CSE 3221
**Operating System Fundamentals**

## No.4

# CPU scheduling

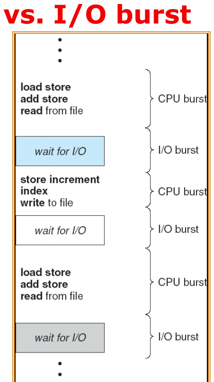
*Prof. Hui Jiang*
*Dept of Computer Science and Engineering*
*York University*

# CPU Scheduling

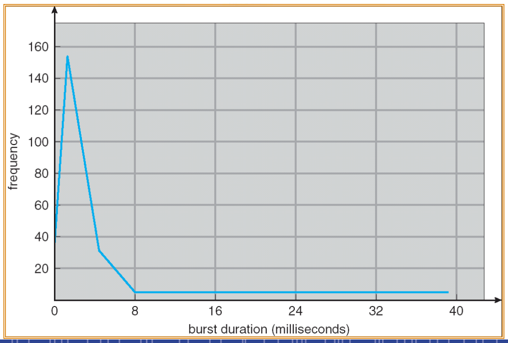
- CPU scheduling is the basis of multiprogramming
- CPU scheduling consists of two components:
  - CPU scheduler: when CPU becomes idle, the CPU scheduler must select from among the processes in ready queue.
  - Dispatcher: the module which gives control of CPU to the process selected by the CPU scheduler.
    - Switching context
    - Switching to user mode
    - Jumping to the proper location in user program to restart
  - Dispatch latency: the time it takes for the dispatcher to stop one process and start another running
    - Dispatcher should be as fast as possible

# CPU burst vs. I/O burst

- Process (thread) execution
  = CPU burst + I/O burst
- Process (thread) alternates between these two states.
- Length of these bursts is very different.



## Histogram of CPU-burst Times




## Non-preemptive vs. Preemptive scheduling

- 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.
- Non-preemptive scheduling takes place under 1 and 4.
  - Once the CPU has been allocated to a process, the process keeps the CPU until it releases CPU.
- Preemptive scheduling takes place in 1,2,3,4.
  - A running process can be preempted by another process
  - Not easy to make OS kernel to support preemptive scheduling
  - How about if the preempted process is updating some critical data structure?
    - Process synchronization
    - Disable interrupt

## Scheduling Criteria

- CPU utilization – keep the CPU as busy as possible.
  - Usage percentage (40% -- 90%)
- Throughput – # of processes that complete their execution per time unit.
- Turnaround time – amount of time to execute a particular process.
  - The interval from the time of submission a process to the time of completion.
- 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* the final output (for time-sharing environment).

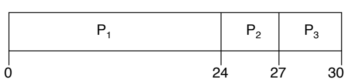
## Scheduling Algorithms

- First-come, first-served (FCFS) scheduling
- Shortest-Job-First (SJF) Scheduling
- Priority Scheduling
- Round-Robin (RR) scheduling
- Multi-level Queue Scheduling
- Multilevel Feedback Queue Scheduling

## First-Come, First-Served (FCFS) Scheduling

| Process | Burst Time |
|---|---|
| $P_1$ | 24 |
| $P_2$ | 3 |
| $P_3$ | 3 |

- Suppose that the processes arrive at time 0 in the order: $P_1$ , $P_2$ , $P_3$
  The Gantt Chart for the scheduling is:



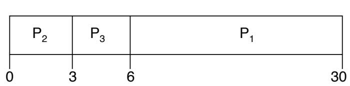

- Waiting time for $P_1$ = 0; $P_2$ = 24; $P_3$ = 27.
- Average waiting time: (0 + 24 + 27)/3 = 17.

## 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.
- FCFS is easy to implement (as a FIFO sequence).
- FCFS results in long wait in most cases and suffers convoy effect.
  - *Convoy effect* : all the other processes wait for one big process to get off the CPU.

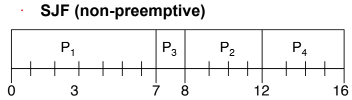
## Shortest-Job-First (SJF) Scheduling

- Associate with each process the length of its next CPU burst. Schedule CPU to process with the shortest time.
  - The shortest one is the first.
- Implementation: ready queue → sorted list.
- Two schemes:
  - nonpreemptive – once CPU given to the process it cannot be preempted until it completes its CPU burst.
  - preemptive – if a new process arrives with CPU burst length less than remaining time of current executing process, it preempts. This scheme is know as the Shortest-Remaining-Time-First (SRTF).
- SJF is optimal – gives minimum average waiting time for a given set of processes.

