Scheduling Part 1
Sample Sorts of Systems

- Simple batch
- Multiprogrammed batch
- Time sharing
- Partitioned servers
- Real time
Scheduling

- Aims
  - provide timely response
  - provide quick response
  - use resources equitably
Timely Response

• “Hard” real time
  – chores *must* be completed on time
    - controlling a nuclear power plant
    - landing (softly) on Mars
Fast Response

• “Soft” real time
  – the longer it takes, the less useful a chore’s result becomes
    - responding to user input
    - playing streaming audio or video
Sharing

• All active threads share processor time equally
Scenario

• Scheduling “jobs”
• Run one at a time
• Running time is known
FIFO
Throughput

- “Goodness” criterion is jobs/hour
- One 168-hour job
- Followed by 168 one-hour jobs
Average Wait Time

- Jobs $J_i$ with processing times $T_i$
- Average wait time ($AWT$)
  - $J_i$ started at time $t_i$
  - $AWT = \frac{\text{sum}(t_i + T_i)}{n}$
    - $t_i = \text{sum}_{j=0}^{i-1}(T_j)$
- For our example
  - $AWT = 252$ hours
Shortest Job First

- $AWT = \frac{\sum(t_i + T_i)}{n}$
  - $t_i = \sum_{j=0}^{i-1}(T_j)$

- $AWT = \frac{(nT_{i0} + (n-1)T_{i1} + \ldots + 2T_{in-2} + T_{in-1})}{n}$

- Minimized when $i_j$ chosen so that
  - $T_{ij} \leq T_{ij+1}$
  - which is shortest job first
SJF and Our Example

Throughput

Jobs/hour

AWT = 86 hours

Time (hours)
Preemption

• Current job may be preempted by others
  – shortest remaining time next (SRTN)
    - optimized throughput
Fairness

- FIFO
  - each job eventually gets processed
- SJF and SRTN
  - a long job might have to wait indefinitely
- What’s a good measure?
Round Robin
Round Robin + FIFO

• AWT?
  – let quantum approach 0:
    - 169 jobs sharing the processor
    - run at 1/169\textsuperscript{th} speed for first week
    - short jobs receive one hour of processor time in 169 hours
    - long job completes in 336 hours
    - AWT = 169.99 hours
    - average deviation = 1.96 hours
    - for FIFO, average deviation = 42.25 hours
Interactive Systems

- Length of “jobs” not known
- Jobs don’t run to completion
  - run till they block for user input
- Would like to favor interactive jobs
Round Robin with Priority
Multi-Level Feedback Queues
Interactive Scheduling

- Time-sliced, priority-based, preemptive
- Priority depends on expected time to block
  - interactive threads should have high priority
  - compute threads should have low priority
- Determine priority using past history
  - processor usage causes decrease
  - sleeping causes increase
Scheduling in Early Unix

- Interactive applications
  - shell, editors
- Lengthier applications
  - compiles
- Long-running applications
  - computing π
6th-Edition Unix Scheduling

• Process priority computation
  – \( p = \frac{(pp->p\_cpu \& 0377)/16}{16}; \)
  – \( p =+ PUSER + pp->p\_nice; \)
  – (numerically low priorities are better than numerically high priorities)

• Every “clock tick”
  – current process: \( p\_cpu++; \)

• Every second
  – all processes: \( p\_cpu = \text{max}(0, p\_cpu - 10) \)

• Every four seconds
  – force rescheduling
    - time quantum
Early BSD Unix Scheduler

• priority = $c_1 + \frac{\text{cpuAvg}}{4} + c_2 \cdot \text{nice}$
  – thread priority, computed periodically

• cpuAvg++
  – every .01 second, while thread is running

• cpuAvg = $\frac{2}{3} \cdot \text{cpuAvg}$
  – computed once/second for each thread

• time quantum is .1 second
Later BSD Unix Scheduler

• priority = \( c_1 + (\text{cpuAvg}/4) + c_2 \cdot \text{nice} \)
  – thread priority, computed periodically

• \text{cpuAvg}++
  – every .01 second, while thread is running

• \text{cpuAvg} = \frac{(2 \cdot \text{load})}{(2 \cdot \text{load} + 1)} \cdot \text{cpuAvg}
  – \text{load} is the short-term average of the sum of the run-queue size and the number of “short-term” sleeping threads
  – computed once/second for all threads

• time quantum still .1 second