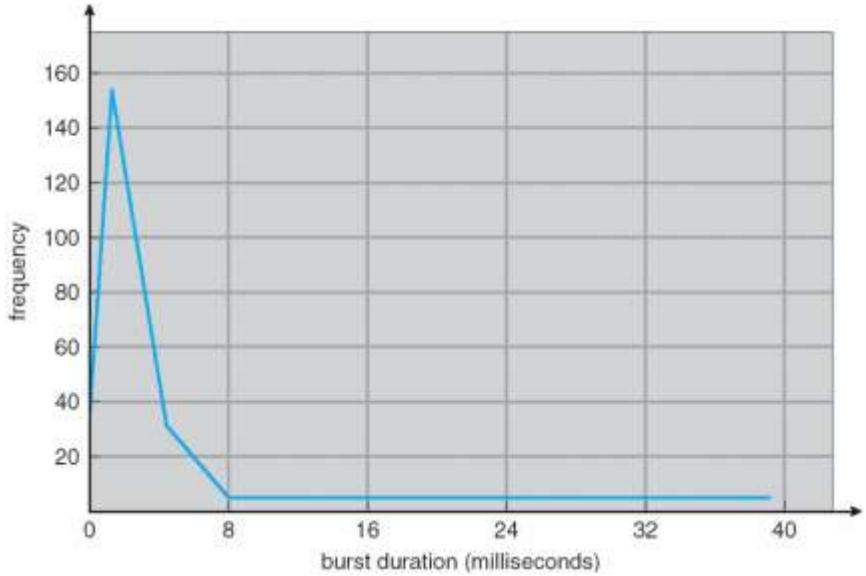
CPU Scheduling

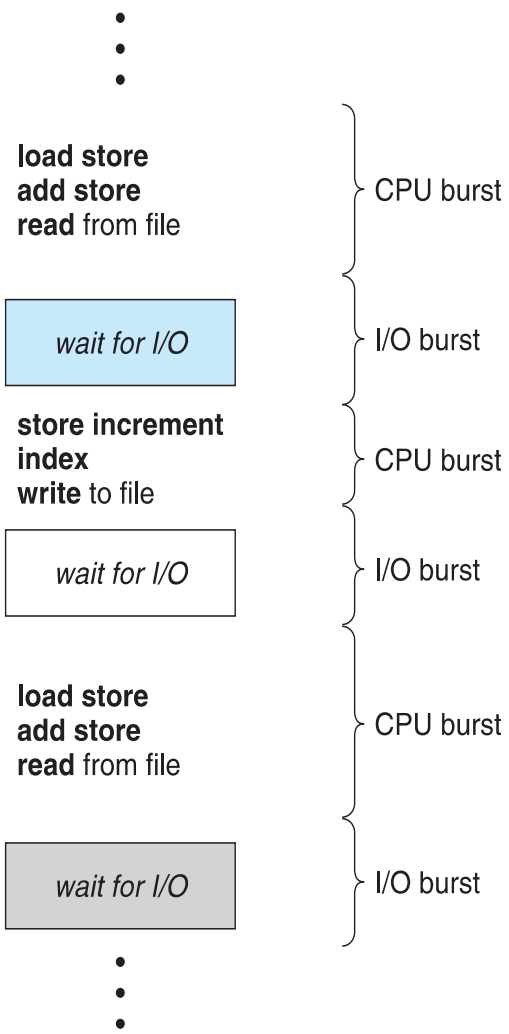
- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
- Operating Systems Examples

Last modification date: 24.11.2019

Basic Concepts

- Maximum CPU utilization obtained with multiprogramming
- CPU–I/O Burst Cycle – Process execution consists of a **cycle** of CPU execution and I/O wait. **CPU burst** followed by **I/O burst**. CPU burst distribution is of main concern
- Selection of CPU scheduling algorithm depends on statistics of CPU-I/O bursts





CPU Scheduler

- **Short-term scheduler** selects from among the processes in ready queue, and allocates the CPU to one of them
  - Queue may be ordered in various ways
- CPU scheduling decisions may take place when a process:
  1. Switches from running to waiting state
  2. Switches from running to ready state
  3. Switches from waiting to ready
  4. Terminates
- Scheduling under 1 and 4 is **non-preemptive**
- All other scheduling is **preemptive**
  - Consider access to shared data
  - Consider preemption while in kernel mode
  - Consider interrupts occurring during crucial OS activities

