files,cachedir=/var/cachefs/
   caches/cache1,dev=4fc0001   1219049688
```

The first mount is our back file system, it's a normal NFS mountpoint. But the second mount is a special one. This one is the consequence of the mount with the `-F cachefs` option.

9.4.3. Statistics about the cache

While using it, you will see the cache structure at `/var/cachefs/caches/cache1` filling up with files. I will explain some of the structure in the next section. But how efficient is this cache? Solaris provides a command to gather some statistics about the cache. With `cachefsstat` you print out data like hit rate and the absolute number of cache hits and cache misses:

```
[root@gandalf:/files]# /usr/bin/cachefsstat

    /files
                cache hit rate:    60% (3 hits, 2 misses)
            consistency checks:      7 (7 pass, 0 fail)
                      modifies:      0
            garbage collection:      0
[root@gandalf:/files]#
```

9.5. The cache

Okay, we have a working CacheFS mount but how and where is the stuff cached by the system? Let's have a look at the cache:

```
[root@gandalf:/var/cachefs/cache1]# ls -l
total 6
drwxrwxrwx   5 root     root         512 Aug 18 10:54
   0000000000044e30
drwx------   2 root     root         512 Aug 11 08:11 lost+
   found
lrwxrwxrwx   1 root     root          16 Aug 11 08:18 theoden:
   _export_files:_files -> 0000000000044e30
```

To ensure that multiple caches using a single cache directory of the time aren't mixing up their data, they are divided at this place. At first a special directory is generated and secondly a more friendly name is linked to this. It's pretty obvious how this name is generated. `theoden:_export_files:_files` can be easily translated to `theoden:/export/files` mounted at `/files`.

Let's assume we've used the cache for another filesystem (e.g. `/export/binaries` on `theoden` mounted to `/binaries`):

```
[root@gandalf:/var/cachefs/cache1]# ls -l
total 10
```

```
drwxrwxrwx   5 root     root          512 Aug 18 10:54
   0000000000044e30
drwxrwxrwx   3 root     root          512 Aug 18 11:18
   0000000000044e41
drwx------   2 root     root          512 Aug 11 08:11 lost+
   found
lrwxrwxrwx   1 root     root           16 Aug 18 11:18 theoden:
   _export_binaries:_binaries -> 0000000000044e41
lrwxrwxrwx   1 root     root           16 Aug 11 08:18 theoden:
   _export_files:_files -> 0000000000044e30
```

With this mechanism, the caches are separated in their respective directories... no mixing up.

When we dig down a little bit deeper to the directories, we will see an additional layer of directories. This is necessary to prevent a situation where a directory contains too many files and thus slows down.

```
[root@gandalf:/var/cachefs/cache1/0000000000044e30
   /0000000000044e00]# ls -l
total 62
-rw-rw-rw-   1 root     root            0 Aug 18 10:54
   0000000000044e66
-rw-rw-rw-   1 root     root         1683 Aug 11 08:24
   0000000000044eaa
-rw-rw-rw-   1 root     root        29417 Aug 11 08:22
   0000000000044eba
```

When you examine these files, you will see that they are just a copy of the original files:

```
[root@gandalf:/var/cachefs/cache1/0000000000044e30
   /0000000000044e00]# cat 0000000000044eaa
[...]
This page is part of the PCRE HTML documentation. It was
   generated automatically
from the original man page. If there is any nonsense in it,
   please consult the
man page, in case the conversion went wrong.
[...]
[root@gandalf:/var/cachefs/cache1/0000000000044e30
   /0000000000044e00]#
```

The "Cache" of CacheFS is a pretty simple structure.

9.6. On-demand consistency checking with CacheFS

Let's assume you share a filesystem with static content (for example a copy of a CD-ROM) or a filesystem that changes on a regular schedule (for example at midnight every day). So it would pose unnecessary load to network to check the consistency every time a file is accessed.

CacheFS knows a special mode of operation for such a situation. It's called *on demand consistency checking*. It does exactly what the name says: It only checks the consistency of files in the cache when you tell the system to do so.

I will demonstrate this with an example:

Let's assume we stay with the normal mode of operation from the example before. We create a file on the fileserver.

```
[root@theoden:/export/files]# date >>
   test_with_consistency_check
[root@theoden:/export/files]# cat test_with_consistency_check
Tue Aug 12 14:59:54 CEST 2008
```

When we go to the NFS client and access the directory, this new file is visible instantaneously. And when we access it, we see the content of the file.

```
[root@gandalf:/files]# cat test_with_consistency_check
Tue Aug 12 14:59:54 CEST 2008
```

Now we go back to the server, and append additional data to the file:

```
[root@theoden:/export/files]# date >>
   test_with_consistency_check
[root@theoden:/export/files]# cat test_with_consistency_check
Tue Aug 12 14:59:54 CEST 2008
Tue Aug 12 15:00:11 CEST 2008
```

Obviously, you will see this change on the client:

```
[root@gandalf:/files]# cat test_with_consistency_check
Tue Aug 12 14:59:54 CEST 2008
Tue Aug 12 15:00:11 CEST 2008
```

Now we unmount it, and remount it:

```
[root@gandalf:/files]# cd /
[root@gandalf:/]# umount /files
[root@gandalf:/]# mount -F cachefs -o backfstype=nfs,backpath=/
   var/cachefs/backpaths/files,cachedir=/var/cachefs/caches/
   cache1,demandconst theoden:/export/files /files
```

You may have noticed the `demandconst` option. This option changes everything. Let's assume you created another file on the NFS server:

```
[root@theoden:/export/files]# date >>
   test_with_ondemand_consistency_check
[root@theoden:/export/files]# cat
   test_with_ondemand_consistency_check
Tue Aug 12 15:00:57 CEST 2008
```

Back on the NFS client you will not even see this file:

```
[root@gandalf:/files]# ls
index.html                          pcre_refcount.html
[...]
pcre_info.html                      pcretest.html
pcre_maketables.html                test_with_consistency_check
```

You have to trigger a consistency check. This is quite easy.

```
[root@gandalf:/files]# cfsadmin -s all

[root@gandalf:/files]# ls
index.html                             pcre_study.html
[..]
pcre_info.html
   test_with_consistency_check
pcre_maketables.html
   test_with_ondemand_consistency_check
pcre_refcount.html
```

Okay, now we can look into the file.

```
[root@gandalf:/files]cat test_with_ondemand_consistency_check
Tue Aug 12 15:00:57 CEST 2008
```

Now we append a new line to the file on the server by executing the following commands on the NFS server

```
[root@theoden:/export/files]date >>
   test_with_ondemand_consistency_check
[root@theoden:/export/files]cat
   test_with_ondemand_consistency_check
Tue Aug 12 15:00:57 CEST 2008
Tue Aug 12 15:02:03 CEST 2008
```

When we check this file on our NFS client, we still see the cached version.

```
[root@gandalf:/files]cat test_with_ondemand_consistency_check
Tue Aug 12 15:00:57 CEST 2008

[root@gandalf:/files]cfsadmin -s all
```

Now we can look into the file again, and you will see the new version of the file.

```
[root@gandalf:/files]cat test_with_ondemand_consistency_check
Tue Aug 12 15:00:57 CEST 2008
Tue Aug 12 15:02:03 CEST 2008
```

Okay, it's pretty obvious this isn't a feature for a filesystem that changes in a constant and fast manner. But it's really useful for situations, where you have control over the changes. As long as a file is cached, the file server will see not a single access for such files. Thus such a file access doesn't add to the load of the server.

There is an important fact here: It doesn't tell CacheFS to check the files right at that moment. It just tells CacheFS to check it at the next access to the file. So you don't have an consistency check storm.

9.7. An practical usecase

Let's take an example: Let's assume you have 50 webservers that serve static content or that serve and execute `.php` files. You may have used scp or rsync to synchronize all of these servers.

With CacheFS the workflow is a little bit different: You simply put all of your files on a fileserver. But you have noticed that there was a large load on this system. To reduce load to the fileserver you use your new knowledge to create a cache on every server. After this you saw a lot of requests for directory and file metadata for the documents directory. You know that this directory is provisioned with the changes only at 3 o'clock in the morning. Thus you write a little script that checks for the file `please_check_me` and starts `cfsadmin -s all` on the client in the case of its existence, causing them to check at the next access to the files if there is a newer version.

9.8. The CacheFS feature in future Solaris Development

CacheFS is one of the hidden gems in Solaris - in fact it's so hidden, that it's been in sustaining mode since 1999. That means that only bugfixes are made for this component

but no new features will find their way into this component. In recent days there was some discussion about the declaration of the End-of-Feature status for CacheFS which will lead to the announcement of the removal of CacheFS. While this isn't a problem for Solaris 10, I strongly disagree with the idea of removing this part of Solaris, as long there is no other boot-persistent non-main memory caching available for Solaris.

9.9. Conclusion

CacheFS is one of the features that even some experienced admins aren't aware of. But as soon as they try it, most of them can't live without it. You should give it a try.

9.9.1. Do you want to learn more ?

Manuals

System Administration Guide: Devices and File Systems - Using The CacheFS File System[1]

man pages docs.sun.com: cfsadmin [2]

docs.sun.com: cachefsstat[3]

docs.sun.com: mount_cachefs[4]

[1] http://docs.sun.com/app/docs/doc/817-5093/fscachefs-70682
[2] http://docs.sun.com/app/docs/doc/816-5166/cfsadmin-1m
[3] http://docs.sun.com/app/docs/doc/816-5166/cachefsstat-1m
[4] http://docs.sun.com/app/docs/doc/816-5166/mount-cachefs-1m

10. The curious case of /tmp in Solaris

Solaris 10/Opensolaris

This article isn't really about a feature, it's about a directory and its misuse. Furthermore it's an article about different default configurations, that lead to misunderstandings. This is a pretty old hat for experienced Solaris admins (many of them learned it the hard tour), but it seems to be totally unknown to many admins new to the business or for people switching from Linux to Solaris, as many distribution are configured in a different way per default. A reader of my blog just found a 2GB .iso in /tmp on a Solaris system and that's not really a good idea. A few days ago, a user in twitter had vast problems with memory usage on a system which boiled down to a crowed `/tmp`.

10.1. tmpfs and its usage

In Linux the `/tmp` directory is a normal directory. Most often this is just a part of the partition containing root. Many admins consider `/tmp` as an additional home directory and put data on it they want to use later on. When you really think about the idea behind `/tmp` is meant as a scratch space, where an application can put some data to process it later. Many applications use this /tmp space to write temporary data to.

Solaris introduced something called tmpfs many years ago, It's a memory based file system. Linux has something with similar functionality and the same name. Its just used differently in the default configuration of many distributions and there are a vast amount of articles in the web that suggests to configure `/tmp` in Linux to use the tmpfs filesystem as this can give you advantages for example with a Mysql for example. (when there is a reasonable amout of SELECTs opting for filesort)

The tmpfs is not a ramdisk. A ram disk just resides in the RAM, a tmpfs resides in the virtual memory, thus it uses the swap when the memory is needed for something else. Furthermore it doesn't have a predefined size, albeit you can define a maximum size for the tmpfs to ensure that someone who writes into the `/tmp` can't eat away all your virtual memory.

It's called `tmpfs` because everything you write into it is temporary, the next boot or unmount will kill all the files on it. When you look at the mount table of a Solaris System you will recognize, that the usual locations for such temporary files are mounted tmpfs:

```
jmoekamp@a380:/var$ mount | grep "swap"
/etc/svc/volatile on swap read/write/setuid/devices/xattr/dev=4
   f00001 on Fri Jul 31 06:33:33 2009
/tmp on swap read/write/setuid/devices/xattr/dev=4f00002 on Fri
    Jul 31 06:34:14 2009
/var/run on swap read/write/setuid/devices/xattr/dev=4f00003 on
    Fri Jul 31 06:34:14 2009
```

Keeping these file systems in virtual memory is a reasonable choice. The stuff in this directory is normally stale after a reboot, most of the time the files are many, but rather small and putting them on disk would just eat away your IOPS budget on your boot disks. As the file system resides in memory, it's much faster and that really helps on jobs with many small files. A good example is compiling software when you use `/tmp` on as the `TMPDIR`.

All this advantages come with a big disadvantage, when you are not aware of the nature of the `/tmp` directory. I assume, you already know why using `/tmp` for storing ISOs is a bad idea. It eats away your memory and later on your swap. And for all the experienced admins: When someone has memory problems, ask at first about the `/tmp` directory, we tend to forget about this, as we've learned this lesson a long time ago and thus don't think about this problem. When you really need a temporary place to store data in it for a while you should use `/var/tmp`. This is a directory on a normal disk based filesystem and thus its content it boot persistent.

10.2. Configuring the maximum size of the tmpfs

When you want to enforce a limit on the size of a `tmpfs` this pretty easy task . You can do this with a mount option. Let's assume you want to provide the `/var/applicationscratch` pad directory for an application, but you want to be sure, that this application can't ruin your day by eating away all your memory:

```
jmoekamp@a380:/var# mkdir /var/applicationscratchpad
jmoekamp@a380:/var# mount -f tmpfs -o size=128m swap /var/
   applicationscratchpad/
jmoekamp@a380:/var# mount | grep "application"
/var/applicationscratchpad on swap read/write/setuid/devices/
   xattr/size=128m/dev=4f00004 on Fri Jul 31 09:32:26 2009
```

When you want to make a boot persistent change to the maximum size of the `/tmp` directory, you have to configure this in the `/etc/vfstab`:

```
swap    -       /tmp    tmpfs   -       yes     size=512m
```

10.3. Conclusion

I hope its now clear, why you shouldnt use `/tmp` as a place for storing big files, at least move them directly somewhere else, when you use /tmp as the target directory to ssh a file to a system. At the other side the `/tmp` in virtual memory gives you some interesting capabilities to speed up your applications. As usual, everything has two sides: Making it default in Solaris gives you speedups per default. But when you are unaware of this default situation your virtual memory may be used for collection of videos ;) Obviously the same is valid, when you configure our Linux system with a tmpfs based `/tmp` and don't tell it to your fellow admins.

10.4. Do you want to learn more?

manpages

docs.sun.com: tmpfs(7f) [1]

docs.sun.com: mount_tmpfs(1M) [2]

[1] `http://docs.sun.com/app/docs/doc/816-5177/tmpfs-7fs`
[2] `http://docs.sun.com/app/docs/doc/816-5166/mount-tmpfs-1m`

11. logadm

Solaris 10/Opensolaris

11.1. The scrolls of of super user castle

The mighty root couldn't sleep at night, so root walked around the castle. Deep down in the foundation of super users castle there was a strange room. It was filled with scrolls and some of the serving daemons of root filled this room with even more scrolls while root was standing there. So root looked in some of them: There were new ones, interesting ones. Then root took some old ones, blew away the dust and after a short look root thought "Damned ... those scrolls are so old, there aren't true anymore. Someone has to clean up this place".

So root spoke a mighty spell of infinite power and another daemon spawned from the ether: "You are the keeper of the scrolls. But don't keep everything. Just the last ones." And so it was done since the day of this sleepless night.

11.2. Housekeeping in your logfile directories

One of the regular task of any decent admin should be the housekeeping in the logfile directory. Logfiles are really useful, when something went wrong, but often they just filling the directories with data. Albeit sometimes it's useful to have even very old logfiles, most of the times you just need the recent ones in direct access.

11.2.1. logadm

One of the big conundrums with solaris is the point that few people know the `logadm` tool. It's available with Solaris since the release of Solaris 9. However it's still one of the well-kept secrets of Solaris despite the fact, that the tool is well documented and already in use Solaris. I'm often wondering what users of Solaris thing, where this `.0`-files were created ;)

Capabilities

For all the usual task surrounding logfile handling you could use the command `logadm` . It's a really capable tool:

- rotating logs (by copying/truncating or moving)
- configuring rules, when a log rotation should take place. This rules can be based on ...
  - the size of the log file
  - the time since last log rotation
- executing command before and after a logfile rotation
- compressing rotated log files based on rules
- specifying your own commands to copy/move the files
- specifying commands that should be used instead of a simple deletion for expiration of files

How it works

`logadm` is the tool to configure it as well it's the tool that do the log rotation. To automatically rotate the logs, `logadm` is executed by `cron` once a day. Let's look into the crontab of the root user.

```
jmoekamp@hivemind:/var/squid/logs# crontab -l
[... CDDL Header omitted ...]
#
10 3 * * * /usr/sbin/logadm
15 3 * * 0 [ -x /usr/lib/fs/nfs/nfsfind ] && /usr/lib/fs/nfs/nfsfind
30 3 * * * [ -x /usr/lib/gss/gsscred_clean ] && /usr/lib/gss/gsscred_clean
30 0,9,12,18,21 * * * /usr/lib/update-manager/update-refresh.sh
```

So as you may have already recognized, the `logadm` script is executed daily at 3:10 am in a Solaris default configuration.

But how is the rotation itself done? Well, exactly as you would do it by typing in commands on the shell. `logadm` does the same job - just automatically. It generates a sequence of commands to rotate the logs and send them to a c shell for execution.

11.3. Practical side of logadm

11.3.1. Configuring it

Albeit it's possible to invoke the rotation totally from command line each time, this isn't a comfortable method to rotate logfiles. In almost every case you would configure the `logadm` facility in order to do the rotation task automatically. So how is this done?

logadm.conf

The daily run of `logadm` depends on a config file. When you don't specify a config file, it's `/etc/logadm.conf` per default. The configuration is rather short and just rotates the usual suspects. The following file is a little bit extended, as the default configuration file doesn't use all important options:

```
jmoekamp@hivemind:~$ cat /etc/logadm.conf
[... CDDL Header omitted ...]
/var/log/syslog -C 8 -P 'Sat Feb 20 02:10:00 2010' -a 'kill -HUP `cat /var/run/syslog
    .pid`'
/var/adm/messages -C 4 -P 'Sat Feb 20 02:10:00 2010' -a 'kill -HUP `cat /var/run/
    syslog.pid`'
/var/cron/log -P 'Thu Dec 17 02:10:00 2009' -c -s 512k -t /var/cron/olog
/var/lp/logs/lpsched -C 2 -N -t '$file.$N'
/var/fm/fmd/errlog -M '/usr/sbin/fmadm -q rotate errlog && mv /var/fm/fmd/errlog.0-
    $nfile' -N -s 2m
/var/fm/fmd/fltlog -A 6m -M '/usr/sbin/fmadm -q rotate fltlog && mv /var/fm/fmd/
    fltlog.0- $nfile' -N -s 10m
smf_logs -C 8 -s 1m /var/svc/log/*.log
/var/adm/pacct -C 0 -N -a '/usr/lib/acct/accton pacct' -g adm -m 664 -o adm -p never
/var/log/pool/poold -N -a 'pkill -HUP poold; true' -s 512k
/var/squid/logs/access.log -P 'Tue Feb 23 06:26:23 2010'  -C 8 -c -p 1d -t '/var/
    squid/logs/access.log.$n' -z 1
```

You can edit this file directly, but it's preferred to change it with the logadm command itself. Let's dissect some lines of this configuration.

Standard log rotation - introducing `-C`,`-P` and `-a`

```
[...]
/var/log/syslog -C 8 -P 'Sat Feb 20 02:10:00 2010' -a 'kill -HUP `cat /var/run/syslog
    .pid`'
[...]
```

This line is responsible for rotating `/var/log/syslog`. The `-C 8` specifies, that the logadm should hold 8 old versions before it expires (read: deletes) old logfiles. With the `-a 'kill -HUP `cat /var/run/syslog.pid`` option the syslog gets an HUP signal after the log rotation. The `syslogd` needs this to recreate the logfile and to restart logging.

The -P isn't there in the pristine version of the file. Those options will are inserted when you run the logadm command. With this option, you tell logadm when when the last rotation was done. logadm inserts it each time it rotates a logfile. It's important to know, that this time is GMT time, so don't wonder about the offset to the local time shown with your logfiles.

In this statement you don't find a configuration of the time or size that leads to a log rotation. In this case the default values "1 byte and 1 week" are kicking in. So /var/log/syslog is rotated when the last rotation was at least one week in the past *a*nd the file is at least one Byte in size.

Explicit definition of a rotation time span - introducing -p

With -p you can control the period between log rotations. For example 1d specifies that you rotate the log files on a daily schedule.

```
[...]
/var/squid/logs/access.log -C 8 -c -p 1d -t '/var/squid/logs/access.log.$n'
[...]
```

So this logfiles is rotated every day by the logadm execution initiated by the entryin the crontab.

A template for the name of the rotated log - introducing -t

-t specifies the way, how the names for logfiles are created by logadm. The template for the new name of the logfile isn't just a fixed string, you can use several variables to control the name of the file.

```
[...]
/var/squid/logs/access.log -C 8 -c -p 1d -t '/var/squid/logs/access.log.$n'
[...]
```

This is a relatively simple example. $n is just the version number of the file, starting with 0. Thus a configuration would lead to a filename like this one:

```
-rw-r----- 1 webservd webservd 652486 2010-02-22 16:36 /var/squid/logs/access.log.0
```

$n is just one possible variable available for use in the templates. The logadm man page specifies further possible variables.

Explicit definition of maximum logfile size - introducing -s

Sometimes you want to set your current logfile a limit based on file size and not based on a time interval. The `-s` option allows you to do so:

```
[...]
/var/cron/log -P 'Thu Dec 17 02:10:00 2009' -c -s 512k -t /var/cron/olog
[...]
```

With this option `logadm` will rotate the logfile as soon it's 512k or larger at the time `logadm` runs.

Specifying both: Space and time

When you specify a maximum logfile size (`-s`) as well as a maximum logfile period (`-p`), both conditions are connected with an AND. So the default "1 byte and 1 week" can be translated: Rotate when the logfile has a size of at least 1 byte AND it's one week old, thus a week old zero-length logfile is not rotated by the default configuration.

Copy and truncate instead of moving - introducing -c

Rotating logfiles isn't unproblematic. Some application don't like it if you simply move the file away. They may use the rotated log instead of a new one, or they simply don't create a new logfile. Thus you have to restart the service or send a signal. There is an alternative. It's called truncating.

```
[...]
/var/cron/log -P 'Thu Dec 17 02:10:00 2009' -c -s 512k -t /var/cron/olog
[...]
```

The `-c` option forces `logadm` to use `cp` instead of `mv` to rotate the log. To get a fresh start in the new log `/dev/null` is copied to the current logfile effectively make a 0 byte file out out it. This circumvents the need for a restart of the application to restart logging.

Compress after rotation - introducing -z

Due to their structure, logfiles can yield an excellent compression ratio. So it's sensible to compress them after rotation. You can configure this with the `-z` session. However often it's somewhat unpractical to work with compressed files, thus the parameter after `-z` forces `logadm` not to compress the stated number of the most recent log files. By doing so you have the recent logfiles available for processing without decompressing, without sacrifying space by leaving all logfiles uncompressed.

```
[...]
/var/squid/logs/cache.log -C 8 -c -p 1d -t '/var/squid/logs/access.log.$n'  -z 1
[...]
```

`-z 1` configures logadm to leave the most recent rotated logfile untouched and compresses the next one. This results to the following files:

```
jmoekamp@hivemind:/var/squid/logs$ ls -l /var/squid/logs/access*
-rw-r----- 1 webservd webservd      0 2010-02-23 07:26 /var/squid/logs/access.log
-rw-r----- 1 webservd webservd 652486 2010-02-22 16:36 /var/squid/logs/access.log.0
-rw-r----- 1 webservd webservd  39411 2010-02-22 09:22 /var/squid/logs/access.log.1.
    gz
```

A log rotation that is never executed automatically - introducing `-p never`

At the first moment, this line looks really strange.

```
[...]
/var/adm/pacct -C 0 -N -a '/usr/lib/acct/accton pacct' -g adm -m 664 -o adm -p never
[...]
```

What is the sense of a configuration line that is never used automatically in a log rotation script. Well, the fact that you don't use it automatically, doesn't mean that you can't use it manually from the command line or that other scripts can't trigger the logadm functionality for log rotation instead of implementing their own.

The config line printed above is such a line enabling the use of logadm log rotation in other scripts. When you look into the script `/usr/lib/acct/turnacct`, you will see a line triggering logadm with `-p now`:

```
[...]
switch)
        pfexec /usr/sbin/logadm -p now /var/adm/pacct
        if test ! -r pacct
        [...]
```

The `turnacct` script triggers the log rotation. Due to the `-p never` in the `logadm` it's the only way to trigger the rotation. I will explain later on what happens due to this command.

Wildcards

The man page of logadm gives a good example for the usage of wildcard and "regular expressions":

```
logadm -w apache -p 1m -C 24\
           -t '/var/apache/old-logs/$basename.%Y-%m'\
           -a '/usr/apache/bin/apachectl graceful'\
           '/var/apache/logs/*{access,error}_log'
```

With this statement you rotate all log files ending on `access_log` or `error_log` in your apache log directory. At the beginning of the line you may have recognized the `apache` name. This is a shorthand for this configuration. So you can simply rotate all logfiles with `logadm -p now apache` instead of the longer name of the directory. Additionally it's important that the command specified by `-a` is executed once after all matching logfiles were rotated and not for each one.

More options

There are several other option controlling the behaviour of a rotation. I will describe those options by short examples.

- `-A 1y`: With `-A` is an additional way to control the expiration of a rotated logfile. For example, this example forces `logadm` to expire rotated logfiles when they are older than a year.
- `-S 100m'`: Perhaps you don't want to specify a expiration limit by age, but by the size all the rotated logfiles takes. You can do so by using the `-S` statement. This example will expire rotated logfiles when they take more than 100 Megabyte.
- `-E 'mv $nfile /longtermstorage'`: This option controls what happens when a rotated logfiles expires. The standard behaviour is a simple deletion. This example moves the expired file to a long term storage, for example an SamFS filesystem mounted via NFS on a different server.
- `-b 'svcadm disable nonbehavingapp' -a 'svcadm enable nonbehavingapp'`: There isn't just an option to execute a command after log rotation with `-a`, there is one for executing commands before log rotation as well. You use the `-b` option to specify this account This is really useful to stop and start a non well behaving application for log rotation.
- `-R 'loganalyser.pl $file'`: the command specified with this option is executed after each log rotation. At first this sounds redundant to `-a` but there is an important difference. `-a` is executed once, even when the line in `logadm.conf` matched several logfiles. The command `-R` is executed for each rotated log. So `-a` is more for signaling a process, that it should start a new log, `-R` is more useful for log processing. When you use `-R` for signaling, the process would get a signal for each logfile you rotate.
- `-M 'mv $file $nfile`: With `-M` you can specify your own command for rotation. This example is the default for a `logadm` run.

11.3.2. Control options

So far i just mentioned the options needed for configuration. But there are some additional options to control what

Changing `logadm.conf` with `logadm`

Okay, how do you put the commands to the `logadm.conf` file. You can edit the file directly , it will work. But `logadm` can be used to add and delete statements to the config file as well. When you add a line to the `logadm.conf` file, prepend the line with a `-w`

```
jmoekamp@hivemind:/var/squid/logs# logadm -w /var/squid/logs/cache.log -C 8 -c -p 1d
    -t '/var/squid/logs/cache.log.$n' -z 1
```

You can check the addition with the `logadm -V` command. `-V` triggers the validation of the file.Thus it's sensible to run this command after a direct addition to the file as well.

```
jmoekamp@hivemind:/var/squid/logs# logadm -V
[...]
/var/squid/logs/cache.log -C 8 -c -p 1d -t '/var/squid/logs/cache.log.$n' -z 1
```

Okay, we've worked a while with this log rotation configuration. You will notice that, that the configuration line has changed.

```
jmoekamp@hivemind:/var/squid/logs# logadm -V
/var/log/syslog -C 8 -P 'Sat Feb 20 02:10:00 2010' -a 'kill -HUP `cat /var/run/syslog
    .pid`'
[...]
/var/squid/logs/access.log -P 'Tue Feb 23 19:19:27 2010' -C 8 -c -p 1d -t '/var/squid
    /logs/access.log.$n' -z 1
/var/squid/logs/cache.log -C 8 -P 'Tue Feb 23 19:59:20 2010' -c -p 1d -t '/var/squid/
    logs/cache.log.$n' -z 1
```

Now you want to get rid of this configuration statement. This is done with the `-R` option in conjunction with the name of the logfile.

```
jmoekamp@hivemind:/var/squid/logs# logadm -r /var/squid/logs/cache.log
jmoekamp@hivemind:/var/squid/logs# logadm -V
/var/log/syslog -C 8 -P 'Sat Feb 20 02:10:00 2010' -a 'kill -HUP `cat /var/run/syslog
    .pid`'
[...]
/var/squid/logs/access.log -P 'Tue Feb 23 19:19:27 2010' -C 8 -c -p 1d -t '/var/squid
    /logs/access.log.$n' -z 1
jmoekamp@hivemind:/var/squid/logs#
```

It's gone .... the `/var/squid/logs/cache.log` will not be rotated the next time `logadm` is executed by `cron` or at the command line.

Forcing an immediate rotation

Sometimes you want to execute an immidiate log rotation. For example because you want to restart a service with a fresh log to ease analysis. `-p now` executes the log rotation right in the moment you start `logadm`

```
jmoekamp@hivemind:/var/squid/logs# date
Dienstag, 23. Februar 2010, 20:48:25 Uhr CET
jmoekamp@hivemind:/var/squid/logs# logadm -p now squid_cachelog
jmoekamp@hivemind:/var/squid/logs# ls -l cache.log*
-rw-r----- 1 webservd webservd      0 2010-02-23 20:48 cache.log
-rw-r----- 1 webservd webservd 636799 2010-02-23 20:41 cache.log.0
jmoekamp@hivemind:/var/squid/logs#
```

With `-p now` you can even execute log rotations that are never executed automatically.

What happens under the hood?

As i wrote at the beginning, logadm works by sending a lot of commands to a c-shell. The `logadm` has an interesting mode of operation. When you use the `-v` option the tools shows the commands `logadm` uses to rotate the logfiles.

```
jmoekamp@hivemind:/var/squid/logs#  logadm -p now -v squid_access
# loading /etc/logadm.conf
# processing logname: squid_access
mkdir -p /var/squid/logs # verify directory exists
cp -fp /var/squid/logs/access.log.2.gz /var/squid/logs/access.log.3.gz # rotate log
    file via copy (-c flag)
cp -f /dev/null /var/squid/logs/access.log.2.gz # truncate log file (-c flag)
mkdir -p /var/squid/logs # verify directory exists
cp -fp /var/squid/logs/access.log.1.gz /var/squid/logs/access.log.2.gz # rotate log
    file via copy (-c flag)
cp -f /dev/null /var/squid/logs/access.log.1.gz # truncate log file (-c flag)
mkdir -p /var/squid/logs # verify directory exists
cp -fp /var/squid/logs/access.log.0 /var/squid/logs/access.log.1 # rotate log file
    via copy (-c flag)
cp -f /dev/null /var/squid/logs/access.log.0 # truncate log file (-c flag)
mkdir -p /var/squid/logs # verify directory exists
cp -fp /var/squid/logs/access.log /var/squid/logs/access.log.0 # rotate log file via
    copy (-c flag)
cp -f /dev/null /var/squid/logs/access.log # truncate log file (-c flag)
touch /var/squid/logs/access.log
chown 80:80 /var/squid/logs/access.log
chmod 640 /var/squid/logs/access.log
#     recording rotation date Tue Feb 23 18:15:44 2010 for /var/squid/logs/access.log
gzip -f /var/squid/logs/access.log.1 # compress old log (-z flag)
# writing changes to /etc/logadm.conf
```

11.4. Tips and Tricks

11.4.1. Multiple instances of logadm

It's perfectly possible to have more than one configuration file. You can choose the config file with `-f`. You just adds an additional entry in the cron table that specifies a different logfile.A colleague of me use this "feature" to have an independent log rotation config file for SamFS, so he don't have to edit the file supplied by the distribution.

11.5. Conclusion

Albeit it's just a really small tool, the possibilities of `logadm` to help you with your logfiles are vast and infinite. I'm really wondering why many people don't know about it. I hope i changed that at least a little bit by writing this tutorial.

11.6. Do you want to learn more?

man pages

docs.sun.com - logadm(1M)[1]

docs.sun.com - logadm.conf(4)[2]

[1] `http://docs.sun.com/app/docs/doc/816-5166/logadm-1m`
[2] `http://docs.sun.com/app/docs/doc/816-5174/logadm.conf-4`

Part III.

Solaris Security

12. Role Based Access Control and Least Privileges

Solaris 10/Opensolaris

12.1. Introduction

12.1.1. The Story of root

And then there was root. And root was almighty. And that wasn't a good thing. root was able to control the world without any control. And root needed control. It was only a short chant between the mere mortals and root. Everybody with the knowledge of the magic chant was able to speak through root.

But root wasn't alone. root had servants called daemons. Some of one them needed divine powers to do their daily job. But root was an indivisible being. So the servants had to work with the powers of root. But the servants wasn't as perfect as root: Some of the servants started to do everything mere mortals said to them if they only said more than a certain amount of prayers at once.

One day, the world of root experienced a large disaster, the negation of being. Top became bottom, left became right, the monster of erem-ef annihilated much of the world. But it got even stranger. root destroyed its own world, and by the power of root the destruction was complete.

Then there was a FLASH. The world restarted, root got a second attempt to reign his world. But this time, it would be different world.

12.1.2. Superuser

The old model of rights in a unix systems is based on a duality. There is the superuser and the normal user. The normal users have a restricted set of rights in the system, the superuser has an unrestricted set of rights. To modify the system, a normal user has to login as root directly or assume the rights of root (by su -). But such a user has unrestricted access to system. Often this isn't desirable. Why should you enable

an operator to modify a system, when all he or she has do to on the system is creating some users from time to time. You've trained him to do useradd or passwd, but it's a Windows admin who doesn't know anything about being an Unix admin. What do you do when he gets to curious. He needs root privileges to create a user or change a password. You need some mechanisms to limit this operator.

But it's get more problematic. Programs have to modify the system to work. A webserver is a nice example. It uses port 80. Ports beneath port number 1024 have a special meaning. They are privileged ports. You need special rights to modify the structures of the system to listen to the port 80. A normal user doesn't have this rights. So the webserver has to be started as root. The children of this process drop the rights of root by running with a normal user. But there is this single instance of the program with all the rights of the user. This process has much rights than needed, a possible attack vector for malicious users.

This led to the development to different models of handling the rights of users in the system: Privileges and Role Based Access Control.

12.1.3. Least privileges

There is a concept in security. It's called least privileges. You give someone only least amount of privileges, only enough to do its tasks. An example of the real world. You won't give the janitor the master key for all the rooms on the campus, when all he has to do is working in Building C. The other way round: There are some trusted people who have access to all rooms in case of emergency.

You have the same concept in computer security. Everyone should have only the least amount of privileges in the system to do its job. The concept of the superuser doesn't match to this. It's an all or nothing. You are an ordinary user with basic privileges or you are an user with unrestricted rights. There is nothing in between. There is no least privileges in this concept.

12.1.4. Role Based Access Control

The example with the key for the janitors is a good example. Let's imagine a large campus. You have janitors responsible for the plumbing (let's call them Lenny and Carl), for the park (let's call him Homer), for the security system (let's call Marge, Lenny helps from time to time).

These roles have different sets of privileges: For example the plumbing janitor has access to all rooms of the heating system. The janitor for the park has only access to the garage with the lawnmover and the cafeteria and the janitor for the security system.

When they start to work in their job, they assume a role. From the privilege perspective it's not important who is the person, but what role the person has assumed. Lenny punches the clock and assumes the role of the plumbing janitor for the next 8 hours. And while he is doing its job he uses the privileges inherent to the role. But he has to do tasks in his office or in his workshop. It's his own room, so he doesn't need the privileges. He doesn't need the special privileges.

Role Based Access Control is quite similar. You login to the system, and then you start work. You read your emails (no special privileges needed), you find an email "Create user xy45345. Your Boss". Okay, now you need special privileges. You assume the role of an User Administrator create the user. Job done, you don't need the privileges anymore. You leave the role and write the "Job done" mail to your boss with your normal users.

Role Based Access Control is all about this: Defining roles, giving them privileges and assigning users to this roles.

12.1.5. Privileges

I've used the word quite often in the article so far. What is a privilege. A privilege is the right to do something. For example, having the keys for the control panel of the heating system.

Unix users are nothing different. Every user has privileges in a unix system. A normal user has the privilege to open, close, read write and delete files when he his allowed to do this (Because he created it, because he belongs to the same group as the create of the file or the creator gave everybody the right to do it). This looks normal to you, but it's privilege based on the login credentials you gave to system. You don't have the privilege to read all files on the system or to use a port number 1024.

Every thing done in the system is based on this privileges. Solaris separated the tasks into many privilege sets. At the moment, there are 70 different privileges in the system. The difference between the normal user is that the users has only a basic set, the root has all.

But it hasn't to be this way. Privileges and users aren't connected with each other. You can give any user the power of the root user, and restrict the privileges of the root user. It's just our binary compatibility guarantee that mandates that the standard configuration of the system resembles the superuser model. There are application out there, which assume that only the root user or the uid 0 as unrestricted rights and exit when they are started with a different user.

12.1.6. RBAC and Privileges in Solaris

Both features have their root in the Trusted Solaris development. Trusted Solaris was a version of Solaris to ensure highest security standards. Today, these mechanisms are part of the normal Solaris in conjunction with the Trusted Extensions. So RBAC is a really old feature: It's in Solaris since version 8 (published in 2000). Privileges found their way into the generic Solaris with the first availability of Solaris 10 in February 2005.

12.2. Some basic terms

As usual the world of RBAC has some special terms. Before using it, im want to explain the jargon. I copy the exact definition from the RBAC manual:

- Rights: A right is the description, to execute an executable as an privileged user.For example the permission to execute the command `reboot` as root.
- Authorization: A permission that enables a user or role to perform a class of actions that could affect security. For example, security policy at installation gives ordinary users the solaris.device.cdrw authorization. This authorization enables users to read and write to a CD-ROM device
- Right Profiles: A collection of administrative capabilities that can be assigned to a role or to a user. A rights profile can consist of authorizations, of commands with security attributes, and of other rights profiles. Rights profiles offer a convenient way to group security attributes.
- Role: A special identity for running privileged applications. The special identity can be assumed by assigned users only. In a system that is run by roles, superuser is unnecessary. Superuser capabilities are distributed to different roles.

12.3. Practical side of RBAC

After so much theory, let's work with roles. After installation of a Solaris system there are no rules assigned to a normal user:

```
$ roles
No roles
```

Let's use the standard example for RBAC: reboot the system. To do this task, you need to be root.

```
$ /usr/sbin/reboot
reboot: permission denied
```

You are not allowed to do this. Okay, until now you would give the root account to all people, who have to reboot the system. But why should someone be able to modify users, when all he or she should to is using the reboot command ?

Okay, at first you create a role. As mentioned before, it's a special user account.

```
# roleadd -m -d /export/home/reboot reboot
64 blocks
```

After creating the role, you have to assign a role password.

```
# passwd reboot
New Password:
Re-enter new Password:
passwd: password successfully changed for reboot
```

Okay, when you look into the /etc/passwd, you see a quite normal user account.

```
# grep reboot /etc/passwd
reboot:x:101:1::/export/home/reboot:/bin/pfsh
```

There is one important difference. You use a special kind of shell. This shell are called profile shells and have special mechanisms to check executions against the RBAC databases.

Okay, we've created the role, now we have to assign them to a user:

```
# usermod -R reboot jmoekamp
UX: usermod: jmoekamp is currently logged in, some changes may
    not take effect until next login.
```

The RBAC system stores the role assignments in the `/etc/user_attr` file

```
# grep "jmoekamp" /etc/user_attr
jmoekamp::::type=normal;roles=reboot
```

But at the moment, this role isn't functional, as this role has no assigned role profile. It's a role without rights an privileges.

At first, lets create a REBOOT role profile. It's quite easy. Just a line at the end of `prof_attr`. This file stores all the attributes of

```
# echo "REBOOT:::profile to reboot:help=reboot.html" >> /etc/
    security/prof_attr
```

Okay, now assign the role profile REBOOT to the role reboot

```
# rolemod -P REBOOT reboot
```

The information of this assignment is stored in the `/etc/usr`. Let's have a look into it:

```
# grep reboot /etc/user_attr
reboot::::type=role;profiles=REBOOT
jmoekamp::::type=normal;roles=reboot
```

But this isn't enough: The profile is empty. You have to assign some administrative command to it.

```
# echo "REBOOT:suser:cmd:::/usr/sbin/reboot:euid=0" >> /etc/
   security/exec_attr
```

12.4. Using the new role

Okay, let's check the role assignments.

```
$ roles
reboot
```

We have still no rights to execute the reboot command.

```
$ /usr/sbin/reboot
reboot: permission denied
```

But now we assume the reboot role.

```
$ su reboot
Password:
```

And as you see ...

```
$ /usr/sbin/reboot
Connection to 10.211.55.3 closed by remote host.
Connection to 10.211.55.3 closed.
```

Connection terminates, systems reboots.

12.5. Authorizations

But RBAC can do more for you. There is an additional concept in it: Authorizations.

Authorizations is a mechanism that needs support of the applications. This application checks if the user has the necessary authorization to use a program.

Let's use the example of the janitor: Rights give him the access to the drilling machine. But this is a rather strange drilling machine. It checks, if the janitor has the permission to drill holes, when he trigger the button.

The concept of authorization is a fine grained system. An application can check for a vast amount of privileges. For example the application can check for the authorization to modify the configuration, to read the configuration or printing the status. A user can have all this authorizations, none or something in between.

It's like the janitors new power screwdriver. It checks if the janitor has the permission to use it at anticlockwise rotation, the permission to use it at clockwise rotation and the permission to set different speeds of rotation.

12.6. Using authorizations for Services

Although applications need support to this models, you can even use it as an admin. SMF has build in support for authorizations. You can assign authorizations to a service. Every role or user with this authorization is allowed to work with the service (restarting,stop,start,status, ...). Let's use Apache for this example.

A normal user has no permission to restart the service:

```
moekamp$ /usr/sbin/svcadm -v disable -s apache2
svcadm: svc:/network/http:apache2: Couldn't modify "general"
   property group (permission denied).
```

Wouldn't it be nice, to have an authorisation that enables an regular user to restart it? Okay, no problem. Let's create one:

```
$ su root
# echo "solaris.smf.manage.apache/server:::Apache Server
   management::" >> /etc/security/auth_attr
```

That's all. Where is the definition of the permission that the authorization means? There is no definition. It's the job of the application to work with.

Now assign this authorization to the user:

```
# usermod -A solaris.smf.manage.apache/server jmoekamp
UX: usermod: jmoekamp is currently logged in, some changes may
   not take effect until next login.
```

Okay, but at the moment no one checks for this authorization, as no application is aware of it. We have to tell SMF to use this authorization.

The authorizations for an SMF servers is part of the general properties of the service. Let's have a look at the properties of this services.

```
# svcprop -p general apache2
general/enabled boolean false
general/entity_stability astring Evolving
```

No authorization configured. Okay ... let's add the authorization we've defined before:

```
svccfg -s apache2 setprop general/action_authorization=astring:
    'solaris.smf.manage.apache/server'
```

Check the properties again:

```
# svcadm refresh apache2
# svcprop -p general apache2
general/enabled boolean false
general/action_authorization astring solaris.smf.manage.apache/
   server
general/entity_stability astring Evolving
```

Okay, a short test. Exit your root shell and login as the regular user you have assigned the authorization.

```
bash-3.2$ svcs apache2
STATE          STIME    FMRI
disabled       22:49:51 svc:/network/http:apache2
```

Okay, I can view the status of the service. Now I try to start it.

```
bash-3.2$ /usr/sbin/svcadm enable apache2
svcadm: svc:/network/http:apache2: Permission denied.
```

What the hell ...? No permission to start the service? Yes, enabling the service is not only a method (the start up script), it's a value of a certain parameter. When you only have the action_authorization you can only do task, that doesn't change the state of the service. You can restart it (no change of the service properties), but not enable or disable it (a change of the service properties). But this is not a problem. You have to login as root again and assign the `solaris.smf.manage.apache/server` authorization to the value authorization.

```
# svccfg -s apache2 setprop general/value_authorization=astring
   : 'solaris.smf.manage.apache/server'
```

With the value authorization SMF allows you to change the state of the service. Try it again.

```
bash-3.2$ /usr/sbin/svcadm enable apache2
bash-3.2$
```

Cool, isn't it ... try this with init.d ... ;)

12.7. Predefined roles

This was a really simple example. Roles can get really complex. But you don't have to define all role profiles at your own. For some standard tasks, there are some predefined roles. Just look at the `/etc/security/prof_attr`. There are 70 role profiles defined in this file. For example the right profile "Software Installation"

```
Software Installation:::Add application software to the system:
   help=RtSoftwareInstall.html; auths=solaris.admin.prodreg.
   read, solaris.admin.prodreg.modify, solaris.admin.prodreg.
   delete,solaris.admin.dcmgr.admin, solaris.admin.dcmgr.read,
   solaris.admin.patchmgr.*
```

This role profile has already some predefined command, that need special security attributes to succeed:

```
Software Installation:solaris:act:::Open;*;JAVA_BYTE_CODE;*;*:
   uid=0;gid=2
Software Installation:suser:cmd:::/usr/bin/ln:euid=0
Software Installation:suser:cmd:::/usr/bin/pkginfo:uid=0
Software Installation:suser:cmd:::/usr/bin/pkgmk:uid=0
Software Installation:suser:cmd:::/usr/bin/pkgparam:uid=0
Software Installation:suser:cmd:::/usr/bin/pkgproto:uid=0
Software Installation:suser:cmd:::/usr/bin/pkgtrans:uid=0
Software Installation:suser:cmd:::/usr/bin/prodreg:uid=0
Software Installation:suser:cmd:::/usr/ccs/bin/make:euid=0
Software Installation:suser:cmd:::/usr/sbin/install:euid=0
Software Installation:suser:cmd:::/usr/sbin/patchadd:uid=0
Software Installation:suser:cmd:::/usr/sbin/patchrm:uid=0
Software Installation:suser:cmd:::/usr/sbin/pkgadd:uid=0;gid=
   bin
Software Installation:suser:cmd:::/usr/sbin/pkgask:uid=0
Software Installation:suser:cmd:::/usr/sbin/pkgchk:uid=0
```

```
Software Installation:suser:cmd:::/usr/sbin/pkgrm:uid=0;gid=bin
```

This is all you need to install software on your system. You can use this predefined role profiles at your will. You don't have to do define all this stuff on your own.

12.8. Privileges

We've talked a lot about RBAC, roles, role profiles. But what are Privileges? Privileges are rights to do an operation in the kernel. This rights are enforced by the kernel. Whenever you do something within the kernel the access is controlled by the privileges.

At the moment, the rights to do something with the kernel are separated into 70 classes:

```
contract_event contract_observer cpc_cpu dtrace_kernel
   dtrace_proc dtrace_user file_chown file_chown_self
   file_dac_execute file_dac_read file_dac_search
   file_dac_write file_downgrade_sl file_flag_set file_link_any
    file_owner file_setid file_upgrade_sl graphics_access
   graphics_map ipc_dac_read ipc_dac_write ipc_owner
   net_bindmlp net_icmpaccess net_mac_aware net_privaddr
   net_rawaccess proc_audit proc_chroot proc_clock_highres
   proc_exec proc_fork proc_info proc_lock_memory proc_owner
   proc_priocntl proc_session proc_setid proc_taskid proc_zone
   sys_acct sys_admin sys_audit sys_config sys_devices
   sys_ip_config sys_ipc_config sys_linkdir sys_mount
   sys_net_config sys_nfs sys_res_config sys_resource sys_smb
   sys_suser_compat sys_time sys_trans_label win_colormap
   win_config win_dac_read win_dac_write win_devices win_dga
   win_downgrade_sl win_fontpath win_mac_read win_mac_write
   win_selection win_upgrade_sl
```

Every UNIX-System does this task hidden behind this privileges. There are many different privileges in the kernel. This privileges are not Solaris specific. It's the way to control the access to this privileges.

12.8.1. Conventional Unix

On conventional unix systems you have a root user, he has all privileges. And you have a normal user, who has only a limited set of privileges. Sometimes you need the rights of an admin to do some tasks. You don't even need to admin the system. You can only traceroute or ping a system, because both tools are setuid tools

```
$ ls -l /usr/sbin/traceroute
-r-sr-xr-x   1 root     bin        42324 Nov 21 00:09 /usr/sbin
   /traceroute
$ ls -l /usr/sbin/ping
-r-sr-xr-x   1 root     bin        51396 Nov 18 19:31 /usr/sbin
   /ping
```

setuid is nothing else than a violation of the security policy. You need a special privilege to ping: The privilege to use access ICMP. On conventional system this right is reserved to the root user. Thus the ping program has to be executed with the rights of root. The problem: At the time of the execution of the program, the program has all rights of the user. Not only to access ICMP, the program is capable to do everything on the system, as deleting files in /etc. This may not a problem with ping or traceroute but think about larger programs. An exploit in a setuid program can lead to the escalation of the users privileges. Setuid root and you are toast.

Let's have a look at the privileges of an ordinary user. There is a tool to get the privileges of any given process in the system, it's called ¡code¿ppriv¡/code¿.$$ is a shortcut for the actual process id (in this case the process id of the shell):

```
bash-3.2$ ppriv -v $$
646:    bash
flags = <none>
        E: file_link_any,proc_exec,proc_fork,proc_info,
           proc_session
        I: file_link_any,proc_exec,proc_fork,proc_info,
           proc_session
        P: file_link_any,proc_exec,proc_fork,proc_info,
           proc_session
        L: contract_event, (..) ,win_upgrade_sl
```

Every process in the system has four sets of privileges that determine if a process is enabled to use a privilege or not. The theory of privileges is quite complex. I would suggest to read the chapter "How Privileges Are Implemented" in the ¡a href="http://docs.sun.com/app/docs/doc/816-4557/prbactm-1?a=view"¿Security Services¡/a¿ manual to learn, how each set controls or is controlled other privilege sets.

At this time, I want only to explain the meaning of the first letter:

- E: effective privileges set
- P: permitted privileges set
- L: limit privileges set
- I: inheritable privileges set

You can think about the privilege sets as keyrings. The effective privilege set are the keys the janitor has on its keyring. The permitted privilege set are the keys the janitor is allowed to put on its keyring. The janitor can decide to remove some of the keys. Perhaps he thinks: I work only in room 232 today. I don't need all the other keys. I leave them in my office. When he looses his keyring he lost only the control about this single room, not about the complete campus.

The inheritable privilege is not a really a keyring. The janitor thinks about his new assistant: "Good worker, but I won't give him my key for the room with the expensive tools." The limited privilege set is the overarching order from the boss of janitor to his team leaders: "You are allowed to give your assistant the keys for normal rooms, but not for the rooms with all this blinking boxes from Sun".

At the moment the most interesting set is the *E:*. This is the effective set of privileges. This is the set of privilege effectively available to process. Compared to the full list of privileges mentioned above the set is much smaller. But this matches your experience when you use a unix system.

12.8.2. Some practical insights to the system

You logged in as a normal user, and you have only a few privileges. It's called the basic set.

```
bash-3.2$ ppriv $$
815:    bash
flags = <none>
        E: basic
        I: basic
        P: basic
        L: all
```

Okay, this example looks different than the one shown before. Nevertheless is has the same meaning. With the switch ¡code¿-v¡/code¿ you can expand the aliases.

```
bash-3.2$ ppriv -v $$
815:    bash
flags = <none>
        E: file_link_any,proc_exec,proc_fork,proc_info,
           proc_session
        I: file_link_any,proc_exec,proc_fork,proc_info,
           proc_session
        P: file_link_any,proc_exec,proc_fork,proc_info,
           proc_session
        L: contract_event,(..),win_upgrade_sl
```

Looks a little bit more familiar? Okay, now let's login as root.

```
$su root
Password:
# ppriv $$
819:    sh
flags = <none>
        E: all
        I: basic
        P: all
        L: all
```

This user has much more privileges. The effective set is much broader. The user has all privileges in the system.

12.8.3. How to give an user additional privileges

Now let's assume, you have an user, that wants to use dtrace. You need three privileges to use Dtrace: `dtrace_kernel,dtrace_proc,dtrace_user`. root has this privileges. A normal user not. Giving root to a developer? God beware! This is a prelude to disaster. But no problem. Assign the matching privileges to the user, and the user is enabled to use dtrace.

```
$ su root
Password:
# usermod -K defaultpriv=basic,dtrace\_kernel,dtrace\_proc,
   dtrace\_user jmoekamp
UX: usermod: jmoekamp is currently logged in, some changes may
   not take effect until next login.
```

Exit to the login prompt and login as the user you've assigned the privilieges.

```
$ ppriv $$
829:    -sh
flags = <none>
        E: basic,dtrace_kernel,dtrace_proc,dtrace_user
        I: basic,dtrace_kernel,dtrace_proc,dtrace_user
        P: basic,dtrace_kernel,dtrace_proc,dtrace_user
        L: all
```

Simple ...

12.8.4. RBAC and privileges combined

Well, but we can do better than that. We've learned there is a thing like RBAC. There is no reason that inhibits the assignment of privileges to a role. At first we create a role `bughunt`, for simplicity we use the `Process Management` role profile. After this we set the role password.

```
# roleadd -m -d /export/home/bughunting -P "Process Management"
    bughunt
# passwd bughunt
New Password:
Re-enter new Password:
```

Now we assign the privileges ...

```
# rolemod -K defaultpriv=basic,dtrace_kernel,dtrace_proc,
   dtrace_user bughunt
```

... and the user to the role.

```
# usermod -R bughunt jmoekamp
UX: usermod: jmoekamp is currently logged in, some changes may
   not take effect until next login.
```

As you might have espected, the user itself doesn't have the privileges to use dtrace.

```
$ ppriv $$
883:    -sh
flags = <none>
        E: basic
        I: basic
        P: basic
        L: all
```

But now assume the role ¡code¿bughunt¡/code¿

```
$ su bughunt
Password:
$ ppriv $$
893:    pfsh
flags = <none>
        E: basic,dtrace_kernel,dtrace_proc,dtrace_user
        I: basic,dtrace_kernel,dtrace_proc,dtrace_user
        P: basic,dtrace_kernel,dtrace_proc,dtrace_user
        L: all
```

And DTrace is at your service.

12.8.5. Privilege-aware programming

The idea of managing the privileges is not limited to users and their shells. In any given system you find dozens of programs as daemons.

These daemons interact in several ways with the privileges. The best way is "Privilege-aware programming". Okay. Let's assume, you code a daemon for your system. For example: You know, that you never will do an exec() call. So you can safely drop this privilege. The process modifies the permitted privilege set. The process can remove a privilege but not add it. Even when someone is able to your code, the attacker can't make an exec() call. The process doesn't even have the privilege to do such a call. And the attacker can't add the privilege again.

Several processes and programs in Solaris are already privilege aware. For example the kernel-level cryptographic framework daemon. Let's look at the privileges of the daemon.

```
# ps -ef | grep "kcfd"
  daemon   125     1   0 14:24:19 ?           0:00 /usr/lib/
     crypto/kcfd
    root   734   728   0 15:54:08 pts/1       0:00 grep kcfd
# ppriv -v 125
125:    /usr/lib/crypto/kcfd
flags = PRIV_AWARE
        E: file_owner,proc_priocntl,sys_devices
        I: none
        P: file_owner,proc_priocntl,sys_devices
        L: none
```

This daemon doesn't have even the basic privileges of a regular user. It has the only the bare minimum of privileges to do its job.

12.8.6. Non-privilege aware processes

But the world isn't perfect. Not every process is privilege aware. Thus you have to limit the privileges by other mechanisms. The service management framework comes to help. The following example is copied from Glen Brunettes Blueprint ¡a href="http://www.sun.com/blueprints/0505/819-2680.pdf"¿Limiting Service Privileges in the Solaris 10 Operating System ¡/a¿

Let's take the Apache Webserver as an example. The apache isn't privilege aware. We start the daemon via the Service Management Framework.

```
# svcadm -v enable -s apache2
svc:/network/http:apache2 enabled.
```

Okay, now we look at the processes of the Apache daemons.

```
# ps -ef | grep "apache"
webservd  1123  1122   0 19:11:54 ?           0:00 /usr/apache2
   /2.2/bin/httpd -k start
webservd  1125  1122   0 19:11:54 ?           0:00 /usr/apache2
   /2.2/bin/httpd -k start
    root  1122     1   1 19:11:50 ?           0:00 /usr/apache2
       /2.2/bin/httpd -k start
webservd  1128  1122   0 19:11:54 ?           0:00 /usr/apache2
   /2.2/bin/httpd -k start
webservd  1127  1122   0 19:11:54 ?           0:00 /usr/apache2
   /2.2/bin/httpd -k start
webservd  1126  1122   0 19:11:54 ?           0:00 /usr/apache2
   /2.2/bin/httpd -k start
webservd  1124  1122   0 19:11:54 ?           0:00 /usr/apache2
   /2.2/bin/httpd -k start
```

Six daemons running as webservd, and one running as root.

```
# ppriv  1122
1122:   /usr/apache2/2.2/bin/httpd -k start
flags = <none>
        E: all
        I: basic
        P: all
        L: all
```

As expected for a root process, this process has the complete set of privileges of a root user. Okay, now one of its children.

```
# ppriv 1124
1124:   /usr/apache2/2.2/bin/httpd -k start
flags = <none>
        E: basic
        I: basic
        P: basic
        L: all
```

Much better ... only basic privileges.

Okay, There is a reason for this configuration. On Unix systems, you have two groups of ports. Privileged ones from 1-1023 and unprivileged ones from 1024 up. You can only

bind to a privileged port with the privilege to do it. A normal user doesn't have this privilege, but root has it. And thus there has to be one process running as root. Do you remember the list of privileges for the apache process running at root. The process has all privileges but needs only one of them, that isn't part of the basic privilege set.

12.8.7. How to get rid of the root apache

Well, but it hasn't to be this way. With Solaris you can give any user or process the privilege to use a privileged port. You don't need the root process anymore.

Now, let's configure it this way. At first we have to deactivate the running apache.

```
svcadm -v disable -s apache2
svc:/network/http:apache2 disabled.
```

I won't explain the Service Management Framework here, but you can set certain properties in SMF to control the startup of a service.

```
# svccfg -s apache2
svc:/network/http:apache2> setprop start/user = astring:
   webservd
svc:/network/http:apache2> setprop start/group = astring:
   webservd
svc:/network/http:apache2> setprop start/privileges = astring:
   basic,!proc_session,!proc_info,!file_link_any,net_privaddr
svc:/network/http:apache2> setprop start/limit_privileges =
   astring: :default
svc:/network/http:apache2> setprop start/use_profile = boolean:
    false
svc:/network/http:apache2> setprop start/supp_groups = astring:
    :default
svc:/network/http:apache2> setprop start/working_directory =
   astring: :default
svc:/network/http:apache2> setprop start/project = astring: :
   default
svc:/network/http:apache2> setprop start/resource_pool =
   astring: :default
svc:/network/http:apache2> end
```

Line 2 to 4 are the most interesting ones. Without any changes, the Apache daemon starts as root and forks away processes with the webservd user. But we want to get rid of the root user for this configuration. Thus we start the daemon directly with the webservd user. Same for the group id.

Now it gets interesting. Without this line, the kernel would deny Apache to bind to port 80. webservd is a regular user without the privilege to use a privileged port. The property start/privileges sets the privileges to start the service. At first, we give the service basic privileges. Then we add the privilege to use a privileged port. The service would start up now.

But wait, we can do more. A webserver shouldn't do any hardlinks. And it doesn't send signals outside its session. And it doesn't look at processes other than those to which it can send signals. We don't need this privileges. proc_session, proc_info and file_link_any are part of the basic privilege set. We remove them, by adding a ¡code¿!¡/code¿ in front of the privilege.

Okay, we have notify the SMF of the configuration changes:

```
# svcadm -v refresh apache2
Action refresh set for svc:/network/http:apache2.
```

Until now, the apache daemon used the root privileges. Thus the ownership of files and directories were unproblematic. The daemon was able to read and write in any directory of file in the system. As we drop this privilege by using a regular user, we have to modify the ownership of some files and move some files.

```
# chown webservd:webservd /var/apache2/2.2/logs/access_log
# chown webservd:webservd /var/apache2/2.2/logs/error_log
mkdir -p -m 755 /var/apache2/run
```

We need some configuration changes, too. We have to move the LockFile and the PidFile. There wasn't one of the two configuration directives in my config file, thus I've simply appended them to the end of the file.

