Processor Scheduling

What Are We Covering?

- Processor Scheduler
  - Short Term - Schedule ready processes to processor(s)
  - Long Term - Schedule processes to reside in memory
- How to design a good processor scheduling algorithm
- Performance Metrics

Quality of Scheduling Algorithms

- Performance Metrics
  - Turnaround Time: average (waiting time + service time)
  - Response Time
    - Similar to turnaround time
    - Used in interactive environment
  - Throughput: Number of jobs completed per unit time
  - Utilization: the percentage of time the processor is busy
  - Turnaround time / Service time
    - Best matches with users' expectation
    - A longer job can have a longer wait time
  - Fairness - no starvation

Constraints on Scheduling

- Nonpreemptive
  - Process will run till terminated or blocked
- Preemptive
  - Process can be interrupted while in execution
  - E.g., preempted by time quantum
- Real-Time Scheduling
  - Each process has to finish before its deadline

Job Characteristics

- CPU bound
  - Takes long computation time
  - Little I/O
- I/O bound
  - Repeat: run for short time; need I/O

Scheduling Algorithms

- First-In-First-Out (FIFO, FCFS)
- Shortest Job First
- Highest Response Ratio First
- Shortest Remaining Time First
- Round-Robin
- Multilevel Feedback Queue

Scheduling Algorithms

- First-in-first-out
  - Easy to implement
  - Short jobs may be stuck behind a long job
- Shortest Job First
  - Non-preemptive
  - Minimize average turnaround time **if** all jobs arrive at the same time
    - A short job still can get stuck behind a long job
  - Standard deviation of turnaround time may not be good
  - May cause starvation
  - Need to predict execution time of each job

Scheduling Algorithms

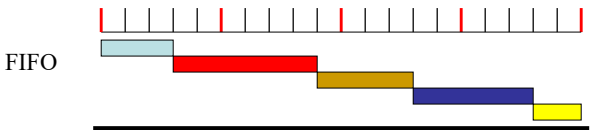
- Shortest Remaining Time First
  - Preemptive
  - Allow a job with shorter remaining time to preempt current running job
  - Minimize average turnaround time
  - Similar problems with SJF + jobs need to be preemptable

Scheduling Algorithms

- Highest Response Ratio Next
  - The ready process with the greatest response ratio (RR) is selected to run
  - RR = (waiting-time + service-time) / service-time
  - Shorter jobs are favored
  - Take waiting time in to account
    - Avoid starvation problem
    - More fair
  - Need to predict execution time of each job
  - Can be non-preemptive or preemptive

Example (FIFO)

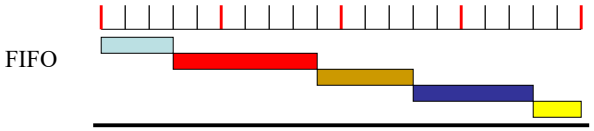
Job #	1	2	3	4	5
Arrival time	0	2	4	6	8
Service time	3	6	4	5	2



Example (FIFO)

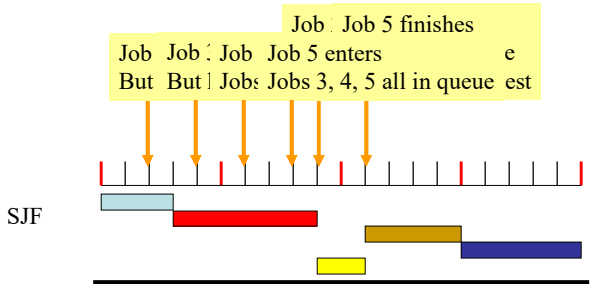
Job #	1	2	3	4	5
Arrival time	0	2	4	6	8
Finish time	3	9	13	18	20
Response time	3	7	9	12	12
Service time	3	6	4	5	2
Wait time	0	1	5	7	10

Average response time = (3+7+9+12+12)/5 = 8.6
Throughput = 5/20 = 1/4
Ave (RespTime/ServTime) = (1+7/6+9/4+12/5+12/2)/5 = 2.564



Example (Shortest Job First)

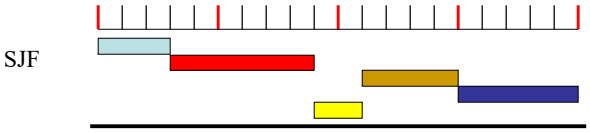
Job #	1	2	3	4	5
Arrival time	0	2	4	6	8
Service time	3	6	4	5	2



Example (Shortest Job First)

