STEVENS INSTITUTE of TECHNOLOGY
THE INNOVATION UNIVERSITY®

CS 492: Operating Systems

Scheduling (2)

Instructor: Iraklis Tsekourakis

Email: itsekour@stevens.edu

Round-Robin Scheduling

- Round-robin
  - Each process is allowed to run specified time quantum.
  - After this time has elapsed, the process is preempted and added to the end of the ready queue.

- Quantum – a time interval, e.g., 100 msec

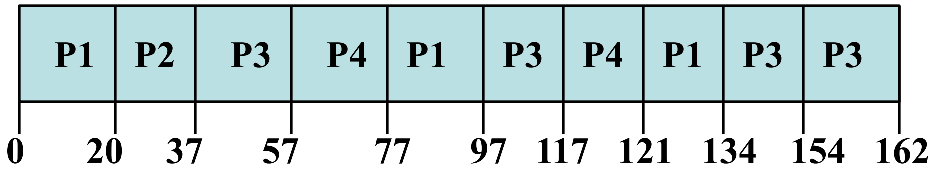
- Advantages:
  - Solution to fairness and starvation
  - Fair allocation of CPU across jobs
  - Low average waiting time when job lengths vary
  - Good for responsiveness if small number of jobs

Round Robin Example

- Time Quantum = 20

Process	CPUTime			
P1	53	**33 left**	**13 left**	
P2	17			
P3	68	**48 left**	**28 left**	**8 left**
P4	24	**4 left**		

Gantt Chart for Schedule



Typically, higher average turnaround time than SJF, but better response time

Question?

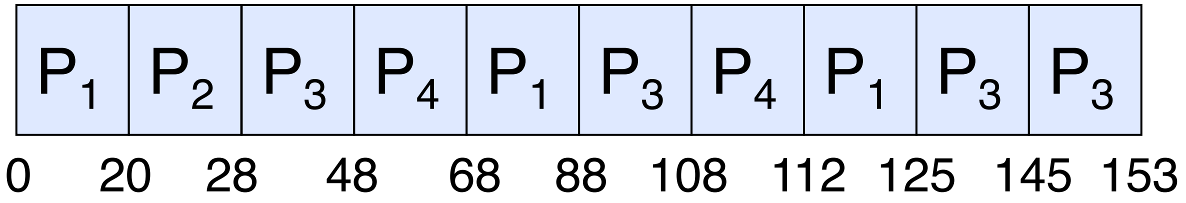
- Explain how time quantum value and context switching time affect each other, in a round-robin scheduling algorithm.

RR: Example

- Example:

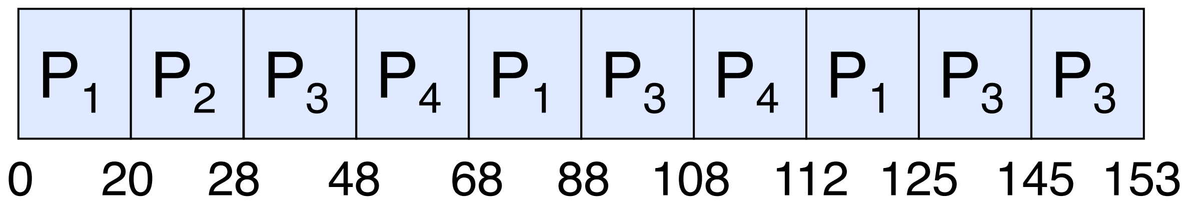
Process	Burst Time
P_1	53
P_2	8
P_3	68
P_4	24

  – The Gantt chart is:



What's the average waiting time?
What's the average turnaround time?

RR: Example (sol)



- Waiting time for P_1=(68-20)+(112-88)=72
  - P_2=(20-0)=20
  - P_3=(28-0)+(88-48)+(125-108)=85
  - P_4=(48-0)+(108-68)=88
- Average waiting time = (72+20+85+88)/4=66¼
- Average turnaround time = (125+28+153+112)/4 = 104½

- Thus, Round-Robin Pros and Cons:
  - Better for short jobs, Fair (+)
  - Context-switching time adds up for long jobs (-)

FCFS Vs RR

- Assuming zero-cost context-switching time, is RR always better than FCFS?
- Simple example:

  10 jobs, each take 100s of CPU time
  RR scheduler quantum of 1s
  All jobs start at the same time