```
# echo "LockFile /var/apache2/2.2/logs/accept.lock" >> /etc/
   apache2/2.2/httpd.conf
# echo "PidFile /var/apache2/2.2/run/httpd.pid" >> /etc/apache2
   /2.2/httpd.conf
```

Okay, everything is in place. Let's give it a try.

```
# svcadm -v enable -s apache2
svc:/network/http:apache2 enabled.
```

Now we check for the running httpd processes:

```
# ps -ef | grep "apache2" | grep -v "grep"
webservd  2239  2235   0 19:29:54 ?           0:00 /usr/apache2
   /2.2/bin/httpd -k start
webservd  2241  2235   0 19:29:54 ?           0:00 /usr/apache2
   /2.2/bin/httpd -k start
```

```
webservd  2235     1   1 19:29:53 ?           0:00 /usr/apache2
   /2.2/bin/httpd -k start
webservd  2238  2235   0 19:29:54 ?           0:00 /usr/apache2
   /2.2/bin/httpd -k start
webservd  2240  2235   0 19:29:54 ?           0:00 /usr/apache2
   /2.2/bin/httpd -k start
webservd  2242  2235   0 19:29:54 ?           0:00 /usr/apache2
   /2.2/bin/httpd -k start
webservd  2236  2235   0 19:29:54 ?           0:00 /usr/apache2
   /2.2/bin/httpd -k start
```

You notice the difference ? There is no httpd running as root. All processes run with the userid webservd. Mission accomplished.

Let's check the privileges of the processes. At first the one, who ran as root before.

```
# ppriv 2235
2235:   /usr/apache2/2.2/bin/httpd -k start
flags = <none>
        E: basic,!file_link_any,net_privaddr,!proc_info,!
           proc_session
        I: basic,!file_link_any,net_privaddr,!proc_info,!
           proc_session
        P: basic,!file_link_any,net_privaddr,!proc_info,!
           proc_session
        L: all
```

Only the least privileges to do the job, no root privileges.

And even the other processes are more secure now:

```
# ppriv 2238
2238:   /usr/apache2/2.2/bin/httpd -k start
flags = <none>
        E: basic,!file_link_any,net_privaddr,!proc_info,!
           proc_session
        I: basic,!file_link_any,net_privaddr,!proc_info,!
           proc_session
        P: basic,!file_link_any,net_privaddr,!proc_info,!
           proc_session
        L: all
```

Before we changed the configuration of the webserver, it has the basic privileges of a regular user. Now we limited even this set.

12.9. The story of root - reprise

And then there was root again. And root was almighty. And this time, it wasn't a bad thing. root was able to distribute the powers between the servants. But no one was as powerful as root. They had only the powers to do their job. But not more. One servant had still the problem with to many prayers, but now this servant wasn't able to destroy the world.

Even the mere mortals learned to use the power of root. Only a small group of trusted priests had the knowledge of magic chant to speak through root. All other had only the knowledge of minor chants to pray for their special wishes.

root got really cautious to give away his powers. From this day forth root casted only the absolutely needed powers to his servants and believers. And the world of root was a more secure place. It was still possible to destroy the world, but now it was much harder as there were now several guardians to protect the power of root.

The End

12.10. Interesting Links

Documentation

System Administration Guide: Security Services - Roles, Rights Profiles, and Privileges

RBAC

SMF and RBAC authorizations

Sun Whitepaper: RBAC in the Solaris Operating Environment

Custom Roles Using RBAC in the Solaris OS

Privileges

Limiting Service Privileges in the Solaris 10 Operating System

Privilege Debugging in the Solaris 10 Operating System

13. The Solaris Security Toolkit

Solaris 10/Opensolaris

When you want to place a system into a network, it's a good practice to harden the system. Hardening is the configuration of a system to minimize the attack vectors for an intruder by closing down services, configuring stricter security policies and activating a more verbose logging or auditing.

But hardening is not a really simple task: You have to switch off as much services as possible and modify the configuration of many daemons. Furthermore you have to know, what your application needs to run, you can't close down a service that another service needs to execute. Those dependencies may be simple for a server with an apache daemon, but to harden a Sun Cluster needs a little bit more knowledge. Furthermore you have to keep the configuration in a way, that's supported by Sun.

13.1. What is the Solaris Security Toolkit?

People at Sun do this at their everyday job, thus we have experience to do such hardings. It would be nice to have this knowledge in a framework to automate all this steps. The Sun Security Toolkit was designed with this objective. As the SST website states:

> The Solaris Security Toolkit, formerly known as the JumpStart Architecture and Security Scripts (JASS) toolkit, provides a flexible and extensible mechanism to harden and audit Solaris Operating Systems (OSs). The Solaris Security Toolkit simplifies and automates the process of securing Solaris Operating Systems and is based on proven security best practices and practical customer site experience gathered over many years. This toolkit can be used to secure SPARC-based and x86/x64-based systems.

The SST is an old tool. I use it for years to harden my own systems and in the past any Jumpstart installation made at customer sites contained this toolkit to give them automatic hardening. Furthermore it's a good practice to use this toolkit on freshly installed systems as a first step in the deployment process of new server hardware before you start to install your application.

¡h4¿How to install the Solaris Security Toolkit?¡/h4¿Installation of the Toolkit is really easy. At first you have to gather it from the Sun Download Center. Sorry, you need a account for it, but you can register for free. You will find it here.Before login in as root, I've copied the file `SUNWjass-4.2.0.pkg.tar.Z` via scp to my freshly installed system with Solaris 10 Update 5

```
# cd /tmp
# ls
SUNWjass-4.2.0.pkg.tar.Z  hsperfdata_root           typescript
hsperfdata_noaccess       ogl_select216
# bash
# uncompress SUNWjass-4.2.0.pkg.tar.Z
# tar xfv SUNWjass-4.2.0.pkg.tar
x SUNWjass, 0 bytes, 0 tape blocks
x SUNWjass/pkgmap, 33111 bytes, 65 tape blocks
[...]
x SUNWjass/install/preremove, 1090 bytes, 3 tape blocks
x SUNWjass/install/tsolinfo, 52 bytes, 1 tape blocks
```

Now let's install the package:

```
# pkgadd -d . SUNWjass

Processing package instance <SUNWjass> from </tmp>

Solaris Security Toolkit 4.2.0(Solaris) 4.2.0
Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
Use is subject to license terms.
Using </opt> as the package base directory.
## Processing package information.
## Processing system information.
## Verifying package dependencies.
## Verifying disk space requirements.
## Checking for conflicts with packages already installed.
## Checking for setuid/setgid programs.

Installing Solaris Security Toolkit 4.2.0 as <SUNWjass>

## Installing part 1 of 1.
/opt/SUNWjass/Audit/disable-IIim.aud
[...]
/opt/SUNWjass/sysidcfg <symbolic link>
[ verifying class <none> ]

Installation of <SUNWjass> was successful.
```

Now I can use the Sun Security Toolkit for system hardening. It's installed at `/opt/SUNWjass/`

13.2. A look into the framework

At the end the SST is a collection of scripts and files and some code to distribute those changes throughout the system.

Each script was developed to execute a single job in the process of system hardening. When you look into the directory `/opt/SUNWjass/Finish` you will find a vast amount of them. For example `enable-bart.fin` for the automatic execution of BART to generate a system baseline or `disable-remote-root-login.fin` to automatically disable root logins, when an admin had activated those logins.

On the other side you will some configuration files in the Sun Security Toolkit as well. Sometimes a service needs some additional configuration for hardening, for example an really paranoid `hosts.deny`. Those configuration templates are contained in the directory `/opt/SUNWjass/Files`.

But you don't use both types of directly ... you use them in collections called drivers. You find those drivers in `/opt/SUNWjass/Drivers`. These drivers execute all such scripts in a certain sequence. Some drivers are really simple ... they just call other drivers. A good example is the `secure.driver` which just calls the `hardening.driver` and the `configuration.driver`. The `hardening.driver` is a better example. This driver calls many of the scripts mentioned before.

At this moment, you should take some time and examine some files in `/opt/SUNWjass/Drivers`, `/opt/SUNWjass/Files` and `/opt/SUNWjass/Finish` to get more insight into the inner workings of SST. Don't be shy, it's just clever shell scripting.

13.3. Use the Solaris Security Toolkit for hardening

A tip from the field at first: Open a second ssh connection to the system, login as root, and leave this window untouched and minimized. This connection is quite handy to have a second limb, when you sawed away the one you sit on.

Okay, how do you use such a driver? This is really simple. But you shouldn't execute the drivers blindly. They can lock you out of your system in the case you use them without caution. So change in to the directory for the Drivers:

```
# cd /opt/SUNWjass/Drivers
```

Now you should look into the desired drivers. An example: The `hardening.driver` contains a like to disable the nscd.

```
disable-nscd-caching.fin
```

But you want another behavior for some reason. You just have to add an `#` in front of the line:

```
#               disable-nscd-caching.fin
```

Well, there is another behavior I don't want. The default locks sshd via tcpwrapper except from accesses from the local host. But there is a better template at `/SUNWjass/Files/etc/hosts.allow` allowing ssh access from all hosts. You can force SST to use it by adding another line to the `hardening.driver`. I've added a line to do so:

```
JASS_FILES="
                /etc/hosts.allow
                /etc/dt/config/Xaccess
                /etc/init.d/set-tmp-permissions
                /etc/issue
                /etc/motd
                /etc/rc2.d/S00set-tmp-permissions
                /etc/rc2.d/S07set-tmp-permissions
                /etc/syslog.conf
"
```

Now the Toolkit copies the file `/opt/SUNWjass/Files/etc/hosts.allow` to `/etc/hosts.allow`. As you may have noticed, the template as to be in the same directory as the file you want to substitute with the difference, that the directory of the template has to be relative to `/opt/SUNWjass/Files/` and not to `/`

Okay, now we have modified our driver, now we can execute it:

```
# cd /opt/SUNWjass/bin
# ./jass-execute secure.driver
[NOTE] The following prompt can be disabled by setting
   JASS_NOVICE_USER to 0.
[WARN] Depending on how the Solaris Security Toolkit is
   configured, it
is both possible and likely that by default all remote shell
   and file transfer
access to this system will be disabled upon reboot effectively
   locking out any
user without console access to the system.
```

```
Are you sure that you want to continue? (yes/no): [no]
yes
```

This warning is not a joke. Know what you do, when you use this toolkit. Hardening means "real hardening" and this process may leave you with a paranoid `hosts.allow` locking you out from accessing the `sshd` on your system. Without console access you would be toast now. But as we use the more sensible template for `hosts.allow`, we can proceed by answering with `yes`:

```
Executing driver, secure.driver

============================================================================

secure.driver: Driver started.
============================================================================




============================================================================

Toolkit Version: 4.2.0
Node name:       gondor
Zone name:       global
Host ID:         1911578e
Host address:    10.211.55.200
MAC address:     0:1c:42:24:51:b9
OS version:      5.10
Date:            Wed Apr 16 16:15:19 CEST 2008
============================================================================
```

After a first status report , a long row of scripts will print log messages to the terminal. For example the finish script `enable-coreadm.fin`:

```
============================================================================

secure.driver: Finish script: enable-coreadm.fin
============================================================================


Configuring coreadm to use pattern matching and logging.

[NOTE] Creating a new directory, /var/core.
[NOTE] Copying /etc/coreadm.conf to /etc/coreadm.conf.JASS
   .20080416161705
```

```
coreadm change was successful.
```

Many reports of this kind will scroll along and at the end `jass.execute` prints out some diagnostic:

```
================================================================================

secure.driver: Driver finished.
================================================================================



================================================================================

[SUMMARY] Results Summary for APPLY run of secure.driver
[SUMMARY] The run completed with a total of 97 scripts run.
[SUMMARY] There were  Failures in   0 Scripts
[SUMMARY] There were  Errors   in   0 Scripts
[SUMMARY] There were  Warnings in   3 Scripts
[SUMMARY] There were  Notes    in  81 Scripts

[SUMMARY] Warning Scripts listed in:
        /var/opt/SUNWjass/run/20080416161504/jass-script-
           warnings.txt
[SUMMARY] Notes Scripts listed in:
        /var/opt/SUNWjass/run/20080416161504/jass-script-notes.
           txt

================================================================================
```

When you look around at you system, you will notice some new files. Every file in the system changed by the SST will be backed up before the change is done. For example you will find a file named `vfstab.JASS.20080416161812` in `/etc`. JASS contains a finish script to limit the size of the `/tmp`. As the `/tmp` filesystem resides in the main memory, this is a sensible thing to do.

Let's check for the differences

```
# diff vfstab vfstab.JASS.20080416161812
11c11
< swap - /tmp tmpfs - yes size=512m
---
> swap  -       /tmp    tmpfs   -       yes     -
```

The script has done its job and added the size=512m option to the mount.

13.4. Effects of the hardening

Many effects are not directly obvious like changes to security or password policies. You will recognize them, when the system forces you to change your password and when it's getting harder to change a new one because of higher requirements for new passwords enforces by Solaris.

A more obvious change is the new message of the day. It doesn't print out the version. The new version is a little bit ... uhmmm ... more unfriendly:

```
aragorn:~ joergmoellenkamp$ ssh jmoekamp@10.211.55.200
Password:

Last login: Wed Apr 16 19:12:19 2008 from 10.211.55.2

|----------------------------------------------------------------------|

| This system is for the use of authorized users only.
              |
| Individuals using this computer system without authority, or
   in |
| excess of their authority, are subject to having all of their
      |
| activities on this system monitored and recorded by system
       |
| personnel.
                                                                    |
|

   |
| In the course of monitoring individuals improperly using this
      |
| system, or in the course of system maintenance, the
   activities  |
| of authorized users may also be monitored.
                         |
|

   |
| Anyone using this system expressly consents to such
   monitoring  |
```

```
| and is advised that if such monitoring reveals possible
          |
| evidence of criminal activity, system personnel may provide
   the |
| evidence of such monitoring to law enforcement officials.
        |
|-----------------------------------------------------------------|
```

13.5. Undo the hardening

All these file ending with `.JASS.` are not just for you to lookup the changes of SST. This files enables the SST to undo the changes and to fall back to a different configuration.

```
# ./jass-execute -u
Executing driver, undo.driver

Please select a Solaris Security Toolkit run to restore through
   :
1.  April 16, 2008 at 16:15:04 (/var/opt/SUNWjass/run
   /20080416161504)
Choice ('q' to exit)?  1
```

The choice is easy, we have just one old version, so we choose `1`:

```
[NOTE] Restoring to previous run from /var/opt/SUNWjass/run
   /20080416161504

================================================================================

undo.driver: Driver started.
================================================================================
```

You may have changed some files since using the toolkit, thus the SST will ask you if you what it should do with those files. For example, I've changed the password of my account, thus the `/etc/shadow` has changed:

```
================================================================================

undo.driver: Undoing Finish Script: set-flexible-crypt.fin
================================================================================
```

```
[NOTE] Undoing operation COPY.
[WARN] Checksum of current file does not match the saved value.
[WARN]    filename = /etc/shadow
[WARN]    current  = 8e27a3919334de7c1c5f690999c35be8
[WARN]    saved    = 86401b26a3cf38d001fdf6311496a48c

Select your course of action:
1. Backup - Save the current file, BEFORE restoring original.
2. Keep   - Keep the current file, making NO changes.
3. Force  - Ignore manual changes, and OVERWRITE current file.

NOTE: The following additional options are applied to this and
   ALL subsequent files:
4. ALWAYS Backup.
5. ALWAYS Keep.
6. ALWAYS Force.

Enter 1, 2, 3, 4, 5, or 6:
2
```

After this command the Solaris Security Toolkit has reverted all changes.

13.6. Conclusion

With the Solaris Security Toolkit you can deploy an certain baseline of security configurations to all your systems in an automatic manner. But it isn't limited to run once after the installation, you can run it as often as you want to ensure that you get to an known secured state of your system after patching or reconfigurations of your system. By doing this automatically, you get a big advantage. Once you've developed your own driver for your site, nobody forgets to set a certain configuration leaving an attack vector open, you assumed as being closed down.

This tutorial demonstrated only a small subset of the capabilties of the toolkit. For example you can integrate it into Jumpstart to automatically harden systems at their installation, you can use it to install a minimal patch cluster on each system where you execute the toolkit. So you should really dig down into the documentation of this toolkit to explore all the capabilities.

13.7. Do you want to learn more?

Documentation

docs.sun.com: Solaris Security Toolkit 4.2 Administration Guide

14. Auditing

Solaris 10/Opensolaris

One of the less known features in Solaris is the Auditing. Auditing solves an important problem: What happens on my system, and whodunit. When something strange happens on your system or you recognize, that you are not the only one who owns your system, it's a good thing to have some logs for analysis.

The nice thing about the auditing in Solaris: It's quite simple to activate. In this article I will give you a short overview to enable and use the auditing in Solaris. This feature is really old, it's in Solaris for since the last century but nevertheless it's a less known Solaris feature.

14.1. Some terms

There are some special terms in auditing. I want to give you a short definition of them as I have to use them in this article. I've copied this definitions from ¡a href="http://docs.sun.com/app/docs/doc/816-4557/auditplan-6?a=view"¿the manual for Solaris Auditing¡/a¿.

- Audit events: A security-related system action that is audited. For ease of selection, events are grouped into audit classes.
- Audit Class: A grouping of audit events. Audit classes provide a way to select a group of events to be audited.
- Audit policy: A set of auditing options that you can enable or disable at your site. These options include whether to record certain kinds of audit data. The options also include whether to suspend auditable actions when the audit trail is full.

14.2. Configuring basic auditing

You have to search for a place in your filesystem. It's a good practice to use an own filesystem, as auditing will eat away your filesystem space until there is nothing left and this is a bad idea for the root file system. But in this example I will omit this step.

At first login as root. Okay, you need a place to store the audit logs. It's important to change the rights of the directory to assure only root can access it.

```
mkdir /var/audit/aragorn-sol
chmod -R 750 /var/audit/aragorn-sol
```

Then go to `/etc/security` and edit the file `/etc/security/audit_control`. This file controls where what classes of information are logged and where you write the log. For example: The `lo` is the audit class for all events in regard of logins and logoffs:

```
dir:/var/audit/aragorn-sol
flags:lo
minfree:20
naflags:lo
```

Okay, configuration is done.But let's have another look the file `/etc/security/audit_startup`. The commands in this script control the audit policies and thus the behavior of the logging and the amount of informations in the log records:

```
/usr/bin/echo "Starting BSM services."
/usr/sbin/auditconfig -setpolicy +cnt
/usr/sbin/auditconfig -conf
/usr/sbin/auditconfig -aconf
```

The second line is the most interesting. Without this line the system would stop user interaction when the system is unable to log. You would deactivate this behavior, when logging is more important than system availability. For the moment we don't change this file.

14.3. Start the auditing

Now activate auditing. You have to reboot after the activation:

```
# ./bsmconv </b>
This script is used to enable the Basic Security Module (BSM).
Shall we continue with the conversion now? [y/n] y
bsmconv: INFO: checking startup file.
bsmconv: INFO: turning on audit module.
```

```
bsmconv: INFO: initializing device allocation.

The Basic Security Module is ready.
If there were any errors, please fix them now.
Configure BSM by editing files located in /etc/security.
Reboot this system now to come up with BSM enabled.
# reboot
```

Two short checks ... auditd runs ...

```
# svcs | grep "auditd"
online          23:30:03 svc:/system/auditd:default
```

... and the system starts to gather audit logs

```
# ls -la
total 6
drwxr-x---   2 root     root          512 Feb  1 23:30 .
drwxr-xr-x   3 root     sys           512 Feb  1 23:18 ..
-rw-r-----   1 root     root          255 Feb  1 23:33
   20080201223003.not_terminated.aragorn-sol
```

Okay, now you have completed the configuration. The system has started to write audit logs.

14.4. Managing the audit logs

Audit logs grows infinitely. To the maximum filesize in the used filesystem or the end of disk capacity ... whatever occurs first. It's a good practice to checkpoint the audit logs in a regular interval. It's quite simple:

```
audit -n
```

With this command the actual file gets closed and a new one gets opened.

```
# cd /var/audit/aragorn-sol/
# ls -l
total 24
-rw-r-----   1 root     root          684 Feb  1 23:55
   20080201223003.20080201225549.aragorn-sol
-rw-r-----   1 root     root          571 Feb  2 00:06
   20080201225549.20080201230639.aragorn-sol
-rw-r-----   1 root     root         2279 Feb  2 00:10
   20080201230834.20080201231010.aragorn-sol
```

```
-rw-r-----   1 root     root         755 Feb  2 00:12
   20080201231010.20080201231245.aragorn-sol
-rw-r-----   1 root     root        4274 Feb  2 08:36
   20080201231245.20080202073624.aragorn-sol
-rw-r-----   1 root     root         200 Feb  2 08:36
   20080202073624.not_terminated.aragorn-sol
```

14.5. Analyzing the audit trails

It doesn't make sense to create audit logs without looking at them. You can't look directly at them as this file are binary ones. You need to command to analyse the audit log. One to extract the data out of the log files based on certain rules and one command to translate it into an human readable format. You use the `auditreduce` command for the first step, and the `praudit` command for the second one.

```
# cd /var/audit/aragorn-sol
auditreduce * | praudit -s
```

This sequence of commands translate all you audit logs into an human readable form. I've cut out some of the lines for an example:

```
header,69,2,AUE_ssh,,localhost,2008-02-01 23:49:17.687 +01:00
subject,jmoekamp,jmoekamp,other,jmoekamp,other
   ,720,3447782834,6969 5632 10.211.55.2
return,success,0
header,77,2,AUE_su,,localhost,2008-02-01 23:49:55.336 +01:00
subject,jmoekamp,root,other,jmoekamp,other,729,3447782834,6969
   5632 10.211.55.2
text,root
return,failure,Authentication failed
header,69,2,AUE_su,,localhost,2008-02-01 23:50:11.311 +01:00
subject,jmoekamp,root,root,root,root,730,3447782834,6969 5632
   10.211.55.2
return,success,0
```

What tells this snippet to you: I've logged into my system as the user jmoekamp, tried to assume root privileges, failed the first time (due wrong password), tried it again and succeeded.

14.6. More auditing

Sometimes it's important to know what users have done on you system. For example: Which programs has been executed. With Solaris auditing it's really easy to collect this information.

At first you have to configure auditing to collect this kind of information:

```
dir:/var/audit/aragorn-sol
flags:lo,ex
minfree:20
naflags:lo,ex
```

The `ex` audit class matches to all events in system in regard to the execution of a program. This tells the auditing subsystem to log all execve() system calls. But you have to signal this change to the audit subsystem to start the auditing of this events. With `audit -s` you notify the audit daemon to read the `/etc/security/audit_control` file again.

```
header,113,2,AUE_EXECVE,,localhost,2008-02-02 00:10:00.623
   +01:00
path,/usr/bin/ls
attribute,100555,root,bin,26738688,1380,0
subject,jmoekamp,root,root,root,root,652,2040289354,12921 71168
    10.211.55.2
return,success,0
```

But this configuration only logs the path of the command, not the command line parameters. You have to configure to log this information. You remember: The audit policy controls the kind of information in the audit logs. Thus we have to modify the audit policy. With the command `auditconfig -setpolicy +argv` you change the policy. You don't have to activate it, it's immediately effective:

```
header,124,2,AUE_EXECVE,,localhost,2008-02-02 00:12:49.560
   +01:00
path,/usr/bin/ls
attribute,100555,root,bin,26738688,1380,0
exec_args,2,ls,-l
subject,jmoekamp,root,root,root,root,665,2040289354,12921 71168
    10.211.55.2
return,success,0
```

To make this behavior persistent, you have add the ¡code¿auditconfig -setpolicy +argv¡/code¿ to the file

14.7. Want to learn more?

This is only a really short introduction to the topic. You will find the documentation for this feature at docs.sun.com: TBD with Part VII Solaris Auditing¡/a¿ is a good place to start.

15. Basic Audit Reporting Tool

Solaris 10/Opensolaris

Apropos auditing. There is a small but cool tool in Solaris. It solves the problem of "No, I haven't changed anything on the system". It's called BART, the Basic Audit Reporting Tool. It a really simple tool and it's really easy to use.

15.1. Usage

Okay, let's assume after some days of work you finally configured all components of your new system. Okay, create a nice place to store the output of the `bart` tool. After this you start `bart` for the first time to create the first manifest of your system.

```
# mkdir /bart-files
# bart create -R /etc > /bart-files/etc.control.manifest
```

The manifest stores all informations about the files. This is the example for the ¡code¿/etc/nsswitch.nisplus¡/code¿:

```
# cat etc.control.manifest | grep "/nsswitch.nisplus"
/nsswitch.nisplus F 2525 100644 user::rw-,group::r--,mask:r--,
   other:r-- 473976b5 0 3 79e8fd689a5221d1cd059e5077da71b8
```

Now lets change some files:

```
# touch /etc/thisisjustatest
# chmod 777 /etc/nsswitch.files
# echo "#just a test" >> /etc/nsswitch.nisplus
```

Okay, enough changes. Let's create a new manifest of the changed `/etc`. Pipe it to a different file.

```
# bart create -R /etc > /bart-files/etc.check20080202.manifest
```

Now we can compare the baseline manifest with the actual manifest.

```
# cd /bart-files
# bart compare etc.control.manifest etc.check20080202.manifest
```

This command prints all differences between the two manifests and thus the difference between the tow states of the system

```
/nsswitch.files:
  mode  control:100644  test:100777
  acl  control:user::rw-,group::r--,mask:r--,other:r--  test:
     user::rwx,group::rwx,mask:rwx,other:rwx
/nsswitch.nisplus:
  size  control:2525  test:2538
  mtime  control:473976b5  test:47a44862
  contents  control:79e8fd689a5221d1cd059e5077da71b8  test:3
     f79176ec352441db11ec8a3d02ef67c
/thisisjustatest:
  add
```

As I wrote before: A really nice tool.

15.2. Want to learn more?

For more information about this tool visit TBD.

16. IPsec

Solaris 10/Opensolaris

16.1. The secrets of root

There were other systems with other roots, other priests and other believers. But how should they communicate with each other without put their secrets in danger. Some chants and documents were really precious. The messengers of root had to be sure, that they gave the chants to the roots with the same belief. But the messengers had another problems: There were many bandits on their way to the other systems. And it was unthinkable that a mere bandit would know the secrets of root.

16.2. Foundations

This will the only article in this series without an explanation of the technologies. I could write for days about it and still leaving out important things. Instead of this, please look at the links section at the end of the article.

Only some short words about this topic. Encryption is essential in networking. The Internet is a inherently insecure media. You have to assume, that you don't talk to the right person as long as you didn't authenticated him, you have to assume, that the data will be read by someone as long you won't encrypt the data.

IPsec solves this problems. I won't tell you that IPsec is an easy protocol. The stuff around IPsec is defined in several RFC and the documentation is rather long. The encryption itself isn't the complex part. But you need a key to encrypt your data. And there starts the complexity. It's absolutely essential, that this key stays secret. How do you distribute keys through an inherently insecure transport channel. And the next problem: How do you negotiate the encryption mechanism? And how to ensure, that you talk with the right system ? Problems ... problems ... problems!

IPsec solves this problems. IPsec isn't a single protocol. It's a suite of protocols consisting out Internet Security Association and Key Management Protocol, this protocol build on the protocol for Internet Key Exchange, this protocol is based on the Oakley protocol

And there is a whole wad of further protocols and mechanisms and algorithms to ensure secure communication.

16.3. IPsec in Solaris

Okay, but now I want to give you an example to configure IPsec on Solaris. Although the matter is complex, it isn't hard to use IPsec within Solaris. IPsec is a rather old feature in Solaris. We've introduced it in Solaris 8 and improved the implementation since this time. So this implementation is now in year eight of its availability.

16.4. Example

The task for this example is to secure all traffic between two hosts. I've used two VM with Solaris Express Build 78 for this configuration. (You need at least an Solaris 10 Update 4 for this tutorial. In this update the ipsecconf command was introduced making the configuration much easier) `theoden` has the IP number `10.211.55.200`. Gandalf has the IP number `10.211.55.201`. I don't want to use manual keying, instead of this the example will use self signed certificates.

16.5. Prepare the installation

At first: Obviously you have to change hostnames and ip numbers and names for certificate corresponding to your own site.

We have to work on two hosts. It's a best practice to open up two terminal windows. Get root in both of them. To keep things apart use the bash shell and modify the shell prompt:

```
# bash
# PS1="[\u@\h:\w]$ "
[root@gandalf:~]$
```

Look at the login prompt in the examples. They designate on which system you have to work on.

Okay ... at first you have to ensure, that the names of the systems can be resolved. It's a good practice to put the names of the systems into the `/etc/hosts`:

```
::1     localhost       loghost
127.0.0.1       localhost       loghost
10.211.55.201 gandalf
10.211.55.200 theoden
```

Okay, we don't want manual keying or some stinking preshared keys. Thus we need to create keys. Login to gandalf and assume the root role:

```
[root@gandalf:~]$ ikecert certlocal -ks -m 1024 -t rsa-md5 -D "
   C=de, O=moellenkamp, OU=moellenkamp-vpn, CN=gandalf" -A IP
   =10.211.55.201
Creating private key.
Certificate added to database.
-----BEGIN X509 CERTIFICATE-----
MIICOzCCAaSgAwIBAgIFAJRpUUkwDQYJKoZIhvcNAQEEBQAwTzELMAkGA1UEBhMC

[ ... some lines omitted ... ]
oi4dO39J7cSnooqnekHjajn7ND7T187k+f+BVcFVbSenIzblq2P0u7FIgIjdlv0
   =
-----END X509 CERTIFICATE-----
```

Do the same on the other host.

```
[root@theoden:~]$ ikecert certlocal -ks -m 1024 -t rsa-md5 -D "
   C=de, O=moellenkamp, OU=moellenkamp-vpn, CN=theoden" -A IP
   =10.211.55.200
Creating private key.
Certificate added to database.
-----BEGIN X509 CERTIFICATE-----
MIICOzCCAaSgAwIBAgIFAIRuR5QwDQYJKoZIhvcNAQEEBQAwTzELMAkGA1UEBhMC

[ ... some lines omitted ... ]
UHJ4P6Z0dtjnToQb37HNq9YWFRguSsPQvc/Lm+S9cJCLwINVg7NOXXgnSfY3k+Q
   =
-----END X509 CERTIFICATE-----
```

You need the output of this commands later, so past them to a text editor or at a save place ...

16.6. Configuration of IPsec

Okay, now we have to tell both hosts to use IPsec when they talk to each other:

```
[root@gandalf:~]$ echo "\{laddr gandalf raddr theoden} ipsec {
   auth_algs any encr_algs any sa shared\}" >> /etc/inet/
   ipsecinit.conf
```

This translates to: When im speaking to theoden, I have to encrypt the data and can use any negotiated and available encryption algorithm and any negotiated and available authentication algorithm.

Such an rule is only valid on one direction. Thus we have to define the opposite direction on the other host to enable bidirectional traffic:

```
[root@theoden:~]$ echo "{laddr theoden raddr gandalf} ipsec {
   auth_algs any encr_algs any sa shared}" >> /etc/inet/
   ipsecinit.conf
```

Okay, the next configuration is file is a little bit more complex. Go into the directory `/etc/inet/ike` and create a file `config` with the following content:

```
cert_trust "10.211.55.200"
cert_trust "10.211.55.201"

p1_xform
  { auth_method preshared oakley_group 5 auth_alg sha encr_alg
     des }
p2_pfs 5

{
 label "DE-theoden to DE-gandalf"
 local_id_type dn
 local_id "C=de, O=moellenkamp, OU=moellenkamp-vpn, CN=theoden"
 remote_id "C=de, O=moellenkamp, OU=moellenkamp-vpn, CN=gandalf
    "

 local_addr  10.211.55.200
 remote_addr 10.211.55.201

 p1_xform
  {auth_method rsa_sig oakley_group 2 auth_alg md5 encr_alg 3
     des\}
}
```

This looks complex, but once you've understand this it's quite easy:

```
cert_trust "10.211.55.200"
cert_trust "10.211.55.201"
```

We use self-signed certificate. The certificate isn't signed by an independent certification authority. Thus there is no automatic method to trust the certificate. You have to configure the iked explicitly to trust this certificates. This both lines tell the iked to trust the certificates with the alternate name 10.221.55.200 and 10.211.55.201. Where did this alternate names came from? You set them! Look in the command line for creating the certificate. You defined this name by using the `-a` switch.

```
label "DE-gandalf to DE-theoden"
local_id_type dn
local_id "C=de, O=moellenkamp, OU=moellenkamp-vpn, CN=gandalf"
remote_id "C=de, O=moellenkamp, OU=moellenkamp-vpn, CN=theoden"
```

Now you define an key exchange. You have to give each connection an unique name. After this you define, what part of the certificate is used to authenticate the remote system. In this example we use the distinguished name. The local system identifies itself with the certificate named `C=de, O=moellenkamp, OU=moellenkamp-vpn, CN=gandalf`to a remote system, and expect a trusted certificate with the distinguished name `C=de, O=moellenkamp, OU=moelle`

```
local_addr  10.211.55.200
remote_addr 10.211.55.201
\end|{lstlisting}
Now the iked knows the ip addresses of the local and the remote
    host.
\begin{lstlisting}
p1_xform
  {auth_method rsa_sig oakley_group 2 auth_alg md5 encr_alg 3
     des}
```

After defining the authentication credentials, we have to define how the system should communicate. This line means: Use the certificates to authenticate the other system. The key determination protocol is based on a prime number. We use md5 as the ground-laying algorithm to authenticate and 3des for encryption. This is the part where you configure the methods for authentication and encryption. They have to be the same on both hosts, otherwise they won't be able to negotiate to a common denominator thus you won't be able to communicate between the both hosts at all.

Now we do the same on the other system.

```
[root@gandalf:/etc/inet/ike]$ cd /etc/inet/ike
[root@gandalf:/etc/inet/ike]$ cat conf
cert_trust "10.211.55.200"
cert_trust "10.211.55.201"

p1_xform
  { auth_method preshared oakley_group 5 auth_alg sha encr_alg
     des }
```

```
p2_pfs 5

{
 label "DE-gandalf to DE-theoden"
 local_id_type dn
 local_id "C=de, O=moellenkamp, OU=moellenkamp-vpn, CN=gandalf"
 remote_id "C=de, O=moellenkamp, OU=moellenkamp-vpn, CN=theoden
    "

 local_addr  10.211.55.201
 remote_addr 10.211.55.200

 p1_xform
  {auth_method rsa_sig oakley_group 2 auth_alg md5 encr_alg 3
     des}
}
```

Obviously you have to swap the numbers for the local and remote system and you have to assign a unique label to it.

Okay, we are almost done. But there is still a missing but very essential thing when you want to use certificates. We have to distribute the certificates of the systems.

```
[root@gandalf:/etc/inet/ike]$ ikecert certdb -l
Certificate Slot Name: 0   Key Type: rsa
        (Private key in certlocal slot 0)
        Subject Name: <C=de, O=moellenkamp, OU=moellenkamp-vpn,
            CN=gandalf>
        Key Size: 1024
        Public key hash: 28B08FB404268D144BE70DDD652CB874
```

At the beginning there is only the local key in the system. We have to import the key of the remote system. Do you remember the output beginning with `-----BEGIN X509 CERTIFICATE-----` and ending with `-----END X509 CERTIFICATE-----`? You need this output now.

The next command won't come back after you hit return. You have to paste in the key. On gandalf you paste the output of the key generation on theoden. On Theoden you paste the output of the key generation on gandalf. Let's import the key on gandalf

```
[root@gandalf:/etc/inet/ike]$ ikecert certdb -a
-----BEGIN X509 CERTIFICATE-----
MIICOzCCAaSgAwIBAgIFAIRuR5QwDQYJKoZIhvcNAQEEBQAwTzELMAkGA1UEBhMC


UHJ4P6Z0dtjnToQb37HNq9YWFRguSsPQvc/Lm+S9cJCLwINVg7NOXXgnSfY3k+Q
   =
```

```
-----END X509 CERTIFICATE-----
[root@gandalf:/etc/inet/ike]$
```

After pasting, you have to hit Enter once and after this you press Ctrl-D once. Now we check for the sucessful import. You will see two certificates now.

```
[root@gandalf:/etc/inet/ike]$ ikecert certdb -l
Certificate Slot Name: 0   Key Type: rsa
        (Private key in certlocal slot 0)
        Subject Name: <C=de, O=moellenkamp, OU=moellenkamp-vpn,
            CN=gandalf>
        Key Size: 1024
        Public key hash: 28B08FB404268D144BE70DDD652CB874

Certificate Slot Name: 1   Key Type: rsa
        Subject Name: <C=de, O=moellenkamp, OU=moellenkamp-vpn,
            CN=theoden>
        Key Size: 1024
        Public key hash: 76BE0809A6CBA5E06219BC4230CBB8B8
\end{listing}
Okay, switch to theoden and import the key from gandalf on this
    system.
\begin{lstlisting}[root@theoden:/etc/inet/ike]$ ikecert certdb
   -l
Certificate Slot Name: 0   Key Type: rsa
        (Private key in certlocal slot 0)
        Subject Name: <C=de, O=moellenkamp, OU=moellenkamp-vpn,
            CN=theoden>
        Key Size: 1024
        Public key hash: 76BE0809A6CBA5E06219BC4230CBB8B8

[root@theoden:/etc/inet/ike]$ ikecert certdb -a
-----BEGIN X509 CERTIFICATE-----
MIICOzCCAaSgAwIBAgIFAJRpUUkwDQYJKoZIhvcNAQEEBQAwTzELMAkGA1UEBhMC


oi4dO39J7cSnooqnekHjajn7ND7T187k+f+BVcFVbSenIzblq2P0u7FIgIjdlv0
   =
-----END X509 CERTIFICATE-----

[root@theoden:/etc/inet/ike]$ ikecert certdb -l
Certificate Slot Name: 0   Key Type: rsa
        (Private key in certlocal slot 0)
        Subject Name: <C=de, O=moellenkamp, OU=moellenkamp-vpn,
            CN=theoden>
```

```
        Key Size: 1024
        Public key hash: 76BE0809A6CBA5E06219BC4230CBB8B8

Certificate Slot Name: 1   Key Type: rsa
        Subject Name: <C=de, O=moellenkamp, OU=moellenkamp-vpn,
            CN=gandalf>
        Key Size: 1024
        Public key hash: 28B08FB404268D144BE70DDD652CB874
```

16.7. Activate the configuration

Okay, now we have to activate this configuration on both systems:

```
[root@gandalf:/etc/inet/ike]$ svcadm enable ike
[root@gandalf:/etc/inet/ike]$ ipsecconf -a /etc/inet/ipsecinit.
   conf
```

and

```
[root@theoden:/etc/inet/ike]$ svcadm enable ike
[root@theoden:/etc/inet/ike]$ ipsecconf -a /etc/inet/ipsecinit.
   conf
```

16.8. Check the configuration

Login into theoden as root and start a packet sniffer:

```
[root@theoden:~]$ snoop host gandalf
Using device ni0 (promiscuous mode)
```

Okay ... now login to gandalf and start some pings to theoden:

```
[root@gandalf:~]$ ping theoden;ping theoden;ping theoden
theoden is alive
theoden is alive
theoden is alive
```

Okay, theoden can speak with gandalf and vice versa. But is it encrypted? In the terminal window for theoden the following output should be printed:

```
gandalf -> theoden      ESP SPI=0x1d2c0e88 Replay=4
theoden -> gandalf      ESP SPI=0x84599293 Replay=4
gandalf -> theoden      ESP SPI=0x1d2c0e88 Replay=5
theoden -> gandalf      ESP SPI=0x84599293 Replay=5
gandalf -> theoden      ESP SPI=0x1d2c0e88 Replay=6
theoden -> gandalf      ESP SPI=0x84599293 Replay=6
```

ESP stands for Encapsulated Security Payload. Mission accomplished.

16.9. Do you want to learn more?

Documentation

System Administration Guide: IP Services ¿¿ IP Security

IPsec at Wikipedia

IPsec

Internet Key Exchange

Oakley Protocol

Internet Security Association and Key Management Protocol

Other

An Illustrated Guide to IPsec

17. Signed binaries

Solaris 10/Opensolaris

One of problems in computer security is the validation of binaries: Is this the original binary or is it a counterfeit binary? Since Solaris 10 Sun electronically signs the binaries of the Solaris Operating Environment. You can check the signature of the binaries with the `elf-sign`-tool.

```
[root@gandalf:/etc]$ elfsign verify -v /usr/sbin/ifconfig
elfsign: verification of /usr/sbin/ifconfig passed.
format: rsa_md5_sha1.
signer: CN=SunOS 5.10, OU=Solaris Signed Execution, O=Sun
   Microsystems Inc.
```

Obviously you have to trust the `elfsign`. But you can check it, when you boot the system from a trusted media (like a original media kit or a checksum validated iso-image. This enables you to check the signature of the `elfsign` independently from the system.

By the way: This certificate and the signature is very important for crypto modules. The crypto framework of solaris just loads modules signed by Sun to prevent the usage of malicious modules (for example to read out the key store and send it somewhere) into the framework.

18. On passwords

Solaris 10/Opensolaris

The password is the key to the system. When you know the username and the password, you can use the system. If not ... well ... go away. You can't overemphasize the value of good passwords. There is something you can do as the admin at the system level to ensure such passwords. At first you can use stronger mechanisms to hash the passwords. And you can force the users of your system to use good password. This tutorial will explain both tasks.

18.1. Using stronger password hashing

Many people are unaware of the fact, that only the first eight characters of a password are used in the default configuration. Don't believe it? Let's try it.

Okay, I've logged into my test machine and change my password:

```
bash-3.2$ passwd jmoekamp
Enter existing login password: oldpassword
New Password: aa3456789
Re-enter new Password: aa3456789
passwd: password successfully changed for jmoekamp
bash-3.2$
```

Now let's try a password that's different at the ninth character by logging into the Solaris system from remote:

```
mymac:~ joergmoellenkamp$ ssh jmoekamp@10.211.55.200
Password: aa3456780
Last login: Wed May 28 11:24:05 2008 from 10.211.55.2
Sun Microsystems Inc.   SunOS 5.11      snv_84  January 2008
```

I've told you ... only the first eight characters are relevant. But it's not that way, that Solaris can't do better than that. It's just the binary compatibility guarantee again. You can't simply change the mechanism encrypting the password. There may be scripts that still need the old unix crypt variant. But in case you are sure, that you haven't such an application you can change it, and it's really simple to do:

When you look into the file `/etc/security/crypt.conf` you will find the additional modules for password encryption.

```
# The algorithm name __unix__ is reserved.

1       crypt_bsdmd5.so.1
2a      crypt_bsdbf.so.1
md5     crypt_sunmd5.so.1
```

The hashing mechanisms are loaded as libraries in the so-called Solaris Pluggable Crypt Framework. It's even possible to develop your own crypting mechanism in the case you don't trust the implementations delivered by Sun.

Table 18.1.: **Cryptographic Mechanisms for password encryption**

Short	Algorithm	Description
1	BSD alike,md5 based	The `crypt_bsdmd5` module is a one-way password hashing module for use with `crypt(3C)` that uses the MD5 message hash algorithm. The output is compatible with `md5crypt` on BSD and Linux systems.
2a	BSD alike, blowfish based	The `crypt_bsdbf` module is a one-way password hashing module for use with `crypt(3C)` that uses the Blowfish cryptographic algorithm.
md5	Sun, md5 based	The `crypt_sunmd5` module is a one-way password hashing module for use with `crypt(3C)` that uses the MD5 message hash algorithm. This module is designed to make it difficult to crack passwords that use brute force attacks based on high speed MD5 implementations that use code inlining, unrolled loops, and table lookup.

Each of the three mechanisms support passwords with up to 255 characters. It's important to know, that the different hashing algorithm can coexist in your password databases. The password hashing for a password will be changed when user change his or her password.

18.1.1. Changing the default hash mechanism

Let's use the `md5`[1] algorithm in our example. But before that, we should look into the actual `/etc/shadow`

```
# grep "jmoekamp" /etc/shadow
jmoekamp:nM2/fPrCTe3F6:14027::::::
```

It's simple to enable a different encryption algorithm for password. You have just to change one lines in `/etc/security/policy.conf`. To edit this file you have to login as root:

```
CRYPT_DEFAULT=md5
```

Okay, now let's change the password.

```
# passwd jmoekamp
New Password: aa1234567890
Re-enter new Password:  aa1234567890
passwd: password successfully changed for jmoekamp
```

When you look in the `/etc/shadow` for the user, you will see a slighly modified password field. It's much longer and between the first and the second `$` you see the used encryption mechanism:

```
# grep "jmoekamp" /etc/shadow
jmoekamp:$md5$vyy8.OVF$$FY4TWzuauRl4.VQNobqMY.:14027::::::
```

Not let's try the login:

```
mymac:~ joergmoellenkamp$ ssh jmoekamp@10.211.55.200
Password: aa1234567890
Last login: Wed May 28 11:38:24 2008 from 10.211.55.2
Sun Microsystems Inc.   SunOS 5.11      snv_84  January 2008
$ exit
Connection to 10.211.55.200 closed.
mymac:~ joergmoellenkamp$ ssh jmoekamp@10.211.55.200
Password: aa1234567891
Password: aa1234567892
Password: aa1234567893
Permission denied (gssapi-keyex,gssapi-with-mic,publickey,
   keyboard-interactive).
mymac:~ joergmoellenkamp$
```

[1] Alec Muffet wrote about the development of this hash mechanism in `http://www.crypticide.com/dropsafe/article/1389`

You see, the correctness of the complete password is tested, not just the first 8 characters.

18.2. Password policies

User have the habit to break any security policy. At least as long you don't enforce it. One of the most annoying habit from the view of the security people is the tendency to choose weak passwords, the name of the boy or girl friend, the prefered brand of cars, birthdays ... you name it. This passwords are everything but secure. But you can configure Solaris to check the new passwords.

18.2.1. Specifing a password policy

There is a central file in Solaris controling the password policy. In `/etc/default/passwd` you define what requirements a password must fulfill before Solaris allows the user to set this password. Let's have a look in the actual file of a standard solaris system. You have to log into your system as root [2]:

```
# cat passwd
[... omitted CDDL header ...]
#
MAXWEEKS=
MINWEEKS=
PASSLENGTH=6
#NAMECHECK=NO
#HISTORY=0
#MINDIFF=3
#MINALPHA=2
#MINNONALPHA=1
#MINUPPER=0
#MINLOWER=0
#MAXREPEATS=0
#MINSPECIAL=0
#MINDIGIT=0
#WHITESPACE=YES
#DICTIONLIST=
#DICTIONDBDIR=/var/passwd
```

[2] One important note for trying out this feature. You need to log into your system as a normal user in a different window.root can set any password without a check by the password policy thus it would look like that your configuration changes had no effect

You enable the checks by uncommenting it and set a reasonable value to the line. When you enable all the checks, it's actually harder to find a valid password than a non-valid one. Whenever thinking about a really hard password policy you should take into consideration, that people tend to make notes about their password when they can't remember it. And a strong password under the keyboard is obviously less secure than a weak password in the head of the user.

Table 18.2.: **/etc/default/password: standard checks**

Parameter	Description
MAXWEEKS	This variable specifies the maximum age for a password.
MINWEEKS	This variable specifies the minimum age for a password. The rationale for this settings gets clearer when I talk about the `HISTORY` setting.
PASSLENGTH	The minimum length for a password
HISTORY	This variable specifies the length of a history buffer. You can specify a length of up to 26 passwords in the buffer. The `MINWEEKS` buffer is useful in conjunction with this parameter. There is a trick to circumvent this buffer and to get you old password back. Just change it as often as the length of the buffer plus one time. The `MINWEEK` parameter prevents this.
WHITESPACE	This variable defines if you are allowed to use a whitespace in your password
NAMECHECK	When you set this variable to `YES`, the system checks if the password and login name are identical. So using the password `root` for the use `root` would be denied by this setting. The default, by the way is, `yes`.

Besides of this basic checks you can use `/etc/default/passwd/` enforce checks for the complexity of passwords. So you can prevent the user from setting to simple passwords.

Table 18.3.: **/etc/default/password: complexity checks**

Parameter	Description
MINDIFF	Let's assume you've used 3 here. If your old password was `batou001`, a new password would be denied, if you try to use `batou002` as only on character was changed. `batou432` would be a valid password.
MINUPPER	With this variable you can force the usage of upper case characters. Let's assume you've specified 3 here, a password like `wasabi` isn't an allowed choice, but you could use `WaSaBi`
MINLOWER	With this variable you enable the check for the amount of lower case characters in your password. In the case you've specified 2 here, a password like `WASABI` isn't allowed, but you can use `WaSaBI`
MAXREPEATS	Okay, some users try to use passwords like `aaaaaa2`. Obviously this isn't really a strong password. When you set this password to 2 you, it checks if at most 2 consecutive characters are identical. A password like `waasabi` would be allowed, but not a password like `waaasabi`
MINSPECIAL	The class `SPECIAL` consists out of characters like `!\=`. Let's assume you've specified 2, a password like `!ns!st` would be fine, but the password `insist` is not a valid choice.
MINDIGIT	With this password you can specify the amount of the numbers in your password. Let's a assume you specify 2, a password like snafu01 would will be allowed. A password like snafu1 will be denied.
MINALPHA	You can check with this variable for a minimum amount of alpha chars (a-z and A-Z) . When you set a value of 2 on this variable, a password like `aa23213` would be allowed, a password like `0923323` would be denied
MINNONALPHA	This checks for the amount of non-alpha characters (0-9 and special chars). A value of 2 would lead to the denial of `wasabi`, but a password like `w2sab!`

18.2.2. Using wordlists

There is another way to force stronger passwords. You can deny every password that is located in a list of words. The program for changing password is capable to compare the new password against a list of words. With this function you can deny the most obvious choices of passwords. But you should initialize the dictionary with a list of words before you can use this feature.

```
# mkpwdict -s /usr/share/lib/dict/words
mkpwdict: using default database location: /var/passwd.
```

The file `/usr/share/lib/dicts/words` is a file in the Solaris Operating System containing a list of words. It's normally used by spell checking tools. Obviously you should use a wordlist in your own language, as user tend do choose words from their own language as passwords. So an English wordlist in Germany may be not that effective.[3] Now you have to tell Solaris to use this lists.

Table 18.4.: **/etc/default/password: Dictionaries**

Parameter	Description
DICTIONLIST	This variable can contain a list of dictionary files separated by a comma. You must specify full pathnames. The words from these files are merged into a database that is used to determine whether a password is based on a dictionary word
DICTIONDBDIR	The directory where the generated dictionary databases reside

When none of the both variables is specified in the `/etc/default/passwd` then no dictionary check is performed.

Let's try it. I've uncommented the `DICTIONDBDIR` line of the `/etc/default/passwd` file and used the standard value `/var/passwd`. One of the word in the dictionary I imported is the word `airplane`

```
$ passwd
passwd: Changing password for jmoekamp
Enter existing login password: chohw!2
```

[3] You find a list of wordlists at http://www.cotse.com/tools/wordlists.htm

```
New Password: airplane
passwd: password is based on a dictionary word.
```

18.3. Conclusion

These are some simple tricks to make your system more secure, just by ensuring that the keys to your server are well-choosen and not simple ones. But as I stated before there is something you should keep in mind. Don't make the passwords too hard to remember.

18.3.1. Do you want to learn more=

Documentation

docs.sun.com: man passwd(1)[4]

docs.sun.com: Changing the Default Algorithm for Password Encryption[5]

Misc. Links

Learning Solaris 10: Solaris Crypt : better password hashing algorithms[6]

[4] http://docs.sun.com/app/docs/doc/816-5165/passwd-1
[5] http://docs.sun.com/app/docs/doc/816-4557/secsystask-42
[6] http://learningsolaris.com/archives/2006/01/19/password-hashing-algorithm/

19. pfexec

Solaris 10/Opensolaris

The Role-Based Access Control (RBAC) scheme in the OpenSolaris OS, Sun's open-source project for the Solaris OS, offers rights profiles. A rights profile is defined as *"a collection of administrative capabilities that can be assigned to a role or to a user"* in the RBAC and Privileges tutorial. Rights profiles can contain authorizations, commands with security attributes, and other rights profiles - a convenient way for grouping security attributes.

With RBAC, you as the system administrator first create profiles and then assign them to roles. Those two tasks are beyond the scope of this article. Finally, you grant users the ability to assume the roles.

You can also assign a profile directly to a user - the method described later in this article. Afterwards, that user can perform the tasks that are defined by the rights profile - even execute root commands without having to log in as superuser. All that person needs to do is prepend the utility `pfexec` to the commands. In effect, pfexec functions as a passwordless su or sudo in Linux.

This article shows you how to delegate administration tasks and assign root capabilities to regular users by way of rights profiles. It is assumed that you are familiar with RBAC concepts and commands in the OpenSolaris OS and have read the RBAC and Privileges tutorial referenced above.

Note: `pfexec` has been available on the Solaris OS since Solaris 8 and Trusted Solaris 8 and is not unique to the OpenSolaris OS.

19.1. Delegating Administration Tasks

Let's assume that user `testuser` would like to regularly share and unshare directories with other systems through a Network File System (NFS). Normal user privileges do not allow him to do so, as shown below:

```
$ /usr/sbin/share /export/home/testuser
Could not share: /export/home/testuser: no permission
```

However, you can add a profile with the share right for `testuser`. Do the following:

1. As root, create a new user and assign a password to that person. In the OpenSolaris OS, the first user you create is already assigned to a profile that allows that person to perform all the root tasks. Following are the related commands and output:

```
# mkdir -p /export/home
# useradd -m -d /export/home/testuser testuser
80 blocks
# passwd testuser
New Password:
Re-enter new Password:
passwd: password successfully changed for testuser
```

2. Log out and log in again as the new user.

3. Look for a matching profile in the `exec\_attr` file with the share command. Here are the command and subsequent output, which shows a match in File System Management:

```
$ grep "share" /etc/security/exec_attr
File System Management:suser:cmd:::/usr/sbin/dfshares:euid
   =0
File System Management:suser:cmd:::/usr/sbin/share:uid=0;
   gid=root
File System Management:suser:cmd:::/usr/sbin/shareall:uid
   =0;gid=root
File System Management:suser:cmd:::/usr/sbin/sharemgr:uid
   =0;gid=root
File System Management:suser:cmd:::/usr/sbin/unshare:uid=0;
   gid=root
File System Management:suser:cmd:::/usr/sbin/unshareall:uid
   =0;gid=root
[...]
```

4. Become root and assign the File System Management rights profile to testuser. Here are the commands and subsequent output:

```
$ su root
Password:
# usermod -P'File System Management' testuser
UX: usermod: testuser is currently logged in, some changes
   may not take
effect until next login.
```

In this case, despite the warning that some changes are only effective after user logout, the user needs not do so. This change is effective immediately.Voila! testuser can now share

and unshare directories by prepending pfexec to the share command without becoming superuser:

```
$ pfexec /usr/sbin/share /export/home/testuser
$ /usr/sbin/share
      - /export/home/testuser rw ""
```

19.2. Granting Root Capabilities to Regular Users

Warning: Even though you can grant regular users root capabilities with pfexec, do so with care and discretion. Ensure that those to whom you grant root capabilities are trusted, knowledgeable administrators and that they protect their own passwords as they would their root passwords.

The Primary Administrator profile in the OpenSolaris OS grants the capabilities traditionally associated with root. That is, users who are assigned that profile can execute root commands without logging in as root. See the profile's entry in the exec_attr file:

```
# cat /etc/security/exec_attr | grep "Primary"
Primary Administrator:suser:cmd:::*:uid=0;gid=0
```

That means all the commands that are executed with this profile with `pfexec` prepended are run with uid=0 and gid=0, hence with root privileges.Granting users the root capabilities through the Primary Administrator rights profile has several advantages:

- You need not reveal the root password to the users.
- To withdraw a user's root privilege, simply delete the Primary Administrator profile from the user setup-no need to set a new root password.
- Users with the Primary Administrator rights profile can set up a root shell and need not prepend root commands with pfexec afterwards.

See this example:

1. As root, assign the Primary Administrator profile to testuser. Here are the commands and subsequent output:

   ```
   $ usermod -P'Primary Administrator' testuser
   UX: usermod: testuser is currently logged in, some changes
      may not take
   effect until next login.
   ```

2. As a test, log in as testuser and execute the `id -a` command twice: once without `pfexec` and once with `pfexec`. Note the output:

```
$ id -a
uid=100(testuser) gid=1(other) groups=1(other)
$ pfexec id -a
uid=0(root) gid=0(root) groups=1(other)
```

Without `pfexec`, `testuser`'s uid and gid values are those of `testuser` and `other`, respectively-that is, nonroot. With `pfexec`, `uid` and `gid` assume the value of `root`.To avoid having to type `pfexec` over and over again, `testuser` can set up a `root` Bash shell on his system, like this:

```
$ pfexec bash
bash-3.2# id
uid=0(root) gid=0(root)
```

To remove the profile, log in as root and type:

```
# usermod -P '' testuser
#
```

Afterwards, `uid` and `gid` revert to the normal values of the user, as reflected in the output of the `id -a` command when executed under the control of `pfexec`. Without an assigned profile, pfexec yields no additional privileges to the user:

```
$ pfexec id -a
uid=100(testuser) gid=1(other) groups=1(other)
```

19.3. An important advice

As you may have recognized, there is no further password protection between a user with the `Primary Administrator` and running programs with `root` capabilities. You shouldn't use such an account for your daily work, when you are using scripts or programms downloaded from the Internet, as such a script has just to prepend a `pfexec` to a command to yield administrative capabilities. On OpenSolaris 200x.y it's a good practice to create a second account for your daily work without the `Primary Administrator` profile.

19.4. Conclusion

Again, passwordless pfexec is the OpenSolaris version of Linux's sudo. Assigning and revoking the root capabilities through the Primary Administrator profile is simple and

straightforward. In addition, you can monitor user actions by logging the executions of pfexec with the auditing subsystem of the Solaris OS.

By default, the user account that you create while installing OpenSolaris 2008.11, the latest version of the OS, is automatically assigned the Primary Administrator rights profile even though the user cannot directly log in as root. Subsequently, that user can run pfexec with the root privilege - a big convenience to all!

Part IV.

Networking

20. Crossbow

Opensolaris

20.1. Introduction

At the moment ZFS, DTrace or Zones are the well known features of Solaris. But in my opinion there will be a fourth important feature soon. Since Build 105 it's integrated (many people will already know which feature i want to describe in this artcle) into Opensolaris. This feature has the project name Crossbow.

It's the new TCP/IP stack of Opensolaris and was developed with virtualisation in mind from ground up. "Virtualisation in mind" does not only lead to the concept of virtual network interface cards, you can configure virtual switches as well and even more important: You can control the resource usage of a virtual client or managing the load by distributing certain TCP/IP traffic to dedicated CPUs. I've already held some talks about Crossbow at different events, thus it's time to write an article about this topic. I will start with the virtualisation part of Crossbow.

20.2. Virtualisation

This part is heavily inspired by this blog entry of Ben Rockwood[1], but he ommited some parts in the course of his article to make a full walk-through out of it, so i extended it a little bit.

Normally a network consists out of switches and networking cards, server and router . It's easy to replicate this in a single system. Networking cards can be simulated by VNICS, switches are called etherstubs in the namespace of Crossbow. Server can be simulated by zones of course, and as router are not much more than special-purpose servers, we could simulate them by a zone as well.

[1] `http://www.cuddletech.com/blog/pivot/entry.php?id=1001`

20.2.1. A simple network

Let's simulate a simple network at first. Just two servers and a router:

Configuration of the network

At first we create two virtual switches. They are called `etherstub0` and `etherstub1`

```
# dladm create-etherstub etherstub0
# dladm create-etherstub etherstub1
```

Okay, now we create virtual nics that are bound to the virtual switch `etherstub0`. These virtual nics are called `vnic1` and `vnic0`.

```
# dladm create-vnic -l etherstub0 vnic1
# dladm create-vnic -l etherstub0 vnic2
```

Now we do the same with our second virtual switch:

```
# dladm create-vnic -l etherstub1 vnic3
# dladm create-vnic -l etherstub1 vnic4
```

Okay, let's look up the configuration of our network.