## Example of Non-Preemptive SJF

| 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 |

- SJF (non-preemptive)

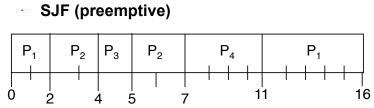



- Average waiting time = (0 + 6 + 3 + 7)/4 = 4

## Example of Preemptive SJF (shortest-remaining-time-first)

| 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 |

- SJF (preemptive)




- Average waiting time = (9 + 1 + 0 +2)/4 = 3

## Determining Length of Next CPU Burst

- Length of next CPU burst is unknown.
- Can only estimate the length.
- Can be done by using the length of previous CPU bursts, using exponential averaging, to predict the next one.

1. $t_n =$ actual lenght of $n^{th}$ CPU burst
2. $\tau_{n+1} =$ predicted value for the next CPU burst
3. $\alpha, 0 \le \alpha \le 1$
4. Define: $\tau_{n+1} = \alpha\, t_n + (1-\alpha)\tau_n.$

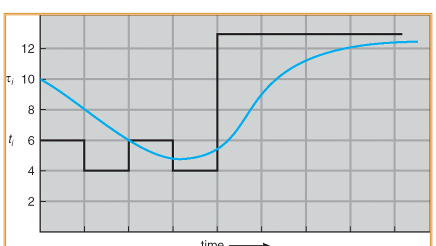
## Examples of Exponential Averaging

- $\alpha=0$
  - $\tau_{n+1} = \tau_n = \ldots = \tau_0$
  - Recent history does not count.
- $\alpha=1$
  - $\tau_{n+1} = t_n$
  - Only the actual last CPU burst counts.
- If we expand the formula, we get:

$$\tau_{n+1} = \alpha\, t_n + (1 - \alpha)\, t_{n-1} + \ldots + (1 - \alpha)^j\, t_{n-j} + \ldots + (1 - \alpha)^{n-1}\, t_0$$

- Since both α and (1 - α) are less than or equal to 1, each successive term has less weight than its predecessor.

## Prediction of the Length of the Next CPU Burst




| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| CPU burst ($t_i$) | | 6 | 4 | 6 | 4 | 13 | 13 | 13 | … |
| "guess" ($\tau_i$) | 10 | 8 | 6 | 6 | 5 | 9 | 11 | 12 | … |

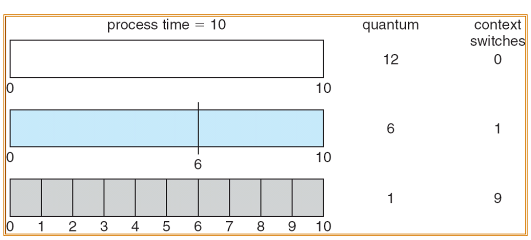
## Priority Scheduling

- 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
  - Nonpreemptive
- SJF is a priority scheduling where priority is the predicted next CPU burst time.
- Problem ➔ Starvation – low priority processes may never execute.
- Solution ➔ Aging – as time progresses increase the priority of the process.

## Round Robin (RR)

- Each process gets a small slice of CPU time (***time quantum***), usually 10-100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue.
  - Ready queue is a circular queue or FIFO queue.
- **Fairness**: 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.
- Performance:
  - *q* large ➔ FCFS
  - *q* small ➔ too many context switches, so overhead is high.
  - *q* must be large with respect to most CPU bursts' lengths.

## Time Quantum and Context Switch Time



## Example of RR with Time Quantum = 20

| Process | Burst Time |
|---|---|
| $P_1$ | 53 |
| $P_2$ | 17 |
| $P_3$ | 68 |
| $P_4$ | 24 |

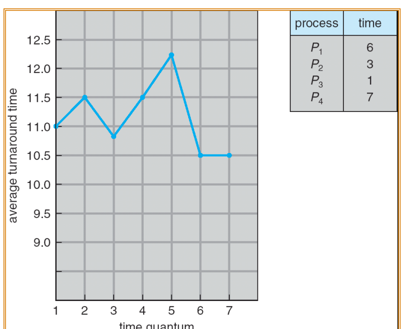
- The Gantt chart is:

| $P_1$ | $P_2$ | $P_3$ | $P_4$ | $P_1$ | $P_3$ | $P_4$ | $P_1$ | $P_3$ | $P_3$ |
|---|---|---|---|---|---|---|---|---|---|

0 20 37 57 77 97 117 121 134 154 162

- Typically, higher average waiting time than SJF, but better *response*.