Job #	1	2	3	4	5
Arrival time	0	2	4	6	8
Finish time	3	9	15	20	11
Response time	**3**	**7**	**11**	**14**	**3**
Service time	3	6	4	5	2
Wait time	**0**	**1**	**7**	**9**	**1**

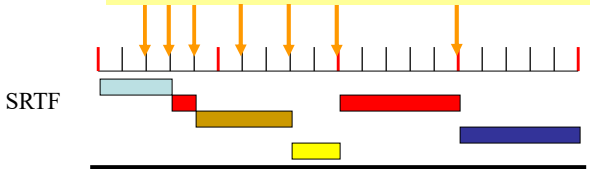
Average response time = (3+7+11+14+3)/5 = 7.6
Throughput = 5/20 = 1/4
Ave (RespTime/ServTime) = (1+7/6+11/4+14/5+3/2)/5 = 1.844



Example (Shortest Remaining Time)

Job #	1	2	3	4	5
Arrival time	0	2	4	6	8
Service time	3	6	4	5	2

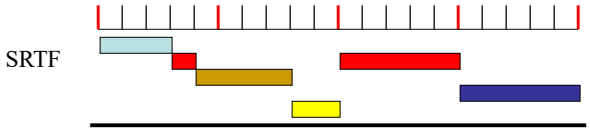
Job 5 enters
Job 3 finishes
Job 5 finishes
Remaining time of Job 2 = 5
Remaining t Job 3 finishes
Choose Job Switch to Job 4



Example (Shortest Remaining Time)

Job #	1	2	3	4	5
Arrival time	0	2	4	6	8
Finish time	3	15	8	20	10
Response time	**3**	**13**	**4**	**14**	**2**
Service time	3	6	4	5	2
Wait time	**0**	**7**	**0**	**9**	**0**

Average response time = (3+13+4+14+2)/5 = 7.2
Throughput = 5/20 = 1/4
Ave (RespTime/ServTime) = (1+13/6+1+14/5+1)/5 = 1.6

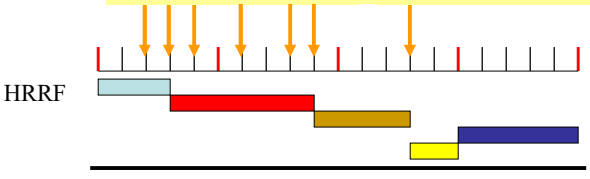


Example (Highest Response Ratio)

RR = (waiting-time + service-time) / service-time

Job #	1	2	3	4	5
Arrival time	0	2	4	6	8
Service time	3	6	4	5	2

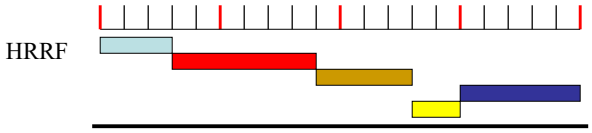
Job 2 finishes
RR of Jo Job 3 finishes
Job 1 Job Jo RR of Jo RR of Job 4 = ((13–6)+5)/5 = 2.4
Jo J Job Jobs Jo RR of Jo RR of Job 5 = ((13–8)+2)/2 = 3.5
Jo S Has Has H Choose J Choose Job 5



Example (Highest Response Ratio)

Job #	1	2	3	4	5
Arrival time	0	2	4	6	8
Finish time	3	9	13	20	15
Response time	**3**	**7**	**9**	**14**	**7**
Service time	3	6	4	5	2
Wait time	**0**	**0**	**7**	**9**	**0**

Average response time = (3+7+9+14+7)/5 = 8
Throughput = 5/20 = 1/4
Ave (RespTime/ServTime) = (1+7/6+9/4+14/5+7/2)/5 = 2.144

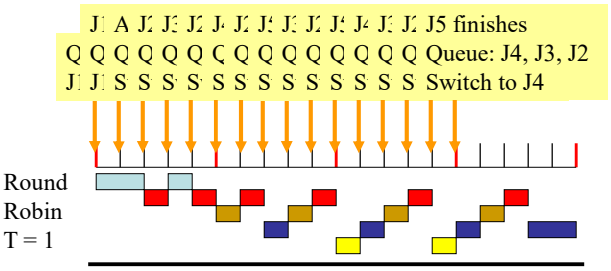


Scheduling Algorithms

- Round-Robin
  - Process take turns
  - Each executes till its time-quantum expires
  - Gives relatively reasonable turnaround time
    - Better in standard deviation of turnaround time
  - Useful when it is hard to estimate the execution time
    - No need to know execution time in advance
  - Overhead on process switching
    - More process switch ⇒ more context switch
  - I/O bound jobs may not use up their time quantum
    - Not fair to I/O bound jobs

