Chapter 5. CPU Scheduling

1. Basic Concepts
2. Scheduling Criteria
3. Scheduling Algorithms
4. Multiple-Processor Scheduling
5. Real-Time Scheduling

1. Basic Concepts

- ☺ *Most valuable resource?*
  - CPU
- ☺ *How to maximize CPU utilization?*
  - Multiprogramming
  - Scheduling of processes

Alternating Sequence of CPU And I/O Bursts

- CPU–I/O burst cycle – Process execution consists of a *cycle* of CPU execution and I/O wait
- CPU-bound programs of long CPU bursts
- I/O-bound programs of short CPU bursts
- CPU burst distribution
- See Figure 5.1 – 5.2.

CPU Scheduler

- **Short-term scheduler** (or **CPU scheduler**)
  - When the CPU becomes idle, it selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them.
- 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, i.e., wherever there is an interrupt
- Scheduling only under 1 and 4 is ***nonpreemptive*** or ***cooperative***
  - Windows 95
  - Embedded systems
- All other scheduling is ***preemptive.***
  - Almost all general purpose operating systems
  - ☺ *How to implement?*
    - Timer
  - Difficult inconsistency problem in data sharing among processes
  - Even kernel processes? General purpose OS and real time OS

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.
  - Must be very short.

2. Scheduling Criteria

1. CPU utilization
   - Keep the CPU as busy as possible
2. Throughput
   - # of processes that complete their execution per time unit
   - Important in batch systems
3. Turnaround time
   - Amount of time to execute a particular process
4. Waiting time
   - Amount of time a process has been waiting in the ready queue
5. 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).

Optimization Criteria

- Max CPU utilization
- Max throughput
- Min turnaround time
- Min waiting time
- Min response time
- Min variance in the response time

3. Scheduling Algorithms

- Mostly in the ready queue, not in the waiting queues

- First-Come, First-Served
- Shortest-Job-First
  - Preemptive
  - Non-preemptive
- Round Robin
- Priority

First-Come, First-Served (FCFS) Scheduling

Process	CPU burst time (ms)
P_1	24
P_2	3
P_3	3

- Suppose that the processes arrive almost at the same time in the order: P_1 , P_2 , P_3 in the ready Q. The **Gantt Chart** for the schedule is:

P_1	P_2	P_3

0 24 27 30

- Waiting time for each process:
  - $P_1 = 0$; $P_2 = 24$; $P_3 = 27$
- Average waiting time:
  - $(0 + 24 + 27)/3 = 17$
- *Convoy effect* short process behind long process – one CPU-bound and many I/O-bound processes -> I/O queue or ready queue could be empty from time to time.
- ☺ *Average turnaround time: ???*

FCFS Scheduling - continued

Suppose that the processes arrive in the order:

$$P_2 , P_3 , P_1$$

- The Gantt chart for the schedule is:

P_2	P_3	P_1	
0	3	6	30

- Waiting time for each process
  - $P_1 = 6; P_2 = 0; P_3 = 3$
- Average waiting time:
  - $(6 + 0 + 3)/3 = 3$
- ☺ *Much better than previous case; why ???*
- ☺ *Average turnaround time: ???*

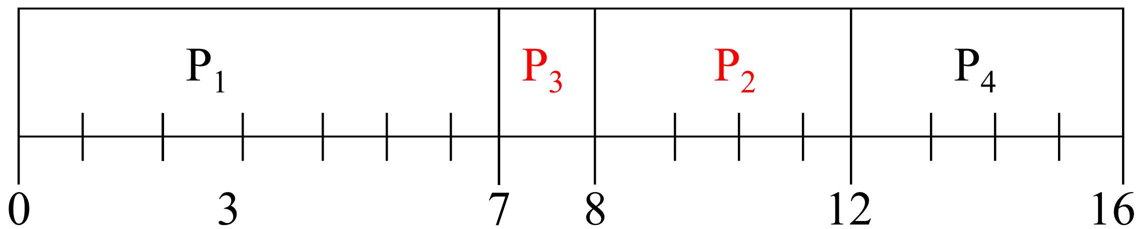
Shortest-Job-First (SJF) Scheduling

- Associate with each process, the length of its next CPU burst. Use these lengths to schedule the process with the shortest time.
- Two schemes:
  - Nonpreemptive
    - Once CPU is 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, preempt. 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)