```
# dladm showlink
LINK         CLASS     MTU    STATE    OVER
ni0          phys      1500   unknown  --
etherstub0   etherstub 9000   unknown  --
etherstub1   etherstub 9000   unknown  --
vnic1        vnic      9000   up       etherstub0
vnic2        vnic      9000   up       etherstub0
vnic3        vnic      9000   up       etherstub1
vnic4        vnic      9000   up       etherstub1
vnic5        vnic      1500   up       ni0
```

Yes, that's all ... but what can we do with it? For example simulating a complete network in your system. Let's create a network with two networks, a router with a firewall and nat and a server in each of the network. Obviously we will use zones for this.

A template zone

At first we create a template zone. This zone is just used for speeding up the creation of other zones. To enable zone creation based on ZFS snapshots, we have to create a filesystem for our zones and mount it at a nice position in your filesystem:

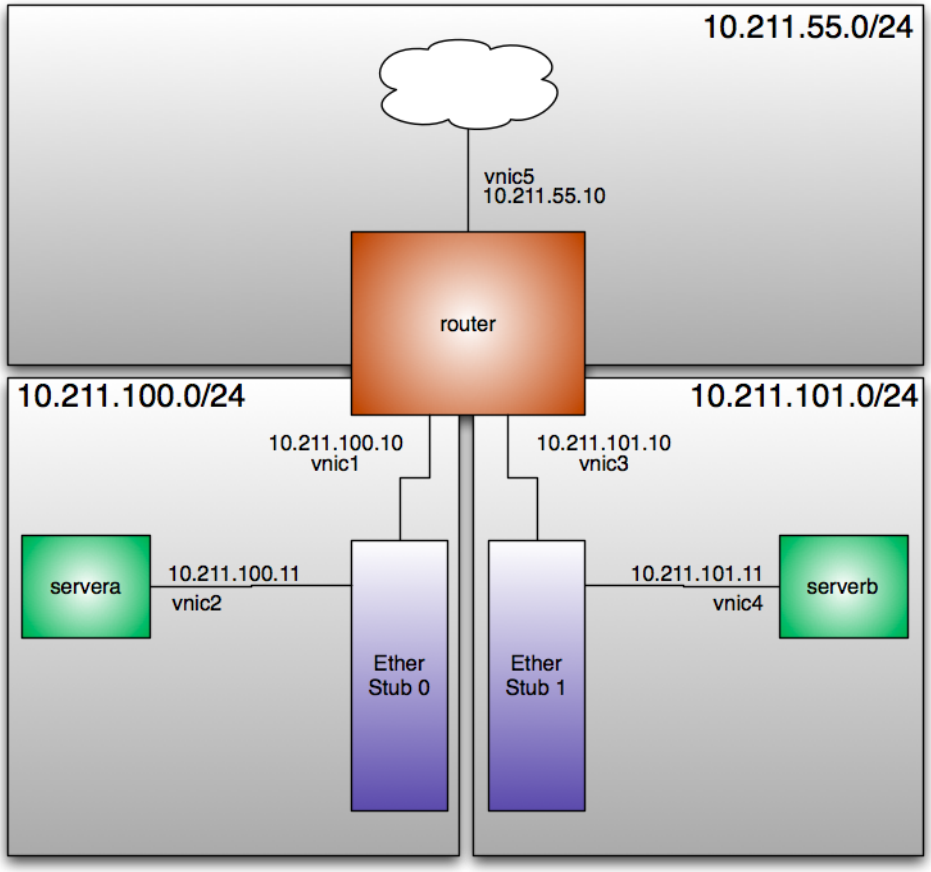

Figure 20.1.: Simple virtual network with Crossbow

```
# zfs create rpool/zones
# zfs set compression=on rpool/zones
# zfs set mountpoint=/zones rpool/zones
```

Now we prepare a command file for the zone creation. The pretty much the standard for a sparse root zone. We don't configure any network interfaces, as we never boot or use this zone. It's just a template as the name alreay states. So at first we create a file called `template` in a working directory. All the following steps assume that you are in this directory as i won't use absolute paths.

```
create -b
set zonepath=/zones/template
set ip-type=exclusive
set autoboot=false
add inherit-pkg-dir
set dir=/lib
end
add inherit-pkg-dir
set dir=/platform
end
add inherit-pkg-dir
set dir=/sbin
end
add inherit-pkg-dir
set dir=/usr
end
add inherit-pkg-dir
set dir=/opt
end
commit
```

Now we create the zone. Depending on your test equipment this will take some times.

```
# zonecfg -z template -f template
# zoneadm -z template install
A ZFS file system has been created for this zone.
Preparing to install zone <template>.
Creating list of files to copy from the global zone.
Copying <3488> files to the zone.
Initializing zone product registry.
Determining zone package initialization order.
Preparing to initialize <1507> packages on the zone.
Initialized <1507> packages on zone.
Zone <template> is initialized.
The file </zones/template/root/var/sadm/system/logs/install_log
   > contains a log of the zone installation.
```

```
#
```

Got a coffee? The next installations will be much faster.We will not boot it as we don't need it for our testbed.

site.xml

While waiting for the zone installation to end we can create a few other files. At first you should create a file called `site.xml`. This files controls which services are online after the first boot. You can think of it like an `sysidcfg` for the Service Management Framework. The file is rather long, so i won't post it in the article directly. You can download my version of this file here[2]

Zone configurations for the testbed

At first we have to create the zone configurations. The files are very similar. The differences are in the zonepath and in the network configuration. The zone `servera` is located in `/zones/serverA` and uses the network interface `vnic2`. This will is called `serverA`:

```
create -b
set zonepath=/zones/serverA
set ip-type=exclusive
set autoboot=false
add inherit-pkg-dir
set dir=/lib
end
add inherit-pkg-dir
set dir=/platform
end
add inherit-pkg-dir
set dir=/sbin
end
add inherit-pkg-dir
set dir=/usr
end
add inherit-pkg-dir
set dir=/opt
end
add net
set physical=vnic2
```

[2] `http://www.c0t0d0s0.org/pages/sitexml.html`

```
end
commit
```

The zone `serverb` uses the directory `/zones/serverB` and is configured to bind to the interface `vnic4`. Obviously i've named the configuration file `serverB`

```
create -b
set zonepath=/zones/serverB
set ip-type=exclusive
set autoboot=false
add inherit-pkg-dir
set dir=/lib
end
add inherit-pkg-dir
set dir=/platform
end
add inherit-pkg-dir
set dir=/sbin
end
add inherit-pkg-dir
set dir=/usr
end
add inherit-pkg-dir
set dir=/opt
end
add net
set physical=vnic4
end
commit
```

We have created both config files for the simulated servers. Now we do the same for our simulated router. The configuration of the `router` zone is a little bit longer as we need more network interfaces. I opened a file called `router` and filled it with the following content:

```
create -b
set zonepath=/zones/router
set ip-type=exclusive
set autoboot=false
add inherit-pkg-dir
set dir=/lib
end
add inherit-pkg-dir
set dir=/platform
end
```

```
add inherit-pkg-dir
set dir=/sbin
end
add inherit-pkg-dir
set dir=/usr
end
add inherit-pkg-dir
set dir=/opt
end
add net
set physical=vnic5
end
add net
set physical=vnic1
end
add net
set physical=vnic3
end
commit
```

sysidcfg files

To speed up installation we create some sysidconfig files for our zones. Without this files, the installation would "go interactive" and you would have to use menus to provide the configuration informations. When you copy place such a file at `/etc/sysidcfg` the system will be initialized with the information provided in the file.

I will start with the sysidcfg file of `router` zone:

```
system_locale=C
terminal=vt100
name_service=none
network_interface=vnic5 {primary hostname=router1 ip_address
   =10.211.55.10 netmask=255.255.255.0 protocol_ipv6=no
   default_route=10.211.55.1}
network_interface=vnic1 {hostname=router1-a ip_address
   =10.211.100.10 netmask=255.255.255.0 protocol_ipv6=no
   default_route=NONE}
network_interface=vnic3 {hostname=router1-b ip_address
   =10.211.101.10 netmask=255.255.255.0 protocol_ipv6=no
   default_route=NONE}
nfs4_domain=dynamic
root_password=cmuL.HSJtwJ.I
security_policy=none
```

```
timeserver=localhost
timezone=US/Central
```

After this, we create a second sysidconfig file for our first server zone. I store the following content into a file called `servera_sysidcfg`:

```
system_locale=C
terminal=vt100
name_service=none
network_interface=vnic2 {primary hostname=server1 ip_address
   =10.211.100.11 netmask=255.255.255.0 protocol_ipv6=no
   default_route=NONE}
nfs4_domain=dynamic
root_password=cmuL.HSJtwJ.I
security_policy=none
timeserver=localhost
timezone=US/Central
```

When you look closely at the `network_interface` line you will see, that i didn't specified a default route. Please keep this in mind. In a last step i create `serverb_sysidcfg`. It's the config file for our second server zone:

```
system_locale=C
terminal=vt100
name_service=none
network_interface=vnic4 {primary hostname=server2 ip_address
   =10.211.101.11 netmask=255.255.255.0 protocol_ipv6=no
   default_route=NONE}
nfs4_domain=dynamic
root_password=cmuL.HSJtwJ.I
security_policy=none
timeserver=localhost
timezone=US/Central
```

Firing up the zones

After creating all this configuration files, we use them to create some zones. The procedure is similar for all zone. At first we do the configuration. After this we clone the `template` zone. As we located the `template` zone in a ZFS filesystem, the cloning takes just a second. Before we boot the zone, we place our configuration files we prepared while waiting for the installation of the `template` zone.

```
# zonecfg -z router -f router
# zoneadm -z router clone template
```

```
Cloning snapshot rpool/zones/template@SUNWzone3
Instead of copying, a ZFS clone has been created for this zone.
# cp router_sysidcfg /zones/router/root/etc/sysidcfg
# cp site.xml /zones/router/root/var/svc/profile
# zoneadm -z router boot
```

We repeat the steps for `servera`.

```
# zonecfg -z servera -f serverA
# zoneadm -z servera clone template
Cloning snapshot rpool/zones/template@SUNWzone3
Instead of copying, a ZFS clone has been created for this zone.
# cp serverA_sysidcfg /zones/serverA/root/etc/sysidcfg
# cp site.xml /zones/serverA/root/var/svc/profile
# zoneadm -z servera boot
```

At last we repeat it for our zone `serverb` again:

```
# zonecfg -z serverb -f serverB
# zoneadm -z serverb clone template
Cloning snapshot rpool/zones/template@SUNWzone3
Instead of copying, a ZFS clone has been created for this zone.
# cp serverb_sysidcfg /zones/serverB/root/etc/sysidcfg
# cp site.xml /zones/serverB/root/var/svc/profile
# zoneadm -z serverb boot
```

After completing the last step, we display the existing zones.

```
# zoneadm list -v
  ID NAME             STATUS     PATH                           BRAND    IP
   0 global           running    /                              native   shared
  13 router           running    /zones/router                  native   excl
  15 servera          running    /zones/serverA                 native   excl
  19 serverb          running    /zones/serverB                 native   excl
```

All zones are up and running.

Playing around with our simulated network

At first a basic check. Let's try to plumb one of the VNICs already used in a zone.

```
# ifconfig vnic2 plumb
vnic2 is used by non-globalzone: servera
```

Excellent. The system prohibits the plumbing. Before we can play with our mini network, we have to activate forwarding and routing on our new router. Since Solaris 10 this is really easy. There is a command for it:

```
# routeadm -e ipv4-forwarding
# routeadm -e ipv4-routing
# routeadm -u
# routeadm
              Configuration   Current              Current
                     Option   Configuration        System State
---------------------------------------------------------------
                IPv4 routing   enabled              enabled
                IPv6 routing   disabled             disabled
             IPv4 forwarding   enabled              enabled
             IPv6 forwarding   disabled             disabled

            Routing services   "route:default ripng:default"

Routing daemons:

                       STATE   FMRI
                    disabled   svc:/network/routing/zebra:quagga
                    disabled   svc:/network/routing/rip:quagga
                    disabled   svc:/network/routing/ripng:default
                    disabled   svc:/network/routing/ripng:quagga
                    disabled   svc:/network/routing/ospf:quagga
                    disabled   svc:/network/routing/ospf6:quagga
                    disabled   svc:/network/routing/bgp:quagga
                    disabled   svc:/network/routing/isis:quagga
                    disabled   svc:/network/routing/rdisc:default
                      online   svc:/network/routing/route:default
                    disabled   svc:/network/routing/legacy-routing:ipv4
                    disabled   svc:/network/routing/legacy-routing:ipv6
                      online   svc:/network/routing/ndp:default
```

This test goes only skin-deep into the capabilities of Solaris in regard of routing. But that is stuff for more than one LKSF tutorial. Now let's look into the routing table of one of our server:

```
# netstat -nr

Routing Table: IPv4
  Destination           Gateway           Flags  Ref     Use      Interface
-------------------- -------------------- ----- ----- ---------- ---------
default              10.211.100.10        UG        1          0 vnic2
10.211.100.0         10.211.100.11        U         1          0 vnic2
127.0.0.1            127.0.0.1            UH        1         49 lo0
```

Do you remember, that i've asked you to keep in mind, that we didn't specified a default route in the sysidcfg? But why have we such an defaultrouter now. There is some automagic in the boot. When a system with a single interfaces comes up without an default route specified in `/etc/defaultrouter` or without being a dhcp client it automatically starts up the router discovery protocol as specified by RPC 1256[3]. By using this protocol the hosts adds all available routers in the subnet as a defaultrouter.

The rdisc protocol is implemented by the `in.routed` daemon. It implements two different protocols. The first one is the already mentioned rdisc protocol. But it implements the

[3] `http://tools.ietf.org/html/rfc1256`

RIP protocol as well. The RIP protocol part is automagically activated when a system has more than one network interface.

```
# ping 10.211.100.11
10.211.100.11 is alive
# traceroute 10.211.100.11
traceroute to 10.211.100.11 (10.211.100.11), 30 hops max, 40
   byte packets
 1  10.211.101.10 (10.211.101.10)  0.285 ms  0.266 ms  0.204 ms
 2  10.211.100.11 (10.211.100.11)  0.307 ms  0.303 ms  0.294 ms
#
```

As you can see ... we've builded a network in a box.

Building a more complex network

Let's extend our example a little bit. We will create an example with more networks and switches, more servers and router. For a quick overview i put the figure 20.2 on page 188.

At first we configure additional etherstubs and VNICs:

```
# dladm create-etherstub etherstub10
# dladm create-vnic -l etherstub1 routerb1
# dladm create-vnic -l etherstub10 routerb10
# dladm create-vnic -l etherstub10 serverc1
# dladm create-vnic -l etherstub1 routerc1
# dladm create-vnic -l etherstub10 routerc2
```

As you see, you are not bound to a certain numbering scheme. You can call a vnic as you want, as long it's beginning with letters and ending with numbers. Now we use an editor to create a configuration file for our `routerB`:

```
create -b
set zonepath=/zones/routerB
set ip-type=exclusive
set autoboot=false
add inherit-pkg-dir
set dir=/lib
end
add inherit-pkg-dir
set dir=/platform
end
add inherit-pkg-dir
set dir=/sbin
```

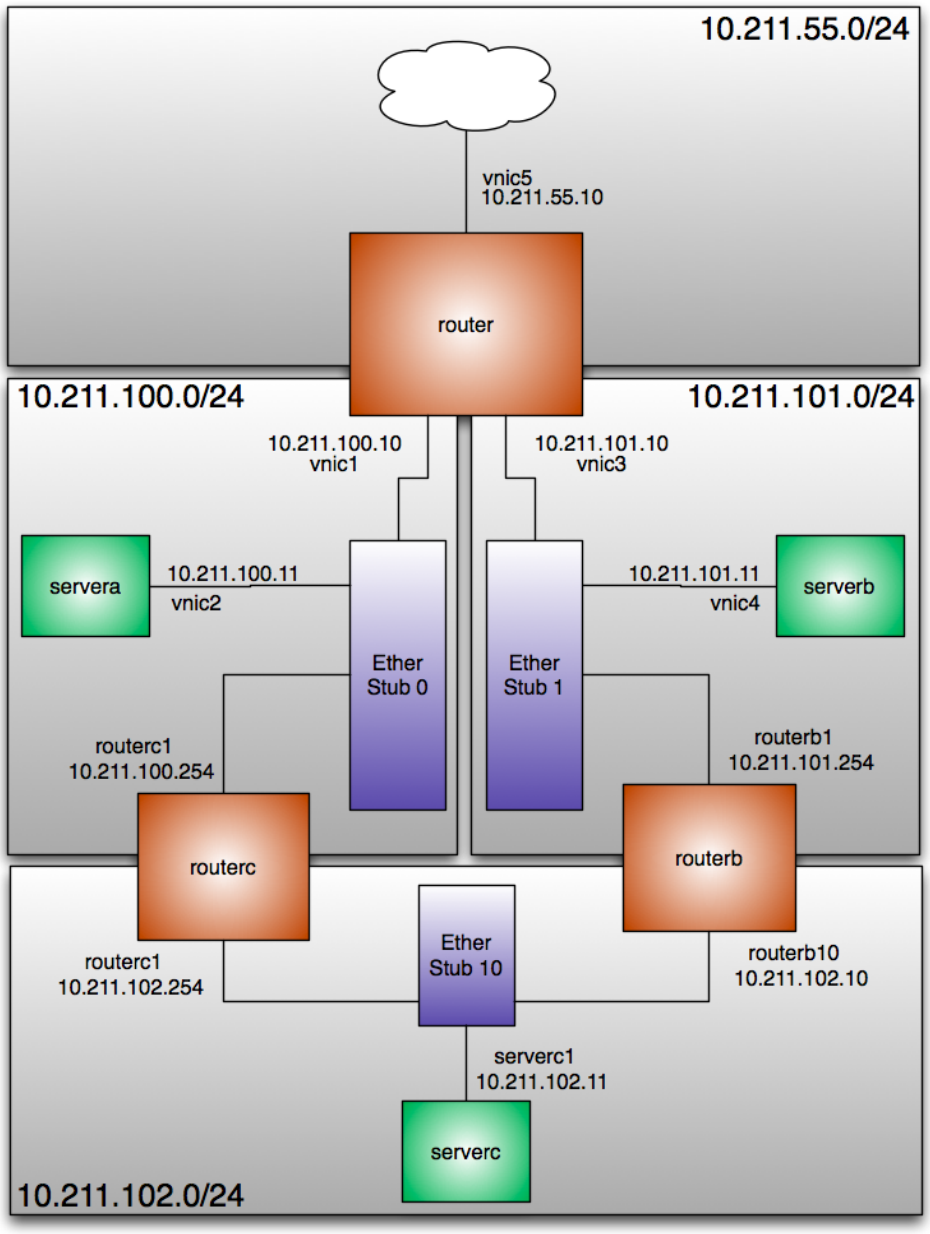

Figure 20.2.: Extended virtual network with Crossbow

```
end
add inherit-pkg-dir
set dir=/usr
end
add inherit-pkg-dir
set dir=/opt
end
add net
set physical=routerb1
end
add net
set physical=routerb10
end
commit
```

We don't have to configure any default router in this `sysidcfg` even when the system is a router itself. The system boots up with a router and will get it's routing tables from the RIP protocol.

```
system_locale=C
terminal=vt100
name_service=none
network_interface=routerb1 {primary hostname=routerb ip_address
   =10.211.101.254 netmask=255.255.255.0 protocol_ipv6=no
   default_route=NONE}
network_interface=routerb10 {hostname=routerb-a ip_address
   =10.211.102.10 netmask=255.255.255.0 protocol_ipv6=no
   default_route=NONE}
nfs4_domain=dynamic
root_password=cmuL.HSJtwJ.I
security_policy=none
timeserver=localhost
timezone=US/Central
```

Okay, we can fire up the zone.

```
# zonecfg -z routerb -f routerb
# zoneadm -z routerb clone template
Cloning snapshot rpool/zones/template@SUNWzone4
Instead of copying, a ZFS clone has been created for this zone.
# cp routerb_sysidcfg /zones/routerb/root/etc/sysidcfg
# cp site.xml /zones/routerB/root/var/svc/profile/
# zoneadm -z routerb boot
```

Okay, the next zone is the `routerc` zone. We bind it to the matching vnics in the zone configuration:

```
create -b
set zonepath=/zones/routerC
set ip-type=exclusive
set autoboot=false
add inherit-pkg-dir
set dir=/lib
end
add inherit-pkg-dir
set dir=/platform
end
add inherit-pkg-dir
set dir=/sbin
end
add inherit-pkg-dir
set dir=/usr
end
add inherit-pkg-dir
set dir=/opt
end
add net
set physical=routerc1
end
add net
set physical=routerc2
end
commit
```

The same rules as for the `routerb` apply to the `routerc`. We will rely on the routing protocols to provide a defaultroute, so we can just insert `NONE` into the `sysidcfg` for the default route.

```
# cat routerc_sysidcfg
system_locale=C
terminal=vt100
name_service=none
network_interface=routerc1 {primary hostname=routerb ip_address
   =10.211.102.254 netmask=255.255.255.0 protocol_ipv6=no
   default_route=NONE}
network_interface=routerc2 {hostname=routerb-a ip_address
   =10.211.100.254 netmask=255.255.255.0 protocol_ipv6=no
   default_route=NONE}
nfs4_domain=dynamic
root_password=cmuL.HSJtwJ.I
security_policy=none
timeserver=localhost
```

```
timezone=US/Central
```

Okay, i assume you already know the following steps. It's just the same just with other files.

```
# zonecfg -z routerc -f routerC
# zoneadm -z routerc clone template
Cloning snapshot rpool/zones/template@SUNWzone4
Instead of copying, a ZFS clone has been created for this zone.
# cp routerb_sysidcfg /zones/routerC/root/etc/sysidcfg
# cp site.xml /zones/routerC/root/var/svc/profile/
# zoneadm -z routerc boot
```

Okay, this is the last zone configuration in my tutorial. It's the zone for `serverc`:

```
create -b
set zonepath=/zones/serverC
set ip-type=exclusive
set autoboot=false
add inherit-pkg-dir
set dir=/lib
end
add inherit-pkg-dir
set dir=/platform
end
add inherit-pkg-dir
set dir=/sbin
end
add inherit-pkg-dir
set dir=/usr
end
add inherit-pkg-dir
set dir=/opt
end
add net
set physical=serverc1
end
commit
```

Again ... no defaultroute ... as this is a single-interface system we leave it to the ICMP Router Discovery Protocol to find the routers. So create a file called `serverC`.

```
system_locale=C
terminal=vt100
name_service=none
```

```
network_interface=serverc1 {primary hostname=server2 ip_address
   =10.211.102.11 netmask=255.255.255.0 protocol_ipv6=no
   default_route=NONE}
nfs4_domain=dynamic
root_password=cmuL.HSJtwJ.I
security_policy=none
timeserver=localhost
timezone=US/Central
```

Well ... it's zone startup time again ...

```
# zonecfg -z serverc -f routerC
# zoneadm -z serverc clone template
Cloning snapshot rpool/zones/template@SUNWzone4
Instead of copying, a ZFS clone has been created for this zone.
# cp serverc_sysidcfg /zones/serverC/root/etc/sysidcfg
# cp site.xml /zones/serverC/root/var/svc/profile/
# zoneadm -z serverC boot
```

So at first we have to make routers out of our routing zones. Obviously we have to login into the both routing zones and activating forwarding and routing. At first on `routerb`:

```
# routeadm -e ipv4-forwarding
# routeadm -e ipv4-routing
# routeadm -u
```

Afterwards on `routerc`. The command sequence is identical.

```
# routeadm -e ipv4-forwarding
# routeadm -e ipv4-routing
# routeadm -u
```

Now lets login into the console of our server:

```
servera# netstat -nr
Routing Table: IPv4
  Destination           Gateway           Flags  Ref     Use
         Interface
-------------------- -------------------- ----- -----
   ---------- ---------
default              10.211.100.10        UG        1
   0 vnic2
default              10.211.100.254       UG        1
   0 vnic2
10.211.100.0         10.211.100.11        U         1
   0 vnic2
```

```
127.0.0.1             127.0.0.1             UH         1
   49 lo0
```

As you see, there are two default routers in the routing table. The host receives router advertisments from two routers, thus it adds both into the routing table. Now let's have a closer at the routing table of the `routerb` system.

```
routerb# netstat -nr
Routing Table: IPv4
  Destination           Gateway           Flags  Ref     Use
         Interface
-------------------- -------------------- ----- -----
   ---------- ---------
default               10.211.101.10         UG         1
   0 routerb1
10.211.100.0          10.211.102.254        UG         1
   0 routerb10
10.211.101.0          10.211.101.254        U          1
   0 routerb1
10.211.102.0          10.211.102.10         U          1
   0 routerb10
127.0.0.1             127.0.0.1             UH         1
   23 lo0
```

This system has more than one devices. Thus the in.routed starts up as a RIP capable routing daemon. After a short moment the in.routed has learned enough about the network and adds it's routing table to the kernel. And after a short moment the routing tables of our router are filled with the routing informations provided by the routing protocols.

Conclusion

The scope of the virtualisation with crossbow part is wider than just testing. Imagine the following situation: You want to consolidate several servers in a complex networks, but you want or you cant change a configuration file. In regard of the networking configuration you just could simulate it in one machine. And as it's part of a single operating system kernel it is a very efficent way to do it. You don't need virtual I/O servers or something like that. It's the single underlying kernel of Solaris itself doing this job. Another interesting use case for Crossbow was introduced by Glenn Brunette in his concept for the immutable service containers[4]

[4] `http://wikis.sun.com/display/ISC/Home`

20.3. Bandwidth Limiting

20.3.1. Demo environment

I did the demonstration in a simple test environment. `a340` is workstation under my desk connected with Gigabit Ethernet to an Airport Extreme (AE) in bridging mode. The system has the ip address `192.168.178.109` and works as a server in this demo. It's a basic OpenSolaris 2009.06 installation with installed `apache22-packages`. `a330` is a notebook connected via `802.11n` to the same AE and it's used as the client.

20.3.2. The rationale for bandwitdth limiting

One of the basic objects in the new Crossbow stack is the flow. Any network traffic is separated into such flows. And with this flows you can do several interesting things. In this article i want to present two usages of them: Bandwidth Limiting and Flow Accounting

Of course, most of the times you want to transport data as fast as possible. But there are situations, where you want to limit the amount of network traffic. Let's assume you provider shared hosting on a platform and you want to sell certain service levels. For example a service level with unlimited bandwidth, one with 2 MBit/s per second and one with 8 MBit/s. If you don't have any mechanism to limit the bandwidth, anybody would just order the 2 MBit/s service as she or he get unlimited bandwidth in any case.

20.3.3. Configuring bandwidth limiting

Let's measure the unlimited traffic at first to have a baseline for testing the limited transmissions.

```
jmoekamp@a330:/tmp$ curl -o test1 http://192.168.178.109/random.bin
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 5598k  100 5598k    0     0  6433k      0 --:--:-- --:--:-- --:--:-- 6464k
```

As you see we are able to download the data 6464 Kilobyte per second. Okay, let us impose a limit for the http server. At first we create a flow that matches on webserver traffic.

```
jmoekamp@a340:~# flowadm add-flow -l e1000g0 -a transport=tcp,
   local_port=80 httpflow
```

When you dissect this flow configuration you get to the following ruleset:

- the traffic is on the ethernet interface `e1000g0`
- it is `tcp` traffic
- the local port is `80`
- for future reference the flow is called `httpflow`

With flowadm show-flow we can check the current configuration of flows on our system.

```
jmoekamp@a340:~# flowadm show-flow
FLOW         LINK         IPADDR                            PROTO  PORT    DSFLD
httpflow     e1000g0      --                                tcp    80      --
```

This is just the creation of the flow. To enable the bandwidth limiting we have to set some properties on this flow. To limit the traffic we have to use the maxbw property. For our first test, we set it to 2 Megabit/s:

```
jmoekamp@a340:~# flowadm set-flowprop -p maxbw=2m httpflow
```

A quick check, if we did everything correct:

```
jmoekamp@a340:~# flowadm show-flowprop
FLOW          PROPERTY        VALUE          DEFAULT        POSSIBLE
httpflow      maxbw               2          --             2m
httpflow      priority        --             --
```

Now i use my laptop as a test client and download the file again:

```
jmoekamp@a330:/tmp$ curl -o test1 http://192.168.178.109/random.bin
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 5598k  100 5598k    0     0   265k      0  0:00:21  0:00:21 --:--:--  266k
```

As you see ... 266 Kilobyte per second that's, roughly 2 MBit/s. Okay, now we try 8 Megabit/s as a limit:

```
jmoekamp@a340:~# flowadm set-flowprop -p maxbw=8m httpflow
```

We check again for the properties of the httpflow

```
jmoekamp@a340:~# flowadm show-flowprop
FLOW          PROPERTY        VALUE          DEFAULT        POSSIBLE
httpflow      maxbw               8          --             8m
httpflow      priority        --             --
```

Okay, a quick test again:

```
jmoekamp@a330:/tmp$ curl -o test1 http://192.168.178.109/random.bin
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 5598k  100 5598k    0     0   936k      0  0:00:05  0:00:05 --:--:--  933k
```

Okay, we yield 933 Kilobyte/s. That's a little bit less than 8 Mbit/s

20.4. Accouting

Okay, all the traffic in Crossbow is separated in flows (when it's not part of a configured flow, it's part of the default flow). It would be nice to use this flow information for accounting. Before doing the testing i activated the accounting with the following command line:

```
jmoekamp@a340:~# acctadm -e extended -f /var/log/net.log net
```

Now i can check for bandwidth usage. For example when i want to know the traffic usage between 18:20 and 18:24 on June 20th 2009 i can use the `flowadm show-usage` account you yield this data from the file i've configured before (in my case `/var/log/net.log`

```
jmoekamp@a340:~#  flowadm show-usage -s 06/20/2009,18:20:00
-e 06/20/2009,18:24:00 -f /var/log/net.log
FLOW         START          END            RBYTES   OBYTES   BANDWIDTH
httpflow     18:20:27       18:20:47       0        0            0 Mbps
httpflow     18:20:47       18:21:07       0        0            0 Mbps
httpflow     18:21:07       18:21:27       104814   6010271      2.446 Mbp
httpflow     18:21:27       18:21:47       0        0            0 Mbps
httpflow     18:21:47       18:22:07       0        0            0 Mbps
httpflow     18:22:07       18:22:27       0        0            0 Mbps
httpflow     18:22:27       18:22:47       0        0            0 Mbps
httpflow     18:22:47       18:23:07       0        0            0 Mbps
httpflow     18:23:07       18:23:27       121410   5333673      2.182 Mbp
httpflow     18:23:27       18:23:47       15246    676598   0.276 Mbps
```

The capability to do accounting on a per flow basis makes this feature really interesting even when you don't want to configure a traffic limit. So i configured an additional flow for SMTP traffic and now the accounting is capable to separate between the HTTP and the SMTP traffic:

```
jmoekamp@a340:~# flowadm show-flow -s
FLOW             IPACKETS   RBYTES   IERRORS     OPACKETS      OBYTES        OERRORS
httpflow         1168       77256    0           4204          6010271       0
smtpflow         18         1302     0           13            1250          0
```

21. IP Multipathing

Solaris 10/Opensolaris

21.1. The bridges at SuperUser Castle

root was sitting in SuperUser castle and everything was fine in the kingdom. But then a loud squeaking and creaking noise found root's attention. The demons of hypertext wrote into the Scrolls of Log, that they couldn't fulfill their work any longer, as the sole bridge into the vast kingdom of root lowered at this moment was broken.

Root spoke: "Use the other bridge ... there are two for a reason. Do I have to think for you all?". But the demons replied: "We can't do that ... only the infinite power of root can lower the bridge". Thus root lowered the bridge but thought "I have to do more important things than lowering bridges".

Thus root spoke a chant of infinite power and a daemon was spawned from the ether. root told the daemon "You are the guardian of the link! Protect it. Guard it. And when everything else fails, you are allowed to lower the second bridge to SuperUser castle."

21.2. Introduction

Before people start to think about clusters and load balancers to ensure the availability, they should start with the low hanging fruits. Such a low hanging fruit is the protection of the availability of the network connection. Solaris has an integrated mechanism to ensure this availability. It's called IP Multipathing.

IP Multipathing is an important part of the solution for an ever reoccurring problem , as almost all applications interact with the outside world on one way or the other. Thus ensuring the mechanisms of communication is a part of almost all architectures.

Even when you have other availability mechanisms like balancers, you want to use a protection of the IP connection out of a simple reason: Many applications have a session context and not all software architectures can replicate those session contexts to another system to enable a failover without loosing the session. So do you really want to loose this context just because of a failing network card or because of a admin unplugging a cable?

Or do you really want to provoke a cluster failover because of a failing network card? IPMP can keep such failures on a low level without needing high-availability mechanisms with a much larger impact.

Out of this reason IP Multipathing is an important part for most HA infrastructures. This tutorial wants to give you an introduction in this topic. It's not really an "less known feature" because for many people working with Solaris, IPMP is a daily part of their work. But many people new to Solaris or OpenSolaris aren't aware of the fact that Solaris has an integrated mechanism for IP Multipathing[1]. Furthermore this tutorial wants to give some insights into new developments in the field of IP Multipathing.

21.2.1. Where should I start?

This tutorial will explain two mechanisms, because the realm of "IP Multipathing" is a topic in flux at the moment. The implementation in Solaris 10 and older releases of Opensolaris (before Build 107) is vastly different to the implementation in current releases of Opensolaris(Build 107 and up).

I thought a while about the problem, what method should make the start in this tutorial. At the end I decided to explain the new IPMP mechanism first as the concepts of multipathing are a little bit more obvious in the new implementation.

21.2.2. Basic Concept of IP Multipathing

The fundamental concepts of both implementations are pretty much the same, thus I will start with a short introduction into the nomenclature of IPMP:

- **IP Link:** An IP link is the logical connection to an IP network. Think of a router, that has to legs ... one to the Internet and one to the inside network. Such a router has two IP links (even when the router has multiple connections to both networks).
- **Physical Interface:** The physical interface is the foundation of all networking, but it isn't really the basic entity in IPMP. The basic entity is the IP interface that is bound to a physical interface. Or to simply it: It's the IP address, not the cable that is managed by IPMP. Of course, you need physical interfaces. At best two or more of them, because with one path you can't do multipathing.[2]

[1] As well as most newbies to Solaris aren't aware of MPxIO ... the counterpart of IPMP for storage
[2] Albeit I can think of remote cases were IPMP with one path can be useful

- **IPMP Group:** Now you have physical interfaces on some network interface cards into several IP Links. How do you tell the system, that certain interfaces are redundant connections into the same IP link?

  The concept of the IPMP group solves this problem. You put all interfaces into a IPMP group that connect into an IP link into a IPMP group. All interfaces in this group are considered as redundancy to each other, so the IPMP can use them to receive and transmit the traffic out of this network.

- **Failure:** Okay, you may think, this one is so obvious you don't have to talk about it. Well, not really. This is one of the most frequent errors in HA. Buying or using a HA product without thinking about the failure modes that are addressed by the mechanism.

  You have to think about what failures are in-scope of IPMP and which one are out-of-scope. IPMP is called IP Multipathing for a reason: It's a tool for IP, it isn't meant for other protocols. So it protects the availability of IP services against failures. But for this task it uses information of other layers of the stack, for example the information if there is a link on the physical connection. Primarily it uses this information to speed up failover. There is no need to check upper layers if you already know that lower layers went away.

- **Failure Detection:** You do IPMP for a reason. You want protect your system from loosing its network connection in the case a networking component fails. One of the most important component of an automatic availability protection mechanism is it's capability to detect the need of doing something like switching the IP configuration to another physical interface. Without such a mechanism it's just an easier interface to switch such configuration manually.

  That said, IPMP provides two mechanisms to detect failures:

  – **Link based:** As the name suggests, the link based failure detection checks if the physical interface has an active and operational link to the physical network. When a physical interface looses its link - for example by problems with the cabling or a switch powered down - IPMP considers the interface as failed and starts to failover to a operational link.

    The monitoring mechanism for the link state is quite simple. It's done by monitoring the `RUNNING` flag of an IP interface. When you look at a functional interface with `ifconfig` you will recognize this flag:

    ```
    e1000g0: flags=209040843<UP,BROADCAST,RUNNING,MULTICAST,DEPRECATED,IPv4,
        NOFAILOVER,CoS> mtu 1500 index 11
            inet 192.168.56.201 netmask ffffff00 broadcast 192.168.56.255
            groupname production0
            ether 8:0:27:11:34:43
    ```

    When you unplug the cable, the `RUNNING` flag is missing:

```
e1000g0: flags=219040803<UP,BROADCAST,MULTICAST,DEPRECATED,IPv4,NOFAILOVER,
    FAILED,CoS> mtu 1500 index 11
        inet 192.168.56.201 netmask ffffff00 broadcast 192.168.56.255
        groupname production0
        ether 8:0:27:11:34:43
```

This method of monitoring the interfaces mandates an capability of the networking card driver to use link-based IPMP. They have to set and unset the `RUNNING` flag based on the link state.[3]

– **Probe based:** The probe base mechanism itself is independent from the hardware. It checks the IP layer on the IP layer. The basic idea is: When you are able to communicate via IP to other systems, it's safe to assume that the IP layer is there.

The probing itself is a simple mechanism. The probe based failure detection sends ICMP messages to a number of systems. As long the other systems react on those ICMP packets, a link is considered as ok. When those other systems don't react in a certain time, the link is considered as failed and the failover takes place

I will talk about the advantages and disadvantages of both in a later section.

- **Data Address:** In IPMP terminology the data addresses are the addresses that are really used for communication. An IPMP group be used for multiple data addresses. However, all data addresses have to be in the same IP link.
- **Test Address:** When you send ICMP messages to detect a failure, you need a sourcing IP address for those messages. So each physical interface needs an address that is just used for testing purposes. This address is called test address.
- **Repair and Repair detection:** When you talk about failures, you have to talk about repairs as well. When an interface is functional again - for example by using another cable or a different switch - you have to detect this situation and reactivate the interface. Without repairs and the detection of repairs you would run out of interfaces pretty soon. The repair detection is just the other side of the failure dection, just that you check for probes getting through or a link that's getting up again.
- **Target systems:** A target system is the matching opposite part of the test address. When you want to check the availability of a network connection via sending probe messages via ICMP, you need a source as well as a target for this ICMP communication.

[3] hme, eri, ce, ge, bge, qfe, dmfe, e1000g, ixgb, nge, nxge, rge, xge definitely work, ask the provider of the driver for other cards

In IPMP speak a target system is a system that is used to test the availability of an IP interface. The IPMP mechanism tries to ping the target system in order to evaluate if the network interface is still fully functional. This is done for each interface by choosing the test address as the source address of the IPMP request.

Target systems are chosen by the IPMP mechanism. The mechanism to do so is quite simple:

- Routers in an ip link are chosen as target systems automatically.
- When there are no routers connected to the IP-link, the IPMP mechanism tries to find hosts in the neighborhood. A ping is sent to the "all hosts"-multicast address `224.0.0.1`.[4]

  ```
  jmoekamp@hivemind:~$ ping -s 224.0.0.1
  PING 224.0.0.1: 56 data bytes
  64 bytes from hivemind-prod (192.168.178.200): icmp_seq=0. time=0.052 ms
  64 bytes from 192.168.178.22: icmp_seq=0. time=0.284 ms
  64 bytes from 192.168.178.114: icmp_seq=0. time=20.198 ms
  ```

  The first few systems replying to this ping are chosen as target systems.
- The automatic mechanism doesn't always choose the most optimal system for this check, thus you can specify them in the case you think a manual configuration ensures that the target system really represent a set of system, whose availability represents a check the availability of the network. Manually defined hosts have always precedence over routers, so manually defining such systems can reduce the ICMP load on your router. However, in most cases the automatic mechanism yields reasonable and sufficient results.

21.2.3. Link based vs. probe based failure/repair detection

As I wrote before, there are two methods of failure detection. Link based failure detection and probe based failure detection. Both have advantages and disadvantages:

Link based

The link based method is the fasted method of both. Whenever the link goes down, the IPMP gets a notification of the state change of the interface almost immediately. So it can react instantaneously on such failures.

Furthermore it doesn't need any test addresses. It doesn't check the availability on the IP layer and so there is no need for the interface to communicate independently from the data address.

[4] This address is specified by RFC 1112 `http://tools.ietf.org/html/rfc1112`

But there is a big disadvantage. The challenge lies in the point that it doesn't check the health of your IP connection, it just checks if there is a link. It's like a a small signal light, that indicates that there's power on the plug, but doesn't tell you if it's 220v or 110v.

There are situations when a purely link-based mechanism is misguiding, especially when the networks are getting more complex. Just think about the following network: Let's assume that link 1 fails. Obviously the link at the physical interface goes down. The link based mechanism can detect this failure and the system can react to this problem and switch over to the other networking card. But now let's assume that link 2 fails. The link on the connection 1 is still up and the system considers the connection to the network as functional. There is no change in the flags of the IP interface. However your networking connection is still broken as your defaultrouter is gone. A link means nothing when you can't communicate over it.

At first such scenarios doesn't sound so common and an intelligent network design can prevent such situations. Yes, that's correct, but just think about el-cheapo media converters from fibre to copper, that doesn't take down the link on the copper side when the link is down on the fibre side[5]. Or small switches that are misused as media converters[6]

Probe based

So how you can circumvent this problem? The solution is somewhat obvious. Don't check only the link on the physical layer. Check it on the layer that really matters. In the case of networking: Don't check if there's a physical link ... check if you can reach other systems with the IP protocol. And the probe base failure detection does exactly this.

As i wrote before, the probe based failure detection uses ICMP messages to check a functional IP network. So it can check if you really have an IP connection to your default router and not just a link to a switch somewhere between the server and the router.

But this method has a disadvantage as well: You need vastly more IP-addresses. Every interface in the IPMP address needs a test address. The test address is used to test the connection and stays on the interface even in the case of a failure[7].

The IP address consumption is huge. Given you have n interfaces you need n test addresses.An IPMP group with four connections needs 4 test addresses. However you

[5] Albeit any decent media converter has a feature that mirrors the link down state from one side to the other to ease management and to notify the connected system about problems

[6] Dont' laugh about it ... I found dusty old 10BASET hubs in raised floors working perfectly as media converters for years and years

[7] Obviously you need the test mechanism to check if the physical link was repaired by the admin

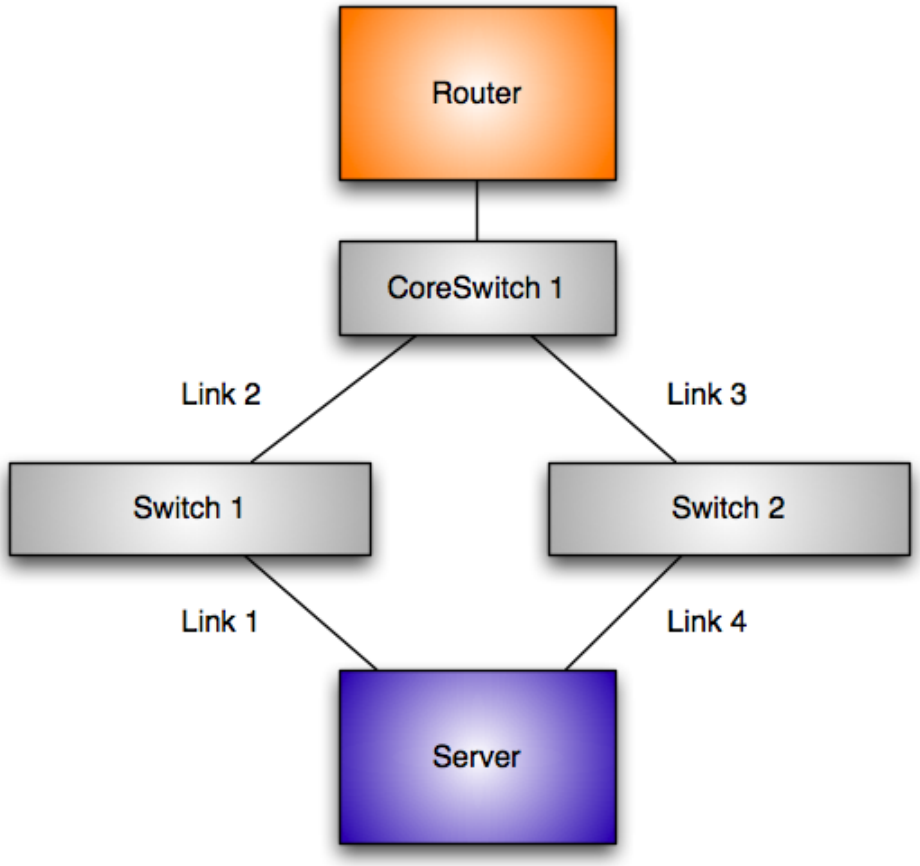

Figure 21.1.: Simple network with redundant server connection

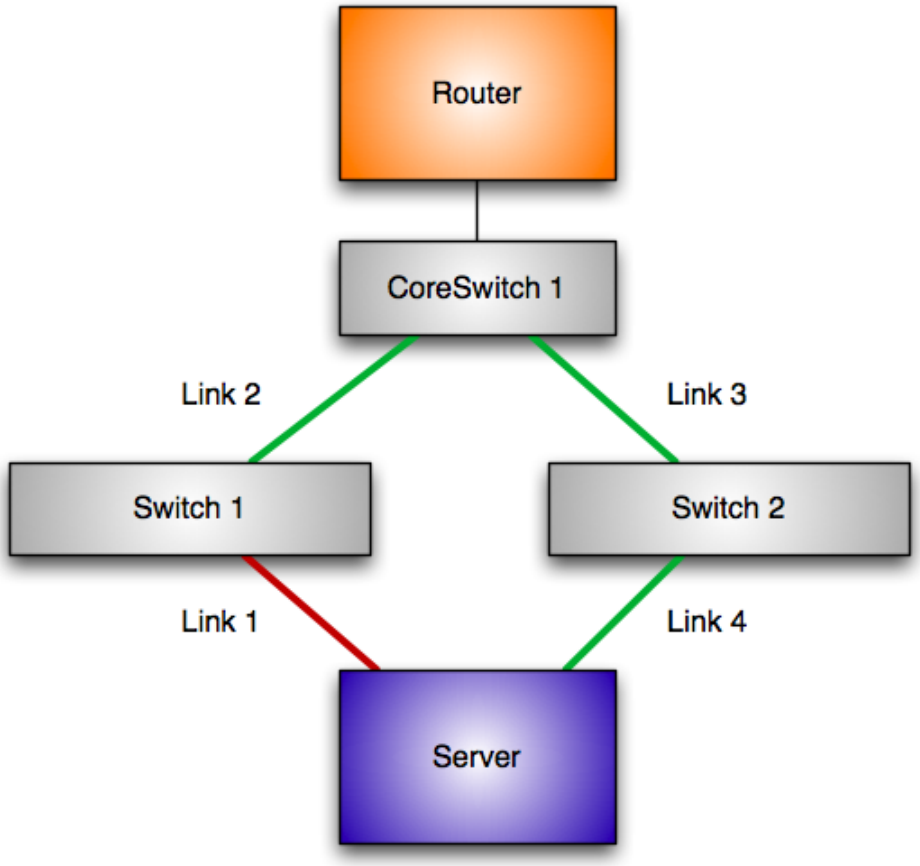

Figure 21.2.: Simple network with redundant server connection

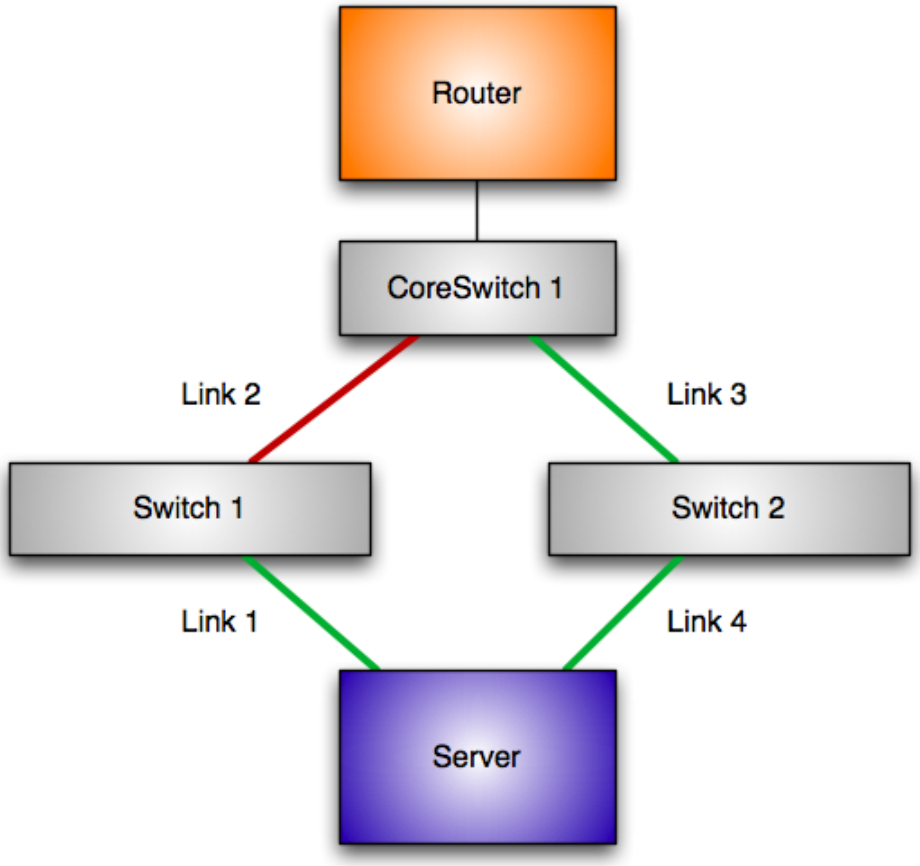

Figure 21.3.: Simple network with redundant server connection

can ease the consumption of IP-Address by using a private network for the test addresses different to the network containing the data addresses. But I will get to this at the end of this chapter.

21.2.4. Failure/Repair detection time

Another interesting question is the speed of the failure and repair detection. It's different for both mechanisms.

For link based failure detection it's easy, as soon as the IPMP subsystem gets aware of the situation, that an interface lost the `RUNNING` flag, it's considered down. It's nearly instantaneous. Even probe-based IPMP uses this mechanism to speed up the failover. Link-based failure detection is still in action, even when you use probe-based failure detection.

But what's with the reaction time of the probe based failure detection? As i've told you before, the mechanism is based on ICMP messages. There are two simple rules:

- When 5 consecutive probes fail, the interface is considered as failed
- When 10 consecutive probes get through on an interface considered as failed, it's considered as repaired

In the default configuration, probing takes place roughly every 2 seconds. You can observe this by `snoop`ing the interface when you have put it into a IPMP group.

```
jmoekamp@hivemind:~# snoop -d e1000g0 -t a -r  icmp
Using device e1000g0 (promiscuous mode)
18:56:10.82015 192.168.178.201 -> 192.168.178.1 ICMP Echo request (ID: 11017 Sequence
     number: 27065)
18:56:10.82045 192.168.178.1 -> 192.168.178.201 ICMP Echo reply (ID: 11017 Sequence
    number: 27065)
18:56:12.64018 192.168.178.201 -> 192.168.178.1 ICMP Echo request (ID: 11017 Sequence
     number: 27066)
18:56:12.64053 192.168.178.1 -> 192.168.178.201 ICMP Echo reply (ID: 11017 Sequence
    number: 27066)
```

Given the 2 seconds between the probes, a failure is detected in 10 seconds by default, a repair is detected in 20 seconds. However you can change this number in the case you need a faster failure. I will explain that on page 232 in section 21.9.4

21.2.5. IPMP vs. Link aggregation

Link aggregation is available on many switches for quite a while now. With link aggregation it is possible to bundle a number of interfaces into a single logical interface. There is a failure protection mechanism in Link Aggregation as well. At start it was

somewhat similar to the link based failure detection. When the link is down on a member of an aggregation, the switch takes the link out of the aggregation and put it's back as soon as the link get's up again. Later something similar to the probe-based mechanism found it's way into the Ethernet standards. It's called LACP. With LACP special frames are used on a link to determine if the other side of the connection is in the same aggregate[8] and if there is really an Ethernet connection between both switches.I won't go in the details now, as this will be the topic of another tutorial in the next days.

But the main purpose of link aggregation is to create a bigger pipe when a single Ethernet connection isn't enough.

So ... why should you use IPMP? The reason is a simple one. When you use link aggregation, all your connections have to terminate on the same switch, thus this mechanisms won't really help you in the case of a switch failure.

The mechanisms of IPMP doesn't work in the Layer 2 of the network, it works in the third layer and so it doesn't have this constraint. The connections of an IPMP group can end in different switches, they can have different speeds, they could be even of a different technology, as long they use IP (this was more of advantage in the past, today in the "Ethernet Everything" age this point lost its appeal).

I tend to say that link aggregation is a performance technology with some high availability capabilities, where as IPMP is a high-availability technology with some performance capabilities.

21.3. Loadspreading

A source of frequent questions is the load spreading feature in IPMP. Customers have asked me if this comparable to the aggregation. My answer is "Yes, but not really!"

Perhaps this is the right moment to explain a thing about IPMP. When you look at the interfaces of a classic IPMP configuration, it looks like the IP addresses are assigned to physical interfaces. But that isn't the truth. When you send out data on such an interface, it's spread on all active[9] interfaces of such a group.

[8] It was common configuration error in early times to have non-matching aggregation configuration

[9] Active doesn't mean functional. An interface can be functional but it isn't used by IP traffic. An interface can be declared as a standby interface, thus it may be functional but the IPMP subsystem wouldn't use it. That's useful when you have a 10 GBe Interface and a 1 GBe Interface. You don't want the 1 GBE interface for normal use, but it's better than nothing in the case the 10 GBe interface fails

But you have to be cautious: IPMP can do this only for outbound traffic. As IPMP is a server-only technology, there is no counterpart for it on the switch. So there is no load spreading on the switch.

The switches doesn't know about this situation. When an inbound packet reaches the default gateway, the router uses the usual mechanisms to get the ethernet address of the IP address and sends the data to this ethernet address. As there can be just one ethernet address for every IP address, the inbound communication will always use just one interface.

This isn't a problem for many workloads as many server applications send more data than they receive[10]. But as soon your application receives at lot of data[11], you should opt for another load distribution mechanism.

However there is a trick to circumvent this constraint: A single IPMP group can provide several data addresses. By carefully distributing this data addresses over the physical interfaces you are able to distribute the inbound load as well. So when you are able to use multiple IP addresses you could do such a manual spreading of the inbound load.

However real load spreading mechanisms with the help of the switches[12] will yield a much better distribution for the inbound traffic in many cases.

But this disadvantage comes with an advantage: You are not bound to a single switch to use this load spreading. You could terminate every interface of you server in a separate switch and the IPMP group still spreads the traffic on all interfaces. That isn't possible with the standard link aggregation technologies of Ethernet.

I want to end this section a short warning: Both aggregation technologies will not increase your bandwidth when you have just a single IP data stream. Both technologies will use the same Ethernet interface for a communication relation between client and server. It's possible to separate them even based on Layer 4 properties, but at the end the single ftp download will use just one of your lines. This is necessary to prevent out-of-order packets[13] due to different trip times of the data on separate links.

21.3.1. Classic IPMP vs. new IPMP

There are many similar concepts in Classic IPMP and New IPMP. But the both implementations have important differences as well.

The most important difference is the binding of the data address to the interfaces.

[10] For example a webserver
[11] For example a webserver
[12] Like bundling of Ethernet links via LACP
[13] You want to prevent this out of performance reasons

- With classic IPMP the data address is bound to a certain interface. In the case of the failure of an interface, the interface isn't used anymore for outbound traffic and the data address gets switched to an operational and active interface.
- With new IPMP you have a virtual `ipmp` interface in front of the physical network interfaces representing the IPMP group. The `ipmp` interface holds the data address and it isn't switched at any time. A physical interface may have a test address, but they are never configured with a data address. The virtual IPMP interface is your point of administration when you want to snoop network traffic for all interfaces in this group for example.

21.4. in.mpathd

There is a component in both variants that controls all the mechanisms surrounding IP multipathing. It's the `in.mpathd` daemon.

```
jmoekamp@hivemind:~$ ps -ef | grep "mpathd" | grep -v "grep"
    root  4523     1   0   Jan 19 ?           8:22 /lib/inet/in.mpathd
```

This daemon is automatically started by `ifconfig`, as soon you are configuring something in conjunction with IPMP on your system. The `in.mpathd` process is responsible for network adapter failure detection, repair detection, recovery, automatic failover and failback.

21.5. Prerequisites

At first you need a testbed. In this tutorial I will use a system with three interfaces. Two of them are Intel networking cards. They are named `e1000g0` and `e1000g0`. The third interface is an onboard Realtek LAN adapter called `rge0`.

The configuration of the ip network is straight forward. The subnet in this test is `192.168.178.0/24`. I have a router at `192.168.178.1`.

The physical network is a little bit more complex to demonstrate the limits of link-based failure detection. `e1000g0` and `e1000g1` are connected to a first switch called `Switch A`. This switch connects to to a second switch called `Switch B`. The `rge0` interface connects directly to `Switch B`. The router of this network is connected to `Switch B` as well.

To make the configuration a little bit more comfortable, we add a few hosts to our `/etc/hosts` file. We need four adresses while going through the tutorial. At first we need the name for the data address:

```
echo "192.168.178.200 hivemind-prod" >> /etc/hosts
```

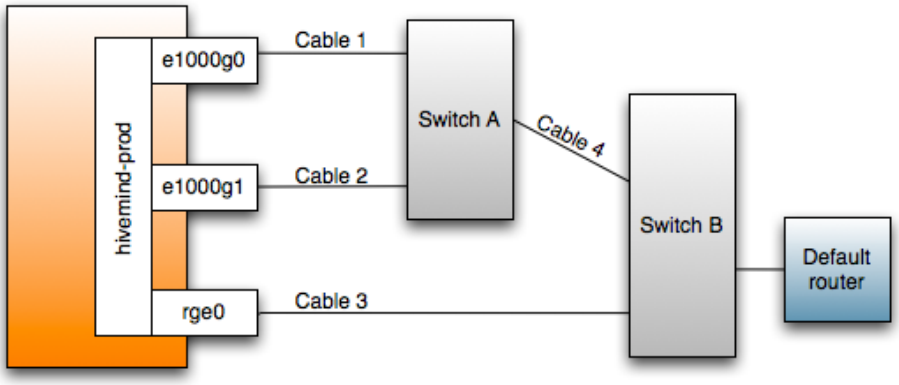

Figure 21.4.: Configuration for the demo

Now we need names for our test addresses. It's a good practice to use the name of the data address appended with the name of the physical address:

```
echo "192.168.178.201 hivemind-prod-e1000g0" >> /etc/hosts
echo "192.168.178.202 hivemind-prod-e1000g1" >> /etc/hosts
echo "192.168.178.203 hivemind-prod-rge0" >> /etc/hosts
```

21.6. New IPMP

When you want to try new IPMP, you need a fairly recent build of OpenSolaris. New IPMP was integrated into Build 107 for the first time.

At first: If you have already a working IPMP configuration, you can simply reuse this config. However it yields a different looking, but functionally equivalent result compared to your system with Classic IPMP. This is made possible by some automagic functions in New IPMP. One example is the implicit creation of the IPMP interface with the name of the IPMP group when there isn't already an IPMP interface in the group. However explicit creation should be prefered as you can choose a better name for your IPMP interface and the dependencies are much more obvious.

As I wrote before, the new IPMP doesn't use a logical interface switching from one interface to the other. You have a special kind of interface for it. It's a virtual interface. It's looking like a real interface but there is no hardware behind this interface.

So ... at first we have to configure this interface:

```
jmoekamp@hivemind:/etc# ifconfig production0 ipmp hivemind-prod up
```

With this command you've configured the IPMP interface. You can use any name for it you want, it just has to begin with a letter and has to end on a number. I have chosen the name `production0` for this tutorial

Now let's look at the interface:

```
jmoekamp@hivemind:~# ifconfig production0
production0: flags=8011000803<UP,BROADCAST,MULTICAST,IPv4,FAILED,IPMP> mtu 68 index 6
        inet 192.168.178.200 netmask ffffff00 broadcast 192.168.178.255
        groupname production0
```

As you see, it's pretty much looking like a normal network interface with some specialities: At first it's in the mode `FAILED` at the moment. There are no network interfaces configured to the group, thus you can't connect anywhere over this interface.

The interface is already configured with the data address.[14]. The data address will never move away from there. At the end you see the name of the IPMP group. The default behavior sets the name of the IPMP group and the name of the IPMP interface to the same value.

Okay, now we have to assign some physical interfaces to it. This is the moment where we have to make a decision. Do we want to use IPMP with probes or without probes? As I've explained before it's important to know at this point, what failure scenarios you want to cover with your configuration. You need to know it now, as the configuration is slightly different.

21.6.1. Link based failure detection

I want to explain the configuration of the link based failure detection first not only because it's easier, but to show you the problems of link based failure detection, too.

Configuration

As explained before, the link based failure detection just snoops on certain events of the networking card like a lost link. Thus we just have to configure the interface into the IPMP group that you want to protect against a link failure, but you don't have to configure any IP addresses on the member interfaces of the IPMP group.

Okay, at first we plumb the interfaces we want to use in our IPMP group:

```
jmoekamp@hivemind:/etc# ifconfig e1000g0 plumb
jmoekamp@hivemind:/etc# ifconfig e1000g1 plumb
jmoekamp@hivemind:/etc# ifconfig rge0 plumb
```

[14] Additional data addresses are configured as logical interfaces onto this virtual interface. You won't configure additional virtual IPMP interfaces

Okay, now we add the three member interfaces into the IPMP group:

```
jmoekamp@hivemind:/etc# ifconfig e1000g0 -failover group production0 up
jmoekamp@hivemind:/etc# ifconfig e1000g1 -failover group production0 up
jmoekamp@hivemind:/etc# ifconfig rge0 -failover group production0 up
```

As you may have noticed, we really didn't specify an IP address or a hostname. With link-based failure detection you don't need it. The IP address of the group is located on the IPMP interface we've defined a few moments ago.

But let's have a look at the `ifconfig` statements. There are two parameters you may not know:

- `-failover`: This parameter marks an interface as a non-failover one. In case of a failure, this interface configuration doesn't move. While a little bit strange in the context of a physical interface[15], but the rationale gets clearer with probe-based IPMP.
- `group production0`: the parameter `group` designates the IPMP group membership of an interface.

Let's look at one of the interfaces:

```
rge0: flags=9040843<UP,BROADCAST,RUNNING,MULTICAST,DEPRECATED,IPv4,NOFAILOVER> mtu
    1500 index 5
        inet 0.0.0.0 netmask ff000000 broadcast 0.255.255.255
        groupname production0
```

We find the consequences of both `ifconfig` parameters: The `NOFAILOVER` is obviously the result of the `-failover` and `groupname production0` is the result of the `group production0` statement. But there is another flag that is important in the realm of IPMP. It's the `DEPRECATED` flag.

The `DEPRECATED` flag has a very simple meaning: Don't use this IP interface. When an interface has this flag, the IP address won't be used to send out data[16]. As those IP addresses are just for test purposes, you don't want them to appear in packets to the outside world.

Playing around

Now we need to interact with the hardware, as we will fail network connections manually. Or to say it differently: We will pull some cables.

[15] Moving hardware is a little bit problematic just by software....

[16] Of course there are exceptions like an application specifically binding to the interface. Please look into the man page for further information

But before we are doing this, we look at the initial status of our IPMP configuration. The new IPMP model improved the monitoring capabilities of it's state by introducing a command for this task. It's called `ipmpstat`.