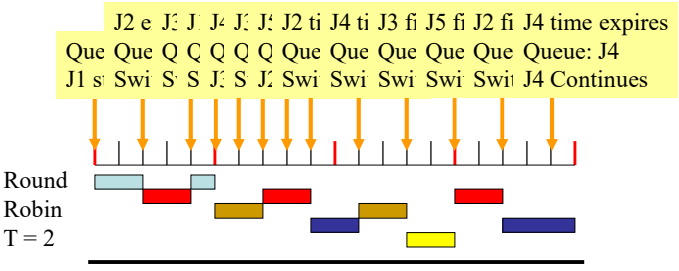
Example (Round Robin, T=1)

Job #	1	2	3	4	5
Arrival time	0	2	4	6	8
Service time	3	6	4	5	2



Example (Round Robin, T=2)

Job #	1	2	3	4	5
Arrival time	0	2	4	6	8
Service time	3	6	4	5	2

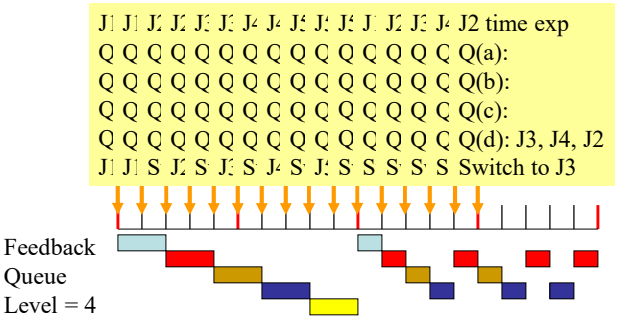


Scheduling Algorithms

- Multilevel Feedback Queue (MLFQ)
  - The system maintains multiple ready queues
    - For example, N priority levels
  - CPU always serves the jobs in the highest priority queue
  - A new process will be assigned a high priority
  - If a process used up its time quantum
    - priority decremented by one (placed in a lower level queue)
  - Favor I/O bound jobs
    - Not use up time quantum → stay at the same priority level
  - Good for time sharing systems
- There are many variations of the algorithm

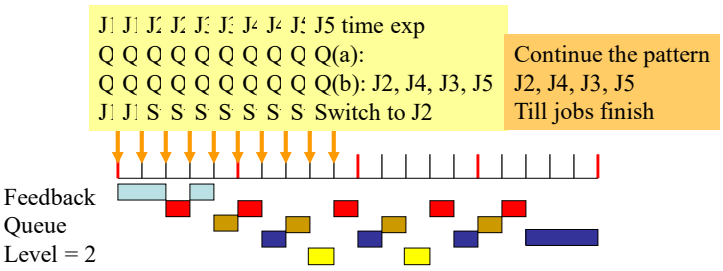
Example (Multi-level Queue, T=1, Q=4)

Job #	1	2	3	4	5
Arrival time	0	2	4	6	8
Service time	3	6	4	5	2



Example (Multi-level Queue, T=1, Q=2)

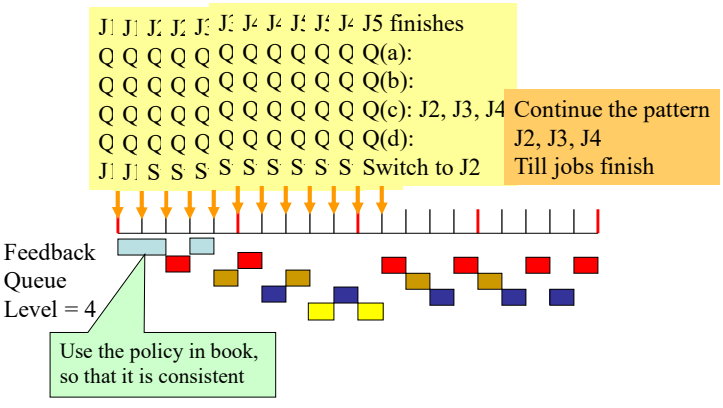
Job #	1	2	3	4	5
Arrival time	0	2	4	6	8
Service time	3	6	4	5	2



Example (Multi-level Q, T=1, Q=4)

Job #	1	2	3	4	5
Arrival time	0	2	4	6	8
Service time	3	6	4	5	2

From book
Empty queue
⇒ not to
drop priority

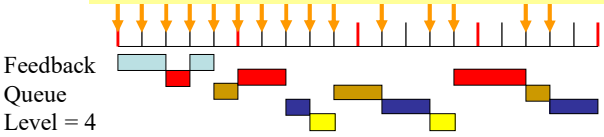


Example (Multi-level Queue, T=2^q, Q=4)