Dispatcher

- **Dispatcher** module gives control of the CPU to the process selected by the short-term scheduler; this involves:
  - switching context
  - switching to user mode
  - jumping to the proper location in the user program to restart that program
- **Dispatch latency** – time it takes for the dispatcher to stop one process and start another running

Scheduling Criteria

System view:

- **CPU utilization** – keep the CPU as busy as possible
- **Throughput** – # of processes that complete their execution per time unit
- **Fairness**
- **Respecting external priorities**
- **Balancing load of resources used**

User view:

- **Turnaround time** – amount of time to execute a particular process
- **Waiting time** – amount of time a process has been waiting in the ready queue
- **Response time** – amount of time it takes from when a request was submitted until the first response is produced, not output (for time-sharing environment)
- **Predictability** – a task is complete within some reasonable time regardless system load.

Scheduling Algorithm Optimization Criteria

- Max CPU utilization
- Max throughput
- Min turnaround time
- Min waiting time
- Min response time

First- Come, First-Served (FCFS) Scheduling

Process	Burst Time
P_1	24
P_2	3
P_3	3

- Suppose that the processes arrive in the order: P_1 , P_2 , P_3
  The **Gantt Chart** for the schedule is:

- Waiting time for P_1 = 0; P_2 = 24; P_3 = 27
- **Average waiting time**: (0 + 24 + 27)/3 = **17**, **max. waiting time = 27**

FCFS Scheduling (Cont.)

Suppose that the processes arrive in the order:

$$P_2 , P_3 , P_1$$

- The Gantt chart for the schedule is:

- Waiting time for $P_1 = 6; P_2 = 0; P_3 = 3$
- **Average waiting time**: (6 + 0 + 3)/3 = **3** , **max. waiting time = 6**
- Much better than previous case
- **Convoy effect** - short process behind long process
  - Consider one CPU-bound and many I/O-bound processes

Shortest-Job-First (SJF) Scheduling

- Associate with each process the length of its next CPU burst
  - Use these lengths to schedule the process with the shortest time

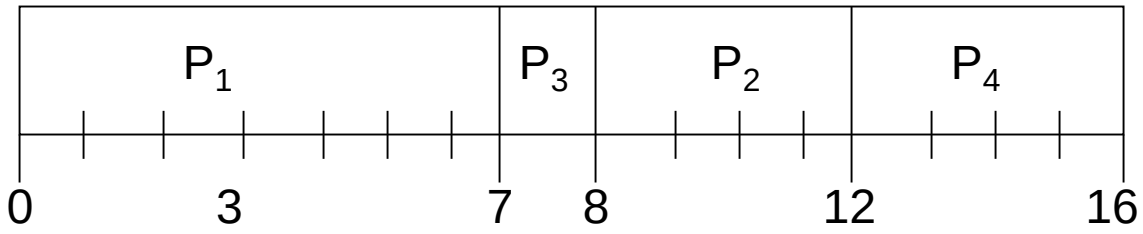
- **SJF is optimal** – gives minimum average waiting time for a given set of processes
  - The difficulty is knowing the length of the next CPU request
  - Could ask the user

- Preemptive version of SJF is called **shortest-remaining-time-first (SRTF)**

Example of SJF

Process		Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF scheduling chart



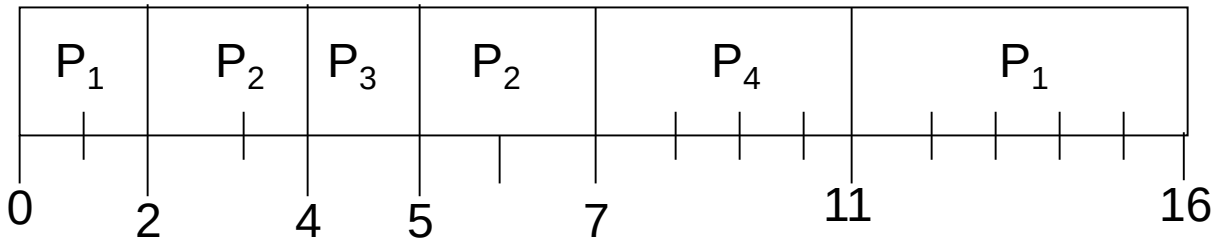
- **Average waiting time** = (0 + 6 + 3 + 7) / 4 = **7**
- **Max. waiting time = 7**

