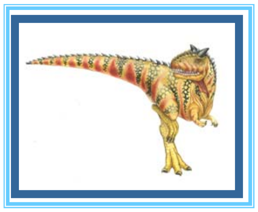
Chapter 5: CPU Scheduling



(slides selected/reordered/modified by R. Doemer, 01/11/11)

Chapter 5: CPU Scheduling

- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
- Thread Scheduling
- Multiple-Processor Scheduling
- Operating Systems Examples
- Algorithm Evaluation

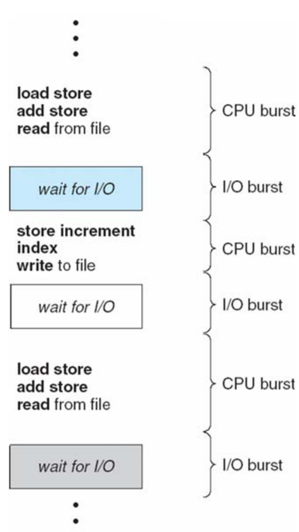
(slide modified by R. Doemer, 01/12/11)

Basic Concepts

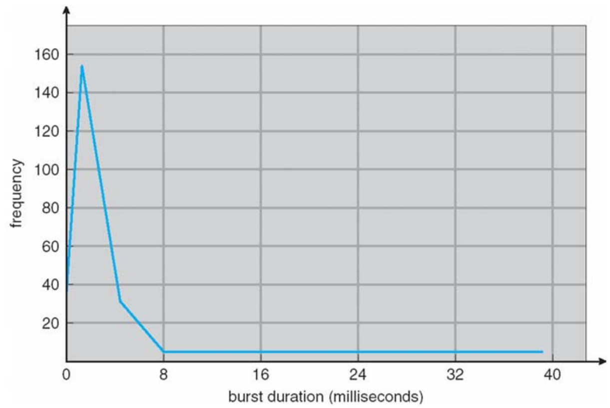
- Maximum CPU utilization obtained with multiprogramming
- **CPU–I/O Burst Cycle** –
  Process execution consists of a *cycle* of
  - CPU execution and
  - I/O wait
- **CPU burst** distribution

(slide modified by R. Doemer, 04/20/10)

Alternating Sequence of CPU And I/O Bursts



Histogram of CPU-burst Times



CPU Scheduler

- 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
- Scheduling under 1 and 4 is **non-preemptive**
- All other scheduling is **preemptive**

(slide modified by R. Doemer, 04/20/10)

Dispatcher

- **Dispatcher** module gives control of the CPU to the process selected by the *short-term scheduler*
- Dispatching 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

(slide modified by R. Doemer, 04/20/10)

Scheduling Algorithm Criteria

- **CPU utilization**
  - keep the CPU as busy as possible
- **Throughput**
  - number 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
- **Response time**
  - amount of time it takes from when a request was submitted until the first response is produced (not the time to output result!)
  - for time-sharing environment

(slide modified by R. Doemer, 01/11/11)

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:

P_1	P_2	P_3
0 – 24	24 – 27	27 – 30

- 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:

P_2	P_3	P_1
0 – 3	3 – 6	6 – 30

- Waiting time for P_1 = 6; P_2 = 0; P_3 = 3
- Average waiting time: (6 + 0 + 3)/3 = 3
- Much better than previous case
- *Convoy effect* short process behind long process

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
  - SJF gives minimum average waiting time for a given set of processes

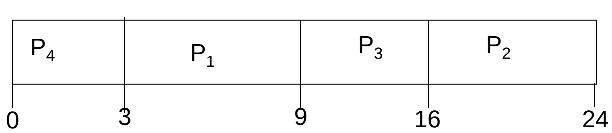
- However, there's a problem:
  - The difficulty is knowing the length of the next CPU request…

(slide modified by R. Doemer, 04/20/10)

Example of SJF

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

- SJF scheduling chart



- Average waiting time = (3 + 16 + 9 + 0) / 4 = 7

(slide fixed by R. Doemer, 01/07/09)

Estimating Length of Next CPU Burst

- Can only *estimate* the length!
  - Note: Text book calls this estimation 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. $\tau_{n+1} =$ predicted value for the next CPU burst
3. $\alpha, 0 \le \alpha \le 1$
4. Define: $\tau_{n+1} = \alpha\, t_n + (1-\alpha)\tau_n$

(slide modified by R. Doemer, 04/21/10)

Examples of Exponential Averaging

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

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

$$\tau_{n+1} = \alpha\, t_n + (1 - \alpha)\alpha\, t_{n-1} + \ldots$$
$$+ (1 - \alpha)^j \alpha\, t_{n-j} + \ldots$$
$$+ (1 - \alpha)^{n+1} \tau_0$$

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

(slide modified by R. Doemer, 04/21/10)

Priority Scheduling

- A **priority number** (an integer) is associated with each process
- The CPU is allocated to the process with the highest priority (smallest integer ≡ highest priority)
  - Preemptive
  - Non-preemptive
- SJF is an example of 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

(slide modified by R. Doemer, 04/20/10)

Round Robin (RR) Scheduling

- 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.
- Performance
  - q large $\Rightarrow$ RR degenerates to FCFS
  - q small $\Rightarrow$ q should be large with respect to context switch, otherwise overhead is too high

(slide modified by R. Doemer, 04/20/10)

Example of RR with Time Quantum = 4

Process	Burst Time
P_1	24
P_2	3
P_3	3

- The Gantt chart is:

P_1	P_2	P_3	P_1	P_1	P_1	P_1	P_1

0 4 7 10 14 18 22 26 30

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

(slide modified by R. Doemer, 04/20/10)

Multilevel Queue Scheduling

- Ready queue is partitioned into separate queues
  - foreground (interactive)
  - background (batch)
- 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

(slide modified by R. Doemer, 04/20/10)

End of Chapter 5