Operating Systems: CPU Scheduling

Neerja Mhaskar

Department of Computing and Software, McMaster University, Canada

Acknowledgements: Material based on the textbook Operating Systems Concepts (Chapter 5)

Basic Concepts

- OS schedules almost all resources available to the system.
- CPU being the primary resource, scheduling processes to execute on the CPU is central to OS design.
- Process execution consists of a **cycle** of CPU execution (**CPU burst**) and I/O wait (**I/O burst**).
- An I/O-bound program typically has many short CPU bursts.
- A CPU-bound program might have a few long CPU bursts
- This distribution is important in the selection of an appropriate CPU-scheduling algorithm.

Non-preemptive Vs. Preemptive

- **Non-preemptive scheduling** - A running task is executed till completion. It cannot be interrupted.

- **Preemptive scheduling** - The currently running process may be interrupted and moved to the Ready queue by the OS.

CPU Scheduler

- **Short-term scheduler (aka CPU scheduler)** selects from among the processes in ready queue and allocates the CPU to one of them.
  - Queue may be ordered in various ways
  - Nodes are PCBs (Process Control Blocks)

Scheduling Algorithms

- First- Come, First-Served (FCFS) Scheduling
- Shortest-Job-First (SJF) Scheduling
  - Shortest-remaining-time-first
- Priority Scheduling
- Round Robin (RR)

Scheduling Criteria

- **CPU utilization** – keep the CPU as busy as possible
- **Throughput** – # of processes that complete their execution per time unit
- **Turnaround time** – amount of time to execute a particular process
- **Waiting time** – amount of time a process has been waiting in the ready queue
- Scheduling Algorithm Optimization Criteria
  - Maximize CPU utilization and throughput
  - Minimize turnaround time and waiting time.

Gantt Chart and various time measurements

Gantt Chart: is a bar chart that illustrates a *particular process schedule*, including the start and finish times of each of the participating processes.

Example:

Suppose processes ***P1, P2, P3*** have CPU burst time **24, 3, 3** respectively.

- Suppose that the processes are scheduled in the order: P_1 , P_2 , P_3

  The Gantt Chart for this schedule is:

- From the chart we calculate
  - Waiting time for each process and average waiting time = waiting time for all processes/total no.of processes.
  - Turnaround time for a process = waiting time + CPU burst time of the process
  - Average turnaround time = turnaround time for all processes/total no.of processes.

First Come First Served Scheduling

- Process requesting the CPU first is allocated the CPU first.
  - Can be implemented using a FIFO queue.
- FCFS scheduling algorithm is non-preemptive
- Disadvantage:
  - Average CPU waiting time for a process is long.

FCFS Scheduling Example

Suppose processes *P1, P2, P3* have ***CPU burst times*** 24, 3, 3 respectively.

- Suppose that the processes arrive in the order: P_1 , P_2 , P_3

  The Gantt Chart for the schedule is:

- The waiting time for P_1 = 0; P_2 = 24; P_3 = 27 msec
- Average waiting time: (0 + 24 + 27)/3 = 17 msec
- Turnaround time for a process = waiting time + CPU burst time
- Average turnaround time: (24 + 27 + 30) /3 = 27 msec

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 $P_1 = 6$ msec; $P_2 = 0$ msec; $P_3 = 3$ msec
- Average waiting time: (6 + 0 + 3)/3 = 3 msec
- Much better than previous case
- **Convoy effect** - short process behind long process
  - Results in lower resource utilization

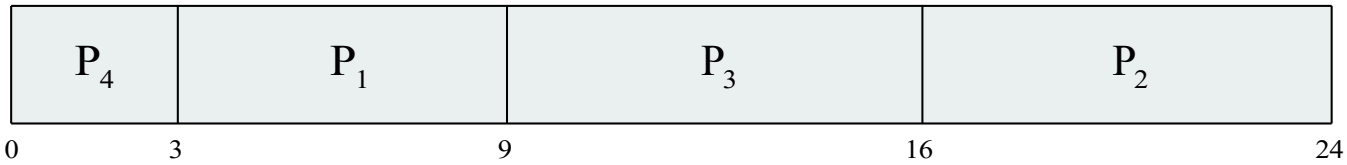
Shortest-Job-First (SJF) Scheduling

- Associated with each process is the length of its *next CPU burst*.
- **Shortest Job First** uses this next CPU burst to schedule the process with the shortest next CPU burst (rather than the total length/time required for the job).
  - If the next CPU bursts of two processes are the same, FCFS scheduling is used to break the tie.
- In fact, shortest-next-CPU-burst scheduling algorithm -- appropriate name for it.
- **SJF is optimal** – gives minimum average waiting time for a given set of processes
- The difficulty is knowing the length of the next CPU burst,
  - Not implemented at the level of short-term CPU scheduling.
  - Frequently used in long-term scheduling.