```
jmoekamp@hivemind:~# ipmpstat -i
INTERFACE   ACTIVE  GROUP       FLAGS     LINK      PROBE     STATE
rge0        yes     production0 -------   up        disabled  ok
e1000g1     yes     production0 -------   up        disabled  ok
e1000g0     yes     production0 --mb---   up        disabled  ok
```

Just to give you a brief tour through the output of the command. The first column reports the name of the interface, the next one reports the state of the interface from the perspective of IPMP. The third column tells you which IPMP group was assigned to this interface.

The next columns gives us some more in-depth information about the interface. The fourth column is a multipurpose column to report a number of states. In the last output, the `--mb--` tells us, that the interface e1000g0 was chosen for sending and receiving multicast and broadcast data. Other interfaces doesn't have a special state, so there are just dashes in the respective `FLAGS` field of these interfaces. The fifth column reveals, that we've disabled probes[17]. The last column details on the state of the interface. In this example it is `OK` and so it's used in the IPMP group.

Okay, now pull the cable from the `e1000g0` interface. It's Cable 1 in the figure. The system automatically switches to e1000g1 as the active interface.

```
jmoekamp@hivemind:~# ipmpstat -i
INTERFACE   ACTIVE  GROUP       FLAGS     LINK      PROBE     STATE
rge0        yes     production0 -------   up        disabled  ok
e1000g1     yes     production0 --mb---   up        disabled  ok
e1000g0     no      production0 -------   down      disabled  failed
```

As you can see, the failure has been detected on the `e1000g0` interface. The link is down, thus it is no longer active. Okay, let's repair it.

Put the cable back to the port of the `e1000g0` interface. After a moments, the link is up. The `in.mpathd` gets aware of the `RUNNING` flag on the interface. `in.mpathd` assumes that the network connection got repaired, so the state of the interface is set to `ok` and thus the interface is reactivated.

```
jmoekamp@hivemind:~# ipmpstat -i
INTERFACE   ACTIVE  GROUP       FLAGS     LINK      PROBE     STATE
rge0        yes     production0 -------   up        disabled  ok
e1000g1     yes     production0 --mb---   up        disabled  ok
e1000g0     yes     production0 -------   up        disabled  ok
```

[17] Or to be more exact, that there is no probing as we didn't configured it so far

The problem with link-based failure detection

Just in case you've played with the ethernet cables, ensure that IPMP chooses an interface connecting via Switch A as the active interface by zipping Cable 3 from the switch B for a moment. When you check with `ipmpstat -i` the `mb` has to be assigned to the interface `e1000g0` or `e1000g1`.

As i wrote before there are failure modes link-based failure detection can't detect. Now let's introduce such a fault. To do so, just remove Cable 4 between switch A and B.

```
jmoekamp@hivemind:~# ipmpstat -i
INTERFACE   ACTIVE  GROUP       FLAGS     LINK      PROBE     STATE
rge0        yes     production0 -------   up        disabled  ok
e1000g1     yes     production0 --mb---   up        disabled  ok
e1000g0     yes     production0 -------   up        disabled  ok
```

As there is still a link on the Cables 1 and 2 everything is fine from the perspective of IPMP. It doesn't switch to the connection via `rge0` which presents the only working connection to the outside world. IPMP is simply not aware of the fact that Switch A was seperated from the IP link `192.168.178.0/24` due to the removal of cable 4.

21.6.2. Probe based failure detection

The probe based detection has some additional capabilities. At first it has all the capabilities of the link-based detection. It switches over to the other network card as soon as the card loses the link. But additionally it checks the availability of the connection by pinging other IP addresses called target systems. When the system doesn't get a reply on the ICMP messages, the interface is assumed to be in failure state and it isn't used anymore. `in.mpathd` switches the data addresses to other interfaces. So how do you configure probe based IPMP?

Configuration

Okay, at first we revert back to the original state of the system. This is easy, we just have to unplumb the interfaces. In my example I'm unplumbing all interfaces. You could reuse the `production0` interface, but I'm including it here just in case you've started reading this tutorial at the beginning of this paragraph[18]. It's important that you unplumb the member interfaces of the group before you unplumb the IPMP interface, otherwise you get an error message:

```
jmoekamp@hivemind:/etc# ifconfig e1000g0 unplumb
jmoekamp@hivemind:/etc# ifconfig e1000g1 unplumb
jmoekamp@hivemind:/etc# ifconfig rge0 unplumb
jmoekamp@hivemind:/etc# ifconfig production0 unplumb
```

[18]In this case, the first three commands will fail, but you have the explicitly defined IPMP interface

Okay, now all the interfaces are away. Now we recreate the IPMP group.

```
jmoekamp@hivemind:/etc# ifconfig production0 ipmp hivemind-prod up
```

We can check the successful creation of the IPMP interface by using the `ipmpstat` command.

```
jmoekamp@hivemind:/etc# ipmpstat -g
GROUP       GROUPNAME   STATE     FDT       INTERFACES
production0 production0 failed    --        --
```

At start there isn't an interface configured into the IPMP group. So let's start to fill the group with some life.

```
jmoekamp@hivemind:/etc# ifconfig e1000g0 plumb hivemind-prod-e1000g0 -failover group
    production0 up
```

There is an important difference. This `ifconfig` statement contains an IP address, that is assigned to the physical interface. This automatically configures IPMP to use the probe based failure detection.

The idea behind the `-failover` setting gets clearer now. Obviously the test addresses of an interface should be failovered by IPMP. They should stay on the logical interface. As the interface has the `FAILOVER` flag, the complete interface including it's IP address is exempted from any failover.

Let's check the ipmp group again:

```
jmoekamp@hivemind:/etc# ipmpstat -g
GROUP       GROUPNAME   STATE     FDT       INTERFACES
production0 production0 ok        10.00s    e1000g0
```

There is now an interface in the group. Of course an IPMP group with just one interface doesn't really make sense. So configure we will configure a second interface into the group. You may have recognized the `FTD` column. `FTD` stands for "Failure Detection Time". Why is there an own column for this number? Due to the dynamic nature of the Failure Detection time, the `FDT` may be different for every group. With this column you can check the the current `FDT`.

```
jmoekamp@hivemind:/etc# ifconfig e1000g1 plumb hivemind-prod-e1000g1 -failover group
    production0 up
```

Let's check again.

```
jmoekamp@hivemind:/etc# ipmpstat -g
GROUP       GROUPNAME   STATE     FDT       INTERFACES
production0 production0 ok        10.00s    e1000g1 e1000g0
```

Now we add the third interface that is connected to the default gateway just via Switch B.

```
jmoekamp@hivemind:/etc# ifconfig rge0 plumb hivemind-prod-rge0 -failover group
    production0 up
```

Let's check again.

```
jmoekamp@hivemind:/etc# ipmpstat -g
GROUP       GROUPNAME   STATE     FDT       INTERFACES
production0 production0 ok        10.00s    rge0 e1000g1 e1000g0
```

All three interfaces are in the IPMP group now. And that's all ... we've just activated failure detection and failover by this four commands. Really simple, isn't it?

Playing around

I hope, you have still the hardware configuration in place, I used to show the problems of link based failure detection. In the case you haven't please create the configuration we've used there.

At first we do a simple test: We simply unplug a cable from the system. In my case I removed the cable 1:

```
jmoekamp@hivemind:~# ipmpstat -i
INTERFACE   ACTIVE  GROUP       FLAGS     LINK      PROBE     STATE
rge0        yes     production0 --mb---   up        ok        ok
e1000g1     yes     production0 -------   up        ok        ok
e1000g0     no      production0 -------   down      failed    failed
```

The system reacts immediately, as the link-based failure detection is still active, even when you use the probe-based mechanism. You can observe this in the `ipmpstat` output by monitoring the state of the link column. It's `down` at the moment and obviously probes can't reach their targets. The state is assumed as `failed`. Now plug the cable back to the system:

```
jmoekamp@hivemind:~# ipmpstat -i
INTERFACE   ACTIVE  GROUP       FLAGS     LINK      PROBE     STATE
rge0        yes     production0 --mb---   up        ok        ok
e1000g1     yes     production0 -------   up        ok        ok
e1000g0     no      production0 -------   up        failed    failed
```

The link is back, but the interface is still failed. IPMP works as designed here. The probing of the interface with ICMP messages still considers this interface as down. As we have now two mechanism to check the availability of the interface, both have to confirm the repair. IPMP doesn't consider an interface as repaired when just one ICMP probe gets through, it waits until 20 ICMP probes were correctly replied by the target system. Due to this probing at repair time instead of just relying on the link, you can prevent that an interface is considered as `OK` when an unconfigured switch brings the link back online, but the configuration of the switch doesn't allow to the server to connect anywhere (because of VLAN configuration for example).

```
jmoekamp@hivemind:~# ipmpstat -i
INTERFACE   ACTIVE  GROUP       FLAGS     LINK      PROBE     STATE
rge0        yes     production0 --mb---   up        ok        ok
e1000g1     yes     production0 -------   up        ok        ok
e1000g0     yes     production0 -------   up        ok        ok
jmoekamp@hivemind:~#
```

As soon as the probing of the interface is successful, it brings the interface back to the `OK` state and everything is fine.

Now we get to a more interesting use case of probe-based failure detection. Let's assume we've repaired everything and all is fine. You should see a situation similar to this one in your `ipmpstat` output:

```
jmoekamp@hivemind:~# ipmpstat -i
INTERFACE   ACTIVE  GROUP       FLAGS     LINK      PROBE     STATE
rge0        yes     production0 -------   up        ok        ok
e1000g1     yes     production0 -------   up        ok        ok
e1000g0     yes     production0 --mb---   up        ok        ok
```

Now unplug cable 4, the cable between the switch A and B. At first nothing happens, but a few seconds later IPMP switches the IP addresses to `rge0` and set the state of the other interfaces to `failed`.

```
jmoekamp@hivemind:~# ipmpstat -i
INTERFACE   ACTIVE  GROUP       FLAGS     LINK      PROBE     STATE
rge0        yes     production0 --mb---   up        ok        ok
e1000g1     no      production0 -------   up        failed    failed
e1000g0     no      production0 -------   up        failed    failed
```

When you look at the output of `ipmpstat` you will notice that the link is still up, but the probe has failed, thus the interfaces were set into the state failed.

When you plug the cable 3 back to the switches nothing will happen at first. You have to wait until the probing mechanism reports that the IPMP messages were correctly returned by the target systems.

```
jmoekamp@hivemind:~# ipmpstat -i
INTERFACE   ACTIVE  GROUP       FLAGS     LINK      PROBE     STATE
rge0        yes     production0 --mb---   up        ok        ok
e1000g1     no      production0 -------   up        failed    failed
e1000g0     no      production0 -------   up        failed    failed
```

After a few seconds it should deliver an `ipmpstat` output reporting everything is well again.

```
jmoekamp@hivemind:~# ipmpstat -i
INTERFACE   ACTIVE  GROUP       FLAGS     LINK      PROBE     STATE
rge0        yes     production0 --mb---   up        ok        ok
e1000g1     yes     production0 -------   up        ok        ok
e1000g0     yes     production0 -------   up        ok        ok
```

21.6.3. Making the configuration boot persistent

As you have recognized for sure, all this configuration took place with the `ifconfig` statement. This configuration is lost when you reboot the system. But there is already an entity that configures the interfaces at system start. It's using the `hostname.*` files. Thus we could use these files for IPMP as well.

Boot persistent link-based configuration

Okay, to recreate our link-based IPMP configuration in a boot persistent, we need to fill the `hostname.*` files with the following statements:

```
jmoekamp@hivemind:/etc# echo "ipmp group production0 hivemind-prod up" > /etc/
    hostname.production0
jmoekamp@hivemind:/etc# echo "group production0 -failover up" > /etc/hostname.e1000g0
jmoekamp@hivemind:/etc# echo "group production0 -failover up" > /etc/hostname.e1000g1
jmoekamp@hivemind:/etc# echo "group production0 -failover up" > /etc/hostname.rge0
```

We reboot the system now to ensure that we did everything correctly. When the system has booted up, we will check if we made an error.

```
jmoekamp@hivemind:~$ ipmpstat -g
GROUP       GROUPNAME   STATE     FDT       INTERFACES
production0 production0 ok        --        rge0 e1000g1 e1000g0
jmoekamp@hivemind:~$ ipmpstat -i
INTERFACE   ACTIVE  GROUP       FLAGS     LINK      PROBE     STATE
rge0        yes     production0 -------   up        disabled  ok
e1000g1     yes     production0 -------   up        disabled  ok
e1000g0     yes     production0 --mb---   up        disabled  ok
```

Looks good. Now let's look into the list of interfaces.

```
jmoekamp@hivemind:~$ ifconfig -a
lo0: flags=2001000849<UP,LOOPBACK,RUNNING,MULTICAST,IPv4,VIRTUAL> mtu 8232 index 1
        inet 127.0.0.1 netmask ff000000
production0: flags=8001000843<UP,BROADCAST,RUNNING,MULTICAST,IPv4,IPMP> mtu 1500
    index 2
        inet 192.168.178.200 netmask ffffff00 broadcast 192.168.178.255
        groupname production0
e1000g0: flags=9040843<UP,BROADCAST,RUNNING,MULTICAST,DEPRECATED,IPv4,NOFAILOVER> mtu
     1500 index 3
        inet 0.0.0.0 netmask ff000000 broadcast 0.255.255.255
        groupname production0
e1000g1: flags=9040843<UP,BROADCAST,RUNNING,MULTICAST,DEPRECATED,IPv4,NOFAILOVER> mtu
     1500 index 4
        inet 0.0.0.0 netmask ff000000 broadcast 0.255.255.255
        groupname production0
rge0: flags=9040843<UP,BROADCAST,RUNNING,MULTICAST,DEPRECATED,IPv4,NOFAILOVER> mtu
    1500 index 5
        inet 0.0.0.0 netmask ff000000 broadcast 0.255.255.255
        groupname production0
lo0: flags=2002000849<UP,LOOPBACK,RUNNING,MULTICAST,IPv6,VIRTUAL> mtu 8252 index 1
        inet6 ::1/128
jmoekamp@hivemind:~$
```

IPMP configured and boot-persistent? Check.

Boot persistent probe-based configuration

We can do the same for the probe-based IPMP:

```
jmoekamp@hivemind:/etc# echo "ipmp group production0 hivemind-prod up" > /etc/
    hostname.production0
jmoekamp@hivemind:/etc# echo "group production0 -failover hivemind-prod-e1000g0 up" >
     /etc/hostname.e1000g0
jmoekamp@hivemind:/etc# echo "group production0 -failover hivemind-prod-e1000g1 up" >
     /etc/hostname.e1000g1
jmoekamp@hivemind:/etc# echo "group production0 -failover hivemind-prod-rge0 up" > /
    etc/hostname.rge0
```

Reboot the system and login afterwards to check the list of interfaces.

```
jmoekamp@hivemind:~$ ifconfig -a
lo0: flags=2001000849<UP,LOOPBACK,RUNNING,MULTICAST,IPv4,VIRTUAL> mtu 8232 index 1
        inet 127.0.0.1 netmask ff000000
production0: flags=8001000843<UP,BROADCAST,RUNNING,MULTICAST,IPv4,IPMP> mtu 1500
    index 2
        inet 192.168.178.200 netmask ffffff00 broadcast 192.168.178.255
        groupname production0
e1000g0: flags=9040843<UP,BROADCAST,RUNNING,MULTICAST,DEPRECATED,IPv4,NOFAILOVER> mtu
     1500 index 3
        inet 192.168.178.201 netmask ffffff00 broadcast 192.168.178.255
        groupname production0
e1000g1: flags=9040843<UP,BROADCAST,RUNNING,MULTICAST,DEPRECATED,IPv4,NOFAILOVER> mtu
     1500 index 4
        inet 192.168.178.202 netmask ffffff00 broadcast 192.168.178.255
        groupname production0
rge0: flags=9040843<UP,BROADCAST,RUNNING,MULTICAST,DEPRECATED,IPv4,NOFAILOVER> mtu
    1500 index 5
        inet 192.168.178.203 netmask ffffff00 broadcast 192.168.178.255
        groupname production0
lo0: flags=2002000849<UP,LOOPBACK,RUNNING,MULTICAST,IPv6,VIRTUAL> mtu 8252 index 1
        inet6 ::1/128
```

Let's check the configuration via `ipmpstat`, too:

```
jmoekamp@hivemind:~$ ipmpstat -i
INTERFACE   ACTIVE  GROUP       FLAGS     LINK      PROBE     STATE
rge0        yes     production0 --mb---   up        ok        ok
e1000g1     yes     production0 -------   up        ok        ok
e1000g0     yes     production0 -------   up        ok        ok
jmoekamp@hivemind:~$
```

Everything is fine.

21.6.4. Using IPMP and Link Aggregation

As I wrote before there is another way to protect your system against the failure of a network connection - Link Aggregation. As i've explained before, there are failure modes that can't be addressed by link aggregation. But you can use both in conjunction. This makes sense, when your main connection is a 10GBe interface and you don't want to

plug a second one into the system and use already existent 1GBe Interfaces as a backup for it instead.

It's pretty straightforward to do so. At first you have to configure the link aggregation.

```
jmoekamp@hivemind:~$ pfexec bash
jmoekamp@hivemind:~# ifconfig e1000g0 unplumb
jmoekamp@hivemind:~# ifconfig e1000g1 unplumb
jmoekamp@hivemind:~# dladm create-aggr -l e1000g0 -l e1000g1 aggregate0
jmoekamp@hivemind:~# dladm show-aggr -x aggregate0
LINK        PORT            SPEED DUPLEX   STATE     ADDRESS             PORTSTATE
aggregate0  --              0Mb   unknown  unknown   0:1b:21:3d:91:f7    --
            e1000g0         0Mb   half     down      0:1b:21:3d:91:f7    standby
            e1000g1         0Mb   half     down      0:1b:21:16:8d:7f    standby
```

The `dladm create-aggr` creates an aggregation, that bundles the interfaces `e1000g0` and `e1000g1` into a single virtual interface. Now I plug both cables into the switch.

```
jmoekamp@hivemind:~# dladm show-aggr -x aggregate0
LINK        PORT            SPEED DUPLEX   STATE     ADDRESS             PORTSTATE
aggregate0  --              100Mb full     up        0:1b:21:3d:91:f7    --
            e1000g0         100Mb full     up        0:1b:21:3d:91:f7    attached
            e1000g1         100Mb full     up        0:1b:21:16:8d:7f    attached
```

Interfaces are up, the aggregation is ready for use.

```
jmoekamp@hivemind:~# ifconfig rge0 unplumb
jmoekamp@hivemind:~# ifconfig production0 ipmp hivemind-prod up
jmoekamp@hivemind:~# ifconfig aggregate0 plumb
jmoekamp@hivemind:~# ifconfig aggregate0 -failover group production0 up
jmoekamp@hivemind:~# ifconfig rge0 plumb
jmoekamp@hivemind:~# ifconfig rge0 -failover group production0 up
```

Looks pretty much like a standard IPMP configuration. You can think of `aggregate0` as a plain-standard physical interface from the perspective the the admin. When we check the IPMP configuration we will see both interfaces.

```
jmoekamp@hivemind:~# ipmpstat -i
INTERFACE   ACTIVE  GROUP       FLAGS     LINK      PROBE     STATE
rge0        yes     production0 -------   up        disabled  ok
aggregate0  yes     production0 --mb---   up        disabled  ok
jmoekamp@hivemind:~# ipmpstat -g
GROUP       GROUPNAME   STATE     FDT       INTERFACES
production0 production0 ok        --        rge0 aggregate0
```

Now we unplug one of the aggregated cables.

```
jmoekamp@hivemind:~# dladm show-aggr -x aggregate0
LINK        PORT            SPEED DUPLEX   STATE     ADDRESS             PORTSTATE
aggregate0  --              100Mb full     up        0:1b:21:3d:91:f7    --
            e1000g0         100Mb full     up        0:1b:21:3d:91:f7    attached
            e1000g1         0Mb   half     down      0:1b:21:16:8d:7f    standby
jmoekamp@hivemind:~# ipmpstat -g
GROUP       GROUPNAME   STATE     FDT       INTERFACES
production0 production0 ok        --        rge0 aggregate0
jmoekamp@hivemind:~# ipmpstat -i
INTERFACE   ACTIVE  GROUP       FLAGS     LINK      PROBE     STATE
rge0        yes     production0 -------   up        disabled  ok
aggregate0  yes     production0 --mb---   up        disabled  ok
```

Everything is still okay. The aggregate hides the fact of the one failed interface from the IPMP subsystem. Now we unplug the second interface.

```
jmoekamp@hivemind:~# ipmpstat -i
INTERFACE   ACTIVE  GROUP       FLAGS     LINK      PROBE     STATE
rge0        yes     production0 --mb---   up        disabled  ok
aggregate0  no      production0 -------   down      disabled  failed
jmoekamp@hivemind:~# ipmpstat -g
GROUP       GROUPNAME   STATE     FDT       INTERFACES
production0 production0 degraded  --        rge0 [aggregate0]
jmoekamp@hivemind:~# ipmpstat -i
INTERFACE   ACTIVE  GROUP       FLAGS     LINK      PROBE     STATE
rge0        yes     production0 --mb---   up        disabled  ok
aggregate0  no      production0 -------   down      disabled  failed
jmoekamp@hivemind:~# dladm show-aggr -x aggregate0
LINK        PORT            SPEED DUPLEX   STATE     ADDRESS            PORTSTATE
aggregate0  --              0Mb   unknown  down      0:1b:21:3d:91:f7   --
            e1000g0         0Mb   half     down      0:1b:21:3d:91:f7   standby
            e1000g1         0Mb   half     down      0:1b:21:16:8d:7f   standby
```

The links are both down, and without a functional interface left, the "link" of the aggregate goes down as well[19]. Of course the IPMP subsystem switches to `rge0` now. When we plug one cable back to the switch, the aggregate is functional again and IPMP detects this and the interface is considered as functional in IPMP again, too.

```
jmoekamp@hivemind:~# dladm show-aggr -x aggregate0
LINK        PORT            SPEED DUPLEX   STATE     ADDRESS            PORTSTATE
aggregate0  --              100Mb full     up        0:1b:21:3d:91:f7   --
            e1000g0         100Mb full     up        0:1b:21:3d:91:f7   attached
            e1000g1         0Mb   half     down      0:1b:21:16:8d:7f   standby
jmoekamp@hivemind:~# ipmpstat -i
INTERFACE   ACTIVE  GROUP       FLAGS     LINK      PROBE     STATE
rge0        yes     production0 --mb---   up        disabled  ok
aggregate0  yes     production0 -------   up        disabled  ok
```

When you plug the second interface into the interface, the aggregate is complete. But it doesn't change a thing from the IPMP side, as the `aggregate0` interface was already functional from the perpective of IPMP with just one interface.

```
jmoekamp@hivemind:~# dladm show-aggr -x aggregate0
LINK        PORT            SPEED DUPLEX   STATE     ADDRESS            PORTSTATE
aggregate0  --              100Mb full     up        0:1b:21:3d:91:f7   --
            e1000g0         100Mb full     up        0:1b:21:3d:91:f7   attached
            e1000g1         100Mb full     up        0:1b:21:16:8d:7f   attached
jmoekamp@hivemind:~#
```

21.6.5. Monitoring the actions of IPMP in your logfiles

All actions of the IPMP subsystem are logged by syslog. In this section I will show you the log messages that you get when a failure occurs and a repair takes place. The `mpathd` is somewhat chatty about the stuff it does.

[19] It stays up, as long as there's a functional interface in the aggregate

Failure and repair of a single interface

```
Jan  8 20:01:06 hivemind in.mpathd[15113]: [ID 215189 daemon.error] The link has gone
     down on rge0
Jan  8 20:01:06 hivemind in.mpathd[15113]: [ID 968981 daemon.error] IP interface
    failure detected on rge0 of group production0
Jan  8 20:01:28 hivemind in.mpathd[15113]: [ID 820239 daemon.error] The link has come
     up on rge0
Jan  8 20:01:43 hivemind in.mpathd[15113]: [ID 341557 daemon.error] IP interface
    repair detected on rge0 of group production0
```

Failure of all interfaces and repair of a single interface

```
Jan  8 20:00:35 hivemind in.mpathd[15113]: [ID 773107 daemon.error] All IP interfaces
     in group production0 are now unusable
Jan  8 20:00:51 hivemind in.mpathd[15113]: [ID 561795 daemon.error] At least 1 IP
    interface (rge0) in group production0 is now usable
```

21.7. Classic IPMP

IPMP itself is a really old feature. It's in Solaris for several versions now. Just the implementation I've described before is a new one. But in Solaris 10 you don't have this new IPMP implementation. Solaris 10 still uses the old implementation. I will call the old implementation classic IPMP. The basic mechanism of new and classic IPMP is pretty much the same: Providing failure detection mechanisms and switch something to do a failover thus the data address stays available. But internally it's a completely different implementation. While the new mechanism is certainly the future of IPMP, I' m pretty sure you will old mechanism more often in the wild.

21.7.1. Prerequisites

This example works with the same configuration, but you need a system with Solaris 10 or an Opensolaris System with a build earlier than 107. I just use Opensolaris on my lab machines, thus i used an virtualized Solaris 10 to explain the configuration of classic IPMP.

I will use the following addresses:

```
192.168.178.200 vhivemind-prod
192.168.178.201 vhivemind-e1000g0
192.168.178.202 vhivemind-e1000g1
```

I will demonstrate this on a recent release of Solaris 10:

```
bash-3.00# cat /etc/release
                        Solaris 10 5/09 s10x_u7wos_08 X86
           Copyright 2009 Sun Microsystems, Inc.  All Rights Reserved.
                        Use is subject to license terms.
                             Assembled 30 March 2009
```

21.7.2. Link based classic IPMP

An important difference to the new IPMP implementation is the point that you doesn't create a distinct IPMP interface because the concept of such a thing doesn't exist in classic IPMP. With link based classic IPMP you just put the interfaces in a group

```
bash-3.00# ifconfig e1000g0 plumb
bash-3.00# ifconfig e1000g1 plumb
bash-3.00# ifconfig e1000g0 vhivemind-prod netmask + broadcast + group production0 up
bash-3.00# ifconfig e1000g1 group production0 up
bash-3.00# ifconfig -a
```

Let's have a short look onto the network configuration.

```
lo0: flags=2001000849<UP,LOOPBACK,RUNNING,MULTICAST,IPv4,VIRTUAL> mtu 8232 index 1
        inet 127.0.0.1 netmask ff000000
e1000g0: flags=201000843<UP,BROADCAST,RUNNING,MULTICAST,IPv4,CoS> mtu 1500 index 15
        inet 192.168.56.200 netmask ffffff00 broadcast 192.168.56.255
        groupname production0
        ether 8:0:27:11:34:43
e1000g1: flags=201000843<UP,BROADCAST,RUNNING,MULTICAST,IPv4,CoS> mtu 1500 index 16
        inet 0.0.0.0 netmask ff000000
        groupname production0
        ether 8:0:27:6d:9:be
```

The data address is directly bound to one of the interfaces. It's important to know,that even when the `ifconfig` output suggest something different, outbound data flows to the network on both interfaces, not just the one which holds the data address.

Now unplug the cable connecting to `e1000g0`

```
bash-3.00# ifconfig -a
lo0: flags=2001000849<UP,LOOPBACK,RUNNING,MULTICAST,IPv4,VIRTUAL> mtu 8232 index 1
        inet 127.0.0.1 netmask ff000000
e1000g0: flags=219000802<BROADCAST,MULTICAST,IPv4,NOFAILOVER,FAILED,CoS> mtu 0 index
    15
        inet 0.0.0.0 netmask 0
        groupname production0
        ether 8:0:27:11:34:43
e1000g1: flags=201000843<UP,BROADCAST,RUNNING,MULTICAST,IPv4,CoS> mtu 1500 index 16
        inet 0.0.0.0 netmask ff000000
        groupname production0
        ether 8:0:27:6d:9:be
e1000g1:1: flags=201000843<UP,BROADCAST,RUNNING,MULTICAST,IPv4,CoS> mtu 1500 index 16
        inet 192.168.56.200 netmask ffffff00 broadcast 192.168.56.255
```

The data address was moved away from `e1000g1` and a logical interface was created to hold it instead.

21.7.3. Probe based classic IPMP

The failure detection by IPMP probes is available in classic IPMP as well. Again all the configuration is done via `ifconfig`

```
bash-3.00# ifconfig e1000g0 plumb
bash-3.00# ifconfig e1000g1 plumb
bash-3.00# ifconfig e1000g0 vhivemind-e1000g0 deprecated -failover netmask +
    broadcast + group production0 up
bash-3.00# ifconfig e1000g0 addif vhivemind-prod netmask +  broadcast + up
Created new logical interface e1000g0:1
bash-3.00# ifconfig e1000g1 vhivemind-e1000g1 deprecated -failover netmask +
    broadcast + group production0 up
bash-3.00# ifconfig -a
```

Please note that you have to use the `deprecated` option to set the `DEPRECATED` flag on your own. New IPMP do this automagically. Forgetting this option leads to interesting, but not always obvious malfunctions. Let's check the network configuration.

```
lo0: flags=2001000849<UP,LOOPBACK,RUNNING,MULTICAST,IPv4,VIRTUAL> mtu 8232 index 1
        inet 127.0.0.1 netmask ff000000
e1000g0: flags=209040843<UP,BROADCAST,RUNNING,MULTICAST,DEPRECATED,IPv4,NOFAILOVER,
    CoS> mtu 1500 index 11
        inet 192.168.56.201 netmask ffffff00 broadcast 192.168.56.255
        groupname production0
        ether 8:0:27:11:34:43
e1000g0:1: flags=201000843<UP,BROADCAST,RUNNING,MULTICAST,IPv4,CoS> mtu 1500 index 11
        inet 192.168.56.200 netmask ffffff00 broadcast 192.168.56.255
e1000g1: flags=209040843<UP,BROADCAST,RUNNING,MULTICAST,DEPRECATED,IPv4,NOFAILOVER,
    CoS> mtu 1500 index 12
        inet 192.168.56.202 netmask ffffff00 broadcast 192.168.56.255
        groupname production0
        ether 8:0:27:6d:9:be
```

Both interfaces have their test addresses. The data address is configured to an additional logical interface. As it's the only interface without the `-failover` statement, this interface is automatically managed by IPMP. Now remove the cable from the `e1000g0` networking card.

```
bash-3.00# ifconfig -a
lo0: flags=2001000849<UP,LOOPBACK,RUNNING,MULTICAST,IPv4,VIRTUAL> mtu 8232 index 1
        inet 127.0.0.1 netmask ff000000
e1000g0: flags=219040803<UP,BROADCAST,MULTICAST,DEPRECATED,IPv4,NOFAILOVER,FAILED,CoS
    > mtu 1500 index 11
        inet 192.168.56.201 netmask ffffff00 broadcast 192.168.56.255
        groupname production0
        ether 8:0:27:11:34:43
e1000g1: flags=209040843<UP,BROADCAST,RUNNING,MULTICAST,DEPRECATED,IPv4,NOFAILOVER,
    CoS> mtu 1500 index 12
        inet 192.168.56.202 netmask ffffff00 broadcast 192.168.56.255
        groupname production0
        ether 8:0:27:6d:9:be
e1000g1:1: flags=201000843<UP,BROADCAST,RUNNING,MULTICAST,IPv4,CoS> mtu 1500 index 12
        inet 192.168.56.200 netmask ffffff00 broadcast 192.168.56.255
```

The virtual interface with the data address has moved from `e1000g0` to `e1000g1`

21.7.4. Making the configuration boot persistent

Making the configuration boot-persistent works works pretty much the same in both implementation. As we used `ifconfig` commands again, we can use the `hostname.*`

files. We just have to translate the command lines accordingly:

Link-based IPMP

At first we configure the `e1000g0` interface by creating the file `/etc/hostname.e1000g0` containing a single line.

```
vhivemind-prod netmask + broadcast + group production0 up
```

Afterwards we do the same for `e1000g1`. We create a file named `/etc/hostname.e1000g1` and put the following line (and just this line) in it:

```
group production0 up
```

Now reboot the system. After a few moments you can get a shell and check your configuration.

```
# ifconfig -a
lo0: flags=2001000849<UP,LOOPBACK,RUNNING,MULTICAST,IPv4,VIRTUAL> mtu 8232 index 1
        inet 127.0.0.1 netmask ff000000
e1000g0: flags=201000843<UP,BROADCAST,RUNNING,MULTICAST,IPv4,CoS> mtu 1500 index 2
        inet 192.168.56.200 netmask ffffff00 broadcast 192.168.56.255
        groupname production0
        ether 8:0:27:11:34:43
e1000g1: flags=201000843<UP,BROADCAST,RUNNING,MULTICAST,IPv4,CoS> mtu 1500 index 3
        inet 0.0.0.0 netmask ff000000 broadcast 0.255.255.255
        groupname production0
        ether 8:0:27:6d:9:be
```

Everything configured as we've planned it.

Probe-based IPMP

Okay, let's do the same for the probe-based IPMP. This is the `/etc/hostname.e1000g0` file configuring the test address on the physical interface and the data address:

```
vhivemind-e1000g0 deprecated -failover netmask + broadcast + group production0 up \
addif vhivemind-prod netmask + broadcast + up
```

The file `/etc/hostname.e1000g1` with the following line will configures the `e1000g1` interface of our system at boot:

```
vhivemind-e1000g1 deprecated -failover netmask + broadcast + group production0  up
```

Okay, reboot your system and you should yield an `ifconfig` output like this one afterwards.

```
# ifconfig -a
lo0: flags=2001000849<UP,LOOPBACK,RUNNING,MULTICAST,IPv4,VIRTUAL> mtu 8232 index 1
        inet 127.0.0.1 netmask ff000000
e1000g0: flags=209040843<UP,BROADCAST,RUNNING,MULTICAST,DEPRECATED,IPv4,NOFAILOVER,
    CoS> mtu 1500 index 2
        inet 192.168.56.201 netmask ffffff00 broadcast 192.168.56.255
        groupname production0
        ether 8:0:27:11:34:43
e1000g0:1: flags=201000843<UP,BROADCAST,RUNNING,MULTICAST,IPv4,CoS> mtu 1500 index 2
        inet 192.168.56.200 netmask ffffff00 broadcast 192.168.56.255
e1000g1: flags=209040843<UP,BROADCAST,RUNNING,MULTICAST,DEPRECATED,IPv4,NOFAILOVER,
    CoS> mtu 1500 index 3
        inet 192.168.56.202 netmask ffffff00 broadcast 192.168.56.255
        groupname production0
        ether 8:0:27:6d:9:be
```

Everything is fine.

21.8. Classic and new IPMP compared

We've configured both mechanisms now. Let's summarize what we have seen so far. When you use the new IPMP the data address is bound to its own interface. Whatever happens to the physical interface there are no changes to the binding of interfaces to ip addresses.

Classic IPMP is different. In the figure I've highlighted the data address by using a bold font. When an interface fails it moves the data address is moved to the functional interface.

There are other differences as well:

- New IPMP provides better observability by the `ipmpstat` tool
- You can assign test addresses with DHCP. This is especially useful when you using a distinct network for your test addresses. As the test addresses are just used for the failure probing you can use ephemeral addresses for them and don't have to manually track them.
- As new IPMP uses an distinct interface, it solves a lot of deficiencies of classic IPMP. To get an overview of this shortcomings you should look at the development portal of the new IPMP mentioned in the sources of this document stated at the end of this chapter.

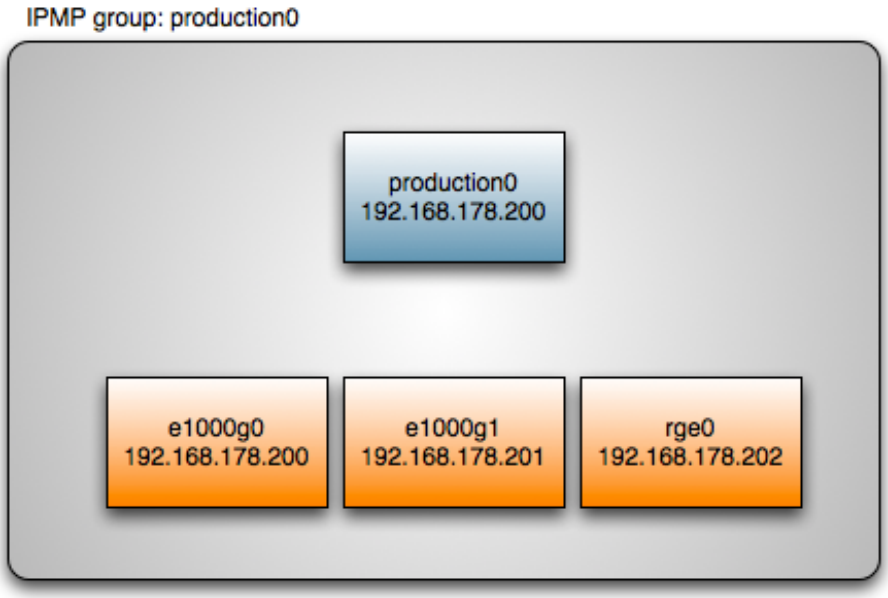

Figure 21.5.: New IPMP - everything okay

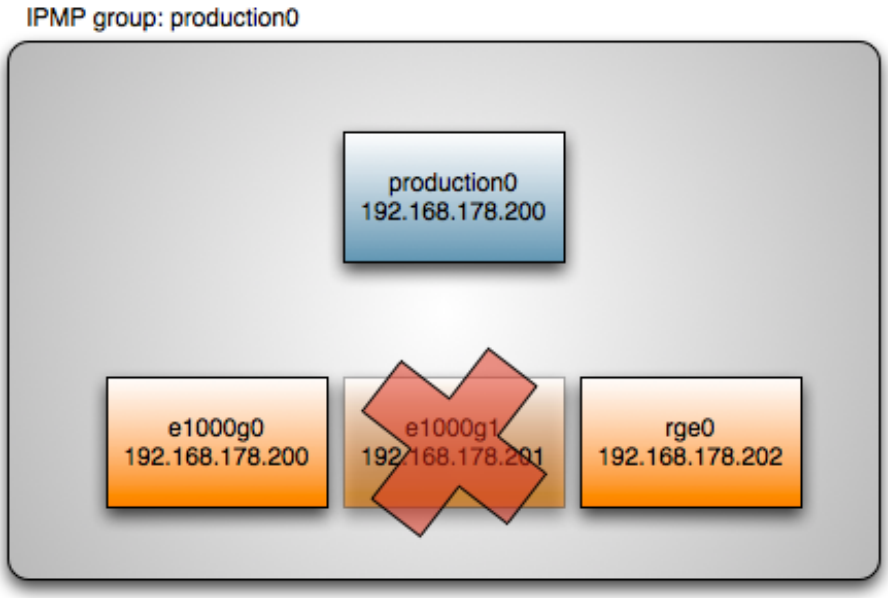

Figure 21.6.: New IPMP - e1000g1 failed

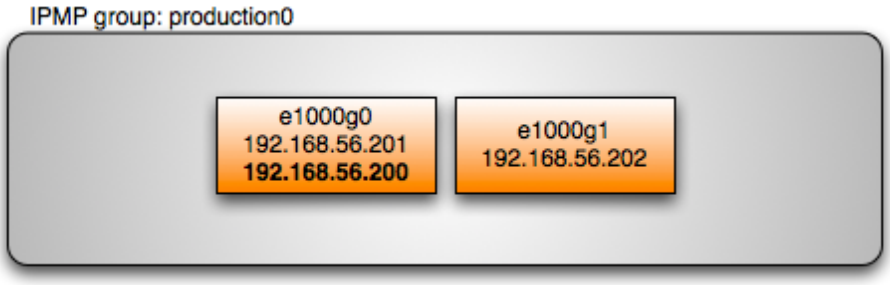

Figure 21.7.: Classic IPMP - everything okay

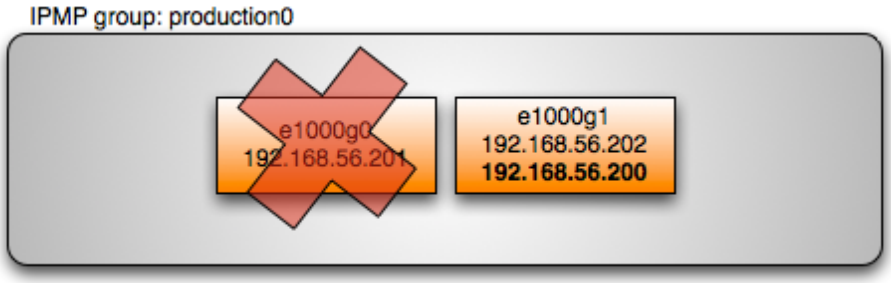

Figure 21.8.: Classic IPMP - e1000g0 failed

21.9. Tips, Tricks and other comments

21.9.1. Reducing the address sprawl of probe based failure detection

The probe based failure detection isn't without a disadvantage. You need a test IP adresses for every interface in an IPMP group. But: The IP addresses doesn't have to be routable ones, they doesn't even have to be in the same subnet as the data address. It's perfectly possible to use a routable IP address as the data address and to use for example the 192.168.1.0/24 address just for the IPMP test addresses . You have just to take care of one additional configuration item: You have to provide test targets in the same network. This can be done by configuring an an additional IP address on your router interfaces for example.

21.9.2. Explicitly configuring target systems

Configuring explicit target systems is easy. You just have to configure host routes to them. Thus `route add -host 192.168.178.2 192.168.178.2 -static` would force `mpathd` to use this system as a target system.

There are two ways to make those routes boot persistant. You could use the `route -p` command as described on page 235 in the section 22. But's thats boring. Let's remember the stuff we learned about transient SMF services in section 5.

A transient SMF service to configure those target systems

So you need a way to ensure that those routes are configured at boot up. You could use a simple legacy ¡code¿init.d¡/code¿ script, but I want to implement it as a configurable transient SMF service. A transient service is a service that is executed once at the start and isn't monitored by the restarter daemon. A transient service is a useful mechanism to do such initial configuration tasks at system start.

The service consists out of a simple manifest. Store it in a file `ipmptargets.xml`:

```
<?xml version="1.0"?>
<!DOCTYPE service_bundle SYSTEM "/usr/share/lib/xml/dtd/service_bundle.dtd.1">
<service_bundle type='manifest' name='ipmptargets'>
<service
        name='network/ipmptargets'
        type='service'
        version='1'>

       <dependency
         name='network'
```

```
  grouping='require_all'
  restart_on='none'
  type='service'>
        <service_fmri value='svc:/milestone/network:default' />
</dependency>

<exec_method
  type='method'
  name='start'
  exec='/lib/svc/method/ipmptargets %m'
  timeout_seconds='60' />

 <exec_method
   type='method'
   name='refresh'
   exec='/lib/svc/method/ipmptargets %m'
   timeout_seconds='60' />

 <exec_method
   type='method'
   name='stop'
   exec='/lib/svc/method/ipmptargets %m'
   timeout_seconds='60' />

 <property_group name='startd' type='framework'>
        <propval
          name='duration' type='astring' value='transient' />
  </property_group>
  <property_group name='general' type='framework'>
          <propval
            name='action_authorization'
            type='astring'
            value='solaris.smf.manage.ipmptargets' />
           <propval name='value_authorization'
             type='astring'
             value='solaris.smf.manage.ipmptargets' />
  </property_group>

 <instance name='target1' enabled='false'>
        <property_group
          name='config_params' type='application'>
                <propval
                name='ip' type='astring' value='192.168.178.1'/>
        </property_group>
 </instance>

 <instance name='target2' enabled='false'>
        <property_group
          name='config_params' type='application'>
                <propval
                  name='ip' type='astring' value='192.168.178.20'/>
        </property_group>
 </instance>

 <stability value='Unstable' />
 <template>
        <common_name>
                <loctext xml:lang='C'>
                        system-wide configuration of IP routes for IPMP
                </loctext>
        </common_name>
        <documentation>
                <manpage
```

```
                                title='ifconfig'
                                section='1M'
                                manpath='/usr/share/man' />
                </documentation>
        </template>
</service>
</service_bundle>
```

The specific host routes are implemented as instances of this service. So it is possible to control the routes with a fine granularity.

Okay, obviously we need the script mentioned in the exec methods of the manifest. So put the following script into the the file `/lib/svc/method/ipmptargets`:

```
#!/bin/sh

. /lib/svc/share/smf_include.sh

getproparg() {
val=`svcprop -p $1 $SMF_FMRI`
[ -n "$val" ] && echo $val
}

if [ -z "$SMF_FMRI" ]; then
echo "SMF framework variables are not initialized."
exit $SMF_EXIT_ERR_CONFIG
fi

OPENVPNBIN='/usr/sbin/route'
IP=`getproparg config_params/ip`


if [ -z "$IP" ]; then
echo "config_params/ip property not set"
exit $SMF_EXIT_ERR_CONFIG
fi

case "$1" in
'start')
route add -host $IP $IP -static
;;

'stop')
echo "not implemented"
route delete -host $IP $IP -static
;;

'refresh')
 route delete -host $IP $IP -static
 route add -host $IP $IP -static
;;

*)
echo $"Usage: $0 {start|refresh}"
exit 1
;;

esac
exit $SMF_EXIT_OK
```

Okay. Now we have to import the the SMF manifest into the repository.

```
jmoekamp@hivemind:~# svccfg import  ipmp_hostroutes.xml
```

It's ready to use now. You can enable and disable your IPMP host routes as you need them:

```
jmoekamp@hivemind:~# svcadm enable ipmptargets:target1
jmoekamp@hivemind:~# netstat -nr

Routing Table: IPv4
  Destination           Gateway           Flags  Ref     Use     Interface
-------------------- -------------------- ----- ----- ---------- ---------
default              192.168.178.1        UG        23     496909
127.0.0.1            127.0.0.1            UH         2       2796 lo0
192.168.56.0         192.168.56.1         U          2          0 vboxnet0
192.168.178.0        192.168.178.9        U          3          0 production0
192.168.178.1        192.168.178.1        UGH        1          0

Routing Table: IPv6
  Destination/Mask            Gateway                   Flags Ref   Use    If
--------------------------- --------------------------- ----- --- ------- -----
::1                         ::1                         UH      2      20 lo0
jmoekamp@hivemind:~# svcadm disable ipmptargets:target1
jmoekamp@hivemind:~# netstat -nr

Routing Table: IPv4
  Destination           Gateway           Flags  Ref     Use     Interface
-------------------- -------------------- ----- ----- ---------- ---------
default              192.168.178.1        UG        24     496941
127.0.0.1            127.0.0.1            UH         2       2796 lo0
192.168.56.0         192.168.56.1         U          2          0 vboxnet0
192.168.178.0        192.168.178.9        U          3          0 production0

Routing Table: IPv6
  Destination/Mask            Gateway                   Flags Ref   Use    If
--------------------------- --------------------------- ----- --- ------- -----
::1                         ::1                         UH      2      20 lo0
```

21.9.3. Migration of the classic IPMP configuration

Albeit new IPMP should be configured like described above a correct configuration for the classic IPMP will setup the new IPMP correctly as well.

21.9.4. Setting a shorter or longer Failure detection time

When i talked about probe based failure detection, i've told you that you have to wait a certain time until a failure or a repair is detected by the `in.mpathd`. The default is 10 seconds for failure detection. The time for repair detection is always twice the time of the failure detection, so it's 20 seconds per default. But sometimes you want a faster reaction to failures and repairs. You can control the failure detection time by editing `/etc/default/mpathd`. You will find a configuration item controlling this time span in the file

```
#
# Time taken by mpathd to detect a NIC failure in ms. The minimum time
# that can be specified is 100 ms.
#
FAILURE_DETECTION_TIME=10000
```

By using a smaller number, you can speed up the failure detection but you have a much higher load of ICMP probes on you system. Keep in mind that i've told you that if 5 consecutive probes fail, the interfaces is considered as failed. When the failure detection time is 10000 ms, the probes have to be sent every 2000 ms. When you configure 100 ms, you will see a probe every 20ms. Furthermore this probing is done on every interface. Thus at 100ms failure detection time, the targets will see 3 ping requests every 20 milliseconds.

So keep in mind that lowering this number will increase load on other systems. So choose your the failure detection based on your business and application needs, not on the thought "I want the lowest possible time".

Just to demonstrate this effect and to learn how you set the failure detection time, you should modify the value in the line `FAILURE_DETECTION_TIME` to 100. Restart the `in.mpathd` afterwards by sending a HUP signal to it with verb=.pkill -HUP in.mpathd=. When you start a snoop with `snoop -d e1000g0 -t a -r  icmp` you will have an output on your display scrolling at a very high speed.

21.10. Conclusion

The nice thing about IPMP is: It's simply there. You can use it. When you have more than one interface in your system and you care about the availability of your network, it takes you just a few seconds to activate at least the link-based variant of IPMP. This fruit is really hanging just a few centimeters above the ground.

21.11. Do you want to learn more?

Documentation

docs.sun.com - Solaris 10 - System Administration Guide: IP Services - Part VI: IPMP[20]

man pages

docs.sun.com - ifconfig(1M)[21]

[20] `http://docs.sun.com/app/docs/doc/816-4554/ipmptm-1?l=en&a=view`
[21] `http://docs.sun.com/app/docs/doc/816-5166/ifconfig-1m?a=view`

docs.sun.com - in.mpathd(1M)[22]

docs.sun.com - ipmpstat(1M)[23]

Misc.

Project Clearview: IPMP Rearchitecture[24]

[22] http://docs.sun.com/app/docs/doc/816-5166/in.mpathd-1m?l=en&a=view
[23] http://docs.sun.com/app/docs/doc/819-2240/ipmpstat-1m?a=view
[24] http://hub.opensolaris.org/bin/view/Project+clearview/ipmp

22. Boot persistent routes

Solaris 10/Opensolaris

22.1. Introduction

When you work with the interesting features of Solaris, you forget about the small features in commands you use day by day. I used a SMF script In my IPMP tutorial to make some host routes persistent ... at foremost to show you that SMF scripts are really easy. But, well ... this script isn't nescessary because of the `-p` option of the `route add` command.

22.2. Configuration

Using the `-p` command makes the route boot persistent:

```
jmoekamp@hivemind:~# route -p add -host 192.168.2.3 192.168.2.3 -static
add host 192.168.2.3: gateway 192.168.2.3
add persistent host 192.168.2.3: gateway 192.168.2.3
```

At first the command tries to configure the route in the live system. When this step was successful, it makes the route boot persistent.

You can check the routes already made persistent by using the `route -p show`command:

```
jmoekamp@hivemind:~# route -p show
persistent: route add -host 192.168.2.3 192.168.2.3 -static
```

The routes are written into the `/etc/inet/static_routes` file:

```
jmoekamp@hivemind:/etc/inet# cat static_routes
# File generated by route(1M) - do not edit.
-host 192.168.2.3 192.168.2.3 -static
```

Every line that isn't empty or commented will be prepended with `route add` and executed at startup. The code responsible for this task is in the method script `/lib/svc/method/net-routing-se` which is used by the SMF service `svc:/network/routing-setup`.

Deletion of a persitant routes is easy, too. Just use the `route delete` command with a `-p`:

```
jmoekamp@hivemind:/etc/inet# route -p delete  -host 192.168.2.3 192.168.2.3 -static
delete host 192.168.2.3: gateway 192.168.2.3
delete persistent host 192.168.2.3: gateway 192.168.2.3
```

22.3. Do you want to learn more?

man pages

docs.sun.com - ifconfig(1M)[1]

[1] http://docs.sun.com/app/docs/doc/816-5166/ifconfig-1m?a=view

23. kssl - an in-kernel SSL proxy

Solaris 10/Opensolaris

There is an interesting feature in Solaris. It is called kssl. One component of this feature is obvious: SSL. So it has something to do with SSL encryption. As you may have already guessed, the `k` at the beginning stands for kernel. And kssl is exactly this: A proxy to do all the encryption stuff in a kernel module.

This feature is already four years old. I've reported about kssl back in December 2005[1] for the first time. After talking with a customer two days ago and looking at some new material about it[2], i thought it could not harm to write a new tutorial about this topic. So much appearances in such a small timeframe are a hint ;) .

23.1. The reasons for SSL in the kernel

There are several good reasons to to encryption in the kernel.

- The kssl proxy uses the Solaris Cryptographic Framework. Even when the application doesnt support cryptographic libraries with PKCS#11 support, you can use the advantages of the framework like the seamless integration of hardware cryptography
- The application server just see unencrypted traffic. The complete handshake is done in the kernel-space. Its done asynchronously and the application isnt involved in this task.
- You dont have to wake up the webserver for the SSL handshake to process all the messages nescessary for SSL.
- By offloading SSL into this kernel module you can yield a much better performance. Krishna Yenduri states in a presentation: *SPECweb05 banking is the main benchmark. Performance gain of more than 20% compared to a user land SSL web server. SPECweb99_SSL showed gains of more than 35%*

[1] `http://www.c0t0d0s0.org/archives/1092-Links-of-the-week.html`
[2] `http://blogs.sun.com/yenduri/resource/kssl_osol.pdf`

23.2. Configuration

The configuration of an SSL proxy is really easy. At first you need a certifcate and the key. For this experiment we will use a self-signed certificate. Ive called the system `a380`, thus i have to use this name in my certificate. Use the name of your own system in your configuration. Furthermore the kssl system expect key and certificate in a single file. We concatenate both files afterwards:

```
# mkdir  /etc/keys
# cd /etc/keys
# openssl req -x509 -nodes -days 365 -subj "/C=DE/ST=Hamburg/L=
   Hamburg/CN=a380" -newkey rsa:1024 -keyout /etc/keys/mykey.
   pem -out /etc/keys/mycert.pem
# cat mycert.pem mykey.pem > my.pem
# chown 600 *
```

Now we configure the kssl proxy:

```
# echo "pass" > /etc/keys/my.pass
# ksslcfg create -f pem -i /etc/keys/my.pem -x 8080 -p /etc/
   keys/my.pass a380 443
```

At first we create a file to automatically answer the passphrase question. Afterwards we configure the kssl service. This configuration statement tells the system to get the keys and from a pem file. The `-i` option specifies the location of the file. `-p` tells the service where it finds the passphrase file. At the end you find `a380 443`. This specifies on which interface and on which port the ssl should listen. At last the `-x 8080` specifies to what port the the unencrypted traffic should be redirected.

After configuring this service, you should see a new service managed by SMF:

```
# svcs -a | grep "kssl"
online          9:03:33 svc:/network/ssl/proxy:kssl-a380-443
```

Obviously we need a webserver in the backend that listens to port 8080. I assume, that you've already installed an Apache on your server. We just add a single line to the configuration file of the webserver.