Example of Shortest-remaining-time-first

- Now we add the concepts of varying arrival times and preemption to the analysis

Process	*Arrival* Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- **Preemptive** SJF Gantt Chart



- **Average waiting time** = [9+1+0+2)]/4 = **3**
- **Max. waiting time = 9**

Determining Length of Next CPU Burst

- Can **only estimate** the length of the next CPU burst

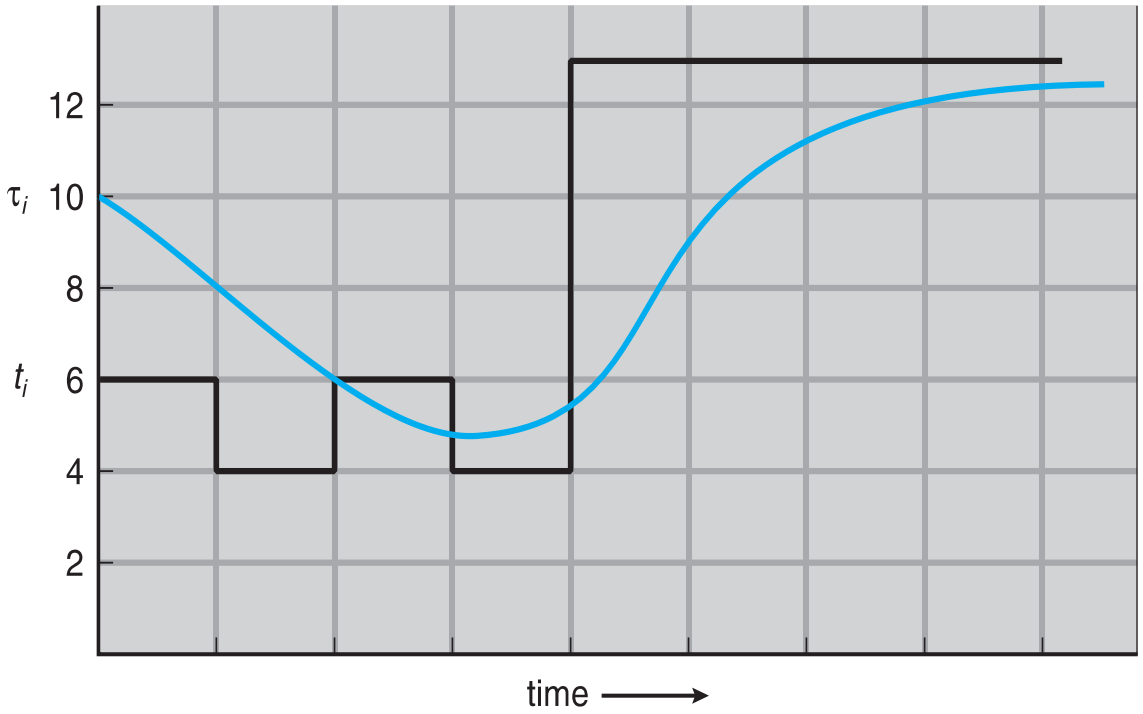
- Prediction can be done by using the length of previous CPU bursts, using exponential averaging

1. t_n = actual length of n^{th} CPU burst
2. τ_{n+1} = predicted value for the next CPU burst
3. $\alpha, 0 \le \alpha \le 1$
4. Define :