Example of SJF

Process	Next CPU Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

- SJF scheduling chart



- Average waiting time = (0+ 3 + 9 + 16) / 4 = 7 msec
- Turnaround time = (3+9+16+24)/4 = 13msec

Example of Shortest-remaining-time-first

- SJF algorithm can either be preemptive or nonpreemptive
- Preemptive version of SJF is called **shortest-remaining-time-first**
- Now we add the concepts of varying arrival times and preemption to the analysis
- When a new process arrives at time **T**, the remaining burst time of all processes which have arrived on or before that time are evaluated and the one with the shortest remaining burst time is executed.

Example of Shortest-remaining-time-first

- Consider the below example with 4 process and their arrival times and next CPU burst times.

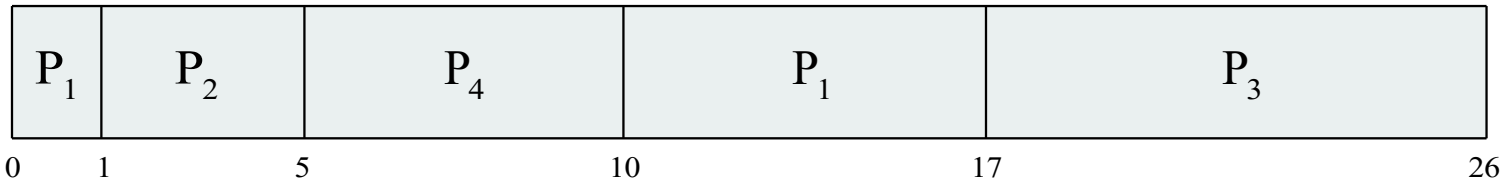
Process	*Arrival* Time T (ms)	Remaining Burst Time (ms)
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

- At time T = 0, P_1 is scheduled first as no other process is around to compare.
- At T=1, P_1 has executed for 1ms, when process P_2 arrives. Since the remaining burst time of P_1 = 7 > remaining burst time of P_2 = 4. P_1 is preempted and P_2 is scheduled to executed on the CPU.

Example of Shortest-remaining-time-first

- Then, at time T= 2, P_3 arrives. At this time, the remaining burst time of $P_1 = 7$, $P_2 = 3$, $P_3 = 9$.
- Since the remaining burst time of P_2 is the smallest, we let it continue to execute.
- Similar logic applies at time T = 3, and hence we let P_2 finish its execution.
- At T = 5, P_2 finishes execution, and the remaining burst times of $P_1 = 7$, $P_3 = 9$, $P_4 = 5$.
- Hence, we schedule P_4 next, then P_1 and finally P_3.

Preemptive SJF (shortest remaining time first) Gantt Chart



- Waiting time for P_2 = (1-1), P_1 = (0+ 10 - 1), P_3 = (17-2), P_4 = (5-3).
- Average waiting time 26/4 = 6.5 msec

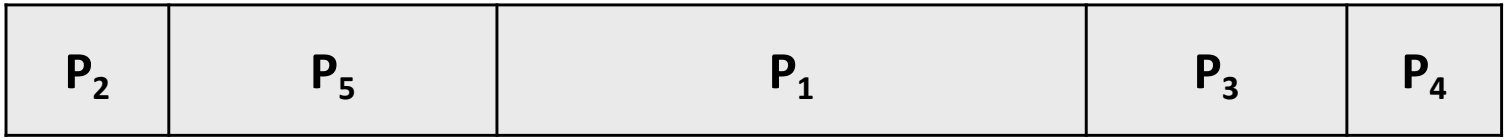
Priority Scheduling

- A **priority number (integer)** is associated with each process
  - Equal priority process scheduled in FCFS order.
- The CPU is allocated to the process with the highest priority (smallest integer ≡ highest priority)
- Priority scheduling can either be preemptive or nonpreemptive
- Problem ≡ **Starvation** – low priority processes may never execute
  - Solution ≡ **Aging** – as time progresses increase the priority of the process

Example of Priority Scheduling – nonpreemptive

Process	Burst Time(ms)	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 = (0 + 1 + 6 + 16 + 18)/5 = 8.2 ms (milliseconds)

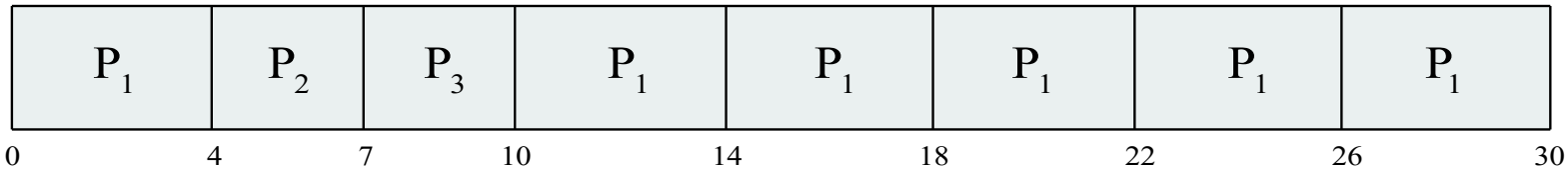
Round Robin (RR)

- Similar to FCFS scheduling, but preemption is added to enable the system to switch between processes.
- Processes are scheduled on FCFS basis from the ready queue, where each process gets a small unit of CPU time (**time quantum *q***)
- After this time has elapsed, the process is preempted and added **to the end** of the ready queue.
- Timer interrupts every quantum to schedule next process
- Preemptive Scheduling

