Scheduling Continued

Azza Abouzied

Last Class: The Completely Fair Scheduler

The Completely-Fair Scheduler

Works by having

1. **a more fluid notion of a time-slice:** the base time slice depends on the number of currently runnable processes (i.e. size of the runqueue) and each process is allocated a weigthed time slice (weighted by priority). Recomputed every few timer ticks.
2. **a more fluid notion of time itself**: uses virtual runtime to track how long a process has been running, but time is relative!

Solution 3: Lottery Scheduling

How does it work?

1. Each job gets a set of tickets.
2. Randomly pick a set of tickets
3. Schedule those

How does it allow for?

1. Priority scheduling
2. Promoting shorter jobs
3. Allowing Cooperation
4. Ensuring Fairness or Preventing Starvation

Interaction effects

The interplay of synchronization and preemption/scheduling can have the following effects:

1. **Priority Inversion Problems.** Low-priority thread acquires lock needed by high-priority thread.
   Solution: **Priority Inheritance***:* When a thread holds a lock that other threads are waiting on, give that thread the priority of the highest-priority thread waiting to get the lock.

2. **Convoy Effects.** A thread acquires the lock, then suspends. Other threads come along, and need to acquire the lock to perform their operations. Everybody suspends until the lock that has the thread wakes up.

Scheduling across Processors

Scheduling across Processors

Two flavors:

1. **Centralized scheduling.** Works best for asymmetric multiprocessing architectures. A dedicated master/scheduler processor: the scheduler can become a bottleneck

2. **Distributed scheduling.** Works for symmetric multiprocessing (SMP) architectures: each processor has its scheduler and its own set of jobs.
   Load Balancing:
   - Job pushing
   - Work stealing

   Processor Affinity: Keep a job on the same processor where all the memory/cache data exists.

Which works best?

The TSA way: Single queue, multiple servers

The Carrefour way: Multiple queues, multiple servers

Design Exercise

Design an elevator

- m elevators: start with $m = 1$ then scale. Be careful of concurrency control.
- n floors
- R_t a request at time time t. Requests are either up or down.

What are you optimizing for? How did that influence your policy?

Questions?