$$\tau_{n=1} = \alpha\, t_n + (1-\alpha)\tau_n .$$

- Commonly, α set to ½

Prediction of the Length of the Next CPU Burst



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

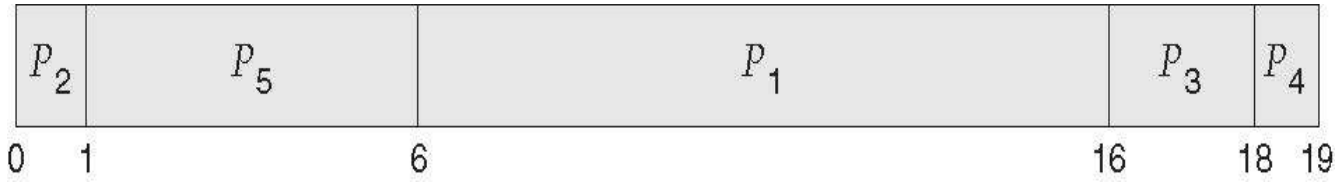
Priority Scheduling

- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority (smallest integer ≡ highest priority)
  - **Preemptive** or
  - **Nonpreemptive**
- SJF is priority scheduling where priority is the inverse of predicted next CPU burst time
- Problem ≡ **Starvation** – low priority processes may never execute
- Solution ≡ **Aging** – as time progresses increase the priority of the process

Example of Priority Scheduling

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

- Priority scheduling Gantt Chart



- **Average waiting time** = 8.2, **max. waiting time = 18**

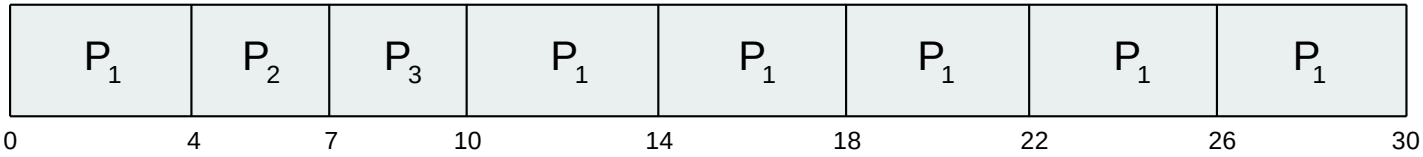
Round Robin (RR)

- Each process gets a small unit of CPU time (**time quantum *q***), usually 10-100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue.
- If there are ***n*** processes in the ready queue and the time quantum is ***q***, then each process gets **1/*n*** of the CPU time in chunks of at most *q* time units at once. No process waits more than **(*n*-1)*q*** time units.
- Timer interrupts every quantum to schedule next process
- Performance
  - ***q*** large $\Rightarrow$ FIFO
  - ***q*** small $\Rightarrow$ ***q*** must be large with respect to context switch, otherwise overhead is too high