Job #	1	2	3	4	5
Arrival time	0	2	4	6	8
Service time	3	6	4	5	2

From book
Empty queue
⇒ not to
drop priority

J1 J1 J2 J2 J3 J3 J4 J2 J3 J5 ti J3 ti J4 J5 fini J2 J3 finishes
Q Q Q Q Q Q Q Q Q(a Q(a Q Q(a): Q Q(a):
Q Q Q Q Q Q Q Q Q(b Q(b Q Q(b): Q Q(b):
Q Q Q Q Q Q Q Q Q(c Q(c Q Q(c): J Q Q(c): J4
Q Q Q Q Q Q Q Q Q(d Q(d Q Q(d): Q Q(d):
J1 J1 S S S S S J2 S S Swi Swi S Switch S Switch to J4



Feedback
Queue
Level = 4

Queue level q: has time quantum 2^q
- Jobs in lower level queue need more CPU time
- Increasing time quantum reduces context switch

MLFQ-like Scheduling

- MLFQ
  - Mostly used algorithm in practice
  - In practice, many variations are used
    - Some consider to move up the task to the next level queue after a certain waiting time w
    - A version of Linux considers $q \times T$ as the time quantum for level q queue (T = 10ms)
    - Unix: MLFQ-like, compute priority as many levels of queues
  - Time quantum
    - Old Unix has 1 second time quantum, then changed to 1/10 second, then changed to 10ms

MLFQ-like Scheduling

- MLFQ-like scheduling algorithms
  - Similar to multilevel feedback queue
  - Use a number to indicate job priority
  - Priority level: 0-127
  - Negative priority value: no interruption
- A process
  - When enters the system, gets a base priority
  - Priority changes depending on CPU usage
    - Can also consider waiting time
  - After time quantum expiration or blocked by IO
    ⇒ Process switch ⇒ Recompute priorities of jobs

MLFQ-like Scheduling

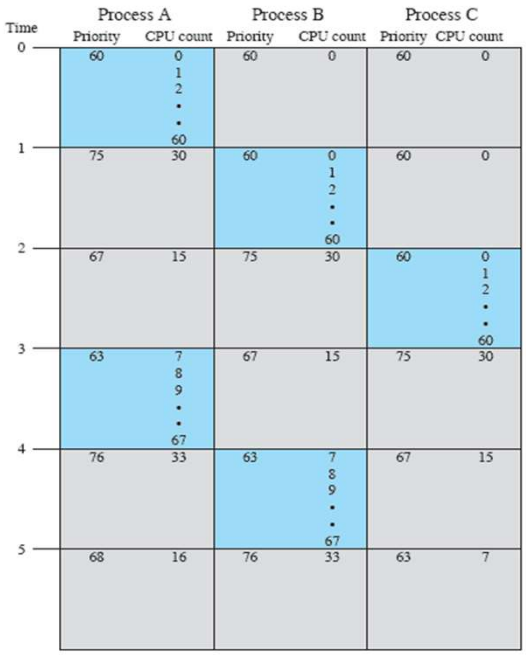
- Priority computation

$$P_{i+1} = base + C_i/2 + nice$$

$$C_i = (U_i + C_{i-1})/2$$

  - C_i : a count to keep track of the CPU usage history
  - U_i : CPU time units used in a time quantum i
    - A time quantum is divided into 60 units
- The priority of a job rises when waiting
  - U_i will be 0 when waiting
  - I/O-bound jobs tend to have higher priorities than CPU-bound jobs

MLFQ-like Scheduling Example



Time	Process A Priority	Process A CPU count	Process B Priority	Process B CPU count	Process C Priority	Process C CPU count
0	60	0 1 2 • • 60	60	0	60	0
1	75	30	60	0 1 2 • • 60	60	0
2	67	15	75	30	60	0 1 2 • • 60
3	63	7 8 9 • • 67	67	15	75	30
4	76	33	63	7 8 9 • • 67	67	15
5	68	16	76	33	63	7

Real Time Scheduling

- Real time systems
  - Examples of computing that requires real time
    - Monitor refreshing - every 16 milliseconds (60Hz) : Periodic
    - Traffic light control : Periodical / Aperiodic
    - Image processing in autonomous driving : Aperiodic
  - Scheduling in RT systems
    - Rate for periodical tasks (how frequent it needs to be activated)
    - Starting time and deadline
    - Preemptive or nonpreemptive
    - Soft or hard deadline