```
# svcadm disable apache22
# echo "Listen 192.168.178.108:8080" >> /etc/apache2/2.2/httpd.
   conf
# svcadm enable apache22
```

When you put `https://a380:443` in your browser, you should see an encrypted "It works" page after you dismissed the dialog warning you of the self-signed certificate. Or to show it to you on the command line:

```
# openssl s_client -connect 192.168.178.108:443
CONNECTED(00000004)
depth=0 /C=DE/ST=Hamburg/L=Hamburg/CN=a380
verify error:num=18:self signed certificate
verify return:1
depth=0 /C=DE/ST=Hamburg/L=Hamburg/CN=a380
verify return:1
---
Certificate chain
 0 s:/C=DE/ST=Hamburg/L=Hamburg/CN=a380
   i:/C=DE/ST=Hamburg/L=Hamburg/CN=a380
---
Server certificate
-----BEGIN CERTIFICATE-----
MIICoTCCAgqgAwIBAgIJAKyJdj/
[...]
V5jX3MU=
-----END CERTIFICATE-----
subject=/C=DE/ST=Hamburg/L=Hamburg/CN=a380
issuer=/C=DE/ST=Hamburg/L=Hamburg/CN=a380
---
No client certificate CA names sent
---
SSL handshake has read 817 bytes and written 328 bytes
---
New, TLSv1/SSLv3, Cipher is RC4-SHA
Server public key is 1024 bit
Compression: NONE
Expansion: NONE
SSL-Session:
    Protocol  : TLSv1
    Cipher    : RC4-SHA
    Session-ID: 32
       CEF20CB9FE2A71C74D40BB2DB5CB304DA1B57540B7CFDD113915B99DBE9812

    Session-ID-ctx:
    Master-Key: 1
       E7B50239095112477 9C5763B5E4BBAF0A9B0693D08DCA8A587B503A5C5027B6FAD9C

    Key-Arg   : None
    Start Time: 1242985143
    Timeout   : 300 (sec)
    Verify return code: 18 (self signed certificate)
---
```

```
GET / HTTP/1.0

HTTP/1.1 200 OK
Date: Fri, 22 May 2009 09:39:13 GMT
Server: Apache/2.2.11 (Unix) mod_ssl/2.2.11 OpenSSL/0.9.8a DAV
   /2
Last-Modified: Thu, 21 May 2009 21:26:30 GMT
ETag: "341f3-2c-46a72cc211a8f"
Accept-Ranges: bytes
Content-Length: 44
Connection: close
Content-Type: text/html

<html><body><h1>It works!</h1></body></html>
read:errno=0
```

Voila, web server encrypted without a single line of SSL configuration in the webserver config files itself.

23.3. Conclusion

Its really easy to add an kssl proxy in front of your webserver. So it isn't difficult to make encrypted webserving more efficient.

23.4. Do you want to learn more?

man pages

ksslcfg(1m) [3]

Other

CZOSUG: KSSL - Solaris Kernel SSL proxy - presentation about KSSL.[4]

KSSL A Kernel Level SSL Server Proxy - a deep dive presentation about KSSL.[5]

[3] http://docs.sun.com/app/docs/doc/816-5166/ksslcfg-1m
[4] http://opensolaris.org/os/project/czosug/events_archive/czosug25_kssl.pdf
[5] http://blogs.sun.com/yenduri/resource/kssl_osol.pdf

Part V.

Storage

24. fssnap - snapshots for UFS

Solaris 10/Opensolaris

An ever recurring problem while doing backups is the problem, that you have to keep the state of the backup consistent. For example: You use the cyrus imapd for storing mails on a system. As it's a system used for years, the message store is still on an UFS file system. Okay, now there is a little problem: You want to make backups, but it's a mail server.

With the amount of mail junkies in modern times, you can't take the system offline for an hour or so, just to make a backup. But you have to take it offline. Especially a mail server is a moving target, as cyrus imapd has indexes for its mailboxes, and index files for the mailboxes itself. Let's assume a backup takes 1 hour, and users delete mails in this time, create mailboxes and so on. It's possible that you backup your mailserver in an inconsistent state, that the mail directory of the user may represent the state one hour ago, but the mailboxes file represent the state of one minute ago.

24.1. fssnap

UFS has a little known feature, that comes to help in such a situation. You can do a filesystem snapshot of the system. This is a non-changing point-in-time view to the filesystem, while you can still change the original filesystem. `fssnap` is a rather old tool. We've introduced in the 1/01 update of Solaris 8.

There is a restriction with this snapshots: This snapshots are solely for the purpose of backups, thus they are not boot persistent. For boot persistent snapshots of a filesystem you will need the Sun Availability Suite.

24.2. A practical example.

Let's assume that we have a directory called `/mailserver`. This directory is the mount-point for a UFS filesystem on `/dev/dsk/c1d1s1`. At the moment it contains some files:

```
# ls -ls
total 10
   2 -rw------T   1 root     root         1024 Apr 23 01:41
      testfile1
   2 -rw------T   1 root     root         1024 Apr 23 01:41
      testfile2
   2 -rw------T   1 root     root         1024 Apr 23 01:41
      testfile3
   2 -rw------T   1 root     root         1024 Apr 23 01:41
      testfile4
   2 -rw------T   1 root     root         1024 Apr 23 01:41
      testindex1
```

Now we want to make a backup. It's sensible to take the mail server offline for a few seconds to keep the files consistent. In this moment we make the filesystem snapshot. This is really easy:

```
# fssnap -o bs=/tmp /mailserver
/dev/fssnap/0
```

With this command we told fssnap to take an snapshot of the filesystem mounted at `/mailserver`. Furthermore we configured that the snapshot uses the `/tmp` for its backing store. In the backing store the changes since the snapshot will be recorded. When `fssnap` is able to create the snapshot, it will return the name for the pseudo device containing the filesystem snapshot. In our case it's `/dev/fssnap/0`. Please remember it, we need it later.

When you look at the /tmp directory you will find an backing store file for this snapshot. It's called `snapshot0` for the first snapshot on the system:

```
# ls -l /tmp
total 910120
-rw-r--r--   1 root     root           30 Apr 22 14:50
   ogl_select305
-rw-------   1 root     root     465976320 Apr 23 01:47
   snapshot0
```

Now we bring the mailserver online again, and after a few seconds we see changes to the filesystem again (okay, in my example I will do this manually):

```
# mkfile 1k testfile5
# mkfile 1k testfile6
# mkfile 2k testindex1
# ls -l
total 16
-rw------T   1 root     root         1024 Apr 23 01:41 testfile1
```

```
-rw------T   1 root     root         1024 Apr 23 01:41 testfile2
-rw------T   1 root     root         1024 Apr 23 01:41 testfile3
-rw------T   1 root     root         1024 Apr 23 01:41 testfile4
-rw------T   1 root     root         1024 Apr 23 01:53 testfile5
-rw------T   1 root     root         1024 Apr 23 01:53 testfile6
-rw------T   1 root     root         2048 Apr 23 01:53
   testindex1
```

Now we want to make the backup itself. At first we have to mount the filesystem. Thus we create a mountpoint

```
# mkdir /mailserver_forbackup
```

Now we mount the snapshot. You will recognize the pseudo device here again. The snapshot is read only thus you have to specify it at mount time:

```
# mount -o ro /dev/fssnap/0 /mailserver_forbackup
```

Okay, when we look into the filesystem, we will see the state of the filesystem at the moment of the snapshot. `testfile5`, `testfile6` and the bigger `testindex1` are missing.

```
# ls -l
total 10
-rw------T   1 root     root         1024 Apr 23 01:41 testfile1
-rw------T   1 root     root         1024 Apr 23 01:41 testfile2
-rw------T   1 root     root         1024 Apr 23 01:41 testfile3
-rw------T   1 root     root         1024 Apr 23 01:41 testfile4
-rw------T   1 root     root         1024 Apr 23 01:41
   testindex1
```

Now we can to a backup without any problems and in a consistent manner:

```
# tar cfv /backup_mailserver_20080424.tar *
a testfile1 1K
a testfile2 1K
a testfile3 1K
a testfile4 1K
a testindex1 1K
```

After this step we should clean-up. We unmount the snapshot and delete the snapshot with its backing store file:

```
# umount /mailserver_forbackup
# fssnap -d /mailserver
Deleted snapshot 0.
# rm /tmp/snapshot0
```

24.3. Conclusion

With the `fssnap` command you have an easy way to do consistent backups on UFS filesystems. While it's not as powerful as the functions of the point-in-time copy functionality in the Availability Suite, it's a perfect match for its job.

24.4. Do you want to learn more?

fssnap

fssnap_ufs

25. Legacy userland iSCSI Target

Solaris 10/Opensolaris

25.1. Introduction

With more and more available bandwidth it gets more and more feasible to use only a single network to transmit data in the datacenter. Networking can get really complex when you have to implement two networks. For example for Ethernet and one for FC. It's getting more complex and expensive, as FC network are mostly optical ones and Ethernet is a copper network in most datacenters.

The idea of using a single network for both isn't really a new one, thus there were several attempts to develop mechanisms to allow the usage of remote block devices. One of the protocols to use block devices over IP networks is iSCSI. iSCSI is a protocol that was designed to allow clients to send SCSI commands to SCSI storage devices on a remote server.

There are other technologies like ATA or FC over Ethernet that were designed , but iSCSI is the most interesting one from my point of view, as you can use already existing and proven methods implemented for IP (for example IPsec) for encrypting the traffic.

The problem with iSCSI is the latency introduced with the usage of IP. But this problem can be solved by the usage of iSER or "iSCSI Extensions for RDMA" to use iSCSI as part of single-fabric network based on Infiniband. We already implement this mechanism in Solaris as a part of the COMSTAR iSER[1] project.

25.2. The jargon of iSCSI

Yet another technology with its own jargon. I will only define the most important:

- *Initiator*: The initiator is the client at iSCSI. It's like the SCSI card in your machine, but instead of a SCSI cable it uses IP for the SCSI commands.
- *Target*: The target is a storage resource on a iSCSI server.

[1] `http://opensolaris.org/os/project/iser/`

- *LUN*: A LUN is the "logical unit number". It represents an individual SCSI device. In iSCSI it's similar. When you want to use an iSCSI disk drive, the initiator connects the target and connects the initiator with the LUN in a iSCSI session.
- *IQN*: To identify a target or an initiator, one of the naming schemes is the iSCSI Qualified Name (IQN). An example for an IQN is iqn.1986-03.com.sun:02:b29a71fb-ff60-c2c6-c92f-ff13555977e6

25.3. The architecture of iSCSI

The combination of all this components can be visualized like in this figure:

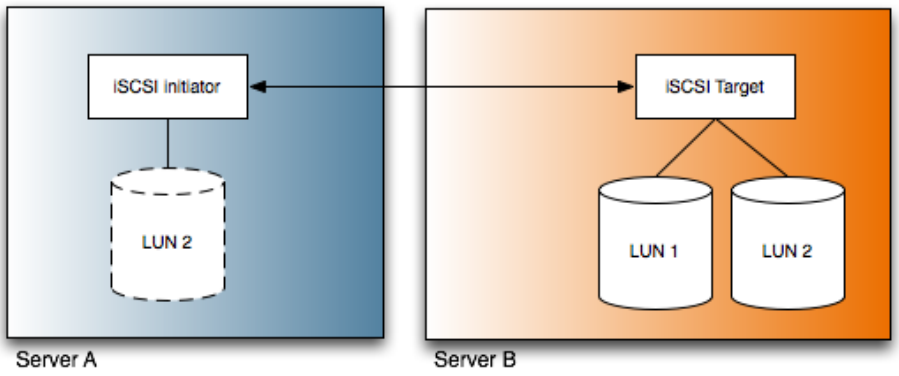

Figure 25.1.: The components of iSCSI

It's a little bit simplified, but this is the core idea of iSCSI: How do I present a LUN on a remote server to another server via an IP network.

25.4. Simple iSCSI

At first I will show you, how to do a really simple iSCSI configuration. This is without any authentication. Thus everybody can connect to the iSCSI targets on the system. But this is sufficient for a test. And it's this is better for having a quick success. ;)

25.4.1. Environment

For this example, I will use my both demo VMs again:

```
10.211.55.200    theoden
10.211.55.201    gandalf
```

Both systems runs with Solaris Express Build 84 for x86, but you can to the same with Solaris Update 4 for SPARC and x86 as well.

In our example, theoden is the server with the iSCSI target. gandalf is the server, which wants to use the LUN via iSCSI on theoden, thus gandalf is the server with the initiator.

25.4.2. Prerequisites

At first, we login to theoden and assume root privileges. Okay, to test iSCSI we need some storage volumes to play around. There is a nice way to create a playground with ZFS. You can use files as devices. But at first we have to create this files

```
# mkdir /zfstest
# cd /zfstest
#  mkfile 128m test1
#  mkfile 128m test2
#  mkfile 128m test3
#  mkfile 128m test4
```

Okay, now we stripe those four files in a zpool:

```
# zpool create testpool /zfstest/test1 /zfstest/test2 /zfstest/
   test3 /zfstest/test4
```

Now we make a short check for the zpool

```
# zpool list
NAME       SIZE   USED  AVAIL    CAP  HEALTH  ALTROOT
testpool   492M    97K   492M     0%  ONLINE  -
```

25.4.3. Configuring the iSCSI Target

We stay at server theoden. Okay, now we have to configure an iSCSI target. We create an emulated volume within the ZFS pool:

```
# zfs create -V 200m testpool/zfsvolume
# zfs list
NAME                  USED  AVAIL  REFER  MOUNTPOINT
testpool              200M   260M    18K  /testpool
testpool/zfsvolume    200M   460M    16K  -
```

The emulated volume has the size of 200M.Okay, it's really easy to enable the iSCSI target. At first we have to enable the iSCSI Target service:

```
# svcadm enable iscsitgt
```

Now we share the volume via iSCSI

```
# zfs set shareiscsi=on testpool/zfsvolume
```

That's all on the target

25.4.4. Configuring the iSCSI initiator

Okay, now we have configure the initiator. We have to login on `gandalf` and assume root privileges as well. At first we have to activate the initiator via SMF:

```
# svcadm enable iscsi_initiator
```

After this we configure the initiator and tell the initiator to discover devices on our iSCSI target.

```
# iscsiadm modify initiator-node -A gandalf
# iscsiadm add discovery-address 10.211.55.200
# iscsiadm modify discovery -t enable
```

25.4.5. Using the iSCSI device

Okay, now tell Solaris to scan for iSCSI devices.

```
# devfsadm -c iscsi
```

The `-c iscsi` limits the scan to iSCSI devices. With the format command we look for the available disks in the system:

```
# format
Searching for disks...done


AVAILABLE DISK SELECTIONS:
       0. c0d0 <DEFAULT cyl 4076 alt 2 hd 255 sec 63>
          /pci@0,0/pci-ide@1f,1/ide@0/cmdk@0,0
       1. c1d0 <DEFAULT cyl 4077 alt 2 hd 255 sec 63>
          /pci@0,0/pci-ide@1f,1/ide@1/cmdk@0,0
       2. c2t0100001C42E9F21A00002A0047E39E34d0 <DEFAULT cyl
          198 alt 2 hd 64 sec 32>
          /scsi_vhci/disk@g0100001c42e9f21a00002a0047e39e34
Specify disk (enter its number): ^C
```

Okay, there is new device with a really long name. We can use this device for a zfs pool:

```
# zpool create zfsviaiscsi
   c2t0100001C42E9F21A00002A0047E39E34d0
# zpool list
NAME          SIZE   USED  AVAIL    CAP  HEALTH  ALTROOT
zfsviaiscsi   187M   480K   187M     0%  ONLINE  -
#
```

As you see, we have created a zfs filesystem via iSCSI on an emulated volume on a zpool on a remote system.

25.5. Bidirectional authenticated iSCSI

Okay, but we are in a public network. How can we protect the target against a user not allowed to use it? iSCSI supports a CHAP[2] based authentication. It's the same scheme as with PPP. With this authentication, target and initiator can authenticate each other based on shared secrets. The configuration is a little bit more complex, as both initiator and target have to know the secrets and their names of each other.

25.5.1. Prerequisites

At first we need some basic data. We need the IQN names of both. At first we look up the IQN of the initiator. Thus we login to `gandalf` and assume root privileges:

[2] `http://en.wikipedia.org/wiki/Challenge-handshake_authentication_protocol`

```
# iscsiadm list initiator-node
Initiator node name: iqn.1986-03.com.sun:01:00000000b89a.47
   e38163
Initiator node alias: gandalf
[...]
```

Now we look up the IQN of the target. Okay, we login to `theoden` and assume root privileges:

```
# iscsitadm list target
Target: testpool/zfsvolume
    iSCSI Name: iqn.1986-03.com.sun:02:b29a71fb-ff60-c2c6-c92f-
       ff13555977e6
iqn.1986-03.com.sun:02:b29a71fb-ff60-c2c6-c92f-ff13555977e6
    Connections: 1
```

Both IQNs are important in the following steps. We need them as a identifier for the systems.

25.5.2. Configuring the initiator

At first we export the zpool and disable the discovery of targets. This steps has to be done on the initiator, in our example `gandalf`:

```
# zpool export zfsviaiscsi
# iscsiadm remove discovery-address 10.211.55.200
# devfsadm -c iscsi -C
```

You don't have to do this steps. The zpool may only get unavailable while we configure the authentication and you will see a few more lines in your logfiles.

Okay, now we configure the CHAP authentication.

```
# iscsiadm modify initiator-node --CHAP-name gandalf
# iscsiadm modify initiator-node --CHAP-secret
Enter secret:
Re-enter secret:
# iscsiadm modify initiator-node --authentication CHAP
```

What have we done with this statements: We told the iSCSI initiator to identify itself as `gandalf`. Then we set the password and tell the initiator to use CHAP to authenticate.

25.5.3. Configuring the target

Okay, now we configure the target to use CHAP as well. This has to be done on the target, in our example `theoden`. But at first we have to set the CHAP name and the CHAP secret of the target itself:

```
# iscsitadm modify admin --chap-name theoden
# iscsitadm modify admin --chap-secret
Enter secret:
Re-enter secret:
```

This isn't an admin login. This is a little misguiding.

Now we create an initiator object on the target.We connect the long IQN with a shorter name.

```
# iscsitadm create initiator --iqn iqn.1986-03.com.sun
   :01:00000000b89a.47e38163 gandalf
```

Now we tell the target, that the initiator on the system `gandalf`will identify itself with the name `gandalf`:

```
# iscsitadm modify initiator --chap-name gandalf gandalf
```

Okay, now we set the password for this initiator. This is the same password we set on the initiator.

```
# iscsitadm modify initiator --chap-secret gandalf
Enter secret:
Re-enter secret:
```

Finally we tell the target, that the system gandalf is allowed to access the `testpool/zfsvolume`:

```
# iscsitadm modify target --acl gandalf test/zfsvolume
```

Now the initiator has to authenticate itself before the target daemon grants access to the target. You could skip the next steps and fast-forward to the section "Reactivation of the zpool" but the authentication is only unidirectional at the moment. The client(initiator) authenticate itself at the server(target).

25.5.4. Configuration of bidirectional configuration

Okay, but it would be nice, that the target identifies himself to initiator as well. Okay, at first we tell the initiator, that the target with the IQN `iqn.1986-03.com.sun:02:b29a71fb-ff60-c2c6-` will authenticate itself with the name theoden. This steps has to be done on the initiator, thus we login into `gandalf` again.

```
# iscsiadm modify target-param --CHAP-name theoden iqn.1986-03.
   com.sun:02:b29a71fb-ff60-c2c6-c92f-ff13555977e6
```

Now we set the secret to authenticate. This is the secret we configured as the CHAP-Secret on the target with `iscsitadm modify admin --chap-secret`:

```
# iscsiadm modify target-param --CHAP-secret iqn.1986-03.com.
   sun:02:b29a71fb-ff60-c2c6-c92f-ff13555977e6
Enter secret:
Re-enter secret:
```

Now we activate the bidirectional authentication for the IQN of theoden:

```
# iscsiadm modify target-param --bi-directional-authentication
   enable iqn.1986-03.com.sun:02:b29a71fb-ff60-c2c6-c92f-
   ff13555977e6
```

At last we tell the initiator to authenticate the target with CHAP.

```
# iscsiadm modify target-param -a chap iqn.1986-03.com.sun:02:
   b29a71fb-ff60-c2c6-c92f-ff13555977e6
```

Okay, now we have completed the configuration for the bidirectional authentication.

25.5.5. Reactivation of the zpool

Okay ... now we can reactivate the zpool. We tell the initiator to discover targets on the target server again, scan for devices and import the zpool again:

```
# iscsiadm add discovery-address 10.211.55.200
# devfsadm -c iscsi -C
# zpool import zfsviaiscsi
# cd /zfsviaiscsi/
```

25.6. Alternative backing stores for iSCSI volumes

Albeit very easy to use iSCSI in conjunction with ZFS emulated volumes it doesn't have to be this way. You can use different kinds of backing store for iSCSI.

25.6.1. File based iSCSI target

One way to provide for storage for an iSCSI target. You can use files for this task. Okay, we have to login as root on `theoden`:

```
# mkdir -p /var/iscsitargets
# iscsitadm modify admin -d /var/iscsitargets
```

At first we've created an directory to keep the files, then we tell the target daemon to use this for storing the target. After this we can create the target:

```
# iscsitadm create target --size 128m smalltarget
```

Now we can check for the iSCSI Target.

```
# iscsitadm list target -v smalltarget
Target: smalltarget
    iSCSI Name: iqn.1986-03.com.sun:02:3898c6a7-1f43-e298-e189
       -83d10f88131d.smalltarget
    Connections: 0
    ACL list:
    TPGT list:
    LUN information:
        LUN: 0
            GUID: 0100001c42e9f21a00002a0047e45145
            VID: SUN
            PID: SOLARIS
            Type: disk
            Size:  128M
            Status: offline
```

Now we switch to the server we use as an initiator. Let's scan for new devices on `gandalf`. As we've activated the discovery of targets before, we've just have to scan for new devices.

```
# devfsadm -c iscsi -C
# format
Searching for disks...done


AVAILABLE DISK SELECTIONS:
       0. c0d0 <DEFAULT cyl 4076 alt 2 hd 255 sec 63>
          /pci@0,0/pci-ide@1f,1/ide@0/cmdk@0,0
       1. c1d0 <DEFAULT cyl 4077 alt 2 hd 255 sec 63>
          /pci@0,0/pci-ide@1f,1/ide@1/cmdk@0,0
```

```
        2. c2t0100001C42E9F21A00002A0047E39E34d0 <SUN-SOLARIS
           -1-200.00MB>
           /scsi_vhci/disk@g0100001c42e9f21a00002a0047e39e34
        3. c2t0100001C42E9F21A00002A0047E45145d0 <DEFAULT cyl
           126 alt 2 hd 64 sec 32>
           /scsi_vhci/disk@g0100001c42e9f21a00002a0047e45145
Specify disk (enter its number):
Specify disk (enter its number): ^C
#
```

Et voila, an additional LUN is available on our initiator.

25.6.2. Thin-provisioned target backing store

One nice thing about ZFS is it's ability to provide thin provisioned emulated ZFS volumes (zvol). You can configure a volume of an larger size than the physical storage you have available. This is useful, when you want to have an volume in it's final size (because resizing would be a pain in the a...) but don't want do spend the money for the disks because you know that much less storage would be needed at first.

It's really easy to create such a kind of a zvol:

```
# zfs create -s -V 2g testpool/bigvolume
```

That's all. The difference is the small `-s`. It tells ZFS to create an sparse (aka thin) provisioned volume.

Well, I won't enable iSCSI for this by `shareiscsi=on` itself. I will configure this manually. As normal volumes zvols are available within the `/dev` tree of your filesystem:

```
# ls -l /dev/zvol/dsk/testpool
total 4
lrwxrwxrwx   1 root     root          35 Mar 22 02:09 bigvolume
    -> ../../../../devices/pseudo/zfs@0:2c
lrwxrwxrwx   1 root     root          35 Mar 21 12:33 zfsvolume
    -> ../../../../devices/pseudo/zfs@0:1c
```

Okay, we can use this devices as a backing store for an iSCSI target as well. We've created a zvol `bigvolume` within the zpool `testpool`. Thus the device is `/dev/zvol/dsk/testpool/bigvolume`:

```
# iscsitadm create target -b /dev/zvol/dsk/testpool/bigvolume
   bigtarget
```

Okay, im switching to my root shell on the initiator. Again we scan for devices:

```
# devfsadm -c iscsi -C
# format
Searching for disks...done


AVAILABLE DISK SELECTIONS:
       0. c0d0 <DEFAULT cyl 4076 alt 2 hd 255 sec 63>
          /pci@0,0/pci-ide@1f,1/ide@0/cmdk@0,0
       1. c1d0 <DEFAULT cyl 4077 alt 2 hd 255 sec 63>
          /pci@0,0/pci-ide@1f,1/ide@1/cmdk@0,0
       2. c2t0100001C42E9F21A00002A0047E39E34d0 <SUN-SOLARIS
          -1-200.00MB>
          /scsi_vhci/disk@g0100001c42e9f21a00002a0047e39e34
       3. c2t0100001C42E9F21A00002A0047E45DB2d0 <DEFAULT cyl
          1021 alt 2 hd 128 sec 32>
          /scsi_vhci/disk@g0100001c42e9f21a00002a0047e45db2
```

Let's create an zpool:

```
# zpool create zfsviaiscsi_thin
   c2t0100001C42E9F21A00002A0047E45DB2d0
# zpool list
NAME               SIZE   USED  AVAIL    CAP  HEALTH  ALTROOT
zfsviaiscsi        187M   112K   187M     0%  ONLINE  -
zfsviaiscsi_thin  1.98G   374K  1.98G     0%  ONLINE  -
```

Do you remember, that we used four 128 MB files as the devices for our zpool on our target. Well, you have an 1.98G filesystem running on this files. You can add more storage to the zpool on the target and you have nothing to do on the initiator. Not a real kicker for ZFS, but imagine the same for other filesystem that can't be grown so easy like a zpool.

25.7. Conclusion

Okay, this was a quick introduction to the actual implementation of iSCSI on Solaris. The future will bring changes to the implementation of the iSCSI target feature but new possibilities as well. iSCSI will be an part of the COMSTAR framework[3] in the future besides of an SAS target or an FC target.

[3] http://opensolaris.org/os/comstar

25.8. Do you want to learn more?

Documentation

docs.sun.com: Configuring Solaris iSCSI Targets and Initiators[4]

Using iSCSI Multipathing in the Solaris(tm) 10 Operating System[5]

Misc links.

prefetch.net: Getting to know the Solaris iSCSI stack[6]

[4] http://docs.sun.com/app/docs/doc/819-2723/fmvcd?l=en&q=iscsi&a=view
[5] http://www.sun.com/blueprints/1205/819-3730.pdf
[6] http://prefetch.net/articles/solarisiscsi.html

26. COMSTAR iSCSI Target

Opensolaris

A while ago, i wrote a tutorial about using iSCSI in Opensolaris and Solaris 10. While the tutorial is still valid for Solaris 10, Opensolaris got a new, more efficient iSCSI Target. iSCSI in Opensolaris is a part of a more generic in-kernel framework to provide SCSI Target services. This framework isn't just capable to deliver iSCSI, you can have FCoE, SRP, FC targets as well. This generic approach led to a different administrative model. Thus when you want to configure iSCSI on a OpenSolaris system and you want the new framework (the old userland based version is still available) you have to configure the target side differently. This tutorial will do pretty much the same than the old iSCSI tutorial, just with the new framework to give you an overview about the changes.

26.1. Why does COMSTAR need a different administrative model?

The reason for the this change is somewhat based in the point for it's name: It's a COmmon Multiprotocol Scsi TARget framework. iSCSI, SRP, iSER, SAS, FCoE, FC have something in common. They use SCSI commands. SCSI isn't just a physical interface. It's a way to transmit data over an interface. It's a set of commands. And independently from the physical interface technology you use, the commands are pretty much the same. A good sign for this commonality is the same jargon in all this technologies: The entity executing SCSI command is called target in Fibrechannel jargon as well in iSCSI as well in SCSI over RDMA over Infiniband. The same is valid for the entity issuing commands to a target. It's called initiator in all those protocols as well.

The differences between all the proctols mentioned above is the way, how the SCSI commands are transmitted over the wire. This led to an obvious idea: Separating the layer providing the SCSI target from the layer providing the protcol to access the SCSI target. The component providing this protocols are called "Port Provider" in COMSTAR.

On the other side, COMSTAR isn't just meant to provide access to SCSI disks. Providing tape devices is in the focus as well. The component providing disk or tape devices is

called "logical unit provider". The logical unit is the device within a target responsible for executing SCSI I/O commands. Perhaps you've heard about the acronym LUN in the past. LUN is the Logical Unit Number, a number designating a certain logical unit.

Between the both components is the `stmf`,the SCSI target mode framework. It connects the port provider and the logical unit provider.

What's the advantage: You just have to develop the SCSI target once and for new protocols you just have to develop a relatively simple port provider. This is one of the reasons why we see new port providers quite frequently. In essence this is the reason why the administrative model has changed with the introduction of COMSTAR: The logical unit, the port and the logic to combine both are separate entities thus you configure them separately.

26.2. Prerequisites

Okay, let's look how you configure this stuff. I will use two systems in this example:

```
192.168.56.101 initiator
192.168.56.102 target
```

The system `target` has to use a recent of OpenSolaris. You can use OpenSolaris as well as Opensolaris Community Edition. My testbed used OpenSolaris Developer Build 127, but it should work with any reasonable recent version. On the system `initiator` any Solaris system will do it, as COMSTAR is a change on the target side, it doesn't change anything on the initiator part.

26.3. Preparing the target system

The COMSTAR framework isn't installed per default on OpenSolaris, thus you have to install it on your own. But there is an easy way to install all the components to make a storage server out of your installation. Doing so installs much more than you need for iSCSI, but on the other side it installs all other components, when you want to make a CIFS/iSCSI/NFS/MDNP/etc-capable storage server on your system.

```
jmoekamp@target:~# pkg install storage-server
DOWNLOAD                                  PKGS       FILES    XFER (MB)
Completed                                 24/24   1026/1026    34.3/34.3

PHASE                                       ACTIONS
Install Phase                              2205/2205
```

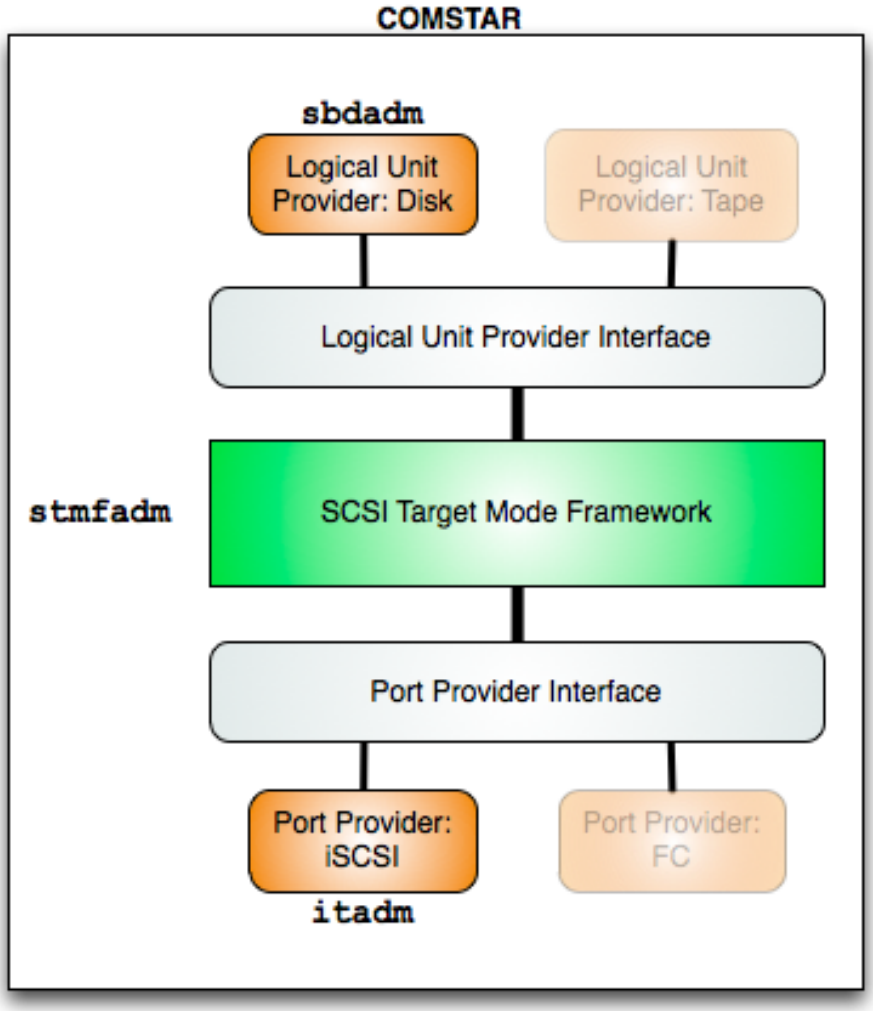

Figure 26.1.: Layering model of COMSTAR iSCSI

Please reboot the system after this step. This framework has many connections to the rest of the system and it's just easier to reboot after the initial install.

Okay, right after the boot the service `stmf` is disabled.

```
jmoekamp@target:~# svcs stmf
STATE          STIME    FMRI
disabled       13:44:42 svc:/system/stmf:default
```

We need this service, so we enable it.

```
jmoekamp@target:~# svcadm enable stmf
jmoekamp@target:~# svcs stmf
STATE          STIME    FMRI
online         13:48:07 svc:/system/stmf:default
```

Now we can check the successful startup of the `stmf` service:

```
jmoekamp@target:~# stmfadm list-state
Operational Status: online
Config Status     : initialized
ALUA Status       : disabled
ALUA Node         : 0
```

Obviously we need a backing store for our iSCSI target. In this case i will use a spare provisioned 10 Terabyte emulated ZFS volume.

```
jmoekamp@target:~# zfs  create -V 10T -s rpool/iscsivol
```

There are other options besides an emulated ZFS volume like a pregenerated file (`mkfile 10T backingstore`), a file that grows from zero to a certain preconfigured file size (`touch backingstore`) and finally just mapping an physical device through the system.

Now we have to configure a logical unit provider in the COMSTAR framework to use our backing store. We want a disk, so we use the disk logical unit provider. We have to use the `sbdadm` command for this task.

```
jmoekamp@target:~# sbdadm create-lu /dev/zvol/rdsk/rpool/iscsivol
Created the following LU:

              GUID                    DATA SIZE           SOURCE
--------------------------------  -------------------  ----------------
600144f0b152cc0000004b080f230004  10995116277760       /dev/zvol/rdsk/rpool/iscsivol
```

Let's do a short check, if everything went well.

```
jmoekamp@target:~# sbdadm list-lu

Found 1 LU(s)

              GUID                    DATA SIZE           SOURCE
--------------------------------  -------------------  ----------------
600144f0b152cc0000004b080f230004  10995116277760       /dev/zvol/rdsk/rpool/iscsivol
```

With the `stmfadm` command you can get some additional insight to your newly created logical unit.

```
jmoekamp@target:~# stmfadm list-lu -v
LU Name: 600144F0B152CC0000004B080F230004
    Operational Status: Online
    Provider Name     : sbd
    Alias             : /dev/zvol/rdsk/rpool/iscsivol
    View Entry Count  : 1
    Data File         : /dev/zvol/rdsk/rpool/iscsivol
    Meta File         : not set
    Size              : 10995116277760
    Block Size        : 512
    Management URL    : not set
    Vendor ID         : SUN
    Product ID        : COMSTAR
    Serial Num        : not set
    Write Protect     : Disabled
    Writeback Cache   : Enabled
    Access State      : Active
```

But at the moment your brand new LU isn't visible. You have to add it to an entity called view. In a view you can define which targets a certain initiator can see. For example you can configure with this views, that all your SAP systems just see the SAP logical units and all MS Exchange Servers just see the Exchange logical units on your storage service. Other logical units aren't visible to the system and they are totally unaware of their existence. For people with a storage background: This is somewhat similar to the LUN masking part ;)

But for this tutorial i will use a simple configuration. I will not impose any access control to this logical unit :

```
jmoekamp@target:~# stmfadm add-view 600144F0B152CC0000004B080F230004
```

Let's check the configuration:

```
jmoekamp@target:~# stmfadm list-view -l 600144F0B152CC0000004B080F230004
View Entry: 0
    Host group   : All
    Target group : All
    LUN          : 1
```

However, the logical unit is still not visible to the outside, as we haven't configured a port provider. The port provider is the component that provides access to the logical unit via iSCSI or FC for example.

26.4. Configuring an iSCSI target

So let's start with the configuration of the iSCSI target. This is the first step where the commands have something to do with iSCSI: `itadm` is the tool to configure an iSCSI target. At first we configure a target portal group. A target portal is an IP and a

portnumber you use to access an iSCSI target. Target portal groups are used to bundle such portals to ease the configuration of access controls.

```
# itadm create-tpg e1000g0 192.168.56.102
```

Okay, now we configure an iSCSI target entity on this portal group:

```
# itadm create-target -t e1000g0
Target iqn.1986-03.com.sun:02:fd9d53cd-6fe5-ecd7-de40-cb3bbb03b422 successfully
    created.
```

This IQN uniquely identifies the target in a network. We will use it from now on when we address this iSCSI target.

That's all. For unauthenticated iSCSI this is all you have to do on the target side: We've configured the logical unit provider and enabled a port provider to give other system access to the target with iSCSI.

26.5. Configuring the initator without authentication

Okay ... now we log into the initiator. Just a quick check ...

```
jmoekamp@inititator:~# format
Searching for disks...done


AVAILABLE DISK SELECTIONS:
       0. c7d0 <DEFAULT cyl 2085 alt 2 hd 255 sec 63>
          /pci@0,0/pci-ide@1,1/ide@0/cmdk@0,0
Specify disk (enter its number): ^C
```

Just our boot disk. At first we have to install the iSCSI initator. This can be done by `pkg install iscsi`.After this step you have to reboot the system.

```
jmoekamp@inititator:~# svcs -a | grep "iscsi"
online         14:08:34 svc:/network/iscsi/initiator:default
```

Okay, the iSCSI service runs on the host `initiator`. Now we have to configure the initiator to discover possible logical units on our target:

```
jmoekamp@inititator:~# iscsiadm add discovery-address 192.168.56.102:3260
jmoekamp@inititator:~# iscsiadm modify discovery --sendtargets enable
```

At first we told the iSCSI initiator to discover logical units at the target portal on `192.168.56.102:3260`. After this step, we configured the iSCSI initiator to use the SendTargets method to discover logical units on the other side. The SendTarget command is a command the initiator sends to the configured hosts to gather all targets available to the initiator issuing the SendTarget command. Every iSCSI target implementation has to support the SendTarget command. This is specified by Appendix D of the RFC 3720.

This makes the configuration of the iSCSI easier, as you don't have to configure all the targets manually. Okay, let's check if the configuration of the discovery was successful:

```
jmoekamp@inititator:~# iscsiadm list discovery
Discovery:
        Static: disabled
        Send Targets: enabled
        iSNS: disabled
```

Now let's look, what targets were discovered by the iSCSI initiator:

```
jmoekamp@inititator:~# iscsiadm list target
Target: iqn.1986-03.com.sun:02:fd9d53cd-6fe5-ecd7-de40-cb3bbb03b422
        Alias: -
        TPGT: 2
        ISID: 4000002a0000
        Connections: 1
```

The iSCSI target IQN you saw at the time when you configured the target on the host `target` reappears here. Okay, now we have to make all discovered logical units for the use as block devices. In Solaris you use the `devfsadm` command for this task. This will populate the `/dev` tree:

```
jmoekamp@inititator:~# devfsadm -C -i iscsi
jmoekamp@inititator:~#
```

When we start format again, we will see that our host `initiator` has a new block device:

```
jmoekamp@inititator:~# format
Searching for disks...done

AVAILABLE DISK SELECTIONS:
       0. c0t600144F0B152CC0000004B080F230004d0 <SUN-COMSTAR-1.0-10.00TB>
          /scsi_vhci/disk@g600144f0b152cc0000004b080f230004
       1. c7d0 <DEFAULT cyl 2085 alt 2 hd 255 sec 63>
          /pci@0,0/pci-ide@1,1/ide@0/cmdk@0,0
Specify disk (enter its number): ^C
jmoekamp@inititator:~#
```

A brand new 10 TB iSCSI volume. Of course we can use it for zfs again:

```
jmoekamp@inititator:~# zpool create testpool c0t600144F0B152CC0000004B080F230004d0
jmoekamp@inititator:~# zfs list
NAME                         USED  AVAIL  REFER  MOUNTPOINT
rpool                       10,1G  5,52G    81K  /rpool
[...]
testpool                      72K  9,78T    21K  /testpool
```

Works as designed. A nice 10 TB ZFS pool on `initiator`.

26.6. Configuring the initator with authentication

Okay, in the non-COMSTAR iSCSI tutorial i explained how to use authentication. With authentication the iSCSI target and the iSCSI initiator can check if the host on the other side of the connection is really the expected node. This works with the CHAP mechanism. Perhaps you know it from PPP for your internet access. It works on the foundation of shared secrets. Only when both partners of a communication know the same secret, the communication partner is authentic. It similar to a secret handshake between two peoples.

Okay, let's configure a bidirectional secret handshake. Just as a good admin we export the pool on it, as the following configuration will terminate the iSCSI connection for a moment.

```
jmoekamp@inititator:~# zpool export testpool
jmoekamp@inititator:~#
```

Okay, to configure the authentication we need to pieces of information at first. Log into the system `target` to gather the iqn of the target.

```
jmoekamp@target:~# itadm  list-target
TARGET NAME                                                  STATE    SESSIONS
<b>iqn.1986-03.com.sun:02:fd9d53cd-6fe5-ecd7-de40-cb3bbb03b422</b>  online   1
```

Now log into the system `initiator` to gather the IQN of the initiator.

```
jmoekamp@inititator:~# iscsiadm list initiator-node
Initiator node name: <b>iqn.1986-03.com.sun:01:809526a500ff.4b07e652</b>
[...]
        Configured Sessions: 1
```

At first we configure the iSCSI target in a way, that it uses chap authentication. We need the IQN of the target here.

```
jmoekamp@target:~# itadm modify-target -a chap iqn.1986-03.com.sun:02:fd9d53cd-6fe5-
    ecd7-de40-cb3bbb03b422
Target iqn.1986-03.com.sun:02:fd9d53cd-6fe5-ecd7-de40-cb3bbb03b422 successfully
    modified
```

Now we have to configure the secrets. The CHAP secrets have be at last 12 characters. At first we set configure the secret, that the initiator will use to authenticate at the target.

```
jmoekamp@target:~# itadm create-initiator -s iqn.1986-03.com.sun:01:809526a500ff.4
    b07e652
Enter CHAP secret: foobarfoobarfoobar
Re-enter secret: foobarfoobarfoobar
```

Now we configure the secret that the target uses to authenticate itself at the initiator.

```
jmoekamp@target:~# itadm modify-target -s iqn.1986-03.com.sun:02:fd9d53cd-6fe5-ecd7-
    de40-cb3bbb03b422
Enter CHAP secret: snafusnafusnafu
Re-enter secret: snafusnafusnafu
Target iqn.1986-03.com.sun:02:fd9d53cd-6fe5-ecd7-de40-cb3bbb03b422 successfully
    modified
```

Okay, we have to something similar on the system `initiator` At first we have to configure the CHAP secret that the initiator uses to authenticate itself

```
jmoekamp@inititator:~# iscsiadm modify initiator-node --CHAP-secret
Enter secret: foobarfoobarfoobar
Re-enter secret: foobarfoobarfoobar
```

Now we tell the system that this initiator should use CHAP authentication to authenticate a target.

```
jmoekamp@inititator:~# iscsiadm modify initiator-node --authentication CHAP
```

The next steps configure the authentication relation between our initiator and a defined target. At we activate an bi-directional authentication. So the initiator has to authenticate at the target as well as the target has to authenticate itself at the initiator.

```
jmoekamp@inititator:~# iscsiadm modify target-param \
 --bi-directional-authentication enable iqn.1986-03.com.sun:02:fd9d53cd-6fe5-ecd7-
     de40-cb3bbb03b422
```

Now we tell the iSCSI initator that the iSCSI target uses CHAP to authenticate the ISCSI initiator.

```
jmoekamp@inititator:~# iscsiadm modify target-param \
 --authentication CHAP iqn.1986-03.com.sun:02:fd9d53cd-6fe5-ecd7-de40-cb3bbb03b422
```

In a last step we configure the shared secret, that the target uses to authenticate the initiator.

```
jmoekamp@inititator:~# iscsiadm modify target-param \
 --CHAP-secret iqn.1986-03.com.sun:02:fd9d53cd-6fe5-ecd7-de40-cb3bbb03b422
Enter secret: snafusnafusnafu
Re-enter secret: snafusnafusnafu
```

Perhaps this figure explain the relation between the secrets better than just command lines can do:

Now we are done. We can do a quick check if our configuration found it's way to the system. At first a quick look at the iSCSI Target on `target`:

```
jmoekamp@target:~# itadm  list-initiator
INITIATOR NAME                                          CHAPUSER  SECRET
iqn.1986-03.com.sun:01:809526a500ff.4b07e652            <none>          set
jmoekamp@target:~# itadm list-target -v
TARGET NAME                                             STATE    SESSIONS
iqn.1986-03.com.sun:02:fd9d53cd-6fe5-ecd7-de40-cb3bbb03b422  online   1
        alias:                  -
        auth:                   chap
        targetchapuser:         -
```

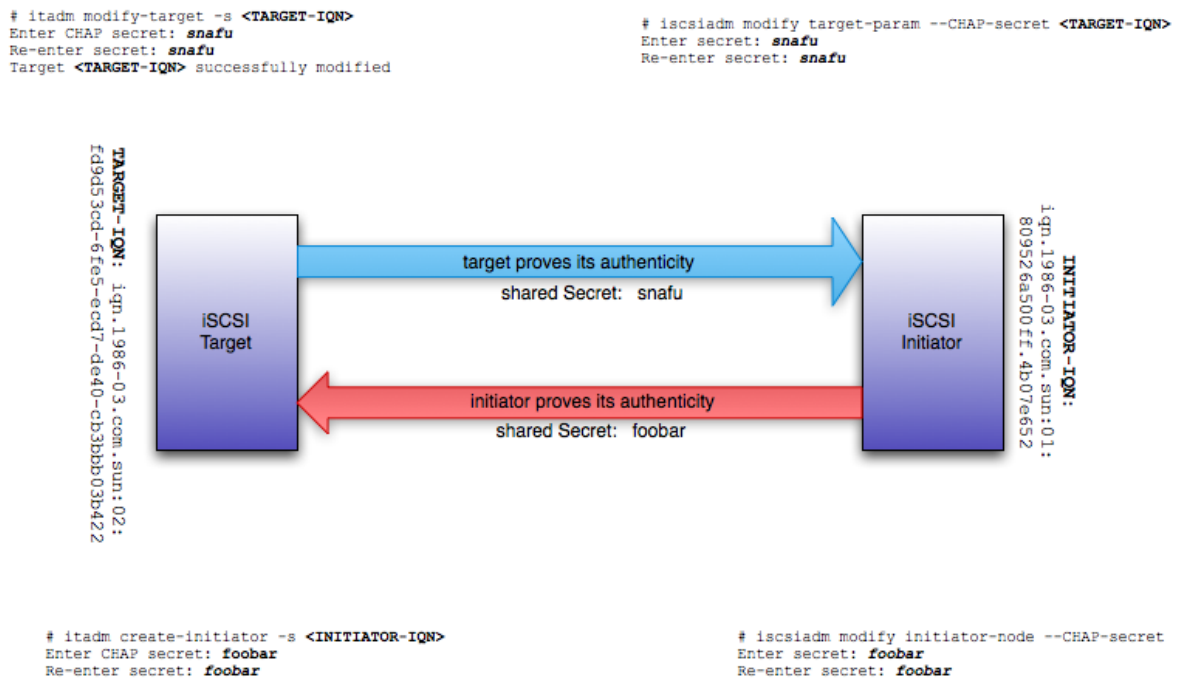

Figure 26.2.: iSCSI Authentication

```
        targetchapsecret:       set
        tpg-tags:               e1000g0 = 2
```

Now we do the same check on `initiator`:

```
jmoekamp@inititator:~# iscsiadm list initiator-node
Initiator node name: iqn.1986-03.com.sun:01:809526a500ff.4b07e652
Initiator node alias: inititator
[...]
        <b>Authentication Type: CHAP
                CHAP Name: iqn.1986-03.com.sun:01:809526a500ff.4b07e652</b>
[...]
        Configured Sessions: 1
```

and

```
jmoekamp@inititator:~# iscsiadm list target-param -v
Target: iqn.1986-03.com.sun:02:fd9d53cd-6fe5-ecd7-de40-cb3bbb03b422
        Alias: -
        <b>Bi-directional Authentication: enabled
        Authentication Type: CHAP
                CHAP Name: iqn.1986-03.com.sun:02:fd9d53cd-6fe5-ecd7-de40-
                    cb3bbb03b422</b>
[...]
        Configured Sessions: 1

jmoekamp@inititator:~#
```

Okay, everything is okay ... now let's just reimport the testpool

```
jmoekamp@inititator:~# zpool import testpool
jmoekamp@inititator:~#
```

We are back in business.

26.7. Conclusion

The configuration of an iSCSI Target isn't more difficult than before. It's just different. I hope i gave you some good insight into this topic.

26.8. Do you want to learn more?

man pages:

docs.sun.com: sbdadm(1M) - SCSI Block Disk command line interface

docs.sun.com: stmfadm(1M)

docs.sun.com: itadm(1M) - administer iSCSI targets

docs.sun.com: iscsiadm(1M)- enable management of iSCSI initiators

Tutorials

c0t0d0s0.org : Less known Solaris Features: iSCSI - tutorial for the userland iSCSI implementation in Solaris.

27. Remote Mirroring with the Availability Suite

Solaris 10/Opensolaris

27.1. Introduction

Solaris was designed with commercial customers in mind. Thus this operating environment has some capabilities that are somewhat useless for the soho user, but absolutely essential for enterprise users. One of this capabilities is remote replication of disk volumes. Imagine the following situation. You have a database and a filesystem for a central application of your company. The filesystem stores binary objects (for example images or something like that). The application is so important for your company, that you plan to build a replicated site. And now your problems start.

Replicating a database is fairly easy. Most databases have some functionality to do a master/replica replication. Filesystems are a little bit harder. Of course you can use rsync, but whats with the data you wrote after the last rsync run and the failure of your main system. And how do you keep the database and the filesystem consistent?

The Solaris operating environment has a feature to solve this problem. It's called Availability Suite (in short AVS). It's a rather old feature, but I would call it a matured feature. The first versions of the tool wasn't really enterprise ready and this led to some rude nick names for this feature, but that's so long ago ....

AVS was designed to give the operating environment the capability to replicate a volume to another site independently from the way it's used. Thus it's irrelevant, if the volume is used by a filesystem, as a raw device for a database. Well you can even use it to give ZFS the capability to do a synchronous replication (a feature missing today)

AVS has a point-in-time-copy feature (something similar to snapshots) as well, but this tutorial will concentrate to the remote mirror capability. Some words about Sun StorageTek Availability Suite at first: We've opensourced the product quite a while ago. While it's a commercial product for Solaris, we've integrated it into Solaris Express Community Edition and Developer Edition.

27.2. Implementation of the replication

The integration of the replication into Solaris is relatively simple. The replication is done by a filter driver in front of the storage device drivers. You can think of it as: Data destined to the harddisk is intercepted by this driver before it goes to the disk to handle the data accordingly to the configuration. This driver can write the data directly to an equivalent driver on a different system or to a queue for later transmission. Furthermore the data is directed to the normal path to the local harddisk.

27.3. Wording

Some definitions are important to understand the following text:

primary host/primary volume: The volume or host which acts as the source of the replication¡

secondary host/secondary volume: The volume or host which acts as the target of the replication

bitmap volume: Each secondary and primary volume has a so called bitmap volume. This bitmap volume is used to store the changes to the primary or secondary volume when replication has failed or was deliberately stopped

27.4. Synchronous Replication

AVS supports two modes of replication. The first mode is the synchronous replication. From the standpoint to ensure that master and replica are identical volume, this is the best method. A system call writing to a replicated volumes doesn't complete until the replica has confirmed, that the data was written to the replicated volume.

By this mode of operation it's ensured that all data committed do disk are on both volumes. This is important for databases for example. But it has an disadvantage as well: A single write takes much longer. You have to factor in the round trip time on the network and the amount of time needed to write the data on the replicated disk.

27.5. Asynchronous Replication

Out of this reason, asynchronous replication was introduced. You can use it in environments with less strict requirements. In this mode the write system calls return when the data has written to the replicated volume and to a per-volume queue. From this queue data is sent to the secondary host. When the secondary host writes the data to the disk, it acknowledges this and the primary host deletes the data from the queue.

This method has the advantage of introducing much less latency to the writes of the system. Especially for long-range replications this is a desired behavior. This comes at a price: In the case of a failure, a committed read from an application may reside in a replication queue but isn't transmitted to the secondary host.

27.6. Choosing the correct mode

Choosing the correct mode is difficult. You have to do an trade-off between performance and integrity. You can speed up synchronous replication by faster networks with a few hops between both hosts, but a soon as you leave your campus this can be a costly endeavor. You should think about one point: Is the need for "up to the last committed" transaction a real or an assumed need.

My experience from the last seven years Sun: As soon as you show people the price tag of "99.999%" availability, the requirements get more differentiated. Or to bring it in a context with remote mirroring: Before ordering this extremely fast and extremely expensive leased line you should talk with the stakeholders with they really need synchronous replication.

27.7. Synchronization

Okay, the replication takes care of keeping the replicated volume and the replica identical, when the software runs. But how to sync both volumes when starting replication or later on, when the replication was interrupted? The process to solve this is called synchronization.

AVS Remote Mirror knows four modes of replication:

Full replication: You do at least one full replication with every volume. It's the first one. The full replication copies all data to the secondary volume.

Update replication : When the volume is in logging mode, the changes to the primary volume is stored on the bitmap volume. Thus with the update replication you can transmit only the changed blocks to the secondary volume.

Full reverse replication This is the other way round. Let's assume you've done a failover to your remote site, and you've worked on the replicated volume for some time. Now you want to switch back to your normal datacenter. You have to transport the changes from the mean time to your primary site as well. Thus there is a replication mode called reverse replication. The full reverse replication copies all data from the secondary volume to the primary volume.

Update reverse replication: The secondary volume has a bitmap volume as well. Thus you can do an update replication from the secondary to the primary volume as well.

Okay, but what mode of replication should you choose? For the first replication it's easy ... full replication. After this, there is a simple rule of thumb: Whenever in doubt of the integrity of the target volume, use the full replication.

27.8. Logging

At last there is another important term in this technology: *Logging*. This has nothing to do with writing log messages about the daemons of AVS. *Logging* is a special mode of operation. This mode is entered when the replication is interrupted. In this case the changes to the primary and secondary will be recorded in the bitmap volume. It's important that *Logging* don't record the change itself. It stores only the information, that a part of the volume has changed. Logging makes the resynchronization of volumes after a disaster more efficient, as you only have to resync the changed parts of a volume as I've explained for the mechanism of update replication before.

27.9. Prerequisites for this tutorial

In this tutorial we need to work with two systems. I will use theoden and gandalf again.

```
10.211.55.200 theoden
10.211.55.201 gandalf
```

27.9.1. Layout of the disks

Both systems have a boot disk and two disks for data. The data disks have a size of 512 MB. In my example this disks are `c1d0` and `c1d1`. I've partitioned each disk in the same manner:

```
# prtvtoc /dev/rdsk/c1d0s2
. /dev/rdsk/c1d0s2 partition map
[..]
.                          First      Sector    Last
. Partition  Tag  Flags    Sector      Count    Sector   Mount
   Directory
       0      0    00        6144      65536     71679
       1      0    00       71680     968704   1040383
       2      5    01           0    1042432   1042431
       8      1    01           0       2048      2047
       9      9    00        2048       4096      6143
```

It's important that the primary and secondary partitions and their respective bitmap partitions are equal in size. Furthermore: Don't use cylinder 0 for partitions under the control of AVS. This cylinder may contain administrative information from other components of the systems. Replication of this information may lead to data loss.

27.9.2. Size for the bitmap volume

In my example I've chosen a 32 MB bitmap partition for a 512 MB data partition. This is vastly too large for the use case.

You can calculate the size for the bitmap size as follows:

$$Size_{\text{Bitmapvolume in kB}} = 1 + (Size_{\text{Datavolume in GB}} * 4)$$

Let's assume a 10 GB volume for Data:

$$Size_{\text{Bitmapvolume in kB}} = 1 + (10 * 4)$$

$$Size_{\text{Bitmapvolume in kB}} = 41kb$$

27.9.3. Usage of the devices in our example

The first partition `c1dXs0` is used as the bitmap volume. The second volume `c1dXs1` is used as the primary volume on the source of the replication respectively the secondary volume on the target of the replication.

27.10. Setting up an synchronous replication

Okay, at first we have to enable AVS on both hosts. At first we activate it on `theoden`

```
[root@theoden:~]$ dscfgadm -e
```

This command may ask for the approval to create the config database, when you run this command for the first time. Answert this question with `y`.After this we switch to gandalf to do the same.

```
[root@gandalf:~]$ dscfgadm -e
```

Now we can establish the replication. We login at theoden first, and configure this replication.

```
[root@theoden:~]$ sndradm -e theoden /dev/rdsk/c1d0s1 /dev/rdsk
   /c1d0s0 gandalf /dev/rdsk/c1d0s1 /dev/rdsk/c1d0s0 ip sync
Enable Remote Mirror? (Y/N) [N]: y
```

What have we configured? We told AVS to replicate the content of `/dev/rdsk/c1d0s1` on `theoden` to `/dev/rdsk/c1d0s1` on `gandalf`. AVS uses the `/dev/rdsk/c1d0s1` volume on each system as the bitmap volume for this application. At the end of this command we configure, that the replication uses IP and it's a synchronous replication.

Okay, but we have to configure it on the targeted system of the replication as well:

```
[root@gandalf:~]$ sndradm -e theoden /dev/rdsk/c1d0s1 /dev/rdsk
   /c1d0s0 gandalf /dev/rdsk/c1d0s1 /dev/rdsk/c1d0s0 ip sync
Enable Remote Mirror? (Y/N) [N]: y
```

We repeat the same command we used on ¡code¿theoden¡/code¿ on `gandalf` as well. Forgetting to do this step is one of the most frequent errors in regard of setting up an remote mirror.

An interesting command in regard of AVS remote mirror is the `dsstat` command. It shows the status and some statistic data about your replication.

```
[root@theoden:~]$ dsstat -m sndr
name              t  s    pct role    kps   tps   svt
dev/rdsk/c1d0s1   P  L 100.00  net      0     0     0
dev/rdsk/c1d0s0                bmp      0     0     0
```

The 100.00 doesn't stand for "100% of the replication is completed". It standing for "100% of the replication to do". We have to start the replication manually. Okay, more formally the meaning of this column is "percentage of the volume in need of syncing". And as we freshly configured this replication it's obivous, that the complete volume needs synchronisation.

Two other columns are important, too: It's the ¡code¿t¡/code¿ and the ¡code¿s¡/code¿ column. The ¡code¿t¡/code¿ column designates the volume type and the ¡code¿s¡/code¿ the status of the volume. In this case we've observed the primary volume and it's in the logging mode. It records changes, but doesn't replicate them right now to the secondary volume.

Okay, so let's start the synchronisation:

```
[root@theoden:~]$  sndradm -m gandalf:/dev/rdsk/c1d0s1
Overwrite secondary with primary? (Y/N) [N]: y
```

We can lookup the progress of the sync with the ¡code¿dsstat¡/code¿ command again:

```
[root@theoden:~]$  dsstat -m sndr
name              t  s    pct role    kps   tps  svt
dev/rdsk/c1d0s1   P SY  97.39  net     Inf     0 -NaN
dev/rdsk/c1d0s0               bmp     Inf     0 -NaN
[root@theoden:~]$  dsstat -m sndr
name              t  s    pct role    kps   tps  svt
dev/rdsk/c1d0s1   P SY  94.78  net     Inf     0 -NaN
dev/rdsk/c1d0s0               bmp     Inf     0 -NaN
[...]
[root@theoden:~]$  dsstat -m sndr
name              t  s    pct role    kps   tps  svt
dev/rdsk/c1d0s1   P SY   3.33  net     Inf     0 -NaN
dev/rdsk/c1d0s0               bmp     Inf     0 -NaN
[root@theoden:~]$  dsstat -m sndr
name              t  s    pct role    kps   tps  svt
dev/rdsk/c1d0s1   P  R   0.00  net       0     0    0
dev/rdsk/c1d0s0               bmp       0     0    0
```

When we start the synchronization the status of the volume switches to `SY` for synchronizing. After a while the sync is complete. The status switches again, this time to `R` for replicating. From this moment all changes to the primary volume will be replicated to the secondary one.

Now let's play around with our new replication set by using the primary volume. Create a filesystem on it for example, mount it and play around with it:

```
 [root@theoden:~]$  newfs /dev/dsk/c1d0s1
newfs: construct a new file system /dev/rdsk/c1d0s1: (y/n)? y
/dev/rdsk/c1d0s1:       968704 sectors in 473 cylinders of 64
   tracks, 32 sectors
        473.0MB in 30 cyl groups (16 c/g, 16.00MB/g, 7680 i/g)
super-block backups (for fsck -F ufs -o b=#) at:
```

```
 32, 32832, 65632, 98432, 131232, 164032, 196832, 229632,
    262432, 295232,
 656032, 688832, 721632, 754432, 787232, 820032, 852832,
    885632, 918432, 951232
[root@theoden:~]$ mount /dev/dsk/c1d0s1 /mnt
[root@theoden:~]$ cd /mnt
[root@theoden:~]$ touch test
[root@theoden:~]$ cp /var/log/* .
[root@theoden:~]$ mkfile 1k test2
```

Okay, in a few seconds I will show you, that all changes really get to the other side.

27.11. Testing the replication

One of the most essential tasks when configuring disaster recovery mechanism is training the procedure. Thus let's test the switch into our remote datacenter. We will simulate a failure now. This will show that the data really gets replicated to a different system ;)

27.11.1. Disaster test

Okay, at first we leave a timestamp in our replicated filesystem, just for testing this feature. I assume, that it's still mounted on the primary host.

