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 $t_i$
  - $AWT = \sum(t_i + T_i)/n$
  - $t_i = \sum_{j=0}^{i-1}(T_j)$
- For our example
  - $AWT = 252$ hours
Shortest Job First

- $AWT = \frac{\text{sum}(t_i + T_i)}{n}$
  - $t_i = \text{sum}_{j=0}^{i-1}(T_j)$
- $AWT = \frac{(nT_{i_0} + (n-1)T_{i_1} + \ldots + 2T_{i_{n-2}} + T_{i_{n-1}})}{n}$
- Minimized when $i_j$ chosen so that
  - $T_{i_j} \leq T_{i_{j+1}}$
  - which is shortest job first
SJF and Our Example

Throughput

AWT = 86 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
Quiz 1

We implement a round-robin scheduler. Jobs are served from the queue in FIFO order with a fixed time quantum. After a job has executed for its quantum, it's preempted and goes to the end of the queue.

Does this scheduler improve the average wait time (compared to SJF) if applied to our example?

a) yes
b) no
Round Robin + FIFO

• AWT?
  – let quantum approach 0:
    - 169 jobs sharing the processor
    - run at $1/169^{th}$ 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 $\pi$
6th-Edition Unix Scheduling

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

• Every “clock tick”
  – current process: \( p_{cpu}++ \)

• Every second
  – all processes: \( p_{cpu} = \max(0, p_{cpu} - 10) \)

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

- priority = $c_1 + (\text{cpuAvg}/4) + c_2 \cdot \text{nice}$
  - thread priority, computed periodically
- cpuAvg++
  - every .01 second, while thread is running
- cpuAvg = $(2/3) \cdot \text{cpuAvg}$
  - computed once/second for each thread
- time quantum is .1 second
Quiz 2

The UNIX schedulers seen so far work on the following principle:

- priority steadily gets worse while a thread is running
- priority steadily gets better while a thread is not running

a) These schedulers work fine under heavy load

b) These schedulers don't work well under heavy load because they use the above principle

c) These schedulers don't work well under heavy load because they improperly implement the above principle
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} = \left( \frac{(2 \cdot \text{load})}{(2 \cdot \text{load}+1)} \right) \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