Example of RR with Time Quantum = 4

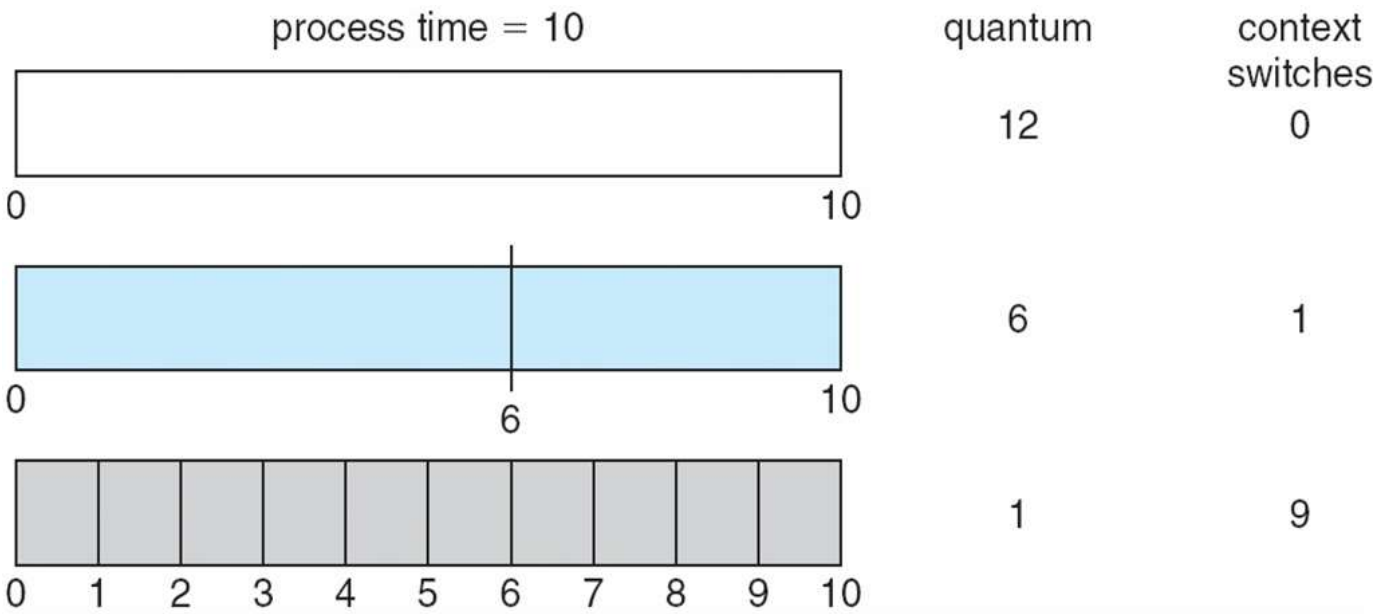
Process	Burst Time
P_1	24
P_2	3
P_3	3

- The Gantt chart is:



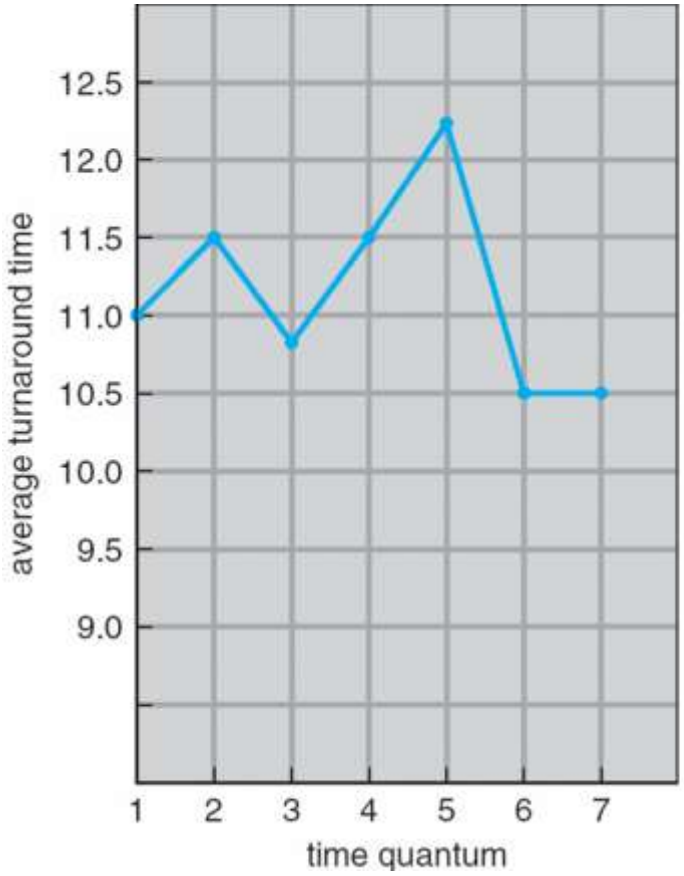
- Typically, higher average turnaround than SJF, but better **response**
- q should be large compared to context switch time
- q usually 10ms to 100ms, context switch < 10 usec

Time Quantum and Context Switch Time



- Context switching is a waste for the system → cannot be too frequent
- Scheduler decision has to be made with short time frame → system data structures have to be designed with this in mind.