```
[root@theoden:~]$# cd /mnt
[root@theoden:~]$ ls
aculog      blah2       lastlog     messages    sulog
   utmpx
blah        ds.log      lost+found  spellhist   test
   wtmpx
[root@theoden:~]$ date > timetest
[root@theoden:~]$ cat timetest
Sat Mar 29 19:28:51 CET 2008
[root@theoden:~]$ cd /
[root@theoden:~]$ umount /mnt
```

Please keep the timestamp in mind. Now we switch both mirrors into the logging mode. As an alternative you can disconnect the network cable. This will have the same effect. Whenever the network link between the both hosts is unavailable, both volume will be set to the logging mode. As I use virtual servers, I can't disconnect a network cable, thus can't do it this way. Okay ...

```
[root@theoden:~]$ sndradm -l
Put Remote Mirror into logging mode? (Y/N) [N]: y
```

When you look at the status of the replication on `theoden`, you will see the logging state again.

```
[root@theoden:~]$ dsstat
name              t  s    pct role   ckps    dkps    tps   svt
dev/rdsk/c1d0s1   P  L   0.00  net      -       0      0     0
dev/rdsk/c1d0s0                bmp      0       0      0     0
```

On `gandalf` it's the same.

```
[root@gandalf:~]$ dsstat
name              t  s    pct role   ckps    dkps    tps   svt
dev/rdsk/c1d0s1   S  L   0.00  net      -       0      0     0
dev/rdsk/c1d0s0                bmp      0       0      0     0
```

Okay, now we mount the secondary volume. Please keep in mind, that we don't mount the volume via network or via a dual ported SAN. It's a independent storage device on a different system.

```
[root@gandalf:~]$ mount /dev/dsk/c1d0s1 /mnt
[root@gandalf:~]$ cd /mnt
[root@gandalf:~]$ ls -l
total 7854
-rw-------   1 root     root           0 Mar 29 16:43 aculog
[..]
-rw-r--r--   1 root     root          29 Mar 29 19:28 timetest
-rw-r--r--   1 root     root        2232 Mar 29 16:43 utmpx
-rw-r--r--   1 root     root       43152 Mar 29 16:43 wtmpx
```

Okay, there is a file called `timetest`. Let's look for the data in the file.

```
[root@gandalf:~]$ cat timetest
Sat Mar 29 19:28:51 CET 2008
```

The file and its content got replicated to the secondary volume instantaniously. Okay, now let's switch back to primary hosts, but we create another file with a timestamp before doing that.

```
[root@gandalf:~]$ date > timetest2
[root@gandalf:~]$ cat timetest2
Sat Mar 29 19:29:10 CET 2008
[root@gandalf:~]$ cd /
[root@gandalf:~]$ umount /mnt
```

Okay, we changed the secondary volume by adding this file, thus we have to sync our primary volume. Thus we do an update reverse synchronisation:

```
[root@theoden:~]$ sndradm -u -r
Refresh primary with secondary? (Y/N) [N]: y
[root@theoden:~]$ dsstat
name              t  s    pct role   ckps    dkps    tps   svt
dev/rdsk/c1d0s1   P  R   0.00  net      -       0      0     0
dev/rdsk/c1d0s0                bmp      0       0      0     0
```

This has two consequence. The changes to the secondary volumes are transmitted to the primary volume (as we use the update sync we just transmit this changes) and the replication is started again. Okay, but let's check for our second timestamp file. We mount our filesystem by using the primary volume.

```
[root@theoden:~]$ mount /dev/dsk/c1d0s1 /mnt
[root@theoden:~]$ cd /mnt
[root@theoden:~]$ ls -l
total 7856
-rw-------   1 root     root           0 Mar 29 16:43 aculog
[...]
-rw-r--r--   1 root     root          29 Mar 29 19:28 timetest
-rw-r--r--   1 root     root          29 Mar 29 19:32 timetest2
[...]
[root@theoden:~]$ cat timetest2
Sat Mar 29 19:29:10 CET 2008
```

Et voila, you find two files beginning with `timetest` and the second version contains the new timestamp we've have written to the filesystem while using the secondary volume on the seondary host. Neat, isn't it?

27.12. Asynchronous replication and replication groups

A new scenario: Okay, the filesystem gets replicated now. Let's assume that we use `/dev/rdsk/c1d0s1` for a database. The filesystem and the database partition are used from the same application and it's important, that the metadata in the database an the binary objects are in sync even when you switched over to the remote site, albeit it's acceptable to lose the last few transactions, as both sites are 1000km away from each other and synchronous replication is not an option.

27.12.1. The problem

When you use synchronous replication, all is well. But let's assume you've chosen asynchronous replication. Under this circumstances a situation can occur, where one queue is processed faster than another, thus the on-disk states of each volume may be consistent in itself, but both volumes may have the state at different point in time, thus leaving the application data model inconsistent.

Such an behavior is problematic, when you have a database volume and a filesystem volume working together for an application, but the results can be catastrophic when you use a database split over several volumes.

The solution of this problem would be a mechanism, that keeps the writes to a group of volumes in order for the complete group. Thus is inconsistencies can't occur.

27.12.2. Replication Group

To solve such problems, AVS supports a concept called Replication Group. Adding volumes to a replication group has some implications:

- All administrative operations to this group are atomic. Thus when you change to logging mode or start an replication, this happens on all volumes in the group
- The writes to any of the primary volumes will be replicated in the same order to the secondary volumes. The scope of this ordering is the complete group, not the single volume.¡/li¿
- Normally every replication relation has its own queue and its own queue flusher daemon. Thus multiple volumes can flush their queue in parallel to increase the performance. In case of the Replication group all I/O operations are routed trough a single queue. This may reduce the performance.

27.12.3. How to set up a replication group?

Okay, at first we login at `theoden`, our primary host in our example. We have to add the existing replication to the replication group and configure another replication relation directly in the correct group. I will create an replication group called `importantapp`.

```
[root@theoden:~]$ sndradm -R g importantapp gandalf:/dev/rdsk/
   c1d0s1
Perform Remote Mirror reconfiguration? (Y/N) [N]: y
```

We've added the group property to the existing mirror, now we create the new mirror directly in the correct group

```
[root@theoden:~]$ sndradm -e theoden /dev/rdsk/c1d1s1 /dev/rdsk
   /c1d1s0 gandalf /dev/rdsk/c1d1s1 /dev/rdsk/c1d1s0 ip sync g
   importantapp
Enable Remote Mirror? (Y/N) [N]: y
```

With `sndradm -P` you can look up the exact configuration of your replication sets:

```
[root@theoden:~]$ sndradm -P
/dev/rdsk/c1d0s1        ->      gandalf:/dev/rdsk/c1d0s1
autosync: off, max q writes: 4096, max q fbas: 16384, async
   threads: 2, mode: sync, group: importantapp, state: syncing
/dev/rdsk/c1d1s1        ->      gandalf:/dev/rdsk/c1d1s1
autosync: off, max q writes: 4096, max q fbas: 16384, async
   threads: 2, mode: sync, group: importantapp, state: syncing
```

Okay, both are in the same group. As before, we have to perform this configuration on both hosts: So we repeat the same steps on the other hosts as well:

```
[root@gandalf:~]$ sndradm -e theoden /dev/rdsk/c1d1s1 /dev/rdsk
   /c1d1s0 gandalf /dev/rdsk/c1d1s1 /dev/rdsk/c1d1s0 ip sync g
   importantapp
Enable Remote Mirror? (Y/N) [N]: y
[root@gandalf:~]$ sndradm -R g importantapp gandalf:/dev/rdsk/
   c1d0s1
Perform Remote Mirror reconfiguration? (Y/N) [N]: y
```

No we start the replication of both volumes. We can to this in a single step by using the name of the group.

```
[root@theoden:~] sndradm -m -g importantapp
Overwrite secondary with primary? (Y/N) [N]: y
```

Voila, both volumes are in synchronizing mode:

```
[root@theoden:~]$ dsstat
name              t  s    pct role   ckps    dkps    tps   svt
dev/rdsk/c1d0s1   P SY  89.37  net       -     Inf      0 -NaN
dev/rdsk/c1d0s0                bmp       0      28      0 -NaN
dev/rdsk/c1d1s1   P SY  88.02  net       -     Inf      0 -NaN
dev/rdsk/c1d1s0                bmp       0      28      0 -NaN
```

Two minutes later the replication has succeeded, we have now a fully operational replication group:

```
[root@theoden:~]$  dsstat
name              t  s    pct role    ckps    dkps    tps   svt
dev/rdsk/c1d0s1   P  R   0.00  net       -       0      0     0
dev/rdsk/c1d0s0                bmp       0       0      0     0
dev/rdsk/c1d1s1   P  R   0.00  net       -       0      0     0
dev/rdsk/c1d1s0                bmp       0       0      0     0
```

Now both volumes are in replicating mode. Really easy, it's just done by adding the group to the replication relations.

27.13. Deleting the replication configuration

In the last parts of this tutorial I've explained to you, how you set up a replication relation. But it's important to know how you deactivate and delete the replication as well. It's quite easy to delete the replication. At first we look up the existing replication configuration.

```
[root@gandalf:~]$ sndradm -P
/dev/rdsk/c1d1s1        <-      theoden:/dev/rdsk/c1d1s1
autosync: off, max q writes: 4096, max q fbas: 16384, async
   threads: 2, mode: sync, state: logging
```

Okay, we can use the local or remote volume as a name to choose the configuration to be deleted:

```
[root@gandalf:~]$ sndradm -d theoden:/dev/rdsk/c1d1s1
Disable Remote Mirror? (Y/N) [N]: y
```

Now we can lookup the configuration again.

```
[root@gandalf:~]$ sndradm -P
```

As you see, the configuration is gone. But you have to do the same on the other host. So login as root to the other host:

```
[root@theoden:~]$ sndradm -P
/dev/rdsk/c1d1s1        ->      gandalf:/dev/rdsk/c1d1s1
autosync: off, max q writes: 4096, max q fbas: 16384, async
   threads: 2, mode: sync, state: logging
[root@theoden:~]$ sndradm -d gandalf:/dev/rdsk/c1d1s1
Disable Remote Mirror? (Y/N) [N]: y
[root@theoden:~]$ sndradm -P
[root@theoden:~]$
```

27.14. Truck based replication

Andrew S. Tannenbaum said "Never underestimate the bandwidth of a truck full of tapes hurling down the highway". This sounds counterintuitive at the first moment, but when you start to think about it, it's really obvious.

27.14.1. The math behind the phrase

Let's assume that you have two datacenters, thousand kilometers apart from each other. You have to transport 48 Terabytes of storage. We will calculate with the harddisk marketing system, 48.000.000 Megabytes. Okay ... now we assume, that we have a 155Mps leased ATM line between the locations. Let's assume that we can transfer 15,5 Megabytes per second of this line under perfect circumstances. Under perfect circumstances we can transfer the amount of data in 3096774 seconds. Thus you would need 35 days to transmit the 48 Terabytes. Now assume a wagon car with two thumpers (real admins don't use USB sticks, they use the X4500 for their data transportation needs)in the trunk driving at 100 kilometers per hour. The data would reach the datacenter within 10 hours. Enough time to copy the data to the transport-thumpers and after the arrival from the thumper to the final storage array.

27.14.2. Truck based replication with AVS

AVS Remote mirror supports a procedure to exactly support such a method: Okay, let's assume you want to migrate a server to a new one. But this new server is 1000km away. You have multiple terabytes of storage, and albeit your line to the new datacenter is good enough for the updates, an full sync would take longer than the universe will exist because of the proton decay.

AVS Remote Mirror can be configured in a way, that relies on a special condition of primary and secondary volumes: The disks are already synchronized, before starting the replication. For example by doing a copy by `dd` to the new storage directly or with the redirection of an transport media like tapes. When you configure AVS Remote Mirror in this way, you don't need the initial full sync.

27.14.3. On our old server

To play around, we create at first a new filesystem:

```
[root@theoden:~]$ newfs /dev/rdsk/c1d1s1
newfs: construct a new file system /dev/rdsk/c1d1s1: (y/n)? y
/dev/rdsk/c1d1s1:       968704 sectors in 473 cylinders of 64
   tracks, 32 sectors
        473.0MB in 30 cyl groups (16 c/g, 16.00MB/g, 7680 i/g)
super-block backups (for fsck -F ufs -o b=#) at:
 32, 32832, 65632, 98432, 131232, 164032, 196832, 229632,
    262432, 295232,
 656032, 688832, 721632, 754432, 787232, 820032, 852832,
    885632, 918432, 951232
```

Now mount it , play around with it and put a timestamp in a file.

```
[root@theoden:~]$ mount /dev/dsk/c1d1s1 /mnt
[root@theoden:~]$ mount /dev/dsk/c1d1s1 /mnt
[root@theoden:~]$ touch /mnt/test1
[root@theoden:~]$ mkfile 1k /mnt/test2
[root@theoden:~]$ mkfile 1k /mnt/test3
[root@theoden:~]$ mkfile 1k /mnt/test4
[root@theoden:~]$ date >> /mnt/test5
```

Okay, now unmount it again.

```
[root@theoden:~]$ umount /mnt
```

Now we can generate a backup of this filesystem. You have to make a image of the volume, making a tar or cpio file backup isn't sufficient.

```
[root@theoden:~]$ dd if=/dev/rdsk/c1d1s1 | gzip > 2migrate.gz
968704+0 records in
968704+0 records out
```

Okay, now activate the replication on the primary volume. Don't activate it on the secondary one! The important difference to a normal replication is the `-E`. When you use this switch, the system assumes that the primary and secondary volume are identical already.

```
[root@theoden:~]$ sndradm -E theoden /dev/rdsk/c1d1s1 /dev/rdsk
   /c1d1s0 gandalf /dev/rdsk/c1d1s1 /dev/rdsk/c1d1s0 ip sync
Enable Remote Mirror? (Y/N) [N]: y
```

Okay, we've used the `-E` switch again to circumvent the need for a full syncronisation. When you look at the status of volume, you will see the volume in the "logging" state:

```
[root@theoden:~]$ dsstat
name              t  s    pct role   ckps    dkps    tps   svt
dev/rdsk/c1d1s1   P  L   0.00  net      -       0      0     0
dev/rdsk/c1d1s0                bmp      0       0      0     0
```

This means, that you can do changes on the volume.

```
[root@theoden:~]$ mount /dev/dsk/c1d1s1 /mnt
[root@theoden:~]$ cat /mnt/test5
Mon Mar 31 14:57:04 CEST 2008
[root@theoden:~]$ date >> /mnt/test6
[root@theoden:~]$cat /mnt/test6
Mon Mar 31 15:46:03 CEST 2008
```

Now we transmit our image of the primary volume to our new system. In my case it's scp, but for huge amount of data sending the truck with tapes would be more sensible.

```
[root@theoden:~]$ scp 2migrate.gz jmoekamp@gandalf:/export/home
   /jmoekamp/2migrate.gz
Password:
2migrate.gz           100% |*****************************|
   1792 KB    00:00
```

27.14.4. On our new server

Okay, when the transmission is completeted, we write the image to the raw device of the secondary volume:

```
[root@gandalf:~]$ cat 2migrate.gz | gunzip | dd of=/dev/rdsk/
   c1d1s1
968704+0 records in
968704+0 records out
```

Okay, now we configure the replication on the secondary host:

```
# sndradm -E theoden /dev/rdsk/c1d1s1 /dev/rdsk/c1d1s0 gandalf
   /dev/rdsk/c1d1s1 /dev/rdsk/c1d1s0 ip sync
```

A short look into the status of replication:

```
[root@gandalf:~]$ dsstat
name              t  s    pct role   ckps    dkps    tps   svt
dev/rdsk/c1d1s1   S  L   0.00  net      -       0      0     0
dev/rdsk/c1d1s0                bmp      0       0      0     0
```

Okay, our primary and secondary volumes are still in logging mode. How do we get them out of this? In our first example we did an full syncronisation, this time we need only an update synchronisation. So login as root to our primary host and initiate such an update sync. This is the moment, where you have to stop working on the primary volume.

```
[root@theoden:~]$ sndradm -u
Refresh secondary with primary? (Y/N) [N]: y
```

After this step all changes we did after creating the image from our primary volume will be synced to the secondary volume.

27.14.5. Testing the migration

Well ... let's test it: Do you remember, that we created `/mnt/test6` after the `dd` for the image? Okay, at first, we put the replication in logging mode again. So login as root on our secondary host.

```
[root@gandalf:~]$ sndradm -l
Put Remote Mirror into logging mode? (Y/N) [N]: y
\begin{lstlisting}
Now we mount the secondary volume:
\begin{lstlisting}
# mount /dev/dsk/c1d1s1 /mnt
[root@gandalf:~]$ cd /mnt
[root@gandalf:~]$ ls
lost+found  test2       test4       test6
test1       test3       test5
```

By the virtues of update synchronisation, the `test6` appeared on the seondary volume.Let's have a look in `/mnt/test6`:

```
[root@gandalf:~]$ cat test6
Mon Mar 31 15:46:03 CEST 2008
```

Cool, isn't it ?

27.15. Conclusion

How can you use this feature? Some use cases are really obvious. It's a natural match for disaster recovery. The Sun Cluster Geographic Edition even supports this kind of remote mirror out of the box to do cluster failover with wider distances than just a campus. But it's usable for other jobs as well, for example for migrations to a new datacenter, when you have to transport a large amount data over long distances without a time window for a longer service interruption.

27.16. Do you want to learn more?

Documentation

Sun StorageTek Availability Suite 4.0 Software Installation and Configuration Guide[1]

Sun StorageTek Availability Suite 4.0 Remote Mirror Software Administration Guide[2]

misc. Links

OpenSolaris Project: Sun StorageTek Availability Suite[3]

[1] `http://docs.sun.com/source/819-6147-10/index.html`
[2] `http://docs.sun.com/source/819-6148-10/index.htm`
[3] `http://opensolaris.org/os/project/avs/`

28. Point-in-Time Copy with the Availability Suite

Solaris 10/Opensolaris

28.1. Introduction

The basic idea of Point-in-Time copies is the idea to freeze the contents of a disk or a volume at a certain time, thus other processes can work on data of a certain point in time, while your application works on the original dataset and changes it.

Why is this important? Let's assume you want to make a backup. The problem is quite simple. When a backup takes longer than a few moments, the files backup first my represent an different state of the system than the files backup last. You backup is inconsistent, as you've done a backup of a moving target. Okay, you could simply freeze your application, copy its data to another disk (via cp or dd) and backup it from there or backup it directly and restart your application, but most of the time, this isn't feasible. Let's assume you have a multi-terabyte database on your system. A simple copy can take quite a time, in this time your application doesn't work (at least when you database has no backup mode).

Okay, simple copy is to ineffective. We have to do it with other methods. This tutorial will show you the usage of a method integrated into OpenSolaris and Solaris.

28.2. Basics

One of this methods is the usage of the point in time copy functionality of the Availability Suite. I've wrote about another function of AVS not long ago when I wrote the tutorial about remote replication. Point-in-time-copy and remote replication are somewhat similar (you detect and record changes and transmit those to a different disk, albeit the procedures are different). Thus it was quite logical to implement both in the AVS.

28.2.1. Availability Suite

The AVS is a Suite consisting out of two important parts: The "Remote Replication" functionality and the "Point-in-time Copy" functionality. Regular readers of this blog will remember the remote replication as I've already written a tutorial about it. The Availability Suite is integrated to Solaris Express Community and Developer Edition. You can use it for free. It's available for Solaris as well, but when you want support for it, you have to buy the product, as it's a add-on product for Solaris 10

28.2.2. The jargon of Point in Time Copies with AVS

Okay, as every technology the mechanisms of Point-in-time copies have their own jargon and I will use it quite regularly in this tutorials.

Master volume: The master volume is the source of the point in time copy. This is the original dataset

Shadow volume: The shadow volume is the volume, which contains the point in time copy

Virtual shadow volume: There are certain methods to establish a point in time copy, that copies only the original data in the case the data is changed on the master volume. But such a shadow volume is incomplete, as the unchanged parts are missing on the shadow volume . For this the virtual shadow volume was introduced.

The idea is simple, but effective. Whenever a block wasn't changed since the last sync of your point-in-time copy, the data is delivered by the master volume. When the block has changed, the data is delivered by the shadow volume. This is transparent to the user or the application, as this virtual shadow volume is created by the AVS point-in-time copy drivers. You access this virtual shadow volume simply by using the name of the shadow volume, even albeit the volume doesn't contain all the needed data.

Bitmap volume: All this mechanisms need a logbook about all the changes made to the master volume. This job is done by bitmap volume. Whenever a block on the disk is changed, AVS marks this in the bitmap volume. The bitmap volume is used at several occasions. By using the bitmap volume it can efficiently sync the master and the shadow volume, you can construct the virtual shadow volume with it in an efficient way.

All types of volumes can be placed on real disk or volumes represented by Veritas Volume Manager or Solaris Volume Manager.

28.2.3. Types of copies

Point-in-time Copy in AVS supports three types of copies:

- independent copies
- dependent copies
- compact independent copies

All three have a basic idea. Using a bitmap to track changes and using it generate a point-in-time copy. But the methods behind it are quite different. In the next three parts of this tutorial I will dig deeper into this methods.

28.3. Independent copy

The most important point about independent copies are in their name. The point in time copy is an independent copy of your original data. You can use it on its own and the copy doesn't need the original data to work.

This method is quite similar of doing a copy with dd or cp. At first a full copy is done by the PiT functionality. Now you've created the first point in time copy. But now the advantages of tracking the changes in a database come in to the game. Whenever data on the master is changed, the system tracks this in the bitmap volume. At the next resync of the shadow with the master, the system only copies the changed blocks to the shadow. This vastly reduces the time for the update

28.3.1. Deeper dive

At first you have your master volume. It's the volume with your data. Before configuration of point-in-time copy both shadow and bitmap copy are uninitialized. We use some special manufactured disks for tutorial purposes. They have only five blocks. :)

When you've configured the independent copy, a full sync takes place. Each block is copied from the master volume to the shadow volume, and the bitmap volume is initialized. At the end of this process master and shadow are identical and the bitmap is in the "clean" state. No differences between the both. There is one important fact: After the initial copy, the bitmap doesn't have to be clean. During the full sync the independent copy behaves like a dependent copy. This is done to enable you to use the master volume directly after initiating the independent copy. So, when you change the master volume during the full sync, you will have a "dirty" bitmap (I will explain this condition in a few moments).

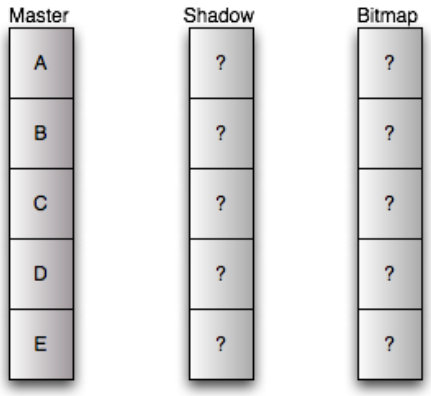

Figure 28.1.: Independent copy: Before configuration

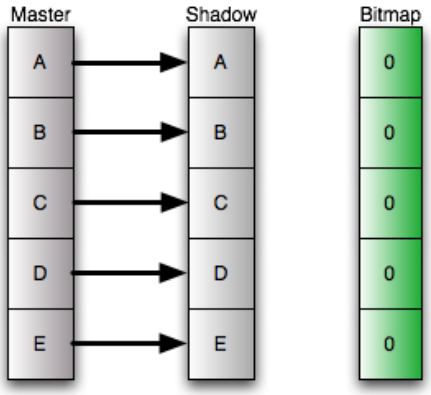

Figure 28.2.: Independent copy: After initialization

Okay, now we change the fourth block of the disk. As the old data is already on the shadow volume, we don't have to move any data. But we log in the bitmap volume, that a block has changed, the block is "dirty". From now the bitmap is in the "dirty" state. The dirty state tells us, that there are differences between the master and the shadow volume.

Okay, we don't need to move data, why do we need the bitmap volume. The bitmap volume makes the synchronization of master and shadow much more efficient. With the bitmap volume you known the position of changed blocks. So when you resync your shadow with the shadow you just have to copy this blocks, and not the whole disk. After copying the block, the adjacent bit in the bitmap is set to zero, the system known that the synced block on master and shadow are identical again.

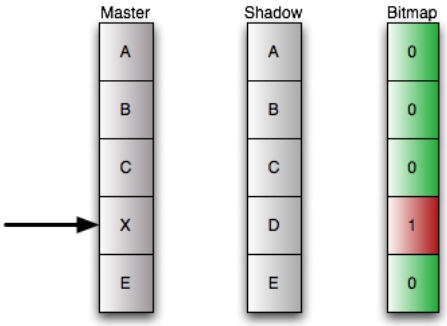

Figure 28.3.: Independent copy: Representation of changed data

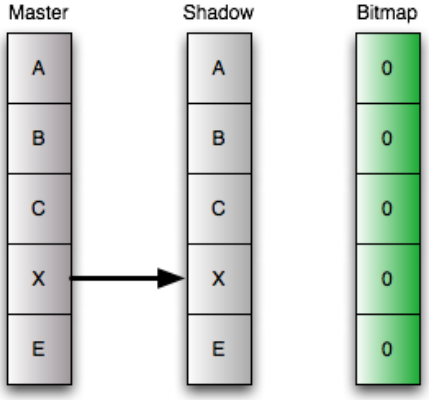

Figure 28.4.: Independent copy: Resynchronization

28.3.2. Advantages and Disadvantages

This has some interesting advantages: You can export this disks in your SAN and use it from a different server. For example you can mount it on a backup server, thus you don't need to transport all the traffic across you local area network.

But there are some disadvantages. You have a high initial amount of data to sync and the size of the master and the shadow have to be equal. This isn't much a problem of the time needed for the sync (because of the fact, that it behaves as a dependent copy) but it poses more load to CPU and I/O system to copy all the data.

28.4. Dependent Copy

The mechanism of dependent copies was introduce to get rid of this initial sync, as there are circumstances where this initial copy would pose to much load to the system.

The dependent copy uses the bitmap a little bit different. The shadow disk doesn't contain all the data, it just contains the changed data. The bitmap is used to keep track of the block on the masters which have a copy on the shadow.

28.4.1. ¡h4¿Deeper dive

The dependent copy is one of the two mechanisms in AVS using the concept of the virtual shadow. Thus the model is a little more complex. Let's assume you create an dependent copy. The initialization is simple. You move no data. You just initialize the bitmap volume. When an user or application access the virtual shadow volume, it checks in the bitmap if the blocks has changed. If the bitmap signals no change, it just delivers the original block of the master volume.

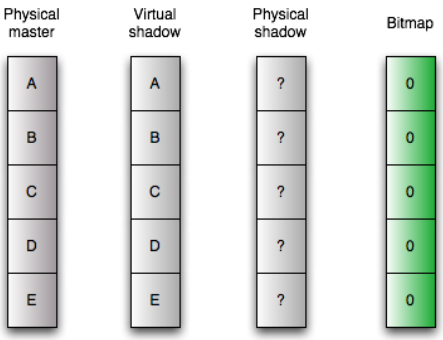

Figure 28.5.: Dependent copy: After initialization

When you change some data on the master volume, AVS starts to copy data. It copies the original content of the block onto the physical shadow volume at the same logical position in the volume. This is the reason, why master and shadow volumes have to have the same size when using dependent copies. Furthermore AVS logs in the bitmap that there is data on the shadow volumes, the block is dirty in the bitmap. When you access the virtual shadow volume now, the bitmap is checked again. But for blocks declared dirty in the bitmap, the data is delivered from the copy on the physical shadow volume, for all other "clean" blocks the data comes from the master volume.

Resyncing the shadow to the master is easy. Just reinitializing the bitmap. Now all data comes from the master volume again, until you change some data on it.

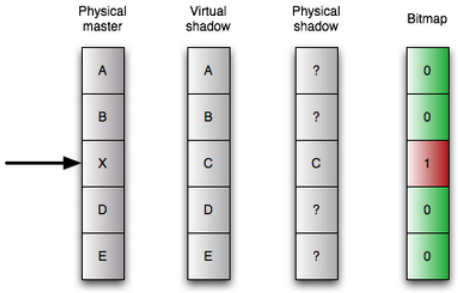

Figure 28.6.: Dependent copy: Changes on the data volume

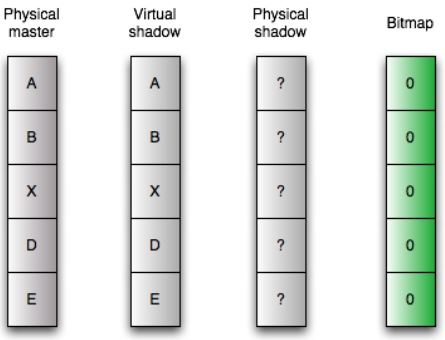

Figure 28.7.: Dependent copy: Resynchronization

28.5. Advantages and Disadvantages

The advantage of this method is its short time of initialization and syncing from master to the shadow. It's virtually zero. But you lose the advantages of the independent copy. The dependent copy can only used in conjunction with the original data. This has two main effects. You can't export the copied volume to a different server and you can't use is as an copy for disaster recovery when your master volume has failed.

Furthermore the master and the slave volume still have to be the same size. But the next method for making an point-in-time copy was derived from the dependent copy exactly to circumvent this problem.

28.6. Compact dependent copy

The compact dependent copy is similar to the normal dependent copy. But this dog knows an additional trick: The shadow and the master doesn't have to be at the same size.

Like the dependent copy this methods uses the concept of the virtual shadow volume. So the bitmap is really important. The bitmap of the compact dependent copy tracks the changes. This exactly as with the normal dependent snapshots. But the bitmap for compact dependent snapshots is enabled to store an important additional information. It's enabled to track, where the system has written the changed data blocks on the shadow volume. So you don't have to write it at the same logical position, you can write it at any position on your shadow volume and it can retrieve the old data with this information.

28.6.1. Deeper dive

Okay, at start this picture looks pretty much the same like its counterpart at the normal dependent snapshots. Please note, the additional information on the bitmap volume. To make the compact dependent copy we just initialize the bitmap to its "clean" state.

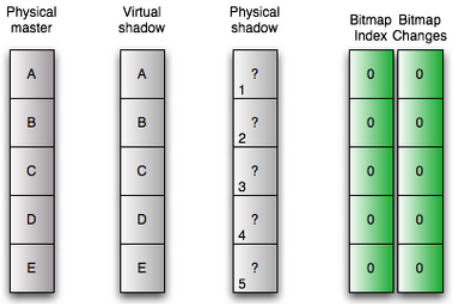

Figure 28.8.: Compact Dependent copy: Initialization

Let's assume that we change the fourth block on our disk. As with the normal copy, the block is declared as dirty. But now starts to work differently. The original data of the master volume is stored to the first free block on the physical shadow volume. In addition to that the position is stored at the bitmap.

The way to read from the shadow volume changes accordingly. When the bitmap signals, that a block is clean, it just passed the data from the master volume to the user or application. When the bitmap signals a dirty, thus changed block, it reads the position

of the block on the physical shadow volume from the bitmap, reads the block from there and delivers it to the application or user.

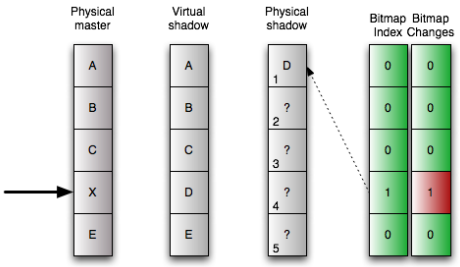

Figure 28.9.: Compact Dependent copy: Change a first block

When we change the next block, for example the third one, the same procedure starts. The original data is stored to the next free block, now the second one, on the physical shadow volume and this position is stored in the bitmap together with the dirty state of the block.

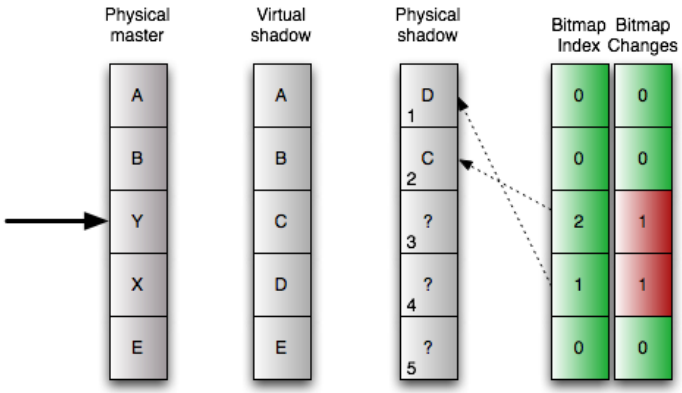

Figure 28.10.: Compact Dependent copy: Change a second block

Okay, resyncing the shadow with the master is easy again. Just initializing the bitmap.

28.6.2. Advantages and Disadvantages

The trick of storing the position with the dirty state has an big advantage. You don't need master and shadow volumes with the same size. When you know, that only a small percentage of blocks change between two point in time copies, you can size the shadow

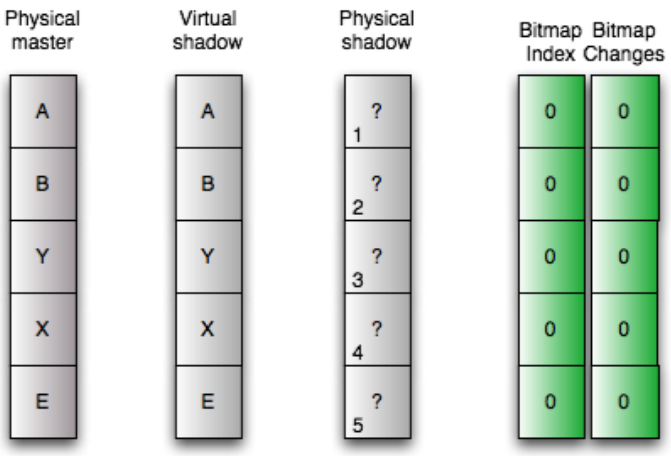

Figure 28.11.: Compact Dependent copy: Resynchronization

volume much smaller, thus saving space on your disk. In my opinion compact dependent copies are the only reasonable way to go when you want more than one copy of your master volume. The disadvantages ares pretty much the same of the normal dependent copies.

28.7. Preparation of the test environment

After all this theory, I will go into more practical stuff. In the following parts of this tutorial I will give you an introduction to point-in-time copies with AVS. But at first we have to prepare some things.

At first: We need only one system for this example, so we don't need any networking configuration. Furthermore you need to assume the root role to do the configuration in this example.

28.7.1. Disklayout

Okay, I will use two harddisks in my example: `/dev/dsk/c1d00` and `/dev/dsk/c1d1`. I've chosen the following layout for the disk.

```
.                          First      Sector     Last
. Partition  Tag  Flags     Sector      Count     Sector  Mount
   Directory
```

```
2       5     01           0   65480940   65480939
3       0     00       48195     273105     321299
4       0     00      321300      80325     401624
5       0     00      401625     273105     674729
6       0     00      674730      80325     755054
8       1     01           0      16065      16064
9       9     00       16065      32130      48194
```

With this configuration I have two 128 mb sized slices. I will use them for data in my example. Additionally I've create two 32 mb small slices for the bitmaps. 32 mb for the bitmaps is too large, but I didn't wanted to calculate the exact size.

28.7.2. Calculation of the bitmap volume size for independent and dependent shadows

You can calculate the size for the bitmap size as follows:

$$Size_{\text{Bitmapvolume in kB}} = 24 + (Size_{\text{Datavolume in GB}} * 8)$$

Let's assume a 10 GB volume for Data:

$$Size_{\text{Bitmapvolume in kB}} = 24 + (10 * 8)$$

$$Size_{\text{Bitmapvolume in kB}} = 104kb$$

28.7.3. Calculation of the bitmap volume size for compact dependent shadows

You can calculate the size for the bitmap size as follows:

$$Size_{\text{Bitmapvolume in kB}} = 24 + (Size_{\text{Datavolume in GB}} * 256)$$

Let's assume a 10 GB volume for Data:

$$Size_{\text{Bitmapvolume in kB}} = 24 + (10 * 256)$$

$$Size_{\text{Bitmapvolume in kB}} = 2584kb$$

28.7.4. Preparing the disks

It's important to have exactly the same layout on the second disk, at least, when you use independent or non-compact dependent copies. Okay, to be more precise, the slices under the control of the point-in-time copy functionality has to have the same size. To simplify the fulfillment of this requirement, I copy the layout from my master disk to the shadow disk.

```
# prtvtoc /dev/dsk/c1d0s2 | fmthard -s - /dev/rdsk/c1d1s2
fmthard:  New volume table of contents now in place.
```

Okay, now let's create a file system for testing purposes on the master disk.

```
# newfs /dev/dsk/c1d0s3
newfs: construct a new file system /dev/rdsk/c1d0s3: (y/n)? y
Warning: 3376 sector(s) in last cylinder unallocated
/dev/rdsk/c1d0s3:       273104 sectors in 45 cylinders of 48
   tracks, 128 sectors
        133.4MB in 4 cyl groups (13 c/g, 39.00MB/g, 18624 i/g)
super-block backups (for fsck -F ufs -o b=#) at:
 32, 80032, 160032, 240032
```

Okay, as an empty filesystem is a boring target for point-in-time copies, we play around a little bit and create some files in our new filesystem.

```
# mount /dev/dsk/c1d0s3 /mnt
# cd /mnt
# mkfile 1k test1
# mkfile 1k test2
# mkfile 1k test3
# mkfile 1k test4
# mkfile 1k testindex1
# ls -l
total 26
drwx------   2 root     root        8192 Apr 25 18:10 lost+
   found
-rw------T   1 root     root        1024 Apr 25 18:10 test1
-rw------T   1 root     root        1024 Apr 25 18:11 test2
-rw------T   1 root     root        1024 Apr 25 18:11 test3
-rw------T   1 root     root        1024 Apr 25 18:11 test4
-rw------T   1 root     root        1024 Apr 25 18:11
   testindex1
```

Okay, that's all ... now let's try point-in-time copies.

28.8. Starting a Point-in-time copy

Okay, before using the merits of point-in-time copies, we have to configure such copies. The configuration of this copies is done with the with the `iiadm` command. In this part of the tutorial I will show you how to configure the different kinds of point-in-time copies.

28.8.1. Common prerequisite

At first we have to enable the Availability suite. This is independent from the method of doing the point-in-time copy. When you've used the AVS before, you don't need this step

```
# dscfgadm
Could not find a valid local configuration database.
Initializing local configuration database...
Successfully initialized local configuration database

If you would like to start using the Availability Suite
   immediately, you may
start the SMF services now.  You may also choose to start the
   services later
using the dscfgadm -e command.

Would you like to start the services now?  [y,n,?] y
```

Please answer the last question with `y` . By doing so, all the services of the AVS we need in the following tutorial are started (besides the remote replication service)

28.8.2. Create an independent copy

Now we can configure the point in time copy. This is really simple.

```
# iiadm -e ind /dev/rdsk/c1d0s3 /dev/rdsk/c1d1s3 /dev/rdsk/
   c1d1s4
```

That's all. What does this command mean: Create an independent copy of the data on the slice `/dev/rdsk/c1d0s3` on `/dev/rdsk/c1d1s3` and use `/dev/rdsk/c1d1s3` for the bitmap. As soon as you execute this command, the copy process starts. We decided to use an independent copy, thus we start a full copy of the master volume to the shadow volume. As long this fully copy hasn't completed, the point-in-time copy behaves like an dependent copy. Now we check the configuration.

```
# iiadm -i
/dev/rdsk/c1d0s3: (master volume)
/dev/rdsk/c1d1s3: (shadow volume)
/dev/rdsk/c1d1s4: (bitmap volume)
Independent copy
Latest modified time: Fri Apr 25 18:16:59 2008
Volume size:   273105
Shadow chunks total: 4267 Shadow chunks used: 0
Percent of bitmap set: 0
        (bitmap clean)
```

The highlighted part is interesting. The bitmap is clean. This means, that there are no changes between the master and the shadow volume.

28.8.3. Create an independent copy

Creating an dependent copy is quite easy,too . You have just alter the command a little bit, you've used to create independent one.

```
# iiadm -e dep /dev/rdsk/c1d0s3 /dev/rdsk/c1d1s3 /dev/rdsk/
   c1d1s4
```

Just substitute the `ind` with the `dep` and you get a dependent copy.

```
# iiadm -i
/dev/rdsk/c1d0s3: (master volume)
/dev/rdsk/c1d1s3: (shadow volume)
/dev/rdsk/c1d1s4: (bitmap volume)
Dependent copy
Latest modified time: Sat Apr 26 23:50:19 2008
Volume size:   273105
Shadow chunks total: 4267 Shadow chunks used: 0
Percent of bitmap set: 0
        (bitmap clean)
```

28.8.4. Create an compact independent copy

How do you get a compact dependent copy? Well, there is no command to force the creation of such a copy. But it's quite easy to get one. When the shadow volume is smaller than the master volume, the system chooses the compact independent copy automatically. We've created two small slices, when we formatted the harddrive. One of the small slices is `/dev/rdsk/c1d1s6`. Let's use it as the shadow volume. This volume

has only a size 32 MB while the master volume is 256 MB large.At first we create an dependent copy again, but with different volumes:

```
# iiadm -e dep /dev/rdsk/c1d0s3 /dev/rdsk/c1d1s6 /dev/rdsk/
   c1d1s4
```

Now let's check the status of the point-in-time copy configuration:

```
# iiadm -i
/dev/rdsk/c1d0s3: (master volume)
/dev/rdsk/c1d1s6: (shadow volume)
/dev/rdsk/c1d1s4: (bitmap volume)
Dependent copy, compacted shadow space
Latest modified time: Sat Apr 26 23:55:05 2008
Volume size:   273105
Shadow chunks total: 1255 Shadow chunks used: 0
Percent of bitmap set: 0
        (bitmap clean)
```

Et voila, you've configured a compact dependent copy.

28.9. Working with point-in-time copies

We've created the point-in-time-copy in the last part of the tutorial, but this is only one half of the story. In this part, we will use the this feature for backup purposes. The procedures are independent from the chosen point-in-time copy mechanism.

Okay, let's play around with our point-in time copy. At first we check the filesystem on our master volume mounted at `/mnt`. Nothing has changed. The AVS doesn't touch the master volume. But now let's create some files.

```
# ls
lost+found  test1       test2       test3       test4
   testindex1
# touch test5
# touch test6
# mkfile 2k testindex2
```

We check our dependent copy again:

```
# iiadm -i
/dev/rdsk/c1d0s3: (master volume)
/dev/rdsk/c1d1s3: (shadow volume)
/dev/rdsk/c1d1s4: (bitmap volume)
Independent copy
```

```
Latest modified time: Fri Apr 25 18:16:59 2008
Volume size:   273105
Shadow chunks total: 4267 Shadow chunks used: 0
Percent of bitmap set: 0
        (bitmap dirty)
```

Please look at the highlighted part. The system detected the changes to the master volume and marked the changed block on the bitmap volumes. The bitmap is "dirty" now.

Okay, now let's use our copy. We create a mountpoint and mount our shadow volume at this mountpoint.

```
# mkdir /backup
# mount /dev/rdsk/c1d1s3 /backup
```

Just for comparison, we have a short look at our master volume again:

```
# cd /mnt
# ls
lost+found  test2       test4       test6       testindex2
test1       test3       test5       testindex1
```

Now we check our copy:

```
# cd /backup
# ls
lost+found  test1       test2       test3       test4
   testindex1
```

We see the state of the filesystem at the moment we've created the point-in-time copy. Please notice the difference. The files created after initiating the copy are not present in the shadow.

You can make everything you want with the filesystem on the shadow volume. You can even write to it. But for this tutorial, we will make a backup from it. Whatever happens with the master volume during this backup, the data on the shadow won't change. Okay, that's isn't so interesting for a few bytes, but important for multi-terabyte databases or filesystems.

```
# tar cfv /backup20080424.tar /backup
a /backup/ 0K
a /backup/lost+found/ 0K
a /backup/test1 1K
a /backup/test2 1K
a /backup/test3 1K
a /backup/test4 1K
```

```
a /backup/testindex1 1K
```

As you see, no `test5`, `test6` or `testindex2`. Okay, we have made our backup, now let's sync our copy.

```
# iiadm -u s /dev/rdsk/c1d1s3
```

That's all. What have we done. We told the AVS to update the shadow copy on `/dev/c1d1s3`. Whenever you specify a disk or volume directly, you use the name of the shadow volume. A master volume can have several shadow volumes, but there can be only one shadow on a volume. So the copy configuration can be specified with the shadow volume. The `-u s` tells AVS to do an update (not a full copy) to the slave (from the master). Okay, now let's check the copy again.

```
# iiadm -i
/dev/rdsk/c1d0s3: (master volume)
/dev/rdsk/c1d1s3: (shadow volume)
/dev/rdsk/c1d1s4: (bitmap volume)
Independent copy
Latest modified time: Fri Apr 25 19:30:19 2008
Volume size:   273105
Shadow chunks total: 4267 Shadow chunks used: 0
Percent of bitmap set: 0
        (bitmap clean)
```

Please look at the highlighted part again. The bitmap is clean again. The master and the shadow are in sync.

Okay, let's check it by mounting the filesystem.

```
# mount /dev/dsk/c1d1s3 /backup
# cd /backup
# ls -l
total 30
drwx------   2 root     root        8192 Apr 25 18:10 lost+
   found
-rw------T   1 root     root        1024 Apr 25 18:10 test1
-rw------T   1 root     root        1024 Apr 25 18:11 test2
-rw------T   1 root     root        1024 Apr 25 18:11 test3
-rw------T   1 root     root        1024 Apr 25 18:11 test4
-rw-r--r--   1 root     root           0 Apr 25 18:20 test5
-rw-r--r--   1 root     root           0 Apr 25 18:20 test6
-rw------T   1 root     root        1024 Apr 25 18:11
   testindex1
-rw------T   1 root     root        2048 Apr 25 18:20
   testindex2
```

It's the exact copy of the filesystem in the moment when you've initiated the copy.

Okay, now let's play again with our point-in-time copy. Let's create some files in our master volume:

```
# cd /mnt
# touch test7
# touch test8
# mkfile 3k testindex2
```

Please note, that I've overwritten the 2k sized version of `testindex2` with a 3k sized version. A quick check of the directories:

```
# ls /mnt
lost+found  test2       test4       test6       test8
   testindex2
test1       test3       test5       test7       testindex1
# ls /backup
lost+found  test2       test4       test6       testindex2
test1       test3       test5       testindex1
```

Okay, the directory are different. Now let's start the backup again.

```
# tar cfv backup20080425.tar /backup
a /backup/ 0K
a /backup/lost+found/ 0K
a /backup/test1 1K
a /backup/test2 1K
a /backup/test3 1K
a /backup/test4 1K
a /backup/testindex1 1K
a /backup/test5 0K
a /backup/test6 0K
a /backup/testindex2 2K
```

Okay, `test7` and `test8` didn't made it into the tarball, as they were created after updating the point-in-time copy. Furthermore we've tared the 2k version of `testindex2` not the 3k version. So you can backup a stable version of your filesystem, even when you modify your master volume during the backup.

Okay, now we can unmount the filesystem again.

```
# cd /
# umount /backup
```

After this we sync the slave volume with the master volume.

```
# iiadm -u s /dev/rdsk/c1d1s3
```

And when we compare the filesystems, they are identical again.

```
# mount /dev/dsk/c1d1s3 /backup
# ls /mnt
lost+found  test2       test4       test6       test8
   testindex2
test1       test3       test5       test7       testindex1
# ls /backup
lost+found  test2       test4       test6       test8
   testindex2
test1       test3       test5       test7       testindex1
```

You can play this game forever, but I will stop now, before it gets boring.

28.10. Disaster Recovery with Point-in-time copies

The process of syncing master and shadow is bidirectional. You can't not only update the shadow from the master, you can update the master from the shadow as well. This is a really neat feature for disaster recovery.

Let's assume, you tried a new version of your software. At first all is well, but a minute later the system is toast. Later you will find out, that there was a race condition in the new code, that only manifested on your production system. But you don't know this now. And to add insult to injury, your face go white after looking into the directory.

```
# ls -l /mnt/testindex*
-rw-r--r--   1 root     root           0 Apr 25 19:40 /mnt/
   testindex1
-rw-r--r--   1 root     root           0 Apr 25 19:41 /mnt/
   testindex2
```

The new code killed your ¡code¿testindex¡/code¿-files. Zero bytes. And you hear the angry guy or lady from customer support shouting your name. But you were cautious, you've created a point-in-time copy before updating the system.

So, calm down and recover before a customer support lynch mob reach your office with forks and torches. Leave the filesystem and unmount it.

```
# cd /
# umount /mnt
```

Now sync the master with the slave. Yes, the other way round.

```
# iiadm -u m /dev/rdsk/c1d1s3
Overwrite master with shadow volume? yes/no yes
```

Okay ... after a few moments the shell prompt appears again. Now you can mount it again.

```
# mount /dev/dsk/c1d0s3 /mnt
# cd /mnt
```

Let's check our work and check the ¡code¿testindex¡/code¿ files.

```
# ls -l /mnt/testindex*
-rw------T   1 root     root         1024 Apr 25 18:11 /mnt/
   testindex1
-rw------T   1 root     root         3072 Apr 25 19:33 /mnt/
   testindex2
```

Phew ... rescued ... and the lynch mob in front of your office throws the torches out of the window, directly on the car of the CEO (of course by accident ;))

28.11. Administration

Okay, there are several administrative procedures with the point-in-time copy functionality. I will describe only the most important ones, as I don't want to substitute the manal with this tutorial.

28.11.1. Deleting a point-in-time copy configuration

Okay, let's assume you used the following configuration so far:

```
# iiadm -l
dep /dev/rdsk/c1d0s3 /dev/rdsk/c1d1s6 /dev/rdsk/c1d1s4
```

It's really easy to delete this config. As I mentioned before, the name of the shadow volume clearly indicates a point-in-time copy configuration, as there can be only one configuration for any given shadow volume. So you use the name of the shadow volume to designate a configuration. Thus the command to delete the configuration is fairly simple:

```
# iiadm -d /dev/rdsk/c1d1s6
```

The `-d` tells `iiadm` to delete the config. When we recheck the current AVS configuration, the config for `/dev/rdsk/c1d1s6` is gone:

```
# iiadm -l
#
```

28.11.2. Forcing a full copy resync of a point-in-time copy

Whenever you are in doubt of the consistency of you point-in-time copy (flaky disks, you've swapped a disk) it may be sensible to force a full copy resync instead of coping only the changed parts. Lets assume the following config of an independent copy:

```
# iiadm -l
ind /dev/rdsk/c1d0s3 /dev/rdsk/c1d1s3 /dev/rdsk/c1d1s4
```

Again you use the name of the shadow volume to designate the configuration. You force the full copy resync with a single command:

```
# iiadm -c s /dev/rdsk/c1d1s3
```

When we check the status of the dependent copy, you will see that a full copy is in progress:

```
# iiadm -i
/dev/rdsk/c1d0s3: (master volume)
/dev/rdsk/c1d1s3: (shadow volume)
/dev/rdsk/c1d1s4: (bitmap volume)
Independent copy, copy in progress, copying master to shadow
Latest modified time: Sun Apr 27 01:49:21 2008
Volume size:   273105
Shadow chunks total: 4267 Shadow chunks used: 0
Percent of bitmap set: 69
        (bitmap dirty)
```

Let's wait for a few moments and check the status again:

```
# iiadm -i
/dev/rdsk/c1d0s3: (master volume)
/dev/rdsk/c1d1s3: (shadow volume)
/dev/rdsk/c1d1s4: (bitmap volume)
Independent copy
Latest modified time: Sun Apr 27 01:49:21 2008
Volume size:   273105
Shadow chunks total: 4267 Shadow chunks used: 0
Percent of bitmap set: 0
       (bitmap clean)
```

The full copy resync has completed.

28.11.3. Grouping point-in-time copies

Sometimes the data of an application is distributed over several disks. For example because you application is rather old can use only volumes sized at 2 Gigabytes each. When you want to make a consistent point-in-time copy of all volumes, you have to do it at the same time. To enable the admin to do so, you can group point-in-time copies. When you use the groupname, all members of the group get the commands at the same time.

Okay, let's assume we have an independent copy so far.

```
# iiadm -l
ind /dev/rdsk/c1d0s3 /dev/rdsk/c1d1s3 /dev/rdsk/c1d1s4
```

Now we want to configure another one for the volume `/dev/rdsk/c1d0s5` with `/dev/rdsk/c1d1s5` as the shadow volume and `/dev/rdsk/c1d1s6` as the bitmap volume.

At first we move the existing configuration into a group. I will name it `database` in my example but you could choose any other name for it.

```
# iiadm -g database -m /dev/rdsk/c1d1s3
```

With `-g` we designate the groupname and with `-m` we move the volume into the group. As usual we use the name of the shadow volume to designate the configuration.

Now we create the point-in-time copy of the second volume. But we will create it directly in the group. To do so, we need the `-g` switch.