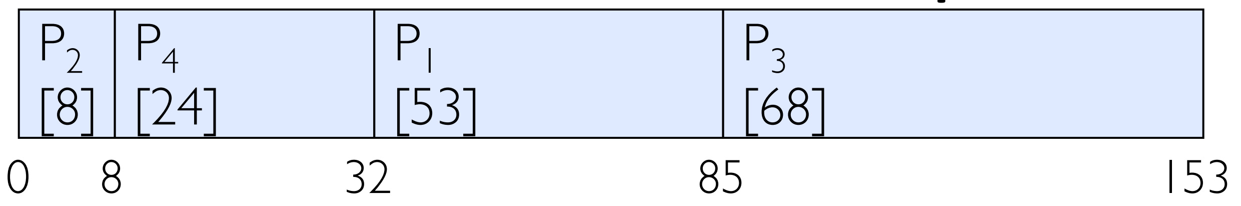
- Turnaround Times:

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 turnaround 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 FIFO
  - Total time for RR longer even for zero-cost switch!

Earliest ex. With different time quantum

Best FCFS:



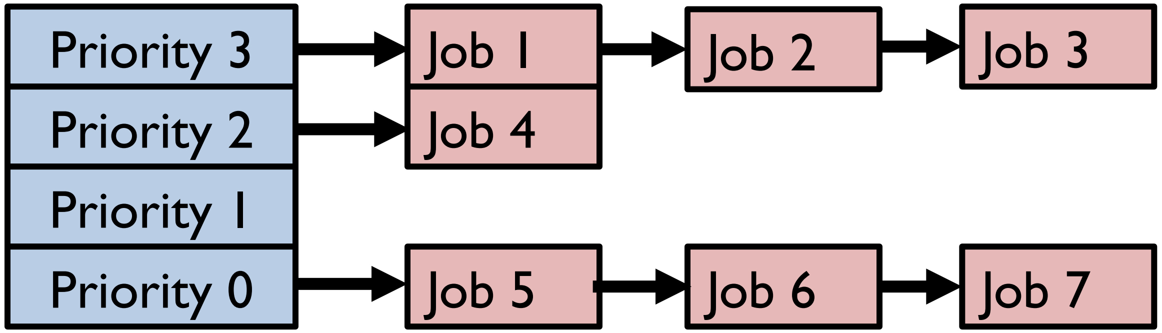
	Quantum	P_1	P_2	P_3	P_4	Average
Wait Time	Best FCFS	32	0	85	8	31¼
	Q = 1	84	22	85	57	62
	Q = 5	82	20	85	58	61¼
	Q = 8	80	8	85	56	57¼
	Q = 10	82	10	85	68	61¼
	Q = 20	72	20	85	88	66¼
	Worst FCFS	68	145	0	121	83½
Turnaround Time	Best FCFS	85	8	153	32	69½
	Q = 1	137	30	153	81	100½
	Q = 5	135	28	153	82	99½
	Q = 8	133	16	153	80	95½
	Q = 10	135	18	153	92	99½
	Q = 20	125	28	153	112	104½
	Worst FCFS	121	153	68	145	121¾

RR discussion

- How do you choose time slice?
  1. What if too big?
  2. What if infinite (*∞)?*
  3. What if time slice too small?

- 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 recently is between 10ms – 100ms
    - Typical context-switching overhead is 0.1ms – 1ms
    - Roughly 1% overhead due to context-switching

Priority Scheduling



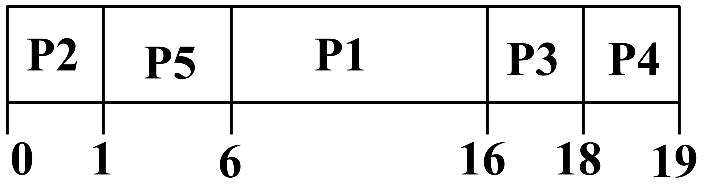
- Execution Plan
  - Always execute highest-priority runnable jobs to completion
  - Each queue can be processed in Round-Robin fashion with some time-quantum
- Problems:
  - Starvation:
    - Lower priority jobs don’t get to run because higher priority tasks always running
  - Deadlock: Priority Inversion
    - Not strictly a problem with priority scheduling, but happens when low priority task has lock needed by high-priority task (busy waiting)
- **How to fix problems?**

Priority Scheduling

- Example

Process	CPUTime	Priority
P1	10	3
P2	1	1
P3	2	4
P4	1	5
P5	5	2

Gantt Chart for Schedule



- Turnaround time
  - P1 = 16;
  - P2 = 1;
  - P3 = 18;
  - P4 = 19;
  - P5 =6
- Average turnaround time (16+1+18+19+6)/5 = 12

How to Assign Priorities?