Turnaround Time Varies With The Time Quantum



process	time
P_1	6
P_2	3
P_3	1
P_4	7

80% of CPU bursts should be shorter than **q**

Multilevel Queue

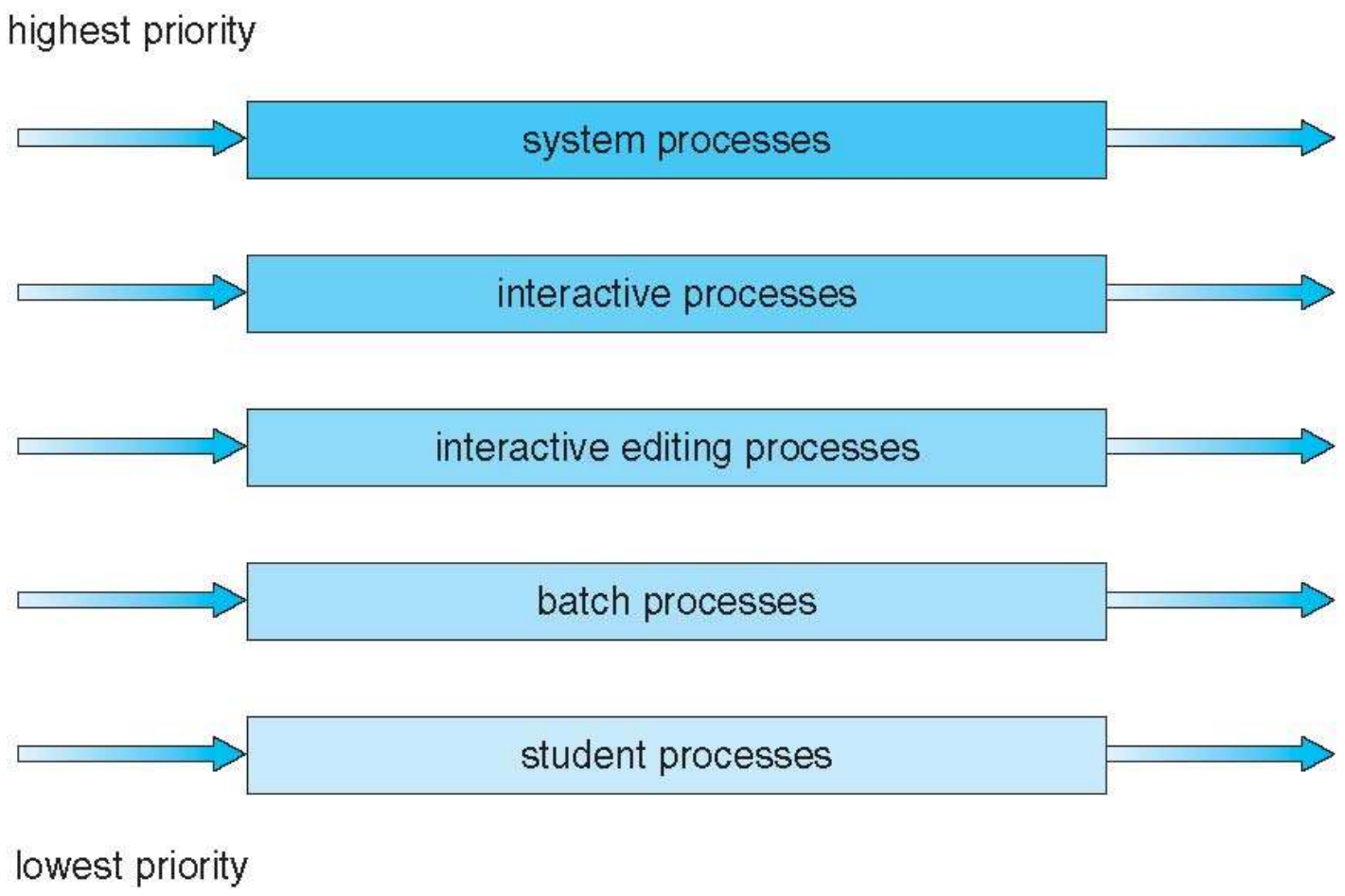
- Ready queue is partitioned into separate queues, eg:
  - **foreground** (interactive)
  - **background** (batch)
- Process permanently in a given queue
- Each queue has its own scheduling algorithm:
  - foreground – RR
  - background – FCFS
- Scheduling must be done between the queues:
  - Fixed priority scheduling; (i.e., serve all from foreground then from background). Possibility of starvation.
  - Time slice – each queue gets a certain amount of CPU time which it can schedule amongst its processes; i.e., 80% to foreground in RR
  - 20% to background in FCFS

Multilevel Feedback Queue

- A process can move between the various queues; aging can be implemented this way

- Multilevel-feedback-queue scheduler defined by the following parameters:
  - number of queues
  - scheduling algorithms for each queue
  - method used to determine when to upgrade a process
  - method used to determine when to demote a process
  - method used to determine which queue a process will enter when that process needs service

