

Saturday, October 4, 2008

Going to lunch - with the eyes of an IT architect

When you work as an architect in the IT industry you tend to look at the non-IT processes in your world with the eyes of an IT architect. And at many occasions you find the same problems. Perhaps the design techniques of efficient CPUs and software can help to do a really efficient lunch distribution ... and i'm sure some of the problems of CPU or software design were solved by observing the people at lunch time.

Let's have just a look at the lunch shops around the Sun office in Hamburg.

Mickleys - Problems with singletonsMickleys is a Turkish deli. This is a nice example for problems introduced by singletons. The processing ("giving you the food") is really efficient. But the cashier is implemented as a singleton, just one cashier but up to three processing threads. The processing threads idle most of the time because of the cashier thread blocking the queue. After a short time the queue gets so long that the processing threads can't dispense any data ... eer ... food.

Sultans - Stateless session transferSultans is a doner shop. They have the same singleton problem like Mickleys but the cashier process is really efficient. But Sultans is a good example for a stateless transfer of sessions between two services. You can look at Sultans as a service separated into two subservices: The cashier service and the doner making service. When you pay for your doner you open the session, but the data of the "session context" isn't distributed to "doner making service". After paying for your doner your session will be requeued to the "doner making service" queue and you specify your doner structure again ("doner plate with fries and spicy sauce"). By doing so the waiting requests can be easily loadbalanced to several doner making services (the one at the Sultans near the office has two doner making services executing independently from each other). By the way: The doner plate at Sultans is really a good lunch.

Oh it's fresh - Scheduling in multiprocessor systemsOkay ... at "Oh it's fresh" (OIF in the following text, but it's called HAM03 sometimes, over the day you find at least on Sun employee there making a short break at any time) you find a good example of process scheduling in multiprocessor systems. At OIF you have two execution units consisting of a CPU (cash processing unit) and a FPU (food processing unit) each. You have a large cache with preprocessed food. All consumer processes are queued in a run queue. All consumers are in a state where they are runnable but want to be scheduled to cash processing unit. When both cash processing units are functional, there is a run queue for every cash processing unit. Often the consumers can be served without waiting for other high latency. As soon as the consumer is scheduled to a Cash processing unit. As long as the lunch is in the cache, the process is completely executed without leaving the cash processing unit. Sometimes there is a high latency event. Sometimes there is a high latency event. For example, when you order a Latte, the job is transferred to the "Coffeemaking Offloading Engine", the consumer process gets preempted and the process is put on the sleep queue. Now the next consumer process is scheduled to the CPU. As soon as the high latency event is over (the Coffeemaking Offloading Engine has prepared the Latte), the actual consumer is rescheduled to the run queue with a priority as high as the actual process. Directly after the completion or the transfer of the new customer to the sleep queue, the longest waiting consumer waiting for the deliverable resource is executed again. It's like in Solaris ... there are conditions in the sleep queue. Only the matching consumers are woken up by seeing a glass of Latte, nevertheless the scheduler is aware of the priority of the consumer in the sleep queue.

I don't really like this shop. Some of the people over there got overconfident about the fact, that they work at a relatively hip deli there. I don't like to get insulted by a guy with a third of my IQ and a tenth of my annual income. When you get violent thoughts like jumping over the counter and putting the mug of coffee into a body hole that wasn't invented for consuming coffee, you should choose a different coffee-dealer shop. By the way: The coffee is horrible over there ... just a wad of water with homeopathic doses of coffee beans used in the process. The substance over there isn't far away from the sign "No coffee beans were harmed during cooking this coffee". Just walk hundred meters and you got much better coffee and a much friendlier CPU/FPU.

Pic-a-deli - The thundering herd problemOIF has a well implemented lunch scheduling mechanism. Now I want to get to an inefficient lunch scheduling. This scheduling doesn't now priorities. So you have an interesting effect in the design of this scheduling mechanism. It's called "thundering herd". The Pic-a-deli has this problem (BTW: all scheduling mechanisms are affected by the "thundering herd" problem, the question is just the severity of the effect). At first you are in the run-queue for the cash processing unit. After this the guests are moved to sleep-queue. The food processing unit is somewhat chaotic. The FPU yells "Pasta", all consumers who have ordered "Pasta" on the sleep queue wake up

and fight for the access to the "Pasta". The scheduling of the FPU is unaware of the sequence thus processes can really starve in the sleep queue. This problem is really similar to the "thundering herd problem". It's not uncommon that a colleague gets it food when others are already finished, despite of ordering it at almost the same time ...

Mc Donalds - Speculative Execution/Branch Prediction When you eat at Mc Donalds you see the effect of branch prediction. Let's assume you want to order a meal ... let's assume you want to order a BigMac (a quarter-pounder with cheese) ... most of the times this is a low-latency event, you get your fries and your burger within seconds, but when you order an McRib you wait a few minutes. This is branch prediction and speculative execution in action. The staff at speculatively executes the preparing of BigMacs, so the cache (the hot bay behind the desk) is filled with a certain amount of BigMacs at any time. In 90% of all cases the consumer decide for a Big Mac. Thus ordering a BigMac is a low-latency event. But in 10% the consumer threads orders a McRib. Well ... now you you have high-latency event ... the cache doesn't contain any McRibs. When there is nothing much to do, you stay at the top of the queue, but the pipeline is stalled for this moment. At lunch time, the Cash Processing Unit does a context switch and processes the next consumer process. When the high-latency event of preparing a McRib is completed, the Cash Processing Unit switches back to the waiting process after the new process has completed. The new process isn't preempted by the completion of the high-latency event.

McDonalds - Scout Threading McDonalds is good example for Scout Threading, too. Let's assume you are stuck at the telephone, but a group of colleagues are almost starved, thus they can't wait any longer, but they are willing to order a McRib for you. You leave the office two minutes later. As you arrive at McDonalds the cache is already warmed, as the scout colleagues already ordered a load of a McRib into the cache two minutes earlier. The complete high-latency event of preparing a McRib is hidden by the scout colleagues doing the upfront order.

The concept of scout threads in computing is exactly the same. A scout thread runs 100 cycles or so upfront to the real thread and warms the cache, so the real thread doesn't have to wait for the data.

Posted by Joerg Moellenkamp in Computing at 15:53