```
# iiadm -g database -e dep /dev/rdsk/c1d0s5 /dev/rdsk/c1d1s5 /
   dev/rdsk/c1d1s6
```

Please notice, that we used a different copy mechanism for the point-in-time copy. The don't have to be identical in the group.

Let's check the state of our copies:

```
# iiadm -i
/dev/rdsk/c1d0s3: (master volume)
/dev/rdsk/c1d1s3: (shadow volume)
/dev/rdsk/c1d1s4: (bitmap volume)
Group name: database
Independent copy
Latest modified time: Sun Apr 27 01:49:21 2008
Volume size:   273105
Shadow chunks total: 4267 Shadow chunks used: 0
Percent of bitmap set: 0
        (bitmap clean)
```

```
--------------------------------------------------------------------------------

/dev/rdsk/c1d0s5: (master volume)
/dev/rdsk/c1d1s5: (shadow volume)
/dev/rdsk/c1d1s6: (bitmap volume)
Group name: database
Dependent copy
Latest modified time: Sun Apr 27 02:05:09 2008
Volume size:   273105
Shadow chunks total: 4267 Shadow chunks used: 0
Percent of bitmap set: 0
        (bitmap clean)
```

Now let's initiate a full copy resync on the group ¡code¿database¡/code¿:

```
# iiadm -c s -g database
```

When you check the state of your copies again, you will recognize that you initiated a full resync on both copies at the same time:

```
# iiadm -i
/dev/rdsk/c1d0s3: (master volume)
/dev/rdsk/c1d1s3: (shadow volume)
/dev/rdsk/c1d1s4: (bitmap volume)
Group name: database
Independent copy, copy in progress, copying master to shadow
Latest modified time: Sun Apr 27 02:08:09 2008
Volume size:   273105
Shadow chunks total: 4267 Shadow chunks used: 0
Percent of bitmap set: 42
        (bitmap dirty)
--------------------------------------------------------------------------------

/dev/rdsk/c1d0s5: (master volume)
/dev/rdsk/c1d1s5: (shadow volume)
/dev/rdsk/c1d1s6: (bitmap volume)
Group name: database
Dependent copy, copy in progress, copying master to shadow
Latest modified time: Sun Apr 27 02:08:09 2008
Volume size:   273105
Shadow chunks total: 4267 Shadow chunks used: 0
Percent of bitmap set: 40
        (bitmap dirty)
```

28.12. Conclusion

I hope I gave you some insight into this really interesting feature of Solaris and OpenSolaris. There are vast possibilities to use it in your daily use. It's not limited to disaster recovery or backups. One of my customers uses this tool to create independent copies of their database. They take a snapshot at midnight and export it on a different database server. The rationale for this process: They run some long running analytics with a huge load on the I/O system on this independent copy. By using the copy the analysis doesn't interfere with the production use of the database. Another customer uses this feature for generating test copies of their production data for testing new software versions. You see, the possibilities are vast and virtually endless.

28.13. Do you want to learn more?

Documentation

docs.sun.com: Sun StorageTek AVS 4.0 Point-in-Time Copy Software Administration Guide[1]

docs.sun.com: Manpage of iiadm[2]

Misc.

blogs.sun.com/AVS: The Blog of the Availability Suite[3]

[1] `http://docs.sun.com/source/819-6149-10/index.html`
[2] `http://docs.sun.com/source/819-6149-10/index.html`
[3] `http://docs.sun.com/source/819-6149-10/index.html`

29. SamFS - the Storage Archive Manager FileSystem

Solaris 10

29.1. Introduction

Okay, this is tutorial isn't really about feature of Solaris itself. But the feature of this tutorial it's deeply coupled with Solaris. Thus you can view it as an optional part of Solaris. This time I will dig into the installation and configuration of SamFS. But a warning, SamFS is a feature monster. This tutorial is equivalent to put your toes in the Atlantic ocean, but when I saw the announcement of the open-sourcing of SamFS I thought, it's time to write this document. In addition to that, it was a nice way to make a reality check on a thought game, I've made some months ago.

29.2. The theory of Hierarchical Storage Management

I think, I should start with the theory of Hierarchical Storage Management(HSM). The technology of managing data and their position on different media isn't really a widespread feature for most administrators. Thus this introduction will be a quite long one.

29.2.1. First Observation: Data access pattern

There is a single fact in data management, that was true the last 40 years and it will be true until we won't store data anymore. The longer you didn't access data, the lesser the probability of an access gets. This matches with daily observation: You work on a text document, you access it every few minutes, you finalize your work and store it on your hard disk. And the microsquaremeters of rotating rust with your document rotate with it. Perhaps you access it a few times in the next weeks to print it again. But after a year it get more and more improbable, that you will access the data. The problems: 1. Murphys Law: A nanosecond can be defined as the time between the time of deletion of

a file and the moment the boss comes into your office wanting the document. 2. In most countries you will find regulations who prohibits the deletion of a file.

29.2.2. Second observation: The price of storage

When you look at the prices for storage, you will see that the price of it will rise with its speed. An enterprise-class 15k rpm FC drive is an expensive device. An enterprise-class 7.2k rpm SATA disk is much cheaper but you can't operate it 24h near of its full load. A LTO4 tape can store 800 GB for 60-70 Euro, whereas even a cheap consumer disk with 750 GB costs round about 100 Euro. The disadvantage: It takes quite a time to get the first byte. You have to load it, wind it into the drive, wind it to the correct position.

29.2.3. Third observation: Capacity

This observation is so simple and so obvious, I won't have to explain it: The amount of data to store never gets smaller. When I remember my first PC, purchased by my dad, it had an 20 Megabyte harddisk. And I thought: Thats pretty much. Today, I need this capacity to store 2 raw images from my DSLR. And in company it's the same. Independently from the provided space, it isn't enough. I know some companies where small amounts of free space on a departmental server is a subject of interdepartment quid-pro-quo deals.

29.2.4. Hierarchical Storage Management

The basic idea of hierarchical storage management leverages these observations. It makes sense to store actual data on the fastest storage available, but it doesn't make sense to use it for data, that didn't have accessed for a year or so. The other way round: It will drive you mad, if actual your text document is stored on a DLT-Tape in a big autoloader, needing 2 minutes to stream the first byte to the client.

Hierarchical Storage Management can use a multitude of storage devices with different access behavior to store your data. You can establish an hierarchy of storage systems. You can use ultra fast FC disks for data in regular usage, cheaper SATA disks for files you need a few times a week, tapes for data needed a few times a year. The kicker behind HSM: The inner workings of HSM are invisible for the user or the application. You just use the file, and the system gathers it from other media.

29.2.5. An analogy in computer hardware

Most of you already use hierarchical mechanism for storing data. It's the hierarchy of memory in your computer. You have ultrafast memory in your processor called registers, then you have an almost as fast memory called cache (most of the time you have several stages like Level 1 to 3 Caches), then you have a much slower main memory. At the end you have the swap space on your harddisk, much slower again than the main memory. HSM does the same for storage.

The advantages and the challenges are pretty much the same: By using the hierarchy in an intelligent manner, you can speed up the access to your data without spending to much for buying only the fastest memory. The challenge: You have to find a way to find the best place for your data, and you have to carefully size the amount of the stages. When it's to small, access times get longer, as you have to access a slower storage/memory.

29.2.6. SamFS

SamFS is an implementation of this concept. It isn't the only one, but from my view it's the best implementation in the unix world. SamFS stands for *S*torage *A*rchive *M*anager *F*ile *S*system. It's an fully POSIX compliant file system, thus an user or an application won't see a different to an UFS for example, with a rich feature set. I would suggest, that you look at the Sun Website for the Sun StorageTek SamFS website for an overview.

29.3. The jargon of SamFS

As usual, this technology has its own jargon. Thus I will start to define the most important words at first.

29.3.1. Lifecycle

Before defining the jargon, it's important to understand, that every file under the control of SamFS follows a certain lifecycle. You create or modify it, the system archives it, after a certain time without an access the system removes it from expensive storage, when it has copies on cheaper ones, when you access it, it will be gathered from the cheaper storage and delivered to you. When you delete it, you have to remove it from all your medias. This cycle is endless until a file is deleted.

29.3.2. Policies

Albeit every file is under the control of the described cycle, the exact life of a file doesnt have to be the same for every file. SamFS knows the concept of policies to describe the way, SamFS should handle a file. How many copies should SamFS make of a file on which media. The most difficult task of configuring SamFS is to find a most adequate policy. You need experience for it, but it's something that you can easily learn on the job.

29.3.3. Archiving

Okay, the first step is archiving. Let's assume you've created a file. The data gets stored into the SamFS filesystem. Okay, but you've defined a policy, that you want two copies on a tape media. The process that does this job is called *archiver*, the process itself is called *archiving*. Archiving moves your files to the desired media. The metadata of the files is augmented with the positions of the file. SamFS can create up to 4 copies of a file. Important to know: SamFS doesn't wait with the archiving process until it needs space on the cache media. It starts the process of archiving files with the next run of the archive (for example every 5 minutes)

29.3.4. Releasing

Okay, let's assume you filesystem is 90% full. You need some space to work. Without SamFS you would move around the data manually. SamFS works similar and differently at the same time. The archiver already moved your data to different places. Thus releasing is the process to delete the data from your filesystem. But it doesn't delete all of it. It keeps a stub of it in the filesystem. This process is called releasing. The metadata (filename, acl, ownership, rights, and the start of the file) stays on disk. Thus you won't see a difference. You can walk around in your directories and you will see all your files. The difference: The data itself isn't in the filesystem anymore, thus it don't consume space in it.

29.3.5. Staging

Okay, after long time (the file was already released) you want to access the data. You go into the filesystem, and open this file. SamFS intercepts this call, and automatically gathers the data from the archive media. In the meantime the reads from this file will be blocked, thus the process accessing the data blocks, too. SamFS uses informations from the metadata to find the media.

29.3.6. Recycling

Okay, the end of the lifetime of a file is its deletion. That's easy for disks. But you can't delete a single file from tape in an efficient manner. Thus SamFS uses a different method: The data on the tape is just marked as invalid, the stub gets deleted. But the data stays on tape. After a while more and more data may get deleted from tape. This may end in a swiss cheese where only a small amount of data is actual data. This would be waste of tape and the access pattern gets slower and slower. Recycling solves this by a single trick. The residual active data gets a special marker. When the archiver runs the next time, the data gets archived again. Now there is no actual data left on the tape. You can erase it by writing a new label to it and you can use it for new data again. This process is called recycling.

29.3.7. The circle of life

Okay, with this jargon we can draw a picture of this processes.

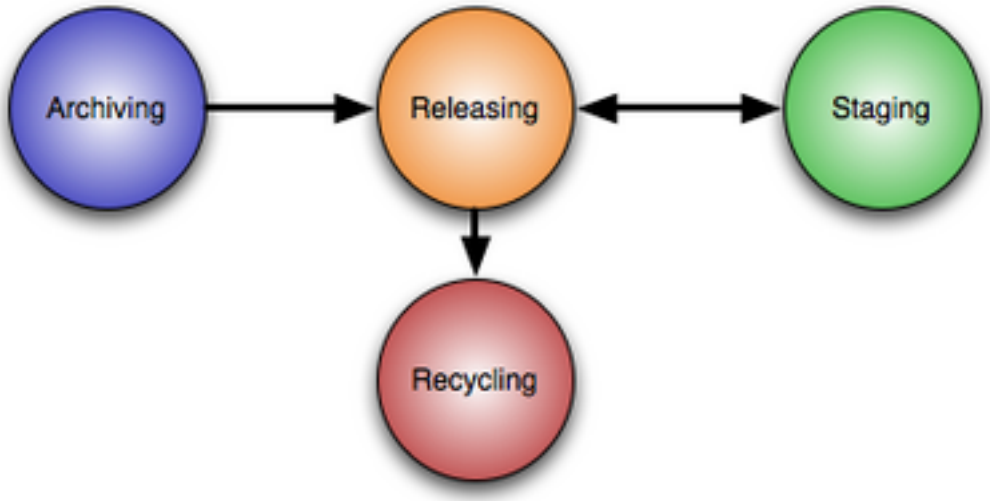

Figure 29.1.: Simplified lifecycle of a file in SamFS

Once a file gets newly written or updated, it gets archived. Based on a combination policies, usage and the caching strategy it's possible it's getting released and staged again and again. And at the end, the tape with the data will be recycled.

29.3.8. Watermarks

Watermarks are an additional, but very important concept in SamFS. The cache is much smaller than the filesystem . Nevertheless you have to provide space for new and updated data. So SamFS implements two important watermarks: Then the cache gets filled to the *high* watermark, the system starts to release the least recently used files with a minimum number of copies on archive media automatically. This process stops, when the *low* water mark is reached. Thus you can ensure that you have at least a certain amount of free capacity to store new or updated data in the filesystem.

29.3.9. The SamFS filesystem: Archive media

When used in conjunction with the Archiver/Stager/Releaser construct, the SamFS filesystem itself isn't much more than a cache for all the data you store in this filesystem. Not the size of the SamFS filesystem is decisive for the size of your file system, the amount of archive media is the limitation of the size. For example. With a 1 GB disk cache and 10 petabyte of T10000 tapes, you can store up to 5 petabyte of data. Why 5 petabyte? Well, it's a best practice to store two copies of every file on your system, just in case a tape gets lost or damaged.

Archive media can be of different nature:

- disk drives
- other SamFS Servers
- tapes (with or without autoloading)
- magneto optical disks (with or without autoloading)

The media doesn't even have to be in reach of an autoloader. SamFS knows the concept of offlined archive media, for example tapes in a safe. When you try to access data on an offlined media, the accessing process blocks and the admin is notified to move its a.. to put the tape into a drive.

29.4. Installation of SamFS

It's quite easy to install SamFS. It's just installing two packages and answering some questions.

29.4.1. Obtaining the binaries

At first, you have to download the iso image with the binaries for SamFS 4.6. You can obtain them at the Sun Download Center. You need an account for the download center, but you can register for free. At the moment you are allowed to test it for 60 days, but you don't need to obtain a test license as there is no license enforcement in the software. Instead of this, this is handled with right-to-use licensing.

29.4.2. Installing the SamFS packages

Let's start the installation of SamFS:

```
# cd sunstorageteksam-fs4.6/
# cd x64
# cd 2.10
# pkgadd -d . SUNWsamfsr SUNWsamfsu</b>

Processing package instance <SUNWsamfsr> from </cdrom/
   sunstorageteksam-fs4.6/x64/2.10>

Sun SAM and Sun SAM-QFS software Solaris 10 (root)(i386) 4.6.5,
   REV=5.10.2007.03.12

         Sun SAMFS - Storage & Archiving Management File System

         Copyright (c) 2007 Sun Microsystems, Inc.

         All Rights Reserved.


       ----------------------------------------------------

In order to install SUNWsamfsr, you must accept the terms of
the Sun License Agreement.
Enter "y" if you do, "n" if you don't, or "v" to view agreement
   . y

- The administrator commands will be executable by root only (
   group bin).

If this is the desired value, enter "y".  If you want to change
the specified value enter "c". y
```

```
By default, elrond is not setup to be remotely managed by
File System Manager. It can only be managed by the File System
Manager if it is installed locally
You can modify the remote management configuration at a later
   time
using the command fsmadm

If you want to keep the default behavior, enter "y". Otherwise
   enter "n". y
## Processing package information.
## Processing system information.
   20 package pathnames are already properly installed.
## Verifying disk space requirements.
## Checking for conflicts with packages already installed.

The following files are already installed on the system and are
    being
used by another package:
  /etc/opt <attribute change only>
  /var/opt <attribute change only>

Do you want to install these conflicting files [y,n,?,q] y
## Checking for setuid/setgid programs.

This package contains scripts which will be executed with super
   -user
permission during the process of installing this package.

Do you want to continue with the installation of <SUNWsamfsr> [
   y,n,?] y

Installing Sun SAM and Sun SAM-QFS software Solaris 10 (root)
   as <SUNWsamfsr>

## Executing preinstall script.


## Installing part 1 of 1.
/etc/fs/samfs/mount
[...]
/var/svc/manifest/application/management/fsmgmt.xml
[ verifying class <none> ]
/opt/SUNWsamfs/sbin/samcmd <linked pathname>
## Executing postinstall script.
```

```
The administrator commands are executable by root only.

-------------------------------------------------------------
-                    PLEASE READ NOW !!!                    -
-------------------------------------------------------------

If you are upgrading from a previous release of SAM and
have not read the README file delivered with this release,
please do so before continuing.  There were significant
restructuring changes made to the system from previous
releases. Failure to convert scripts to conform to these
changes could cause dramatic changes in script behavior.


Installation of <SUNWsamfsr> was successful.

Processing package instance <SUNWsamfsu> from </cdrom/
   sunstorageteksam-fs4.6/x64/2.10>

Sun SAM and Sun SAM-QFS software Solaris 10 (usr)(i386) 4.6.5,
   REV=5.10.2007.03.12

        Sun SAMFS - Storage & Archiving Management File System

        Copyright (c) 2007 Sun Microsystems, Inc.

        All Rights Reserved.

## Executing checkinstall script.
## Processing package information.
## Processing system information.
   10 package pathnames are already properly installed.
## Verifying package dependencies.
## Verifying disk space requirements.
## Checking for conflicts with packages already installed.
## Checking for setuid/setgid programs.

This package contains scripts which will be executed with super
   -user
permission during the process of installing this package.

Do you want to continue with the installation of <SUNWsamfsu> [
   y,n,?] y
```

```
Installing Sun SAM and Sun SAM-QFS software Solaris 10 (usr) as
    <SUNWsamfsu>

## Installing part 1 of 1.
/opt/SUNWsamfs/lib/amd64/libsamconf.so <symbolic link>
[...]
/usr/sfw/bin/tapealert_trap
[ verifying class <none> ]
## Executing postinstall script.

Configuring samst devices.  Please wait, this may take a while.

Adding samst driver
Building samst devices
Issuing /usr/sbin/devfsadm -i samst >> /tmp/SAM_install.log
   2>&1

Adding samioc driver

Adding samaio driver

File System Manager daemon is configured to auto-restart every
   time the daemon dies

Starting File System Manager daemon


Please check the log files for any errors
that were detected during startup

Installation of <SUNWsamfsu> was successful.
```

29.4.3. Installing the SamFS Filesystem Manager

Now we install the FilesystemManager:

```
# cd /cdrom/sunstorageteksam-fs4.6/
# cd x64
# ./fsmgr_setup </b>

Unzipping File System Manager files...
This process may take a while ...
```

```
Start cleaning up to prepare for new software installation
Start installing File System Manager packages...
This process may take a while ...

Processing package instance <SUNWfsmgrr> from </tmp/
   File_System_Manager/2.10/i386>

File System Manager Solaris 10 (root)(i386) 3.0.4,REV
   =5.10.2007.03.01
## Executing checkinstall script.

        Sun SAMFS - Storage & Archiving Management File System

        Copyright (c) 2007 Sun Microsystems, Inc.

        All Rights Reserved.

## Processing package information.
## Processing system information.
   1 package pathname is already properly installed.
## Verifying package dependencies.
## Verifying disk space requirements.

Installing File System Manager Solaris 10 (root) as <SUNWfsmgrr
   >

## Executing preinstall script.
Shutting down Sun Java(TM) Web Console Version 3.0.3 ...
The console is stopped
## Installing part 1 of 1.
/opt/SUNWfsmgr/bin/fsmgr
[...]
/opt/SUNWfsmgr/samqfsui/xsl/svg/storagetier.xsl
[ verifying class <none> ]
## Executing postinstall script.
Extracting online help system files...Archive:  en_samqfsuihelp
   .zip
   creating: en/help/
[...]
  inflating: en/help/stopwords.cfg
done

    Warning: smreg is obsolete and is preserved only for
```

```
    compatibility with legacy console applications. Use wcadmin
        instead.

    Type "man wcadmin" or "wcadmin --help" for more information
       .


    Registering fsmgrAdmin_3.0.


    Warning: smreg is obsolete and is preserved only for
    compatibility with legacy console applications. Use wcadmin
        instead.

    Type "man wcadmin" or "wcadmin --help" for more information
       .

    Registering /opt/SUNWfsmgr/samqfsui/WEB-INF/lib/fsmgmtjni.
       jar
    as fsmgmtjni.jar for scope fsmgrAdmin_3.0

    Enabling logging...

    Warning: smreg is obsolete and is preserved only for
    compatibility with legacy console applications. Use wcadmin
        instead.

    Type "man wcadmin" or "wcadmin --help" for more information
       .


Installation of <SUNWfsmgrr> was successful.

Processing package instance <SUNWfsmgru> from </tmp/
   File_System_Manager/2.10/i386>

File System Manager Solaris 10 (usr)(i386) 3.0.4,REV
   =5.10.2007.03.01
## Executing checkinstall script.

        Sun SAMFS - Storage & Archiving Management File System

        Copyright (c) 2007 Sun Microsystems, Inc.
```

```
        All Rights Reserved.

## Processing package information.
## Processing system information.
   2 package pathnames are already properly installed.
## Verifying package dependencies.
## Verifying disk space requirements.

Installing File System Manager Solaris 10 (usr) as <SUNWfsmgru>

## Installing part 1 of 1.
/usr/lib/libfsmgmtjni.so
/usr/lib/libfsmgmtrpc.so
[ verifying class <none> ]
## Executing postinstall script.
Current session timeout value is 15 minutes, change to 60
   minutes...Set 1 properties for the console application.
done
Starting Sun Java(TM) Web Console Version 3.0.3 ...
The console is running
Appending elrond into /var/log/webconsole/host.conf ...done!

Installation of <SUNWfsmgru> was successful.
Done installing File System Manager packages.


Backing up /etc/security/auth_attr to /etc/security/auth_attr.
   old
Start editing /etc/security/auth_attr ...
Done editing /etc/security/auth_attr

Backing up /etc/user_attr to /etc/user_attr.old
Start editing /etc/user_attr ...
Start editing /etc/user_attr ...
Done editing /etc/user_attr
File System Manager 3.0 and its supporting application
is installed successfully.


****************** PLEASE READ
   **********************************

Please telnet to each Sun StorEdge(TM) QFS servers to be
managed and run the following command:
```

```
    /opt/SUNWsamfs/sbin/fsmadm status


This will determine if the File System Manager daemon is
   running.

If it is not running, please run the following command:

    /opt/SUNWsamfs/sbin/fsmadm config -a


This command will start the File System Manager daemon that
   communicates with
the File System Manager. Failure to do so will prevent File
   System Manager
from communicating with the Sun StorEdge QFS servers.

Remote access to the servers used by the File System Manager is
    now
restricted based on host name or IP address.  If you are
   managing
a Sun StorEdge(TM) QFS Server from a remote management station,
please telnet to the server and run the following command:

    /opt/SUNWsamfs/sbin/fsmadm add <
       management_station_host_name>.<domain>
Press ENTER to continue ...

File System Manager 3.0 supports the following browsers:

  Browser Type                              Operating System
  ========================================================================

  Netscape 7.1/Mozilla 1.7/Firefox 1.5  Solaris OS,
                                        MS Windows 98SE, ME,
                                           2000, and XP
  Internet Explorer 6.0                 MS Windows 98SE, ME,
     2000, and XP

Now launch your web browser and type the following URL:
     https://<hostname>.<domain>:6789

where <hostname> is the host that you have just installed the
```

```
File System Manager.

If you are served with a security related certificate, go ahead
    and
accept it.  Please see user docs for username and password
   details.


It is required to clear the browser cache before accessing the
File System Manager for the first time.  Failure to do so may
   cause
unexpected behavior in various pages.


File System Manager 3.0 has been tested with the
Sun Java(TM) Web Console version 2.2.5 & 3.0.2.
Installing this product with any older Sun Java(TM) Web Console
    version breaks both
applications. This product may work on newer Sun Java(TM) Web
   Console
versions, but this has not been tested.

****************************************************************




Install/Uninstall log file named /var/tmp/fsmgr.setup.log
   .03.23.2008.11:01 is created.
```

29.4.4. Modifying the profile

You have to extend the profile to for additional search paths for binaries and man pages.

```
PATH=$PATH:/opt/SUNWsamfs/bin:/opt/SUNWsamfs/sbin
MANPATH=$MANPATH:/opt/SUNWsamfs/man
export PATH MANPATH
```

That's all about the installation of SamFS.

29.5. The first Sam filesystem

Okay, now we can configure our first filesystem with SamFS.

29.5.1. Prerequisites

Before we can configure SamFS, I want to describe the prerequisites for this task: We need some harddisks for this task. I made my example a little bit more complex, thus I used iSCSI volumes for this task.

I created for this tutorial:

- a 64 MB emulated volume for the storage of metadata
- a 512 MB emulated volume for the filesystem itself
- a 2 GB emulated volumes to use them as archive disks

I assume that you already know the tasks to creating them from the iSCSI tutorial.

The first and the second volume will be used by SamFS directly. You have to use the `format` command to put a label and a partition table on it.

For the both archive volumes, we will use ZFS. Thus I've created a zpool for both:

```
# zpool create samfs_archive_1
   c1t0100001C42E9F21A00002A0047E54035d0
# zpool create samfs_archive_2
   c1t0100001C42E9F21A00002A0047E54036d0
```

29.5.2. The configuration itself

Okay, let's start to create a filesystem. We have to tell SamFS the structure of our filesystem. This is done in the file `/etc/opt/SUNWsamfs/mcf`.

```
samfs1              10     ma    samfs1   -
/dev/dsk/c1t0100001C42E9F21A00002A0047E6642Bd0s0   11     mm
   samfs1   -
/dev/dsk/c1t0100001C42E9F21A00002A0047E54033d0s0   12     md
   samfs1   -
```

Okay, let's dissect this file. At first I want to explain the general meaning of the columns.

- The first column of this file is the equipment identifier. This field serves multiple purposes. You define filesystems, tape drives, disk partitions for the cache or tape robotics in this file. Please note: You do not define media for disk archiving here!
- The second column is the equipment ordinal. This field enumerates every component defined in the `mcf` file. This number has to be unique.
- The third column is the equipment type. SamFS supports a vast amount of device type. You define it by using its shorthand here. `ma`stands for a SamFS/QFS cache disk set with one or more dedicated metadevices.`mo` for example designates an 5 1/4 inch erasable optical drive.
- The forth column is the family set. With the family set name you group devices. For example all disks of a filesystem. All disks of a filesystem have the same name. Another example is the grouping of a tape robotic and all of its tape drive
- The fifth column is the device state.

Okay, what did we describe with our `mcf` file: We defined an filesystem with the name `samfs1`. The name of the family set is `samfs1` as well. The filesystem is of the type "SamFS disk cache with dedicated metadevices. In the next row we've defined that the device `/dev/dsk/c1t0100001C42E9F21A00002A0047E6642Bd0s0` is a device solely for metadata. We gave it the ordinal number 11 and it's part of the `samfs` family, thus a part of the filesystem defined before. The third line configures the `/dev/dsk/c1t0100001C42E9F21A00002A0047E54033d0s0` as the data disk for this filesystem (as the family name is `samfs1` as well.

Yes, you have read it correctly. SamFS is capable to separating the metadata and the data of the files on different disks. The idea behind this concept is to use fast disks for the metadata (e.g. solid state disks) with short access times and slower disks for the data. By this separation the filesystem has doesn't have to step between the position of the metadata and the position of the data when it's updated. The effect: Much better scaling when you use a large amount of disks.

Okay, now we have fully configured the filesystem. Now we modify the `/etc/vfstab` to enable simpler mounting/auto mounting at start.

The device name is the name of the filesystem, in your case `samfs1`. It don't have a raw device. The mountpoint is `/samfs1`, the type samfs and we want to mount it at the start. The options are SamFS specific. They mean: Start to release files (thus freeing space in the cache) when the cache is 80 percent full. Release until the cache is filled only 60 percent.

```
samfs1      -         /samfs1 samfs   -     yes       high=80,low=60
```

Okay, we have to tell SamFS that there is a new configuration:

```
bash-3.00# samd config
Configuring SAM-FS
bash-3.00#
```

Looks good. Now we are able to format the filesystem:

```
# sammkfs samfs1
Building 'samfs1' will destroy the contents of devices:
                /dev/dsk/
                   c1t0100001C42E9F21A00002A0047E6642Bd0s0
                /dev/dsk/
                   c1t0100001C42E9F21A00002A0047E54033d0s0
Do you wish to continue? [y/N]y
total data kilobytes       = 520192
total data kilobytes free  = 520128
total meta kilobytes       = 61440
total meta kilobytes free  = 61024
```

Let's try to mount it.

```
# mkdir /samfs1
# mount /samfs1
# cd /samfs1
```

You should see the obligatory `lost+found` now. But let's do an deeper check of the filesystem:

```
bash-3.00# samfsinfo samfs1
samfsinfo: filesystem samfs1 is mounted.
name:     samfs1       version:     2
time:     Sun Mar 23 15:46:40 CET 2008
count:    2
capacity:       000000000007f000          DAU:          64
space:          0000000000070c40
meta capacity: 000000000000f000          meta DAU:     16
meta space:    000000000000aa80
ord  eq           capacity              space   device
  0  11   000000000000f000   000000000000aa80   /dev/dsk/
     c1t0100001C42E9F21A00002A0047E6642Bd0s0
  1  12   000000000007f000   000000000007efc0   /dev/dsk/
     c1t0100001C42E9F21A00002A0047E54033d0s0
```

Et voila, your first SamFS filesystem.

29.6. Using disk archiving

Okay, we will use the simplest case of archiving. It's disk archiving. In this configuration SamFS doesn't use a tape drive for storing data, it simply uses filesystems, or to be more precise, it uses a directory in a filesystem to store its data.

29.6.1. Prerequisites

Okay, I've created two iSCSI-base diskpool at the start to use them as disk archives. Now I will put some further separation in them by creating directories in it.

```
# mkdir /samfs_archive_1/dir1
# mkdir /samfs_archive_1/dir2
# mkdir /samfs_archive_2/dir2
# mkdir /samfs_archive_2/dir1
```

29.6.2. Configuring the archiver

So, we have to start to configure the archive medias. Every archive media has its `VSN`. VSN is a shorthand for Volume Serial Name. The VSN identifies a media of your archive. In this case we assign the VSN `disk01` with the directory `/samfs_archive_1/dir1/`

```
disk01   /samfs_archive_1/dir1/
disk02   /samfs_archive_1/dir2/
disk03   /samfs_archive_2/dir1/
disk04   /samfs_archive_2/dir2/
```

Now we have usable devices for archiving. But have to configure the archiving as the next step. In this step we define the policies for archiving, control the behavior of he archiver and associate VSNs with archive sets. All this configuration takes place in the file `/etc/opt/SUNWsamfs/archiver.cmd`.

Okay, lets create such a config file for our environment.

```
logfile = /var/opt/SUNWsamfs/archiver/log
interval = 2m
```

Okay, this is easy: The archiver should log its work into the file `/var/opt/SUNWsamfs/archiver/log`. This file is really interesting. I will show you a nifty trick with it later in this tutorial.

The `interval` directive was responsible for defining the interval between the starts of a process for finding new or updated files (`sam-arfind`). This behavior didn't scaled very well with millions of files in a directory.

Today the default is different. The file system itself knows what files have been updated and SamFS stores this information in a list. Today this setting has a similar effect, but with other methods: It's the default setting for the archive aging. It defines the amount of time between the first file added to the todo list for the archiver and the start of the archive. Thus the archiving would start two minutes after adding the first file to the list.

Now we define the archiving policy for the filesystem:

```
fs = samfs1
arset0 .
       1 30s
       2 1200s
```

What does this mean? `arset0` is a name of a so called archive set. The contents of this set are defined later-on. The `.` stands for "every file in the filesystem". Okay, now we tell SamFS to make a first copy to the archiveset `arset0.1` after 30 seconds. The second copy is made to the archiveset `arset0.1` after 1200 seconds (20 minutes). We have just used the name of some archive sets, now we have to declare them:

```
vsns
arset0.1 dk disk01
arset0.2 dk disk03
samfs1.1 dk disk02
endvsns
```

Okay, The translation is quite simple: The archiveset `arset0.1` consists is a disk based set and consists out of the VSN `disk01`. The same for the archive set `arset0.2`. But wait, we didn't used an archiveset `samfs1.1` so far. Well, you haven't defined it explicitly. But it's implicit when you have an archiver configuration for an filesystem. It's the default archive set. You can use it for regular archiving, but as we haven't defined a policy to do so, this archive set is used for storing the meta data of your filesystem. So the association of a VSN to this archive set is mandatory.

So we end up with the following `archiver.cmd`

```
logfile = /var/opt/SUNWsamfs/archiver/log
interval = 2m

fs = samfs1
arset0 .
       1 30s
       2 1200s

vsns
arset0.1 dk disk01
```

```
arset0.2 dk disk03
samfs1.1 dk disk02
endvsns
```

Okay, we've finalized our configuration: Now we have to check the configuration:

```
bash-3.00# archiver -lv
Reading '/etc/opt/SUNWsamfs/archiver.cmd'.
1: logfile = /var/opt/SUNWsamfs/archiver/log
2: interval = 2m
3:
4: fs = samfs1
5: arset0 .
6:       1 30s
7:       2 1200s
8:
9: vsns
10: arset0.1 dk disk01
11: arset0.2 dk disk03
12: samfs1.1 dk disk02
13: endvsns
No media available for default assignment
Notify file: /etc/opt/SUNWsamfs/scripts/archiver.sh
Read timeout: 60
Request timeout: 15m
Stage timeout: 15m

Archive media:
media:dk bufsize: 4 archmax:  512.0M Write timeout: 15m

Archive libraries:
Device: disk archive_drives: 3
  Dictionary:
  dk.disk01               capacity:    2.0G space:    1.9G
  dk.disk02               capacity:    2.0G space:    1.9G
  dk.disk03               capacity:    2.0G space:    1.9G
  dk.disk04               capacity:    0    space:    0


Archive file selections:
Filesystem samfs1  Examine: noscan  Interval: 2m
  archivemeta: on  scanlistsquash: off  setarchdone: off
  Logfile: /var/opt/SUNWsamfs/archiver/log
samfs1   Metadata
  copy: 1  arch_age: 4m
```

```
arset0   path: .
  copy: 1  arch_age: 30
  copy: 2  arch_age: 20m


Archive sets:
[...]

arset0.1
  media: dk
 Volumes:
   disk01 (/samfs_archive_1/dir1/)
 Total space available:    1.9G

arset0.2
  media: dk
 Volumes:
   disk03 (/samfs_archive_2/dir1/)
 Total space available:    1.9G

samfs1.1
  media: dk
 Volumes:
   disk02 (/samfs_archive_1/dir2/)
 Total space available:    1.9G
bash-3.00#
```

Now we tell SamFS to reread its config-files

```
bash-3.00# samd config
Configuring SAM-FS
bash-3.00#
```

And now we have a running archiver. So ... let's have some fun with it. Copy some files in it. I tend do test it by making a recursive copy of the `/var/sadm/pkg` directory. Now let's look onto our archival disks:

```
bash-3.00# ls -l /samfs_archive_1/dir1
total 40734
-rw-------   1 root     root          56 Mar 23 19:39 diskvols.
   seqnum
-rw-------   1 root     root     19608576 Mar 23 17:43 f0
-rw-------   1 root     root     1049088 Mar 23 19:39 f1
bash-3.00# ls -l /samfs_archive_1/dir2
total 13593
```

```
-rw-------   1 root     root           56 Mar 23 19:42 diskvols.
   seqnum
-rw-------   1 root     root      6891520 Mar 23 17:42 f0
-rw-------   1 root     root         4608 Mar 23 19:42 f1
bash-3.00# ls -l /samfs_archive_2/dir1
total 40736
-rw-------   1 root     root           56 Mar 23 19:58 diskvols.
   seqnum
-rw-------   1 root     root      19608576 Mar 23 17:43 f0
-rw-------   1 root     root      1049088 Mar 23 19:58 f1
```

You see, your archival media starts to populate. But where are your files, and what's up with this `f1`. Well, they are written in a very specific, very secret and very closed format: These files are simple tar files. SamFS uses the standard tar format to write the archive file.You can look in it with the standard tar or the tar of SamFS:

```
bash-3.00# star tfv f1
-rw------T root/root   1048576 2008-03-23 19:28 testfile3
```

Please notice, that this isn't a version of Joerg Schilling's star despite of the name.

29.7. Working with SamFS

Now we've configured a file system and we set up the archiver for it. Now let's use it.

29.7.1. Looking up SamFS specific metadata

At first let's create a test file.

```
# mkfile 10m /samfs1/testfile3
```

We now look at the metadata of this file. There is a special version of `ls` that is capable to read the additional information. This version `ls` is called `sls`. So let's check for our test file.

```
[root@elrond:/samfs1]$ sls -D testfile3
testfile3:
  mode: -rw------T  links:   1  owner: root       group: root
  length:   1048576  admin id:       0  inode:      4640.1
  access:      Mar 23 19:28  modification: Mar 23 19:28
  changed:     Mar 23 19:28  attributes:   Mar 23 19:28
  creation:    Mar 23 19:28  residence:    Mar 23 19:28
```

There is nothing new. Okay, let's leave the computer alone, drink a coffee or two, and now we check again:

```
bash-3.00# sls -D testfile3
testfile3:
  mode: -rw------T  links:   1  owner: root      group: root
  length:   1048576  admin id:      0  inode:     4640.1
  archdone;
  copy 1: ----- Mar 23 19:39         1.1    dk disk01 f1
  copy 2: ----- Mar 23 19:58         1.1    dk disk03 f1
  access:      Mar 23 19:28  modification: Mar 23 19:28
  changed:     Mar 23 19:28  attributes:   Mar 23 19:28
  creation:    Mar 23 19:28  residence:    Mar 23 19:28
```

I assume you've already noticed the three additional lines. The archiver did its job:

```
  archdone;
  copy 1: ----- Mar 23 19:39         1.1    dk disk01 f1
  copy 2: ----- Mar 23 19:58         1.1    dk disk03 f1
```

The first line says, that all outstanding archiving for the file is done. The two next lines tells you where the copies are located, when they were archived and tells you about some special flags. The 1.1 means first file in the archive file , starting at the 513th bit of the archive file(the header of tar if 512 byte long, thus the 513th bit is the first usable byte, thus the 1)

29.7.2. Manually forcing the release

Normally a file just get released, when the high watermark is reached or you configured the archiving this way. But you can force the release on the command line.

After this step, the file isn't in the cache any longer. When we look in the metadata, we will the a new information. The file is in the `offline` state:

```
bash-3.00# sls -D testfile3
testfile3:
  mode: -rw------T  links:   1  owner: root      group: root
  length:   1048576  admin id:      0  inode:     4640.1
  offline;  archdone;
  copy 1: ----- Mar 23 19:39         1.1    dk disk01 f1
  copy 2: ----- Mar 23 19:58         1.1    dk disk03 f1
  access:      Mar 23 19:28  modification: Mar 23 19:28
  changed:     Mar 23 19:28  attributes:   Mar 23 19:28
  creation:    Mar 23 19:28  residence:    Mar 24 01:28
```

When we access it again, the file get's staged back to the cache again:

```
bash-3.00# cat testfile3
bash-3.00# sls -D testfile3
testfile3:
  mode: -rw------T  links:   1  owner: root      group: root
  length:   1048576  admin id:      0  inode:     4640.1
  archdone;
  copy 1: ----- Mar 23 19:39         1.1    dk disk01 f1
  copy 2: ----- Mar 23 19:58         1.1    dk disk03 f1
  access:      Mar 24 01:35  modification: Mar 23 19:28
  changed:     Mar 23 19:28  attributes:   Mar 23 19:28
  creation:    Mar 23 19:28  residence:    Mar 24 01:35
```

The `offline` flag has gone away.

29.7.3. Manually forcing the staging of a file

Okay, but you can force the staging as well, let's assume you've released a file.

```
bash-3.00# release testfile3
bash-3.00# sls -D testfile3
testfile3:
  mode: -rw------T  links:   1  owner: root      group: root
  length:   1048576  admin id:      0  inode:     4640.1
  offline;  archdone;
  copy 1: ----- Mar 23 19:39         1.1    dk disk01 f1
  copy 2: ----- Mar 23 19:58         1.1    dk disk03 f1
  access:      Mar 24 01:35  modification: Mar 23 19:28
  changed:     Mar 23 19:28  attributes:   Mar 23 19:28
  creation:    Mar 23 19:28  residence:    Mar 24 01:37
```

A colleague comes into your office, and tells you that he wants to use a large file with simulation data tomorrow. It would be nice, if he don't have to wait for the automatic staging. We can force SamFS to get the file back to the cache.

```
bash-3.00# stage testfile3
```

Okay, let's check for the status of the file:

```
bash-3.00# sls -D testfile3
testfile3:
  mode: -rw------T  links:   1  owner: root      group: root
  length:   1048576  admin id:      0  inode:     4640.1
  archdone;
```

```
copy 1: ----- Mar 23 19:39          1.1    dk disk01 f1
copy 2: ----- Mar 23 19:58          1.1    dk disk03 f1
access:       Mar 24 01:35  modification: Mar 23 19:28
changed:      Mar 23 19:28  attributes:   Mar 23 19:28
creation:     Mar 23 19:28  residence:    Mar 24 01:37
```

Voila, it's in the cache again.

29.8. Usecases and future directions

Okay, after all this configuration. How can you use it? Well, the most obvious usage of SamFS is Archiving. And many customers use it for exactly this task: Archiving audio, archiving video, archiving eMails, documents, xrays, satellite images, simulation data, data of large experimental system. Everything that's worth to be stored for a long long time. That's the job, it was designed for.

29.8.1. Unconventional Usecases

But there are more unconventional use cases. One example: What does a backup software for network use? The ones with a central server and a backup agent? For example for Oracle? Well, the client not really much more than triggering RMAN and transmitting it over the network. The server receives it and stores it into tape. The problem with it? Well, did you recently look into the price list for such an backup agent?

Now think about the following idea: With SamFS you already have something, that controls your tape. everything written to a SamFS filesystem will be written to tape, if you configure it this way. Share this filesystem via NFS. Now you just have to write your RMAN script to backup your data into this NFS filesystem. No backup client involved. By the way, as the backup is written to the cache at first, you don't have the problem of keeping the tape streaming at top speed (the start/stop problem is mostly something of the past, as most drives are capable of changing their speed in discrete steps to adapt the speed of incoming data. Of course you won't use the full speed and the TB/hour capacity of the device). When you completed a backup file, SamFS will write it at full speed of the tape drive to the tape.

The interesting point about SamFS: Once you think about it, about the concept of hierarchical Storage Management, about a filesystem with tapes in its background, you will find more and more interesting use cases for it.

29.8.2. Future directions and ideas

SamFS is a tool that is in active development at Sun, albeit we work on the integration of SamFS like features into ZFS. More interesting will be the hardware side of this system. I've used iSCSI based volumes in my example. Just think about this scenario if I didn't used two 2G volumes but X4500 with 48 TB. Within 2 Racks I could archive almost 1 Petabyte with medium speed disks. (By the way: I've used the ZFS for the diskvols: Thus the transmission of the data to the network and the storage of the data itself is secured by checksums. ZFS Crypto will go beta soon, so you could can create an archive system where no data is unencrypted.) Using such large disk systems for archival is certainly a future direction.

But there is still a need for tapes: Aircraft manufactures have to store their data for 99 years (as far as I know). A normal German company has to store their data for 10 years for our IRS equivalent called Finanzamt. You don't want to store it on rotating rust for this amount of time. But by intelligent policies, you don't have to hassle around with it manually.

29.9. Conclusion

Okay, this was a rather long tutorial and I didn't even talked about the configuration of tape devices. As I told you before: Only the toes in the Atlantic ocean. But I hope, I gave you some insight into a somewhat unconventional topic and a capability of an optional part the Solaris Operating Environment. I assume, with the opensourcing of SamFS we will see a much more widespread use of it.

29.10. Do you want to learn more?

Reading the documentation is even more important than for the other features I presented in this series:

Documentation

docs.sun.com: Sun StorageTek Storage Archive Manager Installation and Upgrade Guide[1]

docs.sun.com: Sun StorageTek SAM File System Configuration and Administration Guide[2]

[1] `http://docs.sun.com/source/819-7932-10/`
[2] `http://docs.sun.com/source/819-7934-10`

docs.sun.com: Sun StorageTek SAM Archive Configuration and Administration Guide[3]

Misc. information

Sun Whitepaper: Sun StorEdge QFS and SAM-FS Software[4]

[3] http://docs.sun.com/source/819-7931-10

[4] http://www.sun.com/storagetek/white-papers/qfs-samfs.pdf

Part VI.

Solaris Administrators Toolbox

30. fuser

Solaris 10/Opensolaris

You know the problem. You try to unmount a filesystem, but all you get is a "Device Busy". How do you find the process blocking the unmount?

30.1. fuser

`fuser` enables you to look for the processes that access a directory or a file. For example we can check for all processes using the `/` filesystem as their "working filesystem":

```
# fuser -c /
/:       701ctm      676ctm      675ctom      672ctom      596ctm
        592ctm      585ctm      584ctom      581ctom      568ctm
        523ctom      521ctom      481ctom      478ctom      477ctom
        469ctom      456ctom      437ctom      425ctm      418ctom
        412ctom      402ctom      401ctom      399ctom      380ctom
        379ctom      366ctm      345ctom      341ctom      338ctom
        333ctom      332ctom      319ctom      272ctom      262ctom
        153ctm      140ctm      133ctom      131ctom      125ctm
        100ctom       18ctm        9ctom        7ctom        1ctm
```

Im sure you already assume, that the numbers stand for the process ids. But what does all that letters mean. I will cite the manpage for this:

- *c* Indicates that the process is using the file as its current directory.
- *m* Indicates that the process is using a file mapped with mmap(2).
- *n* Indicates that the process is holding a non-blocking mandatory lock on the file.
- *o* Indicates that the process is using the file as an open file.
- *r* Indicates that the process is using the file as its root directory.
- *t* Indicates that the process is using the file as its text file.
- *y* Indicates that the process is using the file as its controlling terminal.

30.2. But fuser can do more for you

Okay, now you know which processes uses a filesystem. But let's assume you have to unmount a filesystem for maintaining the storage. But there are still some users, who didn't read their mails and you can't unmount it.

```
# cd /mnt/application
# sleep 1000&
[1] 691
# sleep 1000&
[2] 692
# cd /
# umount /mnt/application
umount: /mnt/application busy
```

Dammit ... Okay, let's check for the offending processes:

```
fuser -u /mnt/application
/mnt/application:      692c(root)     691c(root)
```

And now comes the kicker: fuser can kill all processes using a certain file or directory. You warned your users ...

```
# fuser -k -u /mnt/application
/mnt/application:      692c(root)     691c(root)
[2]+  Killed                    sleep 1000  (wd: /mnt/application
   )
(wd now: /)
[1]+  Killed                    sleep 1000  (wd: /mnt/application
   )
(wd now: /)
#
```

Now we can unmount the directory:

```
# umount /mnt/application
#
```

30.3. A neat trick with fuser

Okay, working with PID may a little bit cumbersome. Pete Shanahan posted a neat trick a long time ago. Let's assume the example with the both `sleep` processes. You've remounted the filesystem an started some programs while being in the `/mnt/application` filesystem:

```
# cd /mnt/application
# sleep 1000&
[1] 726
# sleep 1000&
[2] 727
# cd /
# ps -o pid,args -p "$(fuser /mnt/application 2>/dev/null)"
  PID COMMAND
  726 sleep 1000
  727 sleep 1000
#
```

30.4. Do you want to learn more ?

docs.sun.com: fuser(1M)[1]

Pete Shanahan's fuser trick[2]

[1] http://docs.sun.com/app/docs/doc/816-5166/fuser-1m
[2] http://blogs.sun.com/petesh/date/20050127

31. pfiles

Solaris 10/Opensolaris

This is not a tutorial, just a hint from my toolbox. On customer systems I see the `lsof` tool quite often. But for a quick check for open files you don't need it.

There is a small, but extremely useful tool in the collection of the `p*`-tools: `pfiles` prints all open files of a process. It takes the PID of the process to specify the process.

```
# pfiles 214
214:    /usr/lib/inet/in.iked
  Current rlimit: 256 file descriptors
   0: S_IFDIR mode:0755 dev:102,0 ino:2 uid:0 gid:0 size:512
      O_RDONLY|O_LARGEFILE
      /
   1: S_IFDIR mode:0755 dev:102,0 ino:2 uid:0 gid:0 size:512
      O_RDONLY|O_LARGEFILE
      /
   2: S_IFDIR mode:0755 dev:102,0 ino:2 uid:0 gid:0 size:512
      O_RDONLY|O_LARGEFILE
      /
   3: S_IFREG mode:0600 dev:102,0 ino:28994 uid:0 gid:0 size
      :47372
      O_RDWR|O_APPEND|O_CREAT|O_LARGEFILE
      /var/log/in.iked.log
   4: S_IFSOCK mode:0666 dev:304,0 ino:48934 uid:0 gid:0 size:0
      O_RDWR|O_NONBLOCK
        SOCK_RAW
        SO_SNDBUF(8192),SO_RCVBUF(8192)
        sockname: AF_INET 10.211.55.200  port: 4500
        peername: AF_INET 10.211.55.200  port: 4500
 [..]
   10: S_IFDOOR mode:0777 dev:306,0 ino:0 uid:0 gid:0 size:0
      O_RDWR FD_CLOEXEC  door to in.iked[214]
```

And with the xargs tool there is an easy way to print out all open files on the system.

```
# ps -ef -o pid | sort | xargs pfiles  | more</b>
0:      sched
```

```
  [system process]
1:      /sbin/init
  Current rlimit: 256 file descriptors
   0: S_IFIFO mode:0600 dev:301,3 ino:448255748 uid:0 gid:0
      size:0
      O_RDWR|O_NDELAY
      /var/run/initpipe
 253: S_IFREG mode:0444 dev:298,1 ino:65538 uid:0 gid:0 size:0
      O_RDONLY|O_LARGEFILE FD_CLOEXEC
      /system/contract/process/pbundle
 254: S_IFREG mode:0666 dev:298,1 ino:65539 uid:0 gid:0 size:0
      O_RDWR|O_LARGEFILE FD_CLOEXEC
      /system/contract/process/template
 255: S_IFREG mode:0666 dev:298,1 ino:65539 uid:0 gid:0 size:0
      O_RDWR|O_LARGEFILE FD_CLOEXEC
      /system/contract/process/template
[...]
```

32. Installing Solaris Packages directly via web

Solaris 10/Opensolaris

Until I've finalized my next larger article, I want to give spotlight to a really small, but really useful feature: One relatively unknown feature of recent versions of pkgadd is the ability to load packages directly from web. You just have to specify an URL:

```
# pkgadd -d http://www.blastwave.org/pkg_get.pkg

## Downloading...
..............25%.............50%.............75%.............100%

## Download Complete


The following packages are available:
  1  CSWpkgget     pkg_get - CSW version of automated package
     download tool
                   (all) 3.8.4

[..]

Installation of <CSWpkgget> was successful.
#
```

That's all. As the packages just have to be accessible by http, you can use an existing internal webserver to serve your favorite "must-have" extra packages and install them directly from there. Okay, and solves the problem nicely to get started with Blastwave without moving around the pkg_get package via ftp ;)

33. About crashes and cores

Solaris 10/Opensolaris

No software is without errors. This is a basic law of computer science. And when there is no bug in the software (by a strange kind of luck) your hardware has bugs. And when there are no bugs in the hardware, cosmic rays are flipping bits. Thus an operating system needs some mechanisms to stop a process or the complete kernel at once without allowing the system to write anything back to disk and thus manifesting the corrupted state. This tutorial will cover the most important concepts surrounding the last life signs of a system or an application.

33.1. A plea for the panic

The panic isn't the bug, it's the reaction of the system to a bug in the system. Many people think of panics as the result of an instability and something bad like the bogey man. But: Panics and crash dumps are your friend. Whenever the system detects a inconsistency in its structures, it does the best what it could to: protect your data. And the best way to do this, is to give the system a fresh start, don't try to modify the data on the disk and write some status information to a special device to enable analysis of the problem. The concepts of panic and crash dump were developed to give the admin exactly such tools.

A good example: Imagine a problem in the UFS, a bit has flipped. The operating environment detects an inconsistence in the data structures. You can't work with this error. It would unpredictably alter the data on your disk. You can't shutdown the system by a normal reboot. The flushing of your disks would alter the data on the disk. The only way to get out of the system: Stop everything and restart the system, ergo panic the system and write a crash dump.

Furthermore: Some people look at the core and crash dumps and think about their analysis as an arcane art and see them as a wast of disk space. But it's really easy to get some basic data and hints out of this large heaps of data.

33.2. Difference between Crash Dumps and Cores

Many people use this words synonymously ("The system paniced and wrote a core dump"). Every now and then i'm doing this as well. But this isn't correct. The scope of the dump is quite different:

crash dump A crash dump is the dump of the memory of the complete kernel.

core dump The core dump is the dump of the memory of a process

33.3. Forcing dumps

Okay, a dumps are not only a consequence of errors. You can force the generation of both kinds. This is really useful when you want to freeze the current state of the system or an application for further examination.

33.3.1. Forcing a core dump

Let's assume you want to have an core dump of a process running on your system:

```
# ps -ef  | grep "bash" | grep "jmoekamp"
jmoekamp   681   675   0 20:59:39 pts/1       0:00 bash
```

Okay, now we can trigger the core dump by using the process id of the process.

```
# gcore 681
gcore: core.681 dumped
```

Okay, but the kicker is the fact, that the process still runs afterwards. So you can get an core dump of your process for analysis without interrupting it.

```
# ps -ef  | grep "bash" | grep "jmoekamp"
jmoekamp   681   675   0 20:59:39 pts/1       0:00 bash
```

Neat isn't it. Now you can use the `mdb` to analyse it, for example to print out the backtrace:

```
# mdb core.681
Loading modules: [ libc.so.1 ld.so.1 ]
> $c
libc.so.1`__waitid+0x15(0, 2a9, 8047ca0, 83)
libc.so.1`waitpid+0x63(2a9, 8047d4c, 80)
waitjob+0x51(8077098)
```

```
postjob+0xcd(2a9, 1)
execute+0x77d(80771c4, 0, 0)
exfile+0x170(0)
main+0x4d2(1, 8047e48, 8047e50)
_start+0x7a(1, 8047eec, 0, 8047ef0, 8047efe, 8047f0f)
```

33.3.2. Forcing a crash dump

Okay, you can force a crash dump, too. It's quite easy. You can trigger it with the `uadmin` command.

```
bash-3.2# uadmin 5 0

panic[cpu0]/thread=db47700: forced crash dump initiated at user
    request

d50a2f4c genunix:kadmin+10c (5, 0, 0, db325400)
d50a2f84 genunix:uadmin+8e (5, 0, 0, d50a2fac, )

syncing file systems... 2 1 done
dumping to /dev/dsk/c0d0s1, offset 108593152, content kernel
100% done: 31255 pages dumped, compression ratio 5.31, dump
   succeeded
Press any key to reboot.
```

Why should you do something like that? Well, there are several reasons. For example, when you want to stop a system right at this moment. There is an effect in clusters called "split brain" . This happens, when both systems believe their are the surviving one, because they've lost the cluster interconnect. Sun Cluster can prevent this situation by something called quorum. In a high availability situation the nodes of a cluster try to get this quorum. Whoever gets the quorum, runs the service. But you have to ensure that the other nodes don't even try to write something to disks. The simplest method: Panic the machine.

Another use case would be the detection of an security breach. Let's assume, your developer integrated a security hole as large as the Rhine into a web applicaiton by accident and now someone else owns your machine. The false reaction would be: Switch the system off or trigger a normal reboot. Both would lead to the loss of the memory content and perhaps the hacker had integrated a tool in the shutdown procedure to erase logs. A more feasible possibility: Trigger a crash dump. You keep the content of the memory and you can analyse it for traces to the attacker.

33.4. Controlling the behaviour of the dump facilities

Solaris has mechanisms to control the behaviour of dumps. These mechanisms are different for crash and core dumps.

33.4.1. Crash dumps

You can configure the content of crashdumps, where they are located and what you do with them after the boot. You control this behaviour with the `dumpadm` command. When you use this command without any further option, it prints out the actual state.

```
# dumpadm
      Dump content: kernel pages
       Dump device: /dev/dsk/c0d0s1 (swap)
Savecore directory: /var/crash/incubator
  Savecore enabled: yes
```

This is the default setting: A crash dump contains only the memory pages of the kernel and uses `/dev/dsk/c0d0s1` (the swap device) to store the crash dump in the case of a kernel panic. `savecore` is a special process, that runs at the next boot of the system. In the case of an crash dump at the dump device, it copies the dump to the configured directory to keep it for analysis before it's used for swapping again.

Let's change the behaviour. At first we want to configure, that the complete memory is saved to the crash dump in case of a panic. This is easy:

```
# dumpadm -c all
      Dump content: all pages
       Dump device: /dev/dsk/c0d0s1 (swap)
Savecore directory: /var/crash/incubator
  Savecore enabled: yes
```

Okay, now let's change the location for the crash dump. The actual name is an artefact of my orignal VM image called `incubator`. To get a new test machine I clone this image. I want to use the directory `/var/crash/theoden` for this purpose.

```
# mkdir /var/crash/theoden
# chmod 700 /var/crash/theoden
# dumpadm -s /var/crash/theoden
      Dump content: all pages
       Dump device: /dev/dsk/c0d0s1 (swap)
Savecore directory: /var/crash/theoden
  Savecore enabled: yes
```

Now the system will use the new directory to store the crash dumps. Setting the rights of the directory to `700` is important. The crash dump may contain sensitive information, thus it could be dangerous to make them readable by anyone else than root.

33.4.2. Core dumps

A similar facility exists for the core dumps. You can control the behaviour of the core dumps with the `coreadm` command. Like with `dumpadm` you can get the actual configuration by using `coreadm` without any option.

```
# coreadm
     global core file pattern:
     global core file content: default
       init core file pattern: core
       init core file content: default
            global core dumps: disabled
      per-process core dumps: enabled
      global setid core dumps: disabled
 per-process setid core dumps: disabled
     global core dump logging: disabled
```

This programm has more options than `dumpadm`. I won't go through all options, but some important ones.

From my view the file patterns are the most interesting ones. You can control, where core dumps are stored. The default is to store the core dumps in the working directory of a process. But this may lead to core dumps dispersed over the filesystem.

With core adm you can configure a central location for all your coredumps.

```
# coreadm -i /var/core/core.%n.%f.%u.%p
# coreadm -u
```

With `-i` you tell `coreadm` to set the location for the per-process core dumps. The parameter for this option is the filename for new core dumps. You can use variables in this filename. For example `%n` will be translated to the machine name, `%f` to name of the file, `%u` to the effective user id of the process and `%p` will be substituted with the process id. The `coreadm -u` forces the instant reload the configuration. Otherwise, this setting would get active at the next boot or the next refresh of the coreadm service. Okay, let's try our configuration.

```
# ps -ef | grep "bash" | grep "jmoekamp"
jmoekamp   681   675   0 20:59:39 pts/1       0:00 bash
```

Now we trigger a core dump for a running process.

```
# gcore -p 681
gcore: /var/core/core.theoden.bash.100.681 dumped
```

As you see, the core dump isn't written at the current working directory of the process, it's written at the configured position.

33.4.3. Core dump configuration for the normal user

The both configuration described so far are global ones, so you can do this configuration only with root privileges. But a normal user can manipulate the core dump configuration as well, albeit only for processes owned by her or him.

Let's login as a normal user. Now we check one of our processes for its `coreadm` configuration:

```
$ ps -ef | grep "jmoekamp"
jmoekamp   712   670   0 01:27:38 pts/1       0:00 -sh
jmoekamp   669   666   0 22:29:15 ?           0:00 /usr/lib/ssh
   /sshd
jmoekamp   670   669   0 22:29:16 pts/1       0:00 -sh
jmoekamp   713   712   0 01:27:38 pts/1       0:00 ps -ef
$ coreadm 669
669:    /var/core/core.%n.%f.%u.%p      default
```

Now let's check a process owned by root.

```
$ ps -ef | grep "cron"
jmoekamp   716   670   0 01:28:13 pts/1       0:00 grep cron
    root   322     1   0 22:25:24 ?           0:00 /usr/sbin/
       cron
$ coreadm 322
322: Not owner
```

The system denies the access to this information. Now we change the setting for the process `669` from the first example. It's quite simple:

```
$ coreadm -p /export/home/jmoekamp/cores/core.%n.%f.%u.%p 669
$ coreadm 669
669:    /export/home/jmoekamp/cores/core.%n.%f.%u.%p    default
```

This setting ist inherited to all The per-process core file name pattern is inherited by future child processes of the affected processes.

Why should you set an own path and filename for an application or an user? There are several reasons. For example to ensure that you have the correct rights to an directory

for the cores, when the process starts to dump the core or to seperate the cores from certain applications a different locations.

33.5. Crashdump analysis for beginners

33.5.1. Basic analysis of a crash dump with mdb

Okay, now you have all this crash and core dumps, it would be nice to do something useful with it. Okay, I show you just some basic tricks to get some insight into the state of a system when it wrote a crash dump.

At first we load the dump into the `mdb`

```
# mdb unix.0 vmcore.0
Loading modules: [ unix genunix specfs cpu.generic uppc
   scsi_vhci ufs ip hook neti sctp arp usba nca lofs zfs random
    nsctl sdbc rdc sppp ]
>
```

Solaris has an in-memory buffer for the console messages. In the case you write a crash dump, obviously this messages are written into the crash dump as well. With the `::msgbuf` command of `mdb` you can read this message buffer.

```
> ::msgbuf
MESSAGE
SunOS Release 5.11 Version snv_84 32-bit
Copyright 1983-2008 Sun Microsystems, Inc.  All rights reserved
   .
Use is subject to license terms.
features: 10474df<cpuid,sse3,sse2,sse,sep,cx8,mmx,cmov,pge,mtrr
   ,msr,tsc,lgpg>
mem = 331388K (0x1439f000)
root nexus = i86pc
pseudo0 at root
pseudo0 is /pseudo
[...]
devinfo0 is /pseudo/devinfo@0

panic[cpu0]/thread=db3aea00:
forced crash dump initiated at user request


d5efcf4c genunix:kadmin+10c (5, 0, 0, db5c8a98)
```

```
d5efcf84 genunix:uadmin+8e (5, 0, 0, d5efcfac, )

syncing file systems...
 done
dumping to /dev/dsk/c0d0s1, offset 108593152, content: all
>
```

So it's really easy to get this last messages of a dying system with mdb from the crash dump alone.

A nice information is the backtrace. This helps you to find out, what triggered the crash dump. In this case it's easy. It was the `uadmin` syscall.

```
> $c
vpanic(fea6388c)
kadmin+0x10c(5, 0, 0, db39e550)
uadmin+0x8e()
sys_sysenter+0x106()
```

But it would be nice, to know more of the state of the system, at the moment of the crash. For example we can print out the process table of the system like we would do it with `ps`

```
> ::ps
S    PID   PPID   PGID    SID    UID      FLAGS      ADDR NAME
R      0      0      0      0      0 0x00000001 fec1d3d0 sched
[...]
R    586      1    586    586      0 0x42000000 d55f58a8 sshd
R    545      1    545    545      0 0x42000000 d5601230 fmd
R    559      1    559    559      0 0x42000000 d55fb128
   syslogd
[...]
R    533    494    494    494      0 0x4a014000 d55f19c0 ttymon
```

We can even lookup, which files or sockets where opened at the moment of the crash dump. For example: We want to know the open files of the `ssh` daemon. To get this information, we have to use the address of the process from the process table (the eighth column) and extend it with `"::pfiles"`:

```
> d55f58a8::pfiles
FD   TYPE    VNODE INFO
   0  CHR d597d540 /devices/pseudo/mm@0:null
   1  CHR d597d540 /devices/pseudo/mm@0:null
   2  CHR d597d540 /devices/pseudo/mm@0:null
   3 SOCK db688300 socket: AF_INET6 :: 22
```

And here we look into the open files of the syslog process.

```
> d55fb128::pfiles
FD   TYPE    VNODE INFO
   0  DIR d5082a80 /
   1  DIR d5082a80 /
   2  DIR d5082a80 /
   3 DOOR d699b300 /var/run/name_service_door [door to 'nscd' (
      proc=d5604890)]
   4  CHR db522cc0 /devices/pseudo/sysmsg@0:sysmsg
   5  REG db643840 /var/adm/messages
   6  REG db6839c0 /var/log/syslog
   7  CHR db522840 /devices/pseudo/log@0:log
   8 DOOR db6eb300 [door to 'syslogd' (proc=d55fb128)]
```

As the core dump contains all the pages of the kernel (or more, in the case you configure it) you have a frozen state of your system to investigate everything you want.

And to get back to my security example: With the core dump and `mdb` you can gather really interesting informations. For example, you can see that an ssh connection was open at the time of the crash dump.

```
> ::netstat
TCPv4     St    Local Address         Remote Address        Stack
          Zone
db35f980  0    10.211.55.200.22         10.211.55.2.53811     0
      0
[...]
```

33.5.2. A practical usecase

You can do it like the pros and look at source code and crash dump side by side to find the root cause for an error. Or like some colleagues at the Sun Mission Critical Support Center who wouldn't surprise me, when they find the error by laying their hand on a system).

For all others, there is a more simple way to analyse your crash dump to have at least a little bit more informations to search in a bug database.

I will use a crash I've analyzed a long time ago to show you the trick. Okay, you have to start a debugger. I used `mdb` in this example:

```
bash-3.00# mdb -k unix.4 vmcore.4
```

```
Loading modules: [ unix krtld genunix specfs dtrace cpu.
   AuthenticAMD.15 uppc pcplusmp ufs md ip sctp usba fcp fctl
   nca lofs cpc fcip random crypto zfs logindmux ptm sppp nfs
   ipc ]
```

A prompt appears, just type in `$C` to get a stack trace.

