

Shortest-Remaining-Time (SRT)

- SRT is a preemptive version of SJF
- Policy:
 - Choose the process that has the smallest next CPU burst, and run that process preemptively...
 - (until termination or blocking, or
 - until a process enters the ready queue (either a new process or a previously blocked process))
 - At that point, choose another process to run if one has a smaller expected CPU burst than what is left of the current process' CPU burst

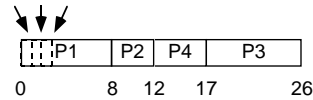
1

Fall 2001, Lecture 19

SJF & SRT Example

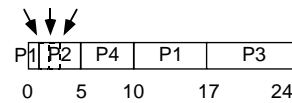
- SJF Example:

Process (Arrival Order)	P1	P2	P3	P4
Burst Time	8	4	9	5
Arrival Time	0	1	2	3



$$\text{average waiting time} = (0 + (8-1) + (12-3) + (17-2)) / 4 = 7.75$$

- Same Example, SRT Schedule:



$$\text{average waiting time} = ((0+(10-1) + (1-1) + (17-2) + (5-3)) / 4 = 6.5$$

2

Fall 2001, Lecture 19

SRT Evaluation

- Preemptive (at arrival of process into ready queue)
- Response time — good
 - Provably *optimal* — minimizes average waiting time for a given set of processes
- Throughput — high
- Fairness — penalizes long processes
 - Note that long processes eventually become short processes
- Starvation — possible for long processes
- Overhead — can be high (recording and estimating CPU burst times)

3

Fall 2001, Lecture 19

Priority Scheduling

- Policy:
 - Associate a priority with each process
 - Externally defined, based on importance, money, politics, etc.
 - Internally defined, based on memory requirements, file requirements, CPU requirements vs. I/O requirements, etc.
 - SJF is priority scheduling, where priority is inversely proportional to length of next CPU burst
 - Choose the process that has the highest priority, and run that process either:
 - preemptively, or
 - non-preemptively
- Evaluation
 - Starvation — possible for low-priority processes
 - Can avoid by *aging* processes: increase priority as they spend time in the system

4

Fall 2001, Lecture 19

Multilevel Queue Scheduling

■ Policy:

- Use several ready queues, and associate a different priority with each queue
- Choose the process from the occupied queue that has the highest priority, and run that process either:
 - preemptively, or
 - non-preemptively
- Assign new processes permanently to a particular queue
 - Foreground, background
 - System, interactive, editing, computing
- Each queue can have a different scheduling policy
 - Example: preemptive, using timer
 - 80% of CPU time to foreground, using RR
 - 20% of CPU time to background, using FCFS

5

Fall 2001, Lecture 19

Multilevel Feedback Queue Scheduling

■ Policy:

- Use several ready queues, and associate a different priority with each queue
- Choose the process from the occupied queue with the highest priority, and run that process either:
 - preemptively, or
 - non-preemptively
- Each queue can have a different scheduling policy
- Allow scheduler to move processes between queues
 - Start each process in a high-priority queue; as it finishes each CPU burst, move it to a lower-priority queue
 - Aging — move older processes to higher-priority queues
 - Feedback = use the past to predict the future — favor jobs that haven't used the CPU much in the past — close to SRT!

6

Fall 2001, Lecture 19

CPU Scheduling in UNIX using Multilevel Feedback Queue Scheduling

■ Policy:

- Multiple queues, each with a priority value (low value = high priority):
 - Kernel processes have negative values
 - Includes processes performing system calls, that just finished their I/O and haven't yet returned to user mode
 - User processes (doing computation) have positive values
- Choose the process from the occupied queue with the highest priority, and run that process preemptively, using a timer (time slice typically around 100ms)
 - Round-robin scheduling in each queue
- Move processes between queues
 - Keep track of clock ticks (60/second)
 - Once per second, add clock ticks to priority value
 - Also change priority based on whether or not process has used more than its "fair share" of CPU time (compared to others)

7

Fall 2001, Lecture 19