- Statically based on process type
- Priority = 1/f
  - f = size of quantum used last
  - The longer a process ran, the lower its priority
  - The process that runs the shortest gets highest priority to run next

Lottery Scheduling

- Yet another alternative: Lottery Scheduling
  - Give each job some number of lottery tickets
  - On each time slice, randomly pick a winning ticket
  - On average, CPU time is proportional to number of tickets given to each job
- How to assign tickets?
  - To approximate SRTF, short running jobs get more, long running jobs get fewer
  - To avoid starvation, every job gets at least one ticket (everyone makes progress)
- Advantage over strict priority scheduling: behaves gracefully as load changes
  - Adding or deleting a job affects all jobs proportionally, independent of how many tickets each job possesses
- Can be used to solve problems that are difficult to handle otherwise

How to evaluate a scheduling algorithm?

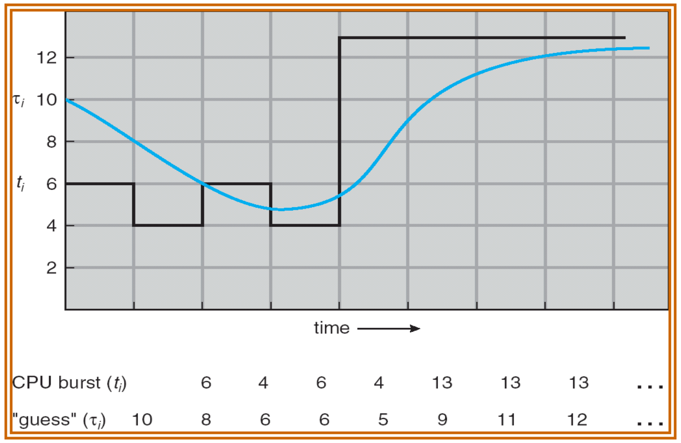
- Deterministic modeling
  - takes a predetermined workload and compute the performance of each algorithm for that workload
- Queueing models
  - Mathematical approach for handling stochastic workloads
- Implementation/Simulation:
  - Build system which allows actual algorithms to be run against actual data – most flexible/general

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
  - Sometimes called "Shortest Time to Completion First" (STCF)
- 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
  - Sometimes called "Shortest Remaining Time to Completion First" (SRTCF)
- 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

Predicting the Length of the Next CPU Burst

- Adaptive: Changing policy based on past behavior
  - CPU scheduling, in virtual memory, in file systems, etc
  - Works because programs have predictable behavior
    - If program was I/O bound in past, likely in future
    - If computer behavior were random, wouldn't help
- Example: SRTF with estimated burst length
  - Use an estimator function on previous bursts:
    Let t_{n-1}, t_{n-2}, t_{n-3}, etc. be previous CPU burst lengths. Estimate next burst $\tau_n = f(t_{n-1}, t_{n-2}, t_{n-3}, \ldots)$
  - Function f could be one of many different time series estimation schemes (Kalman filters, etc)



CPU burst (t_i)		6	4	6	4	13	13	13	. . .
"guess" (τ_i)	10	8	6	6	5	9	11	12	. . .

Problem

Five batch jobs. A through E , arrive at a computer center at almost the same time. They have estimated running times of 10, 6, 2, 4, and 8 minutes. Their (externally determined) priorities are 3, 5, 2, 1, and 4, respectively, with 5 being the highest priority. For each of the following scheduling algorithms, determine the mean process turnaround time. Ignore process switching overhead.

(a) Round robin.

(b) Priority scheduling.

(c) First-come, first-served (run in order 10, 6, 2, 4, 8).

(d) Shortest job first.

For (a), assume that the system is multi-programmed, and that each job gets its fair share of the CPU. For (b) through (d), assume that only one job at a time runs, until it finishes. All jobs are completely CPU bound.

Solution

For round robin, during the first 10 minutes each job gets 1/5 of the CPU. At the end of 10 minutes, C finishes. During the next 8 minutes, each job gets ¼ of the CPU, after which time D finishes. Then each of the three remaining jobs gets 1/3 of the CPU for 6 minutes, until B finishes, and so on. The finishing times for the five jobs are 10, 18, 24, 28, and 30, for an average of 22 minutes.

For priority scheduling, B is run first. After 6 minutes it is finished. The other jobs finish at 14, 24, 26, and 30, for an average of 20 minutes.

If the jobs run in the order A through E, they finish at 10, 16, 18, 22, and 30, for an average of 19.2 minutes.

Finally, shortest job first yields finishing times of 2, 6, 12, 20, and 30, for an average of 14 minutes.