EECS 482
Introduction to Operating Systems

Winter 2019

Baris Kasikci
(Thanks, Harsha Madhyastha for the slides!)
CPU scheduling

- If >1 thread is ready, choose which to run
  - Example, FIFO scheduling in project 2

- Many possible scheduling policies
  - Goal today is to explore fundamental ones
  - Real schedulers often a complex mix of policies
What are good goals for a CPU scheduler?
  - Minimize average response time
  - Maximize throughput
  - Fairness (what does this mean, though)?

“Minimize latency” at odds with “maximize tput”
Maximize Performance

- Minimize average response time
  - Elapsed time to do each job
- Maximize throughput of entire system
  - Rate at which jobs complete in the system
- These are conflicting goals
  - Throughput-response curves common visualization
Throughput-response curves

- Collected from Facebook production service [Chow ‘16]
  - Each colored line: throughput vs. response at different quality
  - Left of graph – adding load has little effect on response time
  - Right of graph – adding load causes exponential increase in latency

March 13, 2019
Load testing (aside)
Fairness

- Share CPU among threads in equitable manner

- How to share between 1 big and 1 small job?
  - Response time proportional to job size?
  - Or equal time for each job?

- Fairness often conflicts with response time
Starvation = extremely unfair

- Starvation can be outcome of synchronization
  - Example: Readers can starve writers

- Starvation can also be outcome of scheduling
  - Example: always run highest-priority thread
  - If many high priority threads, low priority starves
First-come, first-served (FCFS)

- FIFO ordering among jobs

- No preemption (no timer interrupts)
  - Thread runs until it calls yield() or blocks
FCFS Example

- Job A: Arrives at t=0, takes 100 seconds
- Job B: Arrives at t=0+, takes 1 second

- A’s response time = 100
- B’s response time = 101
- Average response time = 100.5
**FCFS Summary**

**Pros:**
- Simple to implement

**Cons:**
- Short jobs can be stuck behind long ones
- Bad for interactive workloads
Round Robin

- Improve average response time for short jobs
- Still FIFO ordering
- Add preemptions (via timer interrupts)
  - Fixed time slice (time quantum)
  - Preempt if still running when time slice is over
Round Robin Example

- Job A: Arrives at t=0, takes 100 seconds
- Job B: Arrives at t=0+, takes 1 second

A’s response time = 101
B’s response time = 2
Average response time = 51.5

March 13, 2019
Choosing a time slice

- What’s the problem with a big time slice?
  - Degenerates to FCFS (poor interactivity)

- What’s the problem with a small time slice?
  - More context switching overhead (low throughput)

- OS typically compromises: e.g., 1ms or 10ms
Round Robin Summary

● Pros:
  • Still pretty simple
  • Good for interactive computing

● Cons?
  • More context-switching overhead

● Comparison: Does RR always reduce response time vs. FCFS?
Round Robin vs. FCFS

- Jobs A and B arrive at t=0, both take 100 secs

\[ \begin{array}{c|c}
A & B \\
\hline
\end{array} \]

\[ \begin{array}{cccc}
& & & \\
\hline
\hline
\hline
\end{array} \]

Which is more fair? RR or FCFS?

- Average response time with FCFS = 150
- Average response time with RR = 199.5

March 13, 2019
STCF

- Shortest time to completion first

- Run job with least work to do
  - Preempt current job if shorter job arrives
  - If will block, work to do is time to next blocking operation

- Finish short jobs first
  - Improves response time of short jobs (by a lot)
  - Hurts response time of long jobs (by a little)

- STCF gives optimal average response time
Analysis of STCF

- Consider 2 jobs: A longer than B
- Average response time \((2A+B)/2\) vs. \((A+2B)/2\)
- \(B < A\), so 2nd has smaller avg. response time
- Apply iteratively (e.g., bubble sort) to minimize
Round Robin Example

- Job A: Arrives at $t=0$, takes 100 seconds
- Job B: Arrives at $t=0^+$, takes 1 second

![](chart)

- A’s response time = 101
- B’s response time = 1
- Average response time = 51

March 13, 2019
STCF

- **Pro:**
  - Optimal average response time

- **Cons?**
  - Potential starvation for long jobs (really unfair!)
  - Needs knowledge of future

- **How to estimate the time a job will run for?**
Predicting job run times

- Ask the job or the user?
  - Strong incentive to lie ("will just take a minute")

- Use past to predict future

- Can assume heavy-tailed distribution
  - If already run for n seconds, likely to run for n more

- OS schedulers often identify interactive apps and boost their priority.
Priority

• Priority
  • Assign external priority to each job
  • Run high-priority jobs before low-priority ones
  • Use, e.g., round-robin for jobs of equal priority
  • Prone to starvation

• Methods for preventing starvation?
  • If job has not run for time t, boost priority
  • Handle priority inversion (lock held by low-priority)
Multimedia: Soft real-time

- Often requires fixed amount of CPU
  - Audio should not skip when compiling projects
  - Preditable: video player plays n frames per second

- Can reserve a share of the CPU
  - X% of the CPU over some time interval
  - Unused CPU split among remaining jobs
Hard real-time scheduling

- Jobs have to complete before deadline
  - Demand / deadline known in advance
  - Example: vehicle control, aviation, etc.

- Earliest-deadline first (EDF)
  - Always run jobs whose deadline is soonest
  - Preempt if newly arriving job has earlier deadline
  - Always succeeds if schedule is feasible
  - But, may be very poor if schedule is infeasible

March 13, 2019
Scheduling: Summary

- Many different policies
  - FCFS
  - Round robin
  - STCF
  - Priority
  - Proportional share
  - EDF

- OS schedulers mix all of these
  - Many heuristics and complex tuning
  - No perfect solution