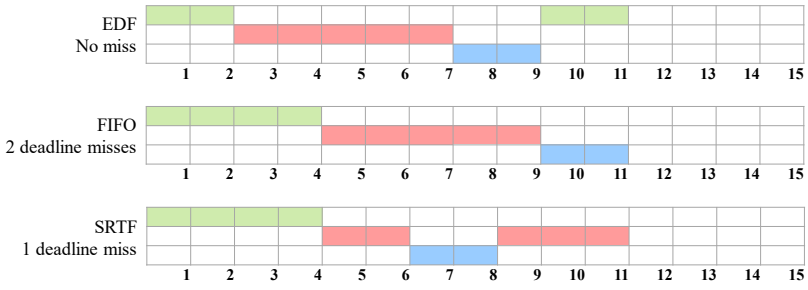
Real Time Scheduling

- Fixed Priority
  - Each task has a predetermined priority
  - Always execute the highest priority task first
  - Preemptive
- Earliest Deadline First
  - The task with the nearest deadline has the highest priority
  - Preemptive
- Rate Monotonic Scheduling
  - Consider periodical tasks
  - Tasks with the highest rates are scheduled first
  - Preemptive

Real Time Scheduling

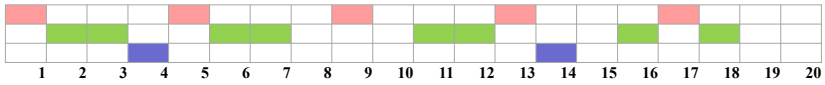
- Earliest Deadline First

Job #	1	2	3
Arrival time	0	2	6
Service time	4	5	2
Deadline	15	8	10



Real Time Scheduling

- Rate monotonic scheduling
  - RM schedulability
    - A system is schedulable if $\sum_i u_i \leq N(2^{1/N} - 1)$
    - $u_i = s_i/p_i$, p_i is the period and s_i is service time
    - N is the number of jobs
    - Example: Jobs 1, 2, 3 has (4,1), (5,2), (10,1)
    - $\sum_i u_i$ = 1/4 + 2/5 + 1/10 = 0.75 $\leq 3(2^{\frac{1}{3}} - 1) \approx 0.78 \Rightarrow$ Schedulable
  - Scheduling algorithm
    - Job with shortest period has highest priority $\Rightarrow$ schedule first



Real Time Scheduling

- Priority based algorithms
  - higher priority task can preempt the execution of lower priority task
  - Priority inversion
    - A high priority task T_h attempts to preempt a lower priority task T_l $\rightarrow$ but cannot $\rightarrow$ because T_l holds a shared resource that T_h needs
  - Unbounded priority inversion
    - T_l holds a shared resource that T_h needs, T_l may be preempted by another task T_m (priority of T_h > priority of T_m > priority of T_l) $\Rightarrow$ T_h's wait time is no longer bounded by the resource usage time T_l may incur $\Rightarrow$ In fact, there is no bound $\Rightarrow$ May cause problem
    - NASA rover Mars Pathfinder uses WindRiver VxWorks RTOS, which is susceptible to priority inversion
    - $\Rightarrow$ Patch the lander software by enabling "**priority inheritance**"

Real Time Scheduling

- NSAS rover priority inversion
  - Tasks and events
    - T_l: meteorological data gathering task: locked the infor-bus for storing the gathered data
    - T_h: infor-bus manager: attempted to lock the infor-bus to read data related to its management task ⇒ blocked, waiting for infor-bus
    - T_m: communicator, does not use infor-bus, but got scheduled to communicate and preempted T_l's execution
    - ⇒ T_h waits for infor-bus locked by T_l, T_l is ready, waits for T_m for finishing its execution
    - T_{other}: watchdog, periodically checks the status of the critical tasks to ensure their proper operation: Noticed that T_h had not been active for some time ⇒ believed that something had gone wrong ⇒ started system reset

Real Time Scheduling

- NSAS rover priority inversion
  - Problem
    - Happened when rover is on Mars ⇒ Took the ground crew many hours to identify the problem and fix it
    - In fact, the same problem had occurred during pre-flight tests, but was believed to be a hardware glitch and got put aside
  - Solution
    - Priority inheritance
    - If a task T_l blocks the execution of a higher priority task T_h ⇒ T_l inherits the priority of T_h (just temporarily)

Readings

- Sections 9.2, 9.3
  - Regular CPU scheduling algorithms
- Section 10.2
  - Real time scheduling algorithms
- Sections 10.3-10.6
  - CPU scheduling in real systems
  - Optional reading, not covered in class