CS162
Operating Systems and Systems Programming
Lecture 8

CPU Scheduling, Protection Address Spaces

February 14, 2011

Ion Stoica

http://inst.eecs.berkeley.edu/~cs162

Review: Last Time

- Scheduling: selecting a waiting process from the ready queue and allocating the CPU to it
- FCFS Scheduling:
  - Run threads to completion in order of submission
  - Pros: Simple (+)
  - Cons: Short jobs get stuck behind long ones (-)
- Round-Robin Scheduling:
  - Give each thread a small amount of CPU time when it executes; cycle between all ready threads
  - Pros: Better for short jobs (+)
  - Cons: Poor when jobs are same length (-)

Goals for Today

- Finish discussion of Scheduling
- Kernel vs User Mode
- What is an Address Space?
- How is it Implemented?

Note: Some slides and/or pictures in the following are adapted from slides ©2005 Silberschatz, Galvin, and Gagne

Round-Robin Discussion

- How do you choose time slice?
  - What if too big?
    - Response time suffers
  - What if infinite (∞)?
    - Get back FIFO
  - What if time slice too small?
    - Throughput suffers!
- Actual choices of timeslice:
  - Initially, UNIX timeslice one second:
    - Worked ok when UNIX was used by one or two people.
    - What if three compilations going on? 3 seconds to echo each keystroke!
  - In practice, need to balance short job performance and long job throughput.
    - Typical time slice today is between 10ms – 100ms
    - Typical context-switching overhead is 0.1ms – 1ms
    - Roughly 1% overhead due to context-switching

Comparisons between FCFS and Round Robin

- Assuming zero cost context switching time, is RR always better than FCFS?
- Simple example: 10 jobs, each takes 100s of CPU time
  RR scheduler quantum of 1s
  All jobs start at the same time
- Completion Times:

Job #	FIFO	RR
1	100	991
2	200	992
…	…	…
9	900	999
10	1000	1000

Both RR and FCFS finish at the same time

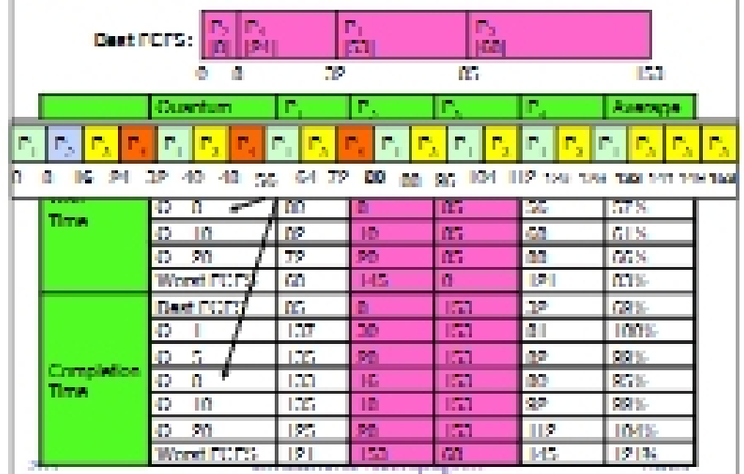
Average response time is much worse under RR!

- Bad when all jobs same length

- Also: Cache state must be shared between all jobs with RR but can be devoted to each job with FCFS

Total time for RR longer even for zero cost switch!

Earlier Example with Different Time Quantum



What if we Knew the Future?

- Could we always mirror best FCFS?
- Shortest Job First (SJF):
  - Run whatever job has the least amount of computation to do
- Shortest Remaining Time First (SRTF):
  - Preemptive version of SJF: if job arrives and has a shorter time to completion than the remaining time on the current job, immediately preempt CPU
- These can be applied either to a whole program or the current CPU burst of each program
  - Idea is to get short jobs out of the system
  - Big effect on short jobs, only small effect on long ones
  - Result is better average response time

Discussion

- SJF/SRTF are the best you can do at minimizing average response time

  Provably optimal (SJF among non preemptive, SRTF among preemptive)

  Since SRTF is always at least as good as SJF, focus on SRTF

- Comparison of SRTF with FCFS and RR

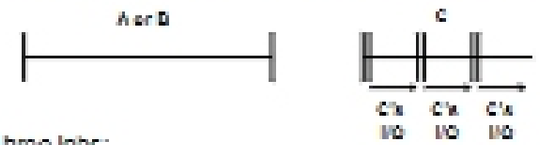
  What if all jobs the same length?

  - SRTF becomes the same as FCFS (i.e. FCFS is best can do if all jobs the same length)

  What if jobs have varying length?

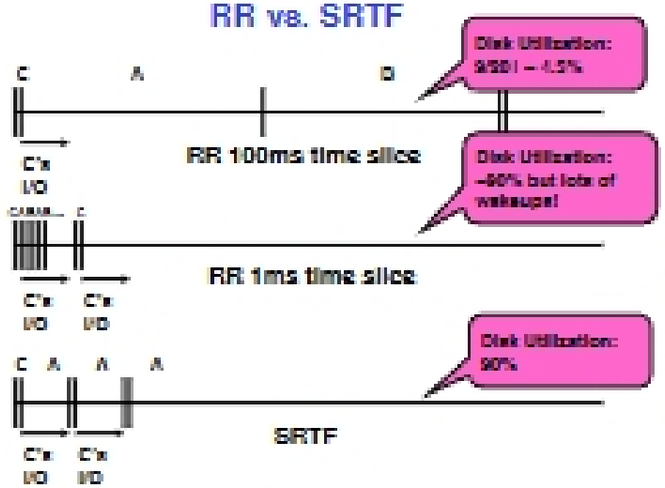
  - SRTF (and RR): short jobs not stuck behind long ones

Example to illustrate benefits of SRTF



- Three jobs:
  - A,B: CPU bound, each run for a week
    C: I/O bound, loop 1ms CPU, 9ms disk I/O
  - If only one at a time, C uses 90% of the disk, A or B could use 100% of the CPU
- With HHO:
  - Once A or B get in, keep CPU for one week each
- What about RR or SHIP?
  - Easier to see with a timeline

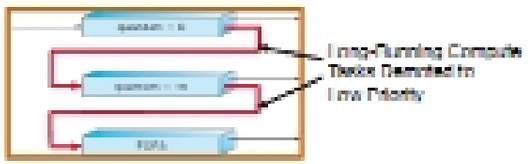
RR vs. SRTF



SRTF Further discussion

- Starvation
  - SRTF can lead to starvation if many small jobs!
  - Large jobs never get to run
- Somehow need to predict future
  - How can we do this?
  - Some systems ask the user
    - When you submit a job, have to say how long it will take
    - To stop cheating, system kills job if takes too long
  - But: even non-malicious users have trouble predicting runtime of their jobs
- Bottom line, can't really know how long job will take
  - However, can use SRTF as a yardstick for measuring other policies
  - Optimal, so can't do any better
- SRTF Pros & Cons
  - Optimal (average response time) (+)
  - Hard to predict future (-)
  - Unfair (-)

Multi-Level Feedback Scheduling



- Multiple queues, each with different priority
  Higher priority queues often considered "foreground" tasks
- Each queue has its own scheduling algorithm
  e.g. foreground RR, background FCFS
- Adjust each job's priority as follows (details vary)
  Job starts in highest priority queue
  If it doesn't finish in its time quantum, drop one level
  If it finishes, push up one level (or to top)