Operating Systems

Tutorial 2 & 16

Michael Tänzer

os-tut@nhng.de
http://os-tut.nhng.de

Calendar Week 48

Outline

True or False

- Multi-threading is always preferable to multiple processes.
- In most cases `CreateProcess` has better performance than `fork` & `execve`.
- The One-to-One model has a higher overhead than Many-to-One because it has to do two switches, one at the user level and one in the kernel.

Explain turnaround, waiting and response time

Turnaround time Time from submission of a process to its completion – includes time spent in waiting queues and running on the CPU

Waiting time Time spent in the ready queue

Response time Time from the submission of a process until it runs for the first time. If processes aren't preempted and never block $\Rightarrow$ waiting time = response time

How does SJF work?

Preemptive vs. non-preemptive SJF

- Select job with the shortest *remaining* time
- Most of the time not possible (total time to completion unknown)

$\Rightarrow$ use estimated length of next CPU burst

- The preemtive version makes a new decision when a new process arrives at the ready queue

What's the basic idea of priority scheduling?

What's the major problem of priority based algorithms?

- Each process is assigned a priority
- Choose the process with the highest priority from the ready queue
- Need another scheduling policy if there are multiple processes with the same priority (e. g. round robin)
- Starvation may occur if there is a process with a low priority and there are always processes with higher priorities ready to run

Explain the multilevel feedback queue algorithm

What kind of processes can be found in the higher and lower queues?

- Multiple queues
- Processes are taken from highest non-empty queue
- Need another scheduling policy to decide which process to take from the queue
- Higher queues $\Rightarrow$ short time slices, lower queues $\Rightarrow$ long time slices
- Process uses the entire time slice $\Rightarrow$ move it down to the next queue
- Process blocks $\Rightarrow$ when it becomes ready again put it in the queue directly above the one it was in when it blocked
- Lower queues will contain CPU bound and higher ones I/O bound processes

How can starvation be avoided?

- Priority ageing: While a process is waiting its priority is increased
- In the case of multilevel feedback queues – when a process is waiting for a long time without getting the CPU it is moved up to the next queue

Describe the idea of lottery scheduling

- Each process gets a number of lottery tickets
- The scheduler draws a ticket
- The process owning that ticket gets the CPU

Enumerate possible advantages of lottery scheduling over priority scheduling

- No starvation (if each process gets at least one ticket)
- Possible to grant a process a specific percentage of CPU time (proportional to the number of tickets)
- Possible to have a hierarchical distribution of CPU time (each user gets *n* tickets which he can assign to the applications he wants to run)
- Possible to give CPU time to another process (e. g. the file server) to allow it to process own requests (by donating tickets to it)

Compute per process and average turnaround time

- Round robin (RR): time slice 1 minute
- Priority-based scheduling
- First come first served (FCFS)
- Shortest job first (SJF)

- Five batch processes *A* to *E* arrive at the same time
- Execution times are 15, 9, 3, 6 and 12 minutes
- Static priorities 4, 7, 3, 1 and 6 with 10 being the highest priority
- All policies (except round robin) non-preemptive and no process makes blocking syscalls

What are common values for the length of a time slice?

From 10 to 100 ms

Short vs. long time slices

In what kind of systems would you use which?

- Long time slice

⇒ fewer context switches (less overhead)

⇒ higher throughput

⇒ good for batch systems

- Short time slice

⇒ ready processes don't have to wait long until they're executed

⇒ better responsiveness of the system

⇒ good for interactive systems

What is Linux' tickless mode?

- The timer interrupt isn't raised at fixed intervals but when it's needed (i. e. the next future event)

$\Rightarrow$ no timer interrupts when the CPU is idle

$\Rightarrow$ deeper sleep states possible (helps to save energy)

What considerations apply to SMP but not single-processor scheduling?

- Shared data between threads (reuse entries in per processor caches)
- Communication between threads (communicating threads should be run in parallel so that they don't have to wait until the partner gets the message)

$\Rightarrow$ need to adapt policy according to the behaviour of the concrete application

Central vs. per processor ready queue

Central queue

- Access has to be synchronized (poor scalability)
- Changes to the ready queue cause caches of it to be invalidated on the other processors

Per processor queue

- Extra mechanisms for load balancing needed
- Better cache utilisation as processes are run on the same CPU (cache affinity)

Prove that SJF provides the minimum turnaround time among all non-preemptive scheduling algorithms

Yay, it's getting mathematical

Questions & Comments

Any questions or comments?

The End

The End