Example of RR with Time Quantum = 4

Process	Burst Time	Time Quantum q = 4
P_1	24	
P_2	3	
P_3	3	

- The Gantt chart is:

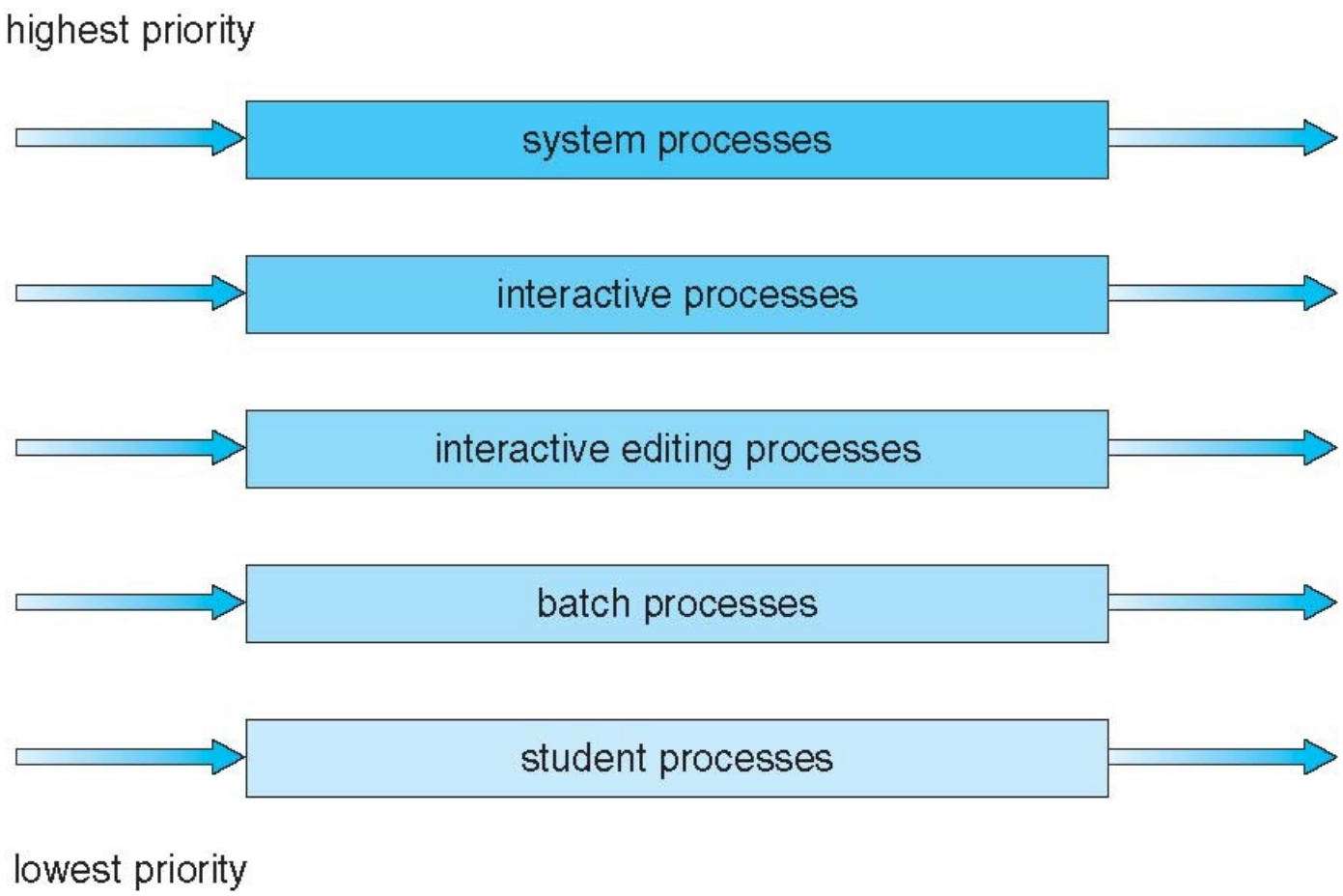


- Typically, higher average *turnaround time* than SJF, but better *response time.*

Multilevel Queue and Multilevel Feedback Queue Scheduling

- **Multilevel Queue Scheduling:** Ready queue is partitioned into various separate queues. Processes reside permanently in a given queue. Each queue has its own scheduling algorithm.
- Disadvantage: Since processes are permanently assigned to a given queue, it is inflexible.
- In contrast, in **Multilevel Feedback Queue Scheduling** a process can move between the various queues.
  - Process in low priority queue can be moved to high priority queue, and *aging* can be implemented this way.

Multilevel Queue and Multilevel Feedback Queue Scheduling



Multiple-Processor Scheduling

- **Symmetric multiprocessing (SMP)** – each processor is self-scheduling, all processes in common ready queue, or each has its own private queue of ready processes
- Currently, most common. Most modern operating systems support SMP, including Windows, Linux, and Mac OS X.