ECE 598 – Advanced Operating Systems
Lecture 23

Vince Weaver

http://www.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

23 April 2015

Announcements

- HW#8 will be out as soon as possible

Memory Barriers

- Not a lock, but might be needed when doing locking
- Modern out-of-order processors can execute loads or stores out-of-order
- What happens a load or store bypasses a lock instruction?
- Processor Memory Ordering Models, not fun

Resources

- If you do not give exclusive access, bad things can happen. Imagine one process printing a document, half done and another task switched in and also starts writing to the printer.
- Pre-emptible resource
- Non-preemptible resource.
- Usually protected by locks.

- More complex if protected by two or more locks (need two resources)

Deadlock

- Two processes both waiting for the other to finish, get stuck
- One possibility is a bad combination of locks, program gets stuck
- P1 takes Lock A. P2 takes Lock B. P1 then tries to take lock B and P2 tries to take Lock A.

Livelock

- Processes change state, but still no forward progress.
- Two people trying to avoid each other in a hall.
- Can be harder to detect

Starvation

- Not really a deadlock, but if there's a minor amount of unfairness in the locking mechanism one process might get "starved" (i.e. never get a chance to run) even though the other processes are properly taking and freeing the locks.

How to avoid Deadlock

- Don't write buggy code
- Reboot the system
- Kill off stuck processes
- Pre-emption (let one of the stuck processes run anyway)
- Rollback (checkpoint occasionally)

Priority Inversion

- Low-importance task interrupts a high-priority one
- Say you have a camera. Low-priority job takes lock to take picture.
- High-priority task wants to use the camera, spins waiting for it to be free. But since it is high-priority, the low priority task can never run to free the lock.

Locking in your OS

- When?
- Interrupts
- Multi-processor
- Pre-emptive kernel (used for lower latencies)
- Big-kernel lock? Fine-grained locking? Transactional memory?

- Semaphores? Mutexes
- Linux futexes?

Does our OS need locks?

- We don't have many shared resources yet.
- Setting/reading the time, if not-atomic and updated by interrupt
- What if multiple processes try to write the console at the same time?

Scheduling

- Picks which jobs to run when
- Complex problem
- Simple: batch scheduling. Each run to completion.
- Multi-tasking.
- Computation often mixed with slow I/O
- Avoid context switching if possible

- Can switch when task voluntarily yields, if kernel blocks on I/O, or if timeslice runs out
- Simple round-robin scheduling
- Different type of processes. Long-running CPU bound where extra latency doesn't matter? Interactive things like GUI interfaces, video games, music playing where too much delay is bad? Real time constraints?

Scheduling Goals

- All: fairness, balance
- Batch: throughput (max jobs/hour), turnaround (time from submission to completion), CPU utilization (want it busy)
- Interactive: fast response, doesn't annoy users
- Real-time: meet deadlines, determinism

Batch Scheduling

- First-come-first-served (what if 2-day long job submitted first)
- Shortest job first
- Many others

Interactive Scheduling

- Round-robin
- Priority – "nice" on UNIX
- Multiple Queues
- Others (shortest process, guaranteed, lottery)
- Fair scheduling – per user rather than per process

Real-time Scheduling

- Complex, more examples in 471 or real time OS course

The Linux Scheduler

- People often propose modifying the scheduler. That is tricky.
- Scheduler picks which jobs to run when.
- Optimal scheduler hard. What makes sense for a long-running HPC job doesn't necessarily make sense for an interactive GUI session. Also things like I/O (disk) get involved.
- You don't want it to have high latency

- Linux originally had a simple circular scheduler. Then for 2.4 through 2.6 had an O(N) scheduler

- Then in 2.6 until 2.6.23 had an O(1) scheduler (constant time, no many how many processes).

- Currently the "Completely Fair Scheduler" (with lots of drama). Is O(log N). Implementation of "weighted fair queuing"

- How do you schedule? Power? Per-task (5 jobs, each get 20%). Per user? (5 users, each get 20%).

Per-process? Per-thread? Multi-processors? Hyper-threading? Heterogeneous cores? Thermal issues?