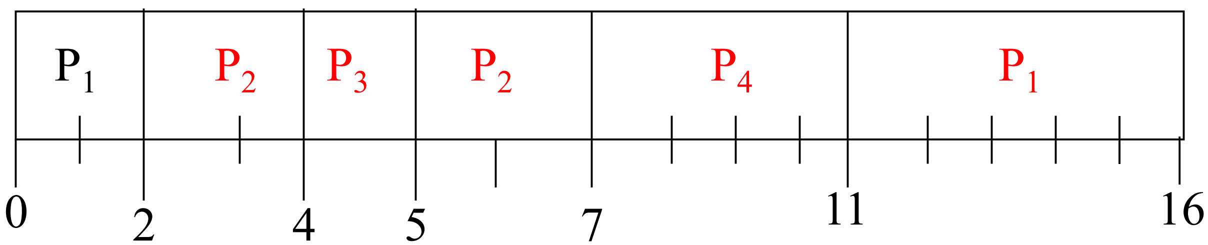


- Average waiting time
  - (0 + 6 + 3 + 7) / 4 = 4
- ☺ *Average turnaround time: ???*

Example of 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 (preemptive)



- Average waiting time
  - (9 + 1 + 0 + 2) / 4 = 3
  - ☺ *??? in FCFS*
- ☺ *Average turnaround time: ???*

Round Robin (RR)

- Each process gets a small unit 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.
- 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.
- ☺ *It is called ???.*
- Performance
  - q large -> FIFO
  - q small -> q must be large with respect to context switch, otherwise the overhead due to context switch is too high.

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

- ☺ *Preemptive or nonpreemptive???*
- Typically, higher average turnaround than SJF, but better *response*
  - ☺ *Average turnaround time in FCFS: ???*
  - ☺ *Average turnaround time in SJF: ???*
  - ☺ *Average turnaround time in RR: ???*

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???*
  - 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

Priority Scheduling

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

- Gantt chart:

P2	*P5*	*P1*	*P3*	*P4*
0–1	1–6	6–16	16–18	18–19

- ☺ *Preemptive or nonpreemptive???*
- Average waiting time
  - ☺ *???*
  - ☺ *??? in FCFS*
- ☺ *Average turnaround time: ???*

Example

Process	Arrival Time	Priority	Burst Time
P1	0	5	10
P2	0	2	2
P3	2	1	1
P4	2	2	4
P5	3	3	3

- FCFS
- Preemptive SJF
- Nonpreemptive SJF
- Preemptive priority
- Nonpreemptive priority
- RR with slice 2

Multilevel Queue Scheduling

- See Figure 5.6.
- Ready queue is partitioned into separate queues:
  - Example
    - Foreground (interactive)
    - Background (batch)
- Each queue has its own scheduling algorithm:
  - Example
    - 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
  - One example: Time slice
    - Each queue gets a certain amount of CPU time which it can schedule amongst its processes.
    - Example: 80% to foreground in RR and 20% to background in FCFS

Multilevel Feedback Queue Scheduling

- 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

- See Figure 5.7.
- 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 which is served FCFS. When it gains CPU, the job receives 8 milliseconds. If it does not finish in 8 milliseconds, the 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.

4. Multiple-Processor Scheduling

- CPU scheduling more complex when multiple CPUs are available
- ☺ *Homogeneous processors* within a multiprocessor system ???
- *Load sharing*
  - ☺ *Multiple ready queues ???*
  - ☺ *Single ready queue ???*

- *Asymmetric multiprocessing*
  - Only one processor accesses the system data structures, alleviating the need for data sharing
  - One master processor and several other processors
- *Symmetric multiprocessing* (*SMP*)
  - Each processor is self-scheduling
  - Single ready queue
  - Process synchronization problem
  - All modern OSes support SMP

5. Real-Time Scheduling

- *Hard real-time* systems – required to complete a critical task within a guaranteed amount of time; very hard to implement
- *Soft real-time* computing – requires that critical processes receive priority over less fortunate ones
  - The priority of real-time processes must not degrade over time.
  - The smaller the dispatch latency, the faster a real-time process can start running.
  - => We need to allow system calls to be preemptible.
  - => One way is to enter preemption points in long-duration system calls, or

  Preemptible kernel:
    - Various synchronization mechanisms are needed for all kernel data structures.
    - => Priority inversion problem – higher-priority processes should wait until lower-priority processes finish the use of kernel data structures, but the lower-priority processes could not be scheduled.
    - => One solution - priority-inheritance protocol – lower-priority processes accessing a kernel data structure wanted by a higher-priority process get the same higher priority.