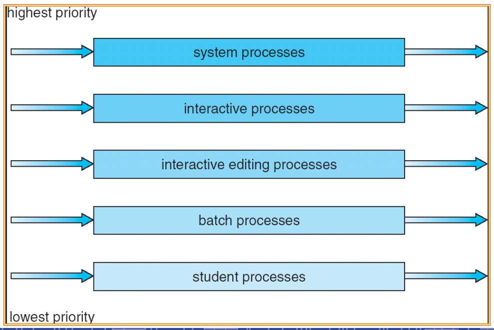
## Turnaround Time Varies With The Time Quantum




## Multilevel Queue

- Ready queue is partitioned into separate queues:
  - foreground (interactive)
  - background (batch)
- Any process is permanently assigned to one of these queues
- Each queue has its own scheduling algorithm, i.e.,
  - 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

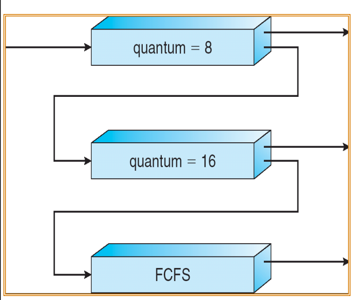
## One example of multilevel Queue Scheduling




## Multilevel Feedback Queue

- A process can move between the various queues; aging can be implemented this way.
  - If used too much CPU time → lower-priority queue
  - If waited too long → higher-priority queue
- Multilevel-feedback-queue scheduler defined by the following parameters:
  - number of queues
  - scheduling algorithms for each queue
  - method to determine when to upgrade a process
  - method to determine when to demote a process
  - method to determine which queue a process will enter when that process needs service
- It is the most general CPU scheduling algorithm. Can be configured to match a specific system under design.

## Example of Multilevel Feedback Queue




- Three queues:
  - $Q_0$ – time quantum 8 milliseconds
  - $Q_1$ – time quantum 16 milliseconds
  - $Q_2$ – FCFS
- Scheduling
  - A new job enters queue $Q_0$. 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 RR and receives 16 additional milliseconds. If it still does not complete, it is preempted and moved to queue $Q_2$.
  - Always preemptive.

## Scheduling in multi-CPU Era

- Multiple-Processor Scheduling
  – Multi-core scheduling
- Scheduling for multiple systems
  – Load balancer (long-term scheduler)
  – Scheduling for distributed systems

## Multiple-Processor Scheduling

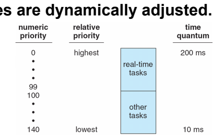
- CPU scheduling more complex when multiple CPUs are available.
- *Homogeneous processors* within a multiprocessor.
  – Any available processor can then be used to run any process in the queue.
- One common ready queue vs. a separate queue for each CPU.
- *Asymmetric multiprocessing* – one processor (master) schedules for all processors
  – only one processor accesses the system data structures
  – alleviating the need for data sharing.
- *Symmetric multiprocessing* – each processor is self-scheduling
  – Each processor select its processes from the queue
  – Process synchronization when accessing common queues

## Real-Time Scheduling

- *Hard real-time* systems – requires to complete a critical task within a guaranteed amount of time.
  – Hard to achieve in a general-purpose computer.
- *Soft real-time* computing – requires that the real-time processes receive priority over others (no aging).
- The dispatch latency must be small → preempt system call (kernel)
  – Adding preemption points (safe points) in system calls
  – Making the entire kernel preemptive by using process synchronization technique to protect all critical region

# Linux Scheduling

- Linux scheduling algorithm is preemptive, priority-based, variable-length RR, with complexity O(1).
- Priority values are dynamically adjusted.




- Use two so-called run-queues for READY queue:


# Scheduling Algorithm Evaluation

- Analytic evaluation: deterministic modeling
  - Given a pre-determined workload, calculate the performance of each algorithm for that workload.
- Queuing Models
  - No static workload available, so use the probabilistic distribution of CPU and I/O bursts.
  - Use queuing-network analysis.
  - The classes of algorithms and distributions that can be handled in this way are fairly limited.
- Simulation: use a simulator to model a computer system
  - simulator is driven by random-number generator according to certain distributions.
  - Simulator is driven by a trace file, which records actual events happened in a real system.