Multilevel Queue Scheduling



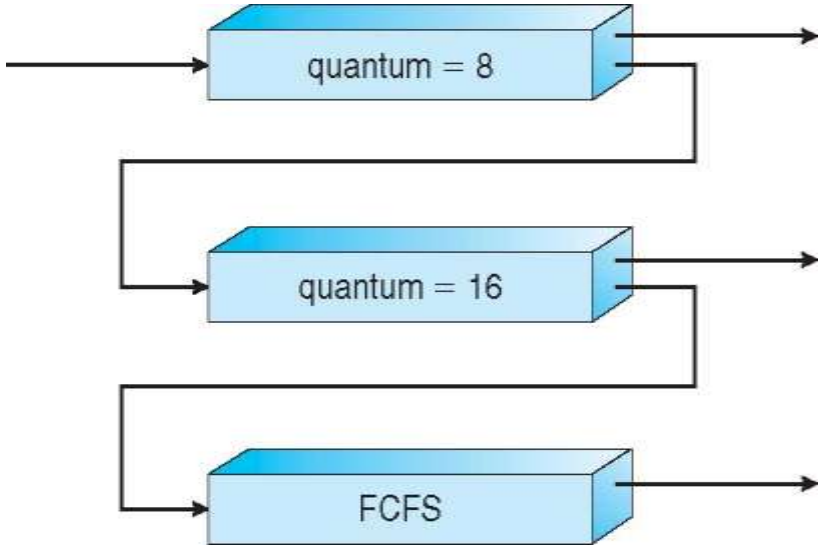
Multilevel Feedback Queue

- A process can move between the various queues; aging can be implemented this way
- Multilevel-feedback-queue scheduler defined by the following parameters:
  - number of queues
  - scheduling algorithms for each queue
  - method used to determine when to upgrade a process
  - method used to determine when to demote a process
  - method used to determine which queue a process will enter when that process needs service

Example of Multilevel Feedback Queue

- Three queues:
  - Q_0 – RR with time quantum 8 milliseconds
  - Q_1 – RR time quantum 16 milliseconds
  - Q_2 – FCFS

- Scheduling
  - A new job enters queue Q_0 which is served FCFS
    - When it gains CPU, job receives 8 milliseconds
    - If it does not finish in 8 milliseconds, job is moved to queue Q_1
  - At Q_1 job is again served FCFS and receives 16 additional milliseconds
    - If it still does not complete, it is preempted and moved to queue Q_2



Windows Scheduling

- Windows uses priority-based preemptive scheduling
- Highest-priority thread runs next
- **Dispatcher** is scheduler
- Thread runs until (1) blocks, (2) uses time slice, (3) preempted by higher-priority thread
- Real-time threads can preempt non-real-time ones
- 32-level priority scheme
- **Variable class** is 1-15, **real-time class** is 16-31
- Priority 0 is memory-management thread
- Queue for each priority
- If no run-able thread, **idle thread** is run
- If wait occurs, priority boost after becoming ready, depends on what was waited for. Foreground window given 3x priority boost
- Windows 7 added **user-mode scheduling** (**UMS**)
  - Applications create and manage threads independent of kernel
  - For large number of threads, much more efficient.

POSIX scheduling

- Each process or thread is controlled by an associated **scheduling policy** and **priority**. Associated with each policy is a **priority range**. Each policy definition specifies the minimum priority range for that policy. The priority ranges for each policy may overlap the priority ranges of other policies. Four scheduling policies are defined; others may be defined by the implementation.
  - SCHED_FIFO : First in-first out (FIFO) scheduling policy.
  - SCHED_RR : Round robin scheduling policy.
  - SCHED_SPORADIC : Sporadic server scheduling policy.
  - SCHED_OTHER : Another scheduling policy (default for normal processes/threads)

- Linux scheduler – a kernel component that decides which **runnable thread** will be executed by CPU next. All scheduling is **preemptive**. Scheduling policies:
  - Normal scheduling policies: SCHED_OTHER, SCHED_BATCH, SCHED_IDLE
  - Real-time scheduling policies: SCHED_FIFO, SCHED_RR, SCHED_DEADLINE (in place of SCHED_SPORADIC)
- More (details) to come - next semester