```
> $C
ffffe80000b9650 vpanic()
ffffe80000b9670 0xffffffffb840459()
ffffe80000b96e0 segmap_unlock+0xe5()
ffffe80000b97a0 segmap_fault+0x2db()
ffffe80000b97c0 snf_smap_desbfree+0x76()
ffffe80000b97e0 dblk_lastfree_desb+0x17()
ffffe80000b9800 dblk_decref+0x66()
ffffe80000b9830 freeb+0x7b()
ffffe80000b99b0 tcp_rput_data+0x1986()
ffffe80000b99d0 tcp_input+0x38()
ffffe80000b9a10 squeue_enter_chain+0x16e()
ffffe80000b9ac0 ip_input+0x18c()
ffffe80000b9b50 i_dls_link_ether_rx+0x153()
ffffe80000b9b80 mac_rx+0x46()
ffffe80000b9bd0 bge_receive+0x98()
ffffe80000b9c10 bge_intr+0xaf()
ffffe80000b9c60 av_dispatch_autovect+0x78()
ffffe80000b9c70 intr_thread+0x50()
```

Okay, now start at the beginning of the trace to strip all lines from the operating system infrastructure for error cases. Okay, `vpanic()` generates the panic. The second line is useless for our purposes to. The next both lines with `segmap` are generated by the error but not the root cause. The interesting line ist `snf_smap_desbfree`

With this name you can go to Sunsolve or bugs.opensolaris.org. Et voila : System panic due to recursive mutex_enter in snf_smap_desbfree trying to re-aquire Tx mutex. When you type this error into the PatchFinder, you will find a patch fixing this bug: 124255-03

Two hints:

- It's a good practice to know `mdb`. It's very useful at compiling open source software in the case your compiled code throw cores, but you don't know why. core files are not just for deleting them.
- Error reports with a stack trace are more usefull than an error report just with "The system paniced when I did this"

33.6. Conclusion

The Solaris Operating Environment has several functions to enable the user or the support engineer in the case something went wrong. Crash and core dumps are an invaluable resource to find the root cause of a problem. Don't throw them away without looking at them.

33.7. Do you want to learn more?

Documentation

coreadm(1M)

dumpadm(1M)

uadmin(1M)

mdb(1)

Solaris Modular Debugger Guide

Books

Solaris Internals: Solaris 10 and Open Solaris Kernel Architecture - Richard McDougall and Jim Mauro

Solaris Performance and Tools: DTrace and MDB Techniques for Solaris 10 and OpenSolaris - Richard McDougall , Jim Mauro and Brendan Gregg

34. Jumpstart Enterprise Toolkit

Solaris 10/Opensolaris

34.1. Automated Installation

As wrote at several occasions in my tutorials, Solaris was designed with the enterprise customer in mind. It's quite common, that a company has dozens of servers and you have to put your customized operating system on all of them. A special management tool, some special admin tools, a certain configuration of the volume manager.

The problem: Let's assume, you have 50 system, all of them will have a slightly different configuration. This is normal, when you install your systems manually. Humans aren't machines, and human work introduces variances in every instance of work. While this is nice for furniture or apparel, this can lead to long searches for needles in haystacks.

On some occasions I had strange problems with systems: One system worked just fine, another made problems. Both were installed with a cookbook. At the end we've used something similar to BART. We've compared both system and found slight differences. On the problematic system, the admin made an typo. I was curious and looked after the working system. Even this system wasn't exactly like demanded by the cookbook.

So: How can you prevent this subtle variances between your systems? The most obvious way is the automation of the installation. Let do the computer, what a computer can do at best: Repetitive tasks with the same quality over and over again. Jumpstart and Jet have several ways and means to do automatic installation, thus making the admins live much easier, after you've done the final setup. And in this tutorial I want to show you, that this setup is really easy.

34.2. About Jumpstart

One basic problem of automated installation is the provisioning of the operating system on the bare metal. Without OS any further configuration is obviously senseless. For several releases Solaris contains a feature called Jumpstart. The basic idea of Jumpstart is the installation of a system by using the network.

34.2.1. The Jumpstart mechanism for PXE based x86

The process of jumpstarting is relatively simple.

1. The system comes. It tries to gather informations about the network
   - Its own IP, netmask and network address
   - the boot server
2. It connects to the boot server via tftp, loads the network boot program (in this case `pxegrub`) and the menu.lst.
3. The `pxegrub` boots the an Solaris environment on the client. The `menu.lst` contains the locations of important sources:
   - config server
   - location of the `sysid_config` file
   - location of the installation media
   - location of the boot environment
4. The mini root starts. From now on,you have a Solaris Kernel running on your systems.
5. It gathers the `rules.ok` from the installation server.
6. It tries to find a matching profile based on the rules in the `rules.ok` file.
7. Based on the jumpstart profile it formats the disks, it mounts all filesystems like on the final system relative to the directory `/a`
8. The installer starts to install the packages relatively to the directory `/a`
9. After all, it writes the boot loader to the disks
10. Now you can reboot the system.

34.3. Jumpstart Server

Albeit the Jumpstart Server is an important part in the process of installing the operating system on the server, it's a passive part. The Jumpstart process itself is executed on the client, not on the server. The server is only necessary to provide information and the installation media. It uses existing protocols for this task as RARP or DHCP for assigning informations about the network or NFS and HTTP for providing access to the installation files.

34.3.1. Development

At the beginning Jumpstart was just able to do the installation. Recent versions[1] include functions to automatically create boot mirrors.

34.4. Control Files for the automatic installation

I will not describe the exact process of native Jumpstart in its multitude of configurations as the Jumpstart Enterprise Toolkit does this job for us, but it's important to know some of the important internals of Jumpstart. I will describe the most important files. There are various others like `/etc/ethers` or the DHCP server configuration, but as you don't touch them manually even with the native configuration of Jumpstart I won't describe them here.

34.4.1. rules

The first important file for the automatic installation is the `rules` file. This files associates system with a installation profile.

```
# rule keywords and rule values       begin script       profile       finish script
# ------------------------------       ------------       --------      -------------
hostname aramaki        setup   webserver               completion
any   -                                 -                 genericprofile  -
```

The first rule can be divided in parts like this: When the hostname of the new server is aramaki, start the script `begin` on the client before starting the installation. For the installation use the profile file `webserver`. After the installation execute the script `completion`

The second line is a catch-all condition. The file is used top down and the process of matching a system to a profile stops at the first match. Thus an installation for `aramaki` would reach the second line. This line can be translated like this. For any other host, use the profile `genericprofile`. There is no begin or finish script.

You can't use the `rules` file directly. The Jumpstart server provides a script to do a syntax check on the `rules`. When the file is correct, the script adds it gets renamed to rules.ok

[1] Recent in Solaris context means Solaris 9 and up ;)

34.4.2. profile

The profile file controls what we install on the system and how we partition the disks for the installation.

```
# profile keywords        profile values
# ----------------        ----------------
  install_type            initial_install
  system_type             standalone
  partitioning            default
  filesys                 any 512 swap
  cluster                 SUNWCprog
  package                 SUNWman delete
  cluster                 SUNWCacc
```

You can have a multitude of profiles in your system. A profile for system with large disks, a profile for system with small disks, a profile with a selection of packages customized for a webserver, a profile customized for a developer workstation. The jumpstart framework will choose the correct one on the basis of the `rules.ok` set

The profile is capable to control almost any important parameter for the installation on the disk. You can define pa

34.4.3. The sysidcfg file

Installing the packages on the system isn't enough to get the operating system up an running. You have to give the system a unique identity. Parameters for this identity are:

- configuration of network interfaces
- locales
- initial root password
- time zone
- etc.

You can type in such information manually, but that wouldn't be a hands-off installation. The installation process of Solaris has a solution for this problem. You can set all the parameters in a file called `sysidcfg`. The Solaris installer will use this file to configure the system accordingly.

```
keyboard=US-English
timezone=US/Central
timeserver=timehost1
terminal=ibm-pc
service_profile=limited_net
```

```
name_service=NIS {domain_name=marquee.central.example.com
                  name_server=nmsvr2(172.25.112.3)}
nfs4_domain=example.com
root_password=URFUni9
```

But: Whenever some essential information is missing, the installer will *go interactive* and ask for the missing information. This obviously is against our objective of an automated installation.

34.5. Jumpstart FLASH

Sometimes you don't to do a new install of a system. You just want to clone a system. For example think about a webserver farm. Let's assume you have thirty of them. You've configured one and now you want to distribute this config to all of your system. You've tuned the system extensively, you changed configurations throughout all components. And you don't want to do this 29 times again.

34.5.1. Full Flash Archives

Solaris Jumpstart knows a special mode of operation for this task: It's called Jumpstart FLASH. The trick of Jumpstart flash is quite easy. At first a normal Jumpstart install and the FLASH install are identical. But when it comes to the installation of the Don't install the packages one by one. Instead jumpstart flash unpacks a archive of a running system on a new system. This archive is called FLASH archive. Technically speaking it's not much more than `cpio` archive of a running system.

34.5.2. Differential Flash Archives

There is an interesting mode of operation for flash archives. You can create differential flash archives. Let's assume you created a basic flash archive and installed all your systems with it: your webserver, your mailserver, your database server. Most parts of the system are identical. Just a few additional binaries and configuration files differentiate your server from each other.

Let's assume you want to create flash archives from all systems. Of course you could create a full flash archive for each system, but this would be waste of disk space. The differential flash archive creation works relatively simple. It compares the content of a flash archive with the actual state of an installed system and just archives the changed

parts. The next time you want to install the system, you use both archives. At first the full archive will be installed on the system, after this you use one or more differential flash archives to complete your installation.

Table 34.1.: **differential archive behavior**

OLD	NEW	ACTION
exists not	exists	File is included in archive
exists	exists but different	The file from the new state is included in archive
exists	exists not	File will be deleted, when the diff archive is used on a server

flar creation is just a big wrapper around `cpio`, thus it's possible to some nifty tricks with it. The current states of the system doesn't have be the active one, and the old states doesn't have to be flar archives.

It's possible to compare an old boot environment and the actual boot environment from Liveupgrade to generate a differential flash archive. This differential can be used to update other servers. You could even compare a remote system via NFS, when don't squash root.[2]

34.5.3. Challenges of Jumpstart Flash for System Recovery

Flash was designed with the task of system cloning in mind. So it removes the identity of the system after the installation by using the `sysidunconfig` command. The need for such a step at system cloning is obvious: One part of the systems identity is the networking configuration. You can't clone the network configuration. TCP/IP duplicates

`sysunconfig` deletes the entire configuration, that makes the installation an unique instance:

- saves a copy of `/etc/hosts` and substitute it with a default one.
- removes any NFS mount from `/etc/vfstab`
- deletes NIS, NIS+,LDAP and DNS name service configuration

[2] I know this has some security implication, but hey ... you should limit the access for such stunts to your admin networks and you can deactivate it afterwards.

- removes the interface configuration of all configured interfaces.
- removes the root password
- removes `/etc/sysidcfg`
- removes `/etc/defaultrouter`
- removes `/etc/ined/netmasks`
- regenerates the ssh-keys
- sets the timezones in `/etc/timezone` to `PST8PDT`

Albeit it's not designed for system recovery, there is a trick you can use to recover the removed information. The knowledge about the removed part is important for the trick, thus I've included a list of them in this tutorial. You will find a script at the end of this tutorial.

34.6. About the Jumpstart Enterprise Toolkit

34.6.1. The basic idea behind JET

The Jumpstart Enterprise Toolkit (JET) was designed as an add-on to the normal Jumpstart process. It solves two major challenges:

- Jumpstart works exceptionally well to install the operating system and to do configurations in conjunction to the operating system, but misses some easy to configure features to execute further configurations of higher-level configuration besides of having hooks for `finish` scripts.
- Some people find it a little bit complex to configure. While I strongly believe, this isn't the case, I have to admit, thats hard to remember the commands when this is not your everyday task

With both issues in mind, the Sun Engineers Michael Ramchand and Marty Lee started to develop the Jumpstart Enterprise Toolkit. Basically it's a bunch of clever shell scripts.

- The Jumpstart Enterprise Toolkit configures the Jumpstart Server accordingly to the configuration in a template.
- Furthermore it provides a framework for application specific modules to install and configure them.

When using JET you don't have to hassle around with all this files explained before, making the installation easier.

34.6.2. Additional features of JET

JET isn't just a config generator for Jumpstart. You can configure JET in a way to fully customize your system. Much of this capability comes from a link called `S99jumpstart` or the respective SMF service. This script has an important role: It executes further actions on the system on following boots. For example, the mirroring of harddisks isn't done by integrated functionalities of Solaris Jumpstart[3]. It's done by a script located on the server and made available by NFS, but executed on the server. This concept makes the JET a very flexible tool.

34.7. Prerequisites

34.7.1. Systems

For these tests, I need quite a few systems. I've used VirtualBox in the preparation of this tutorial.

I've populated my `/etc/hosts` with the following hostnames[4]:

```
192.168.10.1 aramaki
192.168.10.10   togusa
192.168.10.11   ishikawa
```

In my tutorial `aramaki` will serve as the Jumpstart server, `togusa` and `ishikawa` are the installation targets.

34.8. Packages

For this tutorial I've used the following packages and ISOs:

Solaris 10 Update 5 This is the operating environment I will use to demonstrate automated patching

OpenSolaris Community Edition Build 87 `aramaki` runs with this operating environment and it's no problem to jumpstart OpenSolaris CE or DE with JET.

Recommended Patch Cluster To demonstrate automated patching, I've used the recommended patch cluster. You can gather it at `http://sunsolve.sun.com`

[3] Those would make this action unavailable on Solaris 8 for example

[4] In case you wonder about the hostnames, these names are characters from Ghost in a Shell. And just in case you search for kusanagi, this is the name for the system hosting the virtual machines

SUNWjet The Jumpstart Enterprise Toolkit. You can get it at `http://www.sun.com/download/index.jsp?tab=2#J`

SUNWjass The Solaris Security Toolkit. You can get it from `http://www.sun.com/download/index.jsp?tab=2#S`

34.9. Installation of JET

34.9.1. Preparation of the system

A jumpstart server doesn't have to be a big server. But it's good practice to take the following into consideration:

- You have to copy your installation media onto the server. A Solaris version needs up to 4 Gigabyte on disk. Additionally you need space for further products (like other applications) and patches[5].Depending on the amount of Solaris versions you want to provide, it's a good idea not to spend the rest of an existing partition to your Jumpstart server.
- It's a wise choice to have a fast network. As the system isn't hardened and unconfigured at the first install, the paranoid prefer to use a free Gigabit-Ethernet port at my jumpstart server and use a separated VLAN for initial configuration. Or you use your admin LAN for this purpose. Either ways, for an fast installation gigabit ethernet is nice.

34.9.2. The installation

After copying the JET package on our new Jumpstart server, we have to install the package:

```
# pkgadd -d jet.pkg

The following packages are available:
  1  JetEXPLO        jet explo product
                     (sparc) 3.1.11
  2  JetFLASH        JET flash product
                     (sparc) 3.1.8
  3  JetJASS         JASS product
                     (sparc) 3.0.14
  4  JetRBAC         JET RBAC product
```

[5]The actual recommended patch cluster for Solaris 10 x86 is over 300 MB large

```
                    (sparc) 1.1.5
   5  JetSAN         JET san product
                    (sparc) 3.1.7
   6  JetSBD         Secure By Default product
                    (sparc) 1.0.2
   7  JetSDS         JET sds product
                    (sparc,i386) 3.4.4
   8  JetVTS         JET VTS product
                    (sparc) 3.0.11
   9  JetWanBoot     JET WanBoot support
                    (sparc) 1.1.1
  10  JetZONES       JET Zones module
                    (sparc) 1.1.12

... 2 more menu choices to follow;
<RETURN> for more choices, <CTRL-D> to stop display:

  11  SUNWjet        Sun JumpStart Enterprise Toolkit
                    (sparc,i386) 4.4
  12  SUNWjetd       JET Documentation
                    (sparc) 3.3.1

Select package(s) you wish to process (or 'all' to process
all packages). (default: all) [?,??,q]: all
```

JetEXPLO Installs and configures the Sun Explorer

JetFLASH Module to control Jumpstart Flash installed

JetJASS executes the Solaris Security Toolkit on the new host

JetRBAC configures the Role Based Access Control

JetSAN configures the SAN framework of Solaris

JetSBD configures the Secure-by-default setting

JetSDS configures the Solaris Volume Management

JetVTS installs the Sun Validation Test Suite. It tests and validates Sun hardware by verifying the connectivity and functionality of hardware devices, controllers and peripherals.

JetWANboot configures the Jumpstart facilities for installation over the WAN

JetZONES configures Solaris Zones on the newly installed zones.

As the package is quite small with half a megabyte, I always install all packages on a jumpstart server.

```
Processing package instance <SUNWjet> from </root/jet.pkg>

Sun JumpStart Enterprise Toolkit(sparc,i386) 4.4
Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
Use is subject to license terms.

The selected base directory </opt/SUNWjet> must exist before
installation is attempted.

Do you want this directory created now [y,n,?,q] y
Using </opt/SUNWjet> as the package base directory.

[...]

Processing package instance <JetSBD> from </root/jet.pkg>

Secure By Default product(sparc) 1.0.2

#
```

This is all we have to do for the installation.

34.10. Preparations for out first installation

Now you have to copy your Solaris install media. You can do this on two ways.

34.10.1. From a mounted DVD media

When you already have burned media, you can use it for the copy process. Just put it into the drive, the volume management mounts it and you can copy it from there.

```
# /opt/SUNWjet/bin/copy_solaris_media -d /export/install/media/
   solaris/x86/nv87 -n nv87 /cdrom/sol_11_x86
Copying Solaris image....
Verifying target directory...
Calculating the required disk space for the Solaris_11 product
Calculating space required for the installation boot image
Copying the CD image to disk...
Copying Install Boot Image hierarchy...
```

```
Copying /boot netboot hierarchy...
Install Server setup complete

Added Solaris image nv87 at the following location:
        Media:          /export/install/media/solaris/x86/nv87

removing directory /export/install/media/solaris/x86/911
#
```

Let's dissect the command: `-d` specifies the target, where you copy the operating system. `-n` specifies a name for this media. From now on you refer this solaris media as `nv87` in the templates for JET. At the end you specify the location, where the media is located at the moment.

34.10.2. From a .iso file

Okay, you've downloaded your Solaris media. You don't have to burn it, you can use the `.iso` files directly:

```
# ./copy_solaris_media -d /export/install/media/solaris/x86/
   sol10u5 -n sol10u5 -i /export/home/jmoekamp sol-10-u5-ga-x86
   -dvd.iso
Created loopback device /dev/lofi/1 for /export/home/jmoekamp/
   sol-10-u5-ga-x86-dvd.iso
mounted /export/home/jmoekamp/sol-10-u5-ga-x86-dvd.iso at /
   export/install/media/solaris/x86/790/slices/s0 (of type hsfs
   )
Copying Solaris image....
Verifying target directory...
Calculating the required disk space for the Solaris_10 product
Calculating space required for the installation boot image
Copying the CD image to disk...
Copying Install Boot Image hierarchy...
Copying /boot x86 netboot hierarchy...
Install Server setup complete

Added Solaris image sol10u5 at the following location:
        Media:          /export/install/media/solaris/x86/
           sol10u5

Unmounting /export/install/media/solaris/x86/790/slices/s0
removing device /dev/lofi/1
removing directory /export/install/media/solaris/x86/790
```

The command line for using a `.iso` file is quite similar. You just specify with the `-i` that an =.iso= file has to be used and in which directory it should search for it. The last parameter is the name of the `.iso` file itself. The system mounts the dvd image by using the loopback facility of Solaris and copies the media to its target location afterwards.

34.10.3. Looking up the existing Solaris versions

JET provides script to lookup the versions of Solaris you've copied to your Jumpstart Server. With the `list_solaris_locations` script you look up the version and the location of your solaris medias.

```
# ./list_solaris_locations
Version          Location
-------          --------
nv87             /export/install/media/solaris/x86/nv87
sol10u5          /export/install/media/solaris/x86/sol10u5
```

34.11. A basic automated installation

At first we will do a really basic install. No tricks, just the pure operating system. Nevertheless this part will be a little bit longer as I will do a technical deep-dive into the process of installation in this example to show you the inner workings of JET with this installation as an example.

34.11.1. The template for the install

At first we have to create a template for the system. This is really easy.

```
# make_template togusa
Adding product configuration information for
        + base_config
        + custom
        + sds
        + vts
        + explo
        + flash
        + san
        + jass
        + zones
        + sbd
```

```
Updating base_config template specifics
Client template created in /opt/SUNWjet/Templates
```

Okay, this is too much ... at start we don't want all this modules right now. We can add them later, without loosing the configuration. Let's just use the module for the basic configuration:

```
# make_template -f togusa base_config
Adding product configuration information for
        + base_config
Updating base_config template specifics
Client template created in /opt/SUNWjet/Templates
```

Even the basic template is quite long. I've reduced it for this tutorial by deleting all comments, all empty lines and all variables without a value.

```
base_config_ClientArch=i86pc
base_config_ClientEther=08:00:27:97:29:1E
base_config_ClientOS=nv87
base_config_client_allocation="grub"
base_config_sysidcfg_nameservice=NONE
base_config_sysidcfg_network_interface=PRIMARY
base_config_sysidcfg_ip_address=192.168.10.10
base_config_sysidcfg_netmask=255.255.255.0
base_config_sysidcfg_root_password="boajrOmU7GFmY"
base_config_sysidcfg_system_locale="C"
base_config_sysidcfg_timeserver=localhost
base_config_sysidcfg_timezone="Europe/Berlin"
base_config_sysidcfg_terminal=vt100
base_config_sysidcfg_security_policy=NONE
base_config_sysidcfg_protocol_ipv6=no
base_config_sysidcfg_default_route=192.168.10.1
base_config_x86_nowin="yes"
base_config_label_disks="all"
base_config_profile_cluster=SUNWCuser
base_config_profile_usedisk=rootdisk.
base_config_profile_root=free
base_config_profile_swap=256
base_config_ufs_logging_filesys="all"
base_config_profile_del_clusters="SUNWCpm SUNWCpmx SUNWCdial
   SUNWCdialx"
base_config_dns_disableforbuild="yes"
base_config_update_terminal="yes"
base_config_enable_savecore="yes"
base_config_dumpadm_minfree="20000k"
base_config_noautoshutdown="pm_disabled"
```

Let's dissect this template.

Line 1-3 This lines are the most basic ones. The first line defines the architecture of the system. The next line is the Ethernet-Address of the new system. The third one specifies the new operating system.

Line 4 This line specifies, how the new system gathers the most basic informations like its own IP.

Line 5-16 Do you remember the part about `sysidcfg`. The values for this files are defined in this part of the template.

Line 17 This line tells the system to suppress the start of the windowing system.

Line 18 Solaris needs a disk label on the disks for the system. This directive tells the system to write this label to all disks.

Line 19-22 Another known phrase ... profile. Here you specify the partitioning for the system and what packages will be installed on it.

Line 23-end There are several further statements. Please the original file for an explanation.

Okay after this step, we have to generate the configuration for the Jumpstart mechanism. This is really easy:

```
# make_client -f togusa
Gathering network information..
        Client: 192.168.10.10 (192.168.10.0/255.255.255.0)
        Server: 192.168.10.1 (192.168.10.0/255.255.255.0, SunOS
           )
Solaris: client_prevalidate
Solaris: client_build
Creating sysidcfg
Creating profile
Adding base_config specifics to client configuration
Solaris: Configuring JumpStart boot for togusa
         Starting SMF services for JumpStart
Solaris: Configure PXE/grub build
         Adding install client
        Doing a TEXT based install
         Leaving the graphical device as the primary console
         Configuring togusa macro
         Using local dhcp server
         PXE/grub configuration complete
Running '/opt/SUNWjet/bin/check_client  togusa'
        Client: 192.168.10.10 (192.168.10.0/255.255.255.0)
```

```
        Server: 192.168.10.1 (192.168.10.0/255.255.255.0, SunOS
           )
Checking product base_config/solaris
------------------------------------------------------------
Check of client togusa
-> Passed....
```

The nice thing about the `make_client` command: It doesn't just generate the Jumpstart configuration. It checks for the most dumb errors like forget to share the directory of your Solaris media with NFS. So you can detect many problems at an early stage. You don't have to wait until the jumpstart client comes up just to detect, that there is no NFS or no DHCP config.

34.11.2. The generated Jumpstart configuration files

Okay, let's look into the `/tftpboot` directory at first. As the system uses `pxegrub` we need a `menu.lst`

```
-bash-3.2$ cat /tftpboot/menu.lst.0108002797291E
default=0
timeout=2
title Solaris_11 Jumpstart
        kernel /I86PC.Solaris_11-1/platform/i86pc/kernel/unix
           - install nowin -B install_config=192.168.10.1:/opt/
           SUNWjet,sysid_config=192.168.10.1:/opt/SUNWjet/
           Clients/togusa,install_media=192.168.10.1:/export/
           install/media/solaris/x86/nv87,install_boot
           =192.168.10.1:/export/install/media/solaris/x86/nv87
           /boot
        module /I86PC.Solaris_11-1/x86.miniroot
-bash-3.2$
```

In the GRUB configuration we not only load the Kernel, we additionally name the location of the Jumpstart server, the exact location and name of the `sysidconfig` file, the position of our installation media and at last the location of the miniroot. In our example all locations are NFS locations.

Okay, the `install_config` directory is the first important location. We find the `rules.ok` file there.

```
-bash-3.2$ cat /opt/SUNWjet/rules.ok
any any                 Utils/begin     =       Utils/finish
# version=2 checksum=3114
```

Okay, now let's have a look in the specified profile file:

```
-bash-3.2$ cat /opt/SUNWjet/Clients/togusa/profile
#
# This is an automatically generated profile. Please modify the
    template.
#
# Created:      Mon May 19 21:47:50 CEST 2008
#
install_type    initial_install
system_type     server
cluster         SUNWCuser
partitioning    explicit
#
# Disk layouts
#
filesys         rootdisk.s0     free    /
filesys         rootdisk.s1     256     swap
cluster SUNWCpm delete
cluster SUNWCpmx delete
cluster SUNWCdial delete
cluster SUNWCdialx delete
```

As I wrote before, we have to give the system an identity. The `sysidcfg` is responsible for this task, thus we find such a file in our directory. Our new system will use it when the installation has completed.

```
-bash-3.2$ cat /opt/SUNWjet/Clients/togusa/sysidcfg
name_service=NONE
root_password=boajrOmU7GFmY
system_locale=C
timeserver=localhost
timezone=Europe/Berlin
terminal=vt100
security_policy=NONE
nfs4_domain=dynamic
network_interface=PRIMARY {hostname=togusa ip_address
   =192.168.10.10 netmask=255.255.255.0 protocol_ipv6=no
   default_route=192.168.10.1}
```

34.11.3. The installation boot

This leaves you to do one thing. Configure the system to start with PXE in the BIOS of your system. And the system will boot via network and starts to install a system.

After a while the installation will be complete. You can look for the logfile of the installation at `/var/sadm/system/logs`:

```
Configuring disk (c0d0)
        - Creating Fdisk partition table

Fdisk partition table for disk c0d0 (input file for fdisk(1M))
          type: 130  active:  128  offset: 16065   size: 33527655
          type: 100  active:  0    offset: 0       size: 0
          type: 100  active:  0    offset: 0       size: 0
          type: 100  active:  0    offset: 0       size: 0
        - Creating Solaris disk label (VTOC)
        - Processing the alternate sector slice

Creating and checking UFS file systems
        - Creating / (c0d0s0)
Warning: 1608 sector(s) in last cylinder unallocated
/dev/rdsk/c0d0s0:       31744440 sectors in 5167 cylinders of 48 tracks, 128 sectors
        15500.2MB in 323 cyl groups (16 c/g, 48.00MB/g, 5824 i/g)
super-block backups (for fsck -F ufs -o b=#) at:
 32, 98464, 196896, 295328, 393760, 492192, 590624, 689056, 787488, 885920,
Initializing cylinder groups:
......
super-block backups for last 10 cylinder groups at:
 30776480, 30874912, 30973344, 31071776, 31170208, 31268640, 31367072,
 31457312, 31555744, 31654176

Beginning Solaris software installation

Installation of <SUNWkvm> was successful.
[...]
Installation of <SUNWsolnm> was successful.

Solaris 11 software installation succeeded

Solaris 11 packages fully installed
                SUNWkvm
[...]
                SUNWsolnm

Customizing system files
        - Mount points table (/etc/vfstab)
          fd      -       /dev/fd fd      -       no      -
          /proc   -       /proc   proc    -       no      -
          /dev/dsk/c0d0s1         -       -       swap    -       no      -
          /dev/dsk/c0d0s0         /dev/rdsk/c0d0s0        /       ufs     1       no
                  -
          /devices        -       /devices        devfs   -       no      -
          sharefs         -       /etc/dfs/sharetab       sharefs -       no      -
          ctfs    -       /system/contract        ctfs    -       no      -
          objfs   -       /system/object  objfs   -       no      -
          swap    -       /tmp    tmpfs   -       yes     -
        - Network host addresses (/etc/hosts)
        - Environment variables (/etc/default/init)

Cleaning devices

Customizing system devices
        - Physical devices (/devices)
        - Logical devices (/dev)

Installing boot information
        - Updating boot environment configuration file
```

```
        - Installing boot blocks (c0d0)
        - Installing boot blocks (/dev/rdsk/c0d0s0)
Creating boot_archive for /a
updating /a/platform/i86pc/boot_archive
updating /a/platform/i86pc/amd64/boot_archive
```

34.12. A basic automated installation - more polished

Okay, now we've done a basic installation. But mostly we do a standard set of customizations on every system we touch, like installing a few essential tools or integrating the actual recommended patch cluster. So we want to polish the standard installation a little bit. We will extend a basic Solaris 10 Update 5 installation with the following items:

- installing the SUNWstar script
- installing the recommended Patch Cluster
- configuring the Secure-by-default mechanism to limited

34.12.1. Adding the recommended patch cluster

Okay, at first we have to copy the patches into the jumpstart. JET provides the `copy_solaris_patches command` to do so.

```
# copy_solaris_patches 10_x86 /export/home/jmoekamp/10
   _x86_Recommended
Copied....
```

You don't have to configure anything in the templates. Every new Solaris 10 installation on x86 will be installed with the matching recommended patch cluster.

34.12.2. Adding custom packages

Okay, almost everybody installs some customer packages on his/her system. For example, one of the first things im installing on new systems is `joe` to have an WordStar compatible editor:

```
# pkgtrans joe-3.5-sol10-x86-local /tmp all
Transferring <SMCjoe> package instance
# copy_custom_packages /tmp x86 SMCjoe
Transferring <SMCjoe> package instance
Packages copied
```

joe depends on the ncurses library. So we copy this package as well to our JET server.

```
# pkgtrans ncurses-5.6-sol10-x86-local /tmp all
Transferring <SMCncurs> package instance
# copy_custom_packages /tmp x86 SMCncurs
Transferring <SMCncurs> package instance
Packages copied
```

34.12.3. Extending the template

Okay, we need more modules to fulfill this tasks. You don't need to delete the old one and retype all the data in the template. The make_template script can use an old template to create a new one with while retaining all the old values. Basically you use the same name for the old and new template.

```
# ./make_template -f  -T togusa togusa base_config custom sbd
Adding product configuration information for
        + custom
        + sbd
Updating base_config template specifics
Client template created in /opt/SUNWjet/Templates
```

When you look into /opt/SUNWjet/Templates/togusa you will recognize your old configuration, with a large amount of new lines. But we have to change only a few ones:

At first we change the operation system. We've used OpenSolaris in the last example, but there are no patches. But we've copied a Solaris 10 media with the name sol10u5 earlier.

```
base_config_ClientOS=sol10u5
```

Okay, now we want to install the additional packages. You have to add the names of the packages in the line custom_packages.

```
custom_packages="SMCncurs SMCjoe"
```

You don't have to configure the Secure by default module, as this module configures the limited service set when it's used in the template. Patching of the Solaris OE doesn't need configuration as well. So we have to change only this two lines.

34.12.4. The installation

Okay, you can start the installation by starting a network boot. This time the installation takes a little bit longer. First the system starts to install the recommended patch cluster. As I've used a quite actual update, most patches are already installed. but a few ones will find their way into the installation.

```
BASE_CONFIG: Installing base_config....
BASE_CONFIG: Product base_config started
BASE_CONFIG: Installing additional patches from : 10
   _x86_Recommended
BASE_CONFIG: Using patch_order file for patch installation
   sequence
BASE_CONFIG:
BASE_CONFIG: ------ Installing patches for product: 10
   _x86_Recommended ------
BASE_CONFIG:
BASE_CONFIG: Patch 120720-02 is already installed.
BASE_CONFIG: Patch 124458-01 is already installed.
[...]
BASE_CONFIG: <<< 120273-20 (25 of 98) >>>
[...]
BASE_CONFIG: Patch 122175-03 is already installed.
BASE_CONFIG: Patch installation completed.
BASE_CONFIG: No HW specific packages for platform i86pc
BASE_CONFIG: No HW specific patches for platform i86pc
```

Okay, now the system starts to install the additional packages.

```
CUSTOM: Installing custom....
CUSTOM: Installing SMCncurs from: /a/var/opt/sun/jet/js_media/
   pkg/custom/i386
Installation of <SMCncurs> was successful.
CUSTOM: SMCncurs installation complete
CUSTOM: Installing SMCjoe from: /a/var/opt/sun/jet/js_media/pkg
   /custom/i386
Installation of <SMCjoe> was successful.
CUSTOM: SMCjoe installation complete
```

The following module doesn't do really much, as the configuration of the service profile is activated by the `sysidcfg` file.

```
SBD: Installing sbd....
SBD: configured
```

34.12.5. Effects of the new modules

Let's check the results of the installation. At first, we look for the custom packages:

```
# pkginfo | grep "SMC"
application SMCjoe                              joe
application SMCncurs                            ncurses
```

When we look for one of the installed patches, we will see its successful installation to the system:

```
# showrev -p | grep "120273-20"
Patch: 120273-20 Obsoletes:  Requires: 119043-09, 121902-01,
   122532-04 Incompatibles:  Packages: SUNWbzip, SUNWsmagt,
   SUNWsmcmd, SUNWsmmgr
```

34.13. Automatic mirroring of harddisks

Okay, in enterprise computing you wouldn't use a system without redundant boot disks (at least, when you haven't an application that can survive a loss of computing nodes without a problem). So it would be nice to automatically configure a mirror of the boot disks with the Solaris Volume Manager. I assume in the following part, that you have a working knowledge with the . When this not the case, it isn't really a problem, as this part it somewhat self-explanatory.

```
# make_template -f -M -T togusa togusa sds
Adding product configuration information for
        + sds
```

34.13.1. Configuration in the template

As in the last example I will just include the interesting parts of the configuration:

```
base_config_products=" custom sbd sds"
sds_product_version="default"
sds_root_mirror="c1d1"
sds_use_fmthard="yes"
sds_database_locations="rootdisk.s7:3"
sds_database_partition="s7:32"
sds_metadb_size=""
sds_root_alias="rootdisk"
sds_root_mirror_devalias_name="rootmirror"
sds_mirrored_root_flag="yes"
```

At first you have to include the `sds` module to the `base_config_product` line. Then choose the disk you want to use for mirroring, in my case it's `/dev/dsk/c1d1`. Line 4 orders the `sds` module to copy the vtoc of the first mirror to the second one. When there are only two disks in your system you have to specify the `sds_mirror_root_flag` with yes in line 10. want to see the metadb replica copies on at least three disks[6]. You have to tell the system that it shouldn't obey this rule.

34.13.2. Effects of the configuration

Okay, after the installation we can log into the system. You may notice after the installation a vast amount of accesses to the harddisk. The devices for `/` and the `swap` are not longer referenced by an device name. Instead of this you will find the names of the logical devices of the .

```
-bash-3.2$ cat vfstab
#device         device          mount           FS      fsck    mount   mount
#to mount       to fsck         point           type    pass    at boot options
#
fd      -       /dev/fd fd      -       no      -
/proc   -       /proc   proc    -       no      -
/dev/md/dsk/d20 -       -       swap    -       no      -
/dev/md/dsk/d10 /dev/md/rdsk/d10        /       ufs     1       no      logging
/devices        -       /devices        devfs   -       no      -
ctfs    -       /system/contract        ctfs    -       no      -
objfs   -       /system/object  objfs   -       no      -
swap    -       /tmp    tmpfs   -       yes     -
```

Let's look after the exact configuration of the volume manager.

```
-bash-3.2$ metastat
d20: Mirror
    Submirror 0: d21
      State: Okay
    Submirror 1: d22
      State: Okay
    Pass: 1
    Read option: roundrobin (default)
    Write option: parallel (default)
    Size: 530145 blocks (258 MB)

d21: Submirror of d20
    State: Okay
    Size: 530145 blocks (258 MB)
    Stripe 0:
        Device    Start Block  Dbase        State Reloc Hot Spare
        c0d0s1          0     No            Okay   Yes
```

[6] When you want to find out the correct state of an situation you need at least three copies to be sure. With only two copies, one or the other version may be correct, with three you have two correct copies, so there is a good chance that the two copies represent the correct state. Solaris Volume Manager likes to distribute those over at least three disk to ensure that the failure of a disk won't take out exactly half of the copies

```
d22: Submirror of d20
    State: Okay
    Size: 530145 blocks (258 MB)
    Stripe 0:
        Device    Start Block  Dbase        State Reloc Hot Spare
        c1d1s1          0     No            Okay   Yes


d10: Mirror
    Submirror 0: d11
      State: Okay
    Submirror 1: d12
      State: Okay
    Pass: 1
    Read option: roundrobin (default)
    Write option: parallel (default)
    Size: 32836860 blocks (15 GB)

d11: Submirror of d10
    State: Okay
    Size: 32836860 blocks (15 GB)
    Stripe 0:
        Device    Start Block  Dbase        State Reloc Hot Spare
        c0d0s0          0     No            Okay   Yes


d12: Submirror of d10
    State: Okay
    Size: 32836860 blocks (15 GB)
    Stripe 0:
        Device    Start Block  Dbase        State Reloc Hot Spare
        c1d1s0          0     No            Okay   Yes


Device Relocation Information:
Device   Reloc  Device ID
c1d1   Yes      id1,cmdk@AVBOX_HARDDISK=VB37711a7b-00576cc9
c0d0   Yes      id1,cmdk@AVBOX_HARDDISK=VB97a90791-9d191449
```

The default numbering scheme for the Solaris volume manager is quite simple: The mirror is designated with the first number in a decade (e.g. 10,20,30), the parts of a mirror are numbered with the next free numbers in the decade. For example: The first mirror half of the first mirror get the number 11, the second number gets the number 12.

It takes a while until the mirrors are in sync, but after this you have a automatically installed, patched, customized and mirrored system.

34.14. Automatic hardening

34.14.1. Preparing the Jumpstart for installation

At fist you uncompress and untar the distribution.

```
# copy_product_media jass 4.2.0 /export/home/jmoekamp i386
Transferring <SUNWjass> package instance

Packages copied.
```

Okay, but we have to do another step. There is a patch for the version 4.2.0 of the : 122608-xx. At first we have to tell JET that there is a patch for this product and version. We have to modify the file patch.matrix in /opt/SUNWjet/Products/jass.

```
#
# Patch matrix for Solaris Security Toolkit (JASS)
#
# <os>:<arch>:<version>:<patchlist>
#
10:i386:4.2.0:122608
```

Now it's easy to integrate the patch. I've unpacked the patch in the directory /export/home/jmoekamp/pa before:

```
# copy_product_patches jass 4.2.0 /export/home/jmoekamp/
   patch_jass i386

Patches copied.
```

34.14.2. Configuring the template

Okay, you have to configure only a few basic variables to trigger the automatic hardening of your new installation.

```
base_config_products=" custom sbd sds jass"
jass_product_version="4.2.0"
jass_execute="secure.driver"
```

34.14.3. After Jumpstarting

Okay, it's time to reboot the machine we want to install again. At first, all is like at the runs before. But then we see some further lines in the logfile.

```
JASS: Installing jass....
JASS: Installing Solaris Security Toolkit (JASS) 4.2.0...
JASS: Installing SUNWjass from: /a/var/opt/sun/jet/js_media/pkg
   /jass/4.2.0/i386
```

```
Installation of <SUNWjass> was successful.
JASS: SUNWjass installation complete
JASS: Register postinstall script 'postinstall' for boot z
```

It's important to know, that the above configuration installed the `SUNWjass` package on the system, patches it there and then run runs the toolkit installed on the system.

The hardening of the system is started in the background.After a while you will recognize the work of the script. The backup files of the Solaris Security Toolkit are dispersed all over the directories.

```
bash-3.00$ ls -l /etc/*.JASS*
-rw-r--r--   1 root     other         372 May 23 19:48 /etc/
   coreadm.conf.JASS.20080523195314
[...]
-rw-r--r--   1 root     sys           362 May 23 19:43 /etc/
   vfstab.JASS.20080523195420
bash-3.00$
```

After the completion of the background JASS run, you have a automatically installed, patched, customized, mirrored and hardened system.

34.15. Deep Dive to the installation with JET

As I told you before much of the configuration takes place after the installation executed by the Jumpstart mechanism. We used several modules of the JET toolkit so far, thus this is a good moment to do a deep dive into the process, that takes place after the installation.

The concept of this further installs pretty much relies on the concept of a so called finish script. Do you remember the `rules.ok` file? There was a finish script specified in that file for all installations:

```
bash-3.2$ cat rules.ok
any any                 Utils/begin     =       Utils/finish
# version=2 checksum=3114
```

The installation of the Solaris Operating Environment is equal to the normal Jumpstart process, as it relies on the same functions. But then JET comes into the game. After the installation has completed, the script `Utils/finish` is executed. But where is this file. It's relative to an directory we've specified before. Or to be exact, JET did that for is. This is a snippet from the `menu.lst` for our system.

```
title Solaris_11 Jumpstart
        kernel /I86PC.Solaris_11-1/platform/i86pc/kernel/unix
           - install nowin -
B install_config=192.168.10.1:/opt/SUNWjet,sysid_config
   =192.168.10.1:/opt/SUNWje
t/Clients/togusa,install_media=192.168.10.1:/export/install/
   media/solaris/x86/nv
87,install_boot=192.168.10.1:/export/install/media/solaris/x86/
   nv87/boot
        module /I86PC.Solaris_11-1/x86.miniroot
```

The `Utils/finish` is relativ to `install_config`, thus the executed finish script `192.168.10.1:/opt/SU`
The NFS mount specified in `install_config` is one of the first mounts done on the new system and we can use the content of this directory throughout the installation process. By the way: This is the reason, why the `rules.ok` is located at this strange position.

We can study the further process in logfile of the installation. The complete log is located at `/var/opt/sun/jet/` in the file `jumpstart_install.log`. Let's start. At first the finish script starts to take copy some components from the Jumpstart server to the local disk.

```
Creating directory: /a/var/opt/sun/jet/post_install
Creating directory: /a/var/opt/sun/jet/Utils
Creating directory: /a/var/opt/sun/jet/config
Creating directory: /a/var/opt/sun/jet/js_media/patch
Creating directory: /a/var/opt/sun/jet/js_media/pkg
Copying file Clients/togusa/sysidcfg to /a/var/opt/sun/jet/
   config/sysidcfg
Copying file Clients/togusa/profile to /a/var/opt/sun/jet/
   config/profile
Copying file Clients/togusa/host.config to /a/var/opt/sun/jet/
   config/host.config
Copying file Utils/solaris/releaseinfo to /a/var/opt/sun/jet/
   config/releaseinfo
Copying functions to /a/var/opt/sun/jet/Utils/lib
Copying file Clients/togusa/module_hints to /a/var/opt/sun/jet/
   config/module_hints
```

As you see, the JET copies over part of the toolkit as well as configuration files to the new position. But why are all those directories relative to `/a`. Well this is easy. In the netbooted mini root, the local disks are mounted relative to `/a`. The reasoning behind this copy is relatively simple. In the next boots the contents of `/opt/SUNWjet/` aren't available any longer, as the system doesn't mount it in the next steps. The

post installation scripts rely on some helper function. The simplest way to ensure their availability under all circumstances [7] is a simple copy.

The next step is the mounting of the directories with patches and product.

```
Mounting nfs://192.168.10.1/export/install/patches on /var/opt/
   sun/jet/js_media/patch
Mounting nfs://192.168.10.1/export/install/pkgs on /var/opt/sun
   /jet/js_media/pkg
```

Now it get's a little bit complicated, but I will simplify it as far as I can. Depending on the complexity of the setup your configuration may use one or more so called products. A product in JET-speak is a JET module for the installation and configuration of a certain area of the operating system. In any case you will use the product `base_config` but there may be several ones. Our example uses the products `base_config`, `custom`, `sds` and `jass`. The JET framework gathers this information from the configuration template. It's stored in this line:

```
base_config_products=" custom sbd sds jass"
```

The framework takes this information and to execute the `install` script in any of this directories. For example it starts at first the `install` script in `/opt/SUNWjet/Products/base_config/sol` as this is default for every installation, after this it will step forward by executing the `install` script in any of the product directories. The `install` script has two important roles. At first it installs packages, patches and files according to the configuration in the templates. At second it registers so called `post_install` scripts.

34.15.1. Post installation scripts

`post_install scripts` are executed at the next boots. You can order this scripts by specifing a certain boot level. After the execution of all scripts in a boot level, the system reboots. For example all scripts with boot level 1 are executed after the first reboot, all `post_install` scripts with the boot level 2 are executed after the second reboot and so on. These scripts are installed in `/var/opt/sun/jet/post_install`

But how get this scripts to execution at the following reboots. JET copies a `init.d` script to the new system. On Solaris 10 it creates a matching SMF service. The function of this script is quite simple: Gather the actual boot level by reading the file `/var/opt/sun/jet/post_install/Platform`, execute all scripts in the boot level, increment the bootlevel by one and reboot the system.

[7] even when your installation disables the network

34.15.2. An example for boot levels and postinstall scripts

We've done the first boot. It booted from the network. The installation of the Solaris Operating Environment succeeded, thus the script from the `finish` column is executed by the netbooted installation system. After some configuration tasks the system starts to register `postinstall` scripts.

```
SDS: Installing sds....
SDS: Copying file sds_functions to /a/var/opt/sun/jet/Utils/
   sds_functions
SDS: Creating directory: /a/var/opt/sun/jet/post_install/
   Platform/1
SDS: Register postinstall script 'create_fmthard' for boot 1
SDS: Register postinstall script 'set_boot_device' for boot 1
SDS: Register postinstall script 'create_metadb' for boot 1
SDS: Register postinstall script 'create_root_mirror' for boot
   1
SDS: Register postinstall script 'attach_mirrors' for boot z
SDS: Register postinstall script 'create_user_devices' for boot
    1
SDS: Register postinstall script 'attach_user_mirrors' for boot
    z
SDS: WARNING: unable to locate specified md.tab for SDS.
SDS: Enable md:mirrored_root_flag in /etc/system
```

You see, that the SDS product registered some scripts for boot level 1 and some for boot level z. Let's look further into the installation log. This happens after the first reboot:

```
SDS: Running 001.sds.001.create_fmthard
fmthard:  New volume table of contents now in place.
SDS: Running 001.sds.001.set_boot_device
SDS: Setting OBP nvramrc rootdisk path
[...]
SDS: Create 3 copies on c1d1s7
metadb: waiting on /etc/lvm/lock
SDS: Running 001.sds.003.create_root_mirror
SDS: Setting OBP nvramrc rootmirror path
[...]
SDS: Installing GRUB on Alternate Boot Disk.
SDS: Running 001.sds.007.create_user_devices
```

Later on you will recognize the scripts for boot level z

```
Running additional install files for reboot z
SDS: Running 003.sds.001.attach_mirrors
SDS: Attach d12 to d10
```

```
SDS: Attach d22 to d20
SDS: Attach d12 to d10
d10: submirror d12 is attached
SDS: Attach d22 to d20
d20: submirror d22 is attached
SDS: Running 003.sds.002.attach_user_mirrors
```

With this mechanism, you can implement installation processes, that needs package or programm installations that need several boots to fullfill.

34.15.3. The end of the post installation

At the very end the init.d script is deleted together with the matching SMF service.

34.16. Using Jumpstart Flash

As I stated before, the installation mechanism of Jumpstart Flash is quite different from a normal installation. So I will start with a new template on a new server to demonstrate Jumpstart Flash.

34.16.1. Creating a flash archive

The first step to do a flash based install is the creation of a flash archive. Obviously you need an already installed system to create such an archive. To do so, I create a directory for the flar image at first:

```
bash-3.00# mkdir /flardir
bash-3.00# flarcreate -S -n "togusa" -x /flardir -R / /flardir/
   togusa.flar
Full Flash
Checking integrity...
Integrity OK.
Running precreation scripts...
Precreation scripts done.
Creating the archive...
5144848 blocks
Archive creation complete.
Running postcreation scripts...
Postcreation scripts done.
```

```
Running pre-exit scripts...
Pre-exit scripts done.
bash-3.00#
```

The second command generates the flar image itself. With this command, I generate the flash archive `togusa.flar` in the directory `/flar`. The `-x` option excludes the directory `/flardir from the flasharchive. The \verb-R=` specifies that the flash archive should contain all filesystems descending to `/`. The `-S` flag bypasses the size checks.

```
# scp togusa.flar jmoekamp@192.168.10.1:/export/flar/togusa/
   togusa.flar
```

After the creation of the flash archive, you have to transmit it to a server. It doesn't have to be the jumpstart server. it doesn't even have to be an NFS server. It just have to be reachable with HTTP or NFS by the server you plan to install. In this example we will use the Jumpstart Server for this task, thus we will use a share on this system.

Don't forget to share this directory via NFS:

```
# echo "share -F nfs -o anon=0,sec=sys,ro -d \"Installation
   Images\" /export/install" >> /etc/dfs/dfstab
# shareall
```

34.16.2. Preparing the template

Okay, we want to install our node `ishikawa` with the flash image of `togusa`. At first like with all jet installation we add a hostname to our `/etc/hosts`

```
# echo "192.168.10.11 ishikawa" >> /etc/hosts
\end|lstlisting}
Now we generate the template. We need only a small template for
    the basic installation.
\begin{lstlisting}
# make_template ishikawa flash
Adding product configuration information for
        + base_config
        + flash
Updating base_config template specifics
Client template created in /opt/SUNWjet/Templates
```

Okay, we have to fill the templates with standard settings.

```
base_config_ClientArch=i86pc
base_config_ClientEther=08:00:27:DE:91:AB
base_config_ClientOS=sol10u5
```

The use operating system here isn't important. You will not install this operating system. You will install the operating system contained in the flash archive.

Fill out the `sysidsection` of the system like with a normal system, you just need to fill the data to prevent the system from going interactive.

Now we get to the configuration of the flash install. You just define one or more locations of flash archives. When the installation in your flash archive contains all recommended patches you can save some time at the installation and skip the installation by using `yes` for the `flash_skip_recommended_patches` parameter.

```
flash_archive_locations="nfs://192.168.10.1/export/flar/togusa/
   togusa.flar"
flash_skip_recommended_patches="yes"
```

Now we have to create the Jumpstart environment for this client.

```
# /opt/SUNWjet/bin/make_client -f ishikawa
Gathering network information..
        Client: 192.168.10.11 (192.168.10.0/255.255.255.0)
        Server: 192.168.10.1 (192.168.10.0/255.255.255.0, SunOS
           )
Solaris: client_prevalidate
Creating Client directory for ishikawa
Solaris: client_build
Creating sysidcfg
Creating profile
Adding base_config specifics to client configuration
Adding flash specifics to client configuration
FLASH: Modifying client profile for flash install
FLASH: Removing package/cluster/usedisk entries from profile
Solaris: Configuring JumpStart boot for ishikawa
         Starting SMF services for JumpStart
Solaris: Configure PXE/grub build
         Adding install client
        Doing a TEXT based install
         Leaving the graphical device as the primary console
         Configuring ishikawa macro
         Using local dhcp server
         PXE/grub configuration complete
Running '/opt/SUNWjet/bin/check_client  ishikawa'
        Client: 192.168.10.11 (192.168.10.0/255.255.255.0)
```

```
         Server: 192.168.10.1 (192.168.10.0/255.255.255.0, SunOS
            )
Checking product base_config/solaris
Checking product flash
FLASH: Checking nfs://192.168.10.1/export/flar/togusa/togusa.
   flar
------------------------------------------------------------
Check of client ishikawa
-> Passed....
```

Please note the single line with `FLASH:` at the beginning. The JET framework checks for the availability of the flash archive. This prevents one of the most occurring problems with flash... a unaccessible flash archive at the installation.

When we look into the `profile` for `ishikawa` you will recognize, that all statements regarding cluster to install or similar stuff is removed. But there is a new statement. `archive_location` specifies the location of the flar image and `install_type` tells the system to do a flash install.

```
# cat profile
#
# This is an automatically generated profile. Please modify the
    template.
#
# Created:      Sat May 24 12:50:20 CEST 2008
#
install_type    flash_install
archive_location        nfs://192.168.10.1/export/flar/togusa/
   togusa.flar
partitioning    explicit
#
# Disk layouts
#
filesys         rootdisk.s0     free    /
filesys         rootdisk.s1     256     swap
# pwd
/opt/SUNWjet/Clients/ishikawa
#
```

34.16.3. While Jumpstarting

When you look into the logfiles of your system, you will notice a big change. You won't see a large amount of messages from the package installation. Instead of this you will

just see the progress notifications of the extraction of the flar archive.

You will find this information in the middle of: `/var/sadm/system/logs/install_log`

```
[...]
Beginning Flash archive processing

Extracting archive: togusa
        Extracted    0.00 MB (  0% of 2512.13 MB archive)
[...]
        Extracted 2512.13 MB (100% of 2512.13 MB archive)
        Extraction complete
[...]
```

The JET framework augments this with some housekeeping tasks like deleting the Solaris Volume Manager configuration. As usual you can look into the `/var/opt/sun/jet/finish.log` logfile to find the related messages:

```
FLASH: Installing flash....
FLASH: Disabling /a/etc/lvm/mddb.cf -> /a/etc/lvm/mddb.cf.
   disabled
FLASH: Purging entries from /a/etc/lvm/md.tab
FLASH: Disabling mirrored_root_flag in /etc/system
FLASH: Cleanout crash dump area
FLASH: Clear out devfsadm files
```

34.17. Using Jumpstart Flash for System Recovery

34.17.1. The basic trick

My colleague Mario Beck showed it to me several years ago. The problem is the sysunconfig as it deletes the configuration information. The basic trick is simple. Before you create the flash archive, just do a backup of those files in one of the directories covered by the archive creation. Later on you can use this data to recover the configuration by copying those files in their original location.

34.17.2. Using an augmented Flash archive

The flash archive is just a cpio copy of the complete system minus explicitly excluded parts. Everything on the system is included in the flash archive, not just the OS[8]. So we

[8] otherwise the flash archives would make sense

can easily rescue the personality of the system into the flash archive.

To ease up this task, I use the following script:

```
#!/bin/sh
mkdir -p /var/opt/recovery
mkdir -p /var/opt/recovery/etc
cp -p /etc/hosts /var/opt/recovery/etc
cp -p /etc/shadow /var/opt/recovery/etc
cp -p /etc/passwd /var/opt/recovery/etc
cp -p /etc/vfstab /var/opt/recovery/etc
cp -p /etc/nodename /var/opt/recovery/etc
cp -p /etc/hostname.* /var/opt/recovery/etc
cp -p /etc/dhcp.* /var/opt/recovery/etc
cp -p /etc/defaultdomain /var/opt/recovery/etc
cp -p /etc/TIMEZONE /var/opt/recovery/etc
mkdir -p /var/opt/recovery/etc/inet
cp -p /etc/inet/netmasks /var/opt/recovery/etc/inet/netmasks
cp -p /etc/defaultrouter /var/opt/recovery/etc/defaultrouter
mkdir -p /var/opt/recovery/var/ldap
cp -p /etc/.rootkey /var/opt/recovery/etc
cp -p /etc/resolv.conf /var/opt/recovery/etc
cp -p /etc/sysidcfg /var/opt/recovery/etc
cp -p /var/ldap/ldap_client_cache /var/opt/recovery/var/ldap/
   ldap_client_cache
cp -p /var/ldap/ldap_client_file /var/opt/recovery/var/ldap/
   ldap_client_file
cp -p /var/ldap/ldap_client_cred /var/opt/recovery/var/ldap/
   ldap_client_cred
cp -p /var/ldap/cachemgr.log /var/opt/recovery/var/ldap/
   cachemgr.log
mkdir -p /var/opt/recovery/var/nis
cp -p /var/nis/NIS_COLD_START /var/opt/recovery/var/nis
mkdir -p /var/opt/recovery/var/yp
cp -R -p /var/yp/* /var/opt/recovery/var/yp
```

When you create a flar archive after running this script, it will include this directory in the archive, thus a newly installed machine with this flash archive will have this directory as well. So you can use it to recover the old status of the system. The process is simple. Just do a `cp -R /var/opt/recovery/* /`. The process of jumpstarting the server is identical of doing a normal flash install.

34.18. Conclusion

JET and Jumpstart are incredible powerful tools to ease the installation of your Solaris systems. You can install your systems via network, configure and customize them. With tools like Jumpstart FLASH it's easy to clone systems for large computing cluster.

Even when you have only a few systems, it's a good practice to install you systems by a Jumpstart server. A bare metal recovery of your system is much easier, you can use a common template to setup your system and beside the hard disk space the jumpstart server needs only resources at the moment of the installation.

34.18.1. Do you want to learn more?

Documentation

Solaris 10 5/08 Installation Guide: Solaris Flash Archives (Creation and Installation)[9]

Misc.

wikis.sun.com: JET Homepage[10]

jet.maui.co.uk: JET resources [11]

[9] http://docs.sun.com/app/docs/doc/820-4328
[10] http://wikis.sun.com/display/JET/Home
[11] http://jet.maui.co.uk/wiki/index.php/Main_Page

35. Small tricks

Solaris 10/Opensolaris

35.1. Less is more

Less is more in Linux. Or to be exact `less` is `more`. The standard pager in Solaris is `more`, the standard pager in Linux is `less`. So you want `less` when looking into man pages? That's easy: Just put

```
export PAGER="less"
```

into your `.profile` file and execute `source ~/.profile` to get the new setting without logging out and in again.

Part VII.

Nontechnical feature

36. Long support cycles

36.1. The support cycle

Enterprise customers want to protect their investments. They don't want to use an operating system only to be forced in two years to use a different one, because they won't get any support. Most companies have a really long process of validating new configurations and they don't want to go through it without a good reason (bigger hardware updates or when they use a new software). Thus you have to support a commercial unix quite long.

Out of this reason, Sun has an well defined and long life cycle for the releases of our operating environment.

This is the policy of Sun for lifecycles. We won't shorten the time, but often the effective lifetime is much longer, as you will see in the next paragraph.

36.2. An example: Solaris 8

Okay, we started to ship Solaris 8 in February 2000.Last order date (E4) was at November 16, 2006. After that we shipped Solaris 8 Media Kits until February 16, 2007. Solaris 8 entered Retirement support mode Phase I on Mach 31, 2007. Thus it will reach Retirement Support Mode Phase II on March 31, 2009. End of Service Life is on March 31, 2012. Thus Solaris 8 has an service life of 12 years.

Table 36.1.: Events in the lifecycle of a Solaris release

Event	Name	Description
E1	General Availability (GA)	GA is a day of joy, celebrations and marketing. A new major release of Solaris is born, for example the first release of Solaris 10. In at least the next 4 and a half years your will see several updates to this version of the operating system.
E2	End of Life (EOL) Pre-Notification	Okay, one year before we announce the formal End of Life of the product, Sun sends the first notification/warning to our customers.
E3	End of Life (EOL) Announcement	Okay, we announce the end of the development of a major Solaris release. As I told before, this is at least 54 month after the GA, sometimes longer than that. When we announce the EOL, we trigger the start of the next phase in the lifecycle of a Solaris release, the last order phase.
E4	Last Order Date (LOD)	90 days after the EOL announcement is the Last-Order-Date. This is the last day, you can order a Solaris version as an preinstalled image and it's the last a purchased system includes the license for this specific operating system. This Last Order Date isn't effective for Support Contracts. You can order a support contract for a Solaris version until its End of Service Life. With the Last Order day the next phase is started: The last ship phase
E5	Last Ship Date (LSD)	In the next 90 days all orders for a new version has to be fulfilled. Yeah, you can't order an EOLed operating system for delivery a year after its end-of-life (besides of special agreements). With the Last-Order-Date the retirement of the Solaris Version starts.
E6	End of Retirement Support Phase 1	For the next two years you have essentially the same service than before EOL with some exceptions. No fixes for cosmetic bugs, no feature enhancements, no quarterly updates.
E7	End of Retirement Support Phase 2	In the last phase of the lifecycle, you still get telephone support for a version and you can still download patches for the system, but there will be no new patches.
after E7	End of Service Life (EOSL)	After EOSL you can't get further support or patches with the exception of special agreements between the customer and Sun.

36.3. Sidenote

For a customer this long release cycles are optimal, but there is a problem for Sun in it. We don't force our customer to use new versions early. Some customers still use old Solaris 8 versions and they use Solaris 10 like Solaris 8 to keep the processes in sync. There are some technological leaps between 8 and 10, but they don't use the new features. They think they know Solaris, but they know just 8, not 10. The reputation of being somewhat outdated has its root partly in this habit. This is the bad side of the medal, but long support cycles are too important to change this policy...

36.4. Do you want to learn more

Disclaimer: The following documents are the authoritative source. If I made a mistake at summarizing them, the informations in these both documents are the valid ones.

Solaris Operating System Life Cycle

Solaris Operating System Retirement End Of Life Matrix

Part VIII.

Licensing

Part IX.

TODO

- proof reading
- typo hunting

List of Tables

List of Figures