CS415 Module 6 Part C - Case Study: Linux Scheduling

Athens State University

1 Some history

The Linux Scheduler Has A Checkered History

- Remember that Linus Torvalds:
  1. Built the first versions of Linux while a graduate student
  2. And can be a bit of a jerk
- So the early versions of the Linux scheduler was rather primitive
  - Based on scheduler in Classical Linux
  - Didn't do well with SMP and multicore machines
- Many different approaches have been taken to address these problems
  - Current kernels implement the Completely Fair Scheduler (CFS)

Back to reality

- The Linux scheduler tries to be very efficient
- But has does some rather complex processing to make that happen
- Design philosophies
  - Provide large time slices for important processes
  - Adjust time slice based on CPU use
  - Bind processes to CPU cores
  - Try to do everything in $O(1)$ time

2 Processor Scheduling

Processor Scheduling

- Provide separate run queue for processor
- Scheduler only selects processes for a processor from that processor's run queue
  - Means that a processor may go idle even if jobs are waiting in other processors ready queue
- Processor run queues are periodically rebalanced
  - Jobs are moved to other run queues in order to balance the length of the queues

Processor Affinity

- **Processor Affinity**: Setting a preference for which CPU upon which a process will run
- A structure in the process control block can be adjusted to set the affinity
- At process creation, a process is set to run on any CPU, this can be changed by the process
- Process affinity settings are inherited by child processes
- Rebalancing does not override affinity

Basic Scheduling Algorithm

- Find the highest-priority queue with a runnable process
- Find the first process on that queue
- Calculate the size of its time slice
- Let run
- When the time slice expires, put the process on the *expired* queue
- Repeat

Run Queues

- There are, by default, 140 run queues, one for each priority level
- Each queue is paired: *active* and *expired*
- Priorities 0-99 are reserved for *real-time process*
- The remaining priorities are for normal processes, whose priority may be adjusted by the user or system using the `nice()` system call

The Highest Priority Process

- The scheduler maintains a bit map indicating which queues have processes ready to run
- Find the first bit that's set
  - 140 queues implies 5 integers
  - Do compares to find the first value that is non-zero
  - Time depends on number of priority levels, not the number of processes!

Algorithm for Computing Timeslices

- The quantum is computing by looking at the static process priority
- If static priority SP is less 120, then the quantum is computed as $Q = (140 - SP) * 20$
- Otherwise it is computed as $Q = (140 - SP) * 5$
- Basic idea is that important processes should run longer so higher priority processes get longer quanta

Dynamic Priority

- Dynamic priority is calculated from the SP and a*average sleep time*
- Keep record of time a process was sleeping, up to some maximum time
- Decrement that value on each timer tick while process is running
- Result is a number between 0 and 10 that measures the percentage of time that a process was sleeping recently
  - Five is neutral, 10 means up priority by 5, 0 hurts priority by 5

Interactive Processes

- The Dynamic Priority bonus is used to determine if a process is interactive
- Low-priority processes have a hard time becoming interactive
- A default priority process will be flagged as being an interactive process when its sleep time is greater than 700ms

Using Quanta

- At each time tick, decrease the quantum of the current running process
- If that time goes to zero, process is done
- At that point, a non-interactive process goes into the expired queue
- An interactive process goes to the end of the current priority queue
- If nothing else is in that queue, the interactive process runs immediately
- But the more it runs, the DP bonus decreases and so does the priority and interactive status of the process

Avoiding Indefinite Overtaking

- Two sets of 140 queues: active and expired
- System only runs processes on the active queues
- When a priority level of the active queue is empty, the scheduler looks for the next-highest priority queue
- After running all of the active queues, the active and expired queues are swapped
- This avoids need for aging, which is an $O(n)$ task that must be run on each clock tick