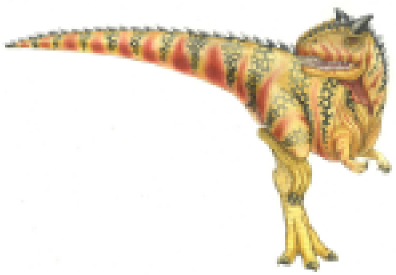
Chapter 5a: CPUScheduling



Outline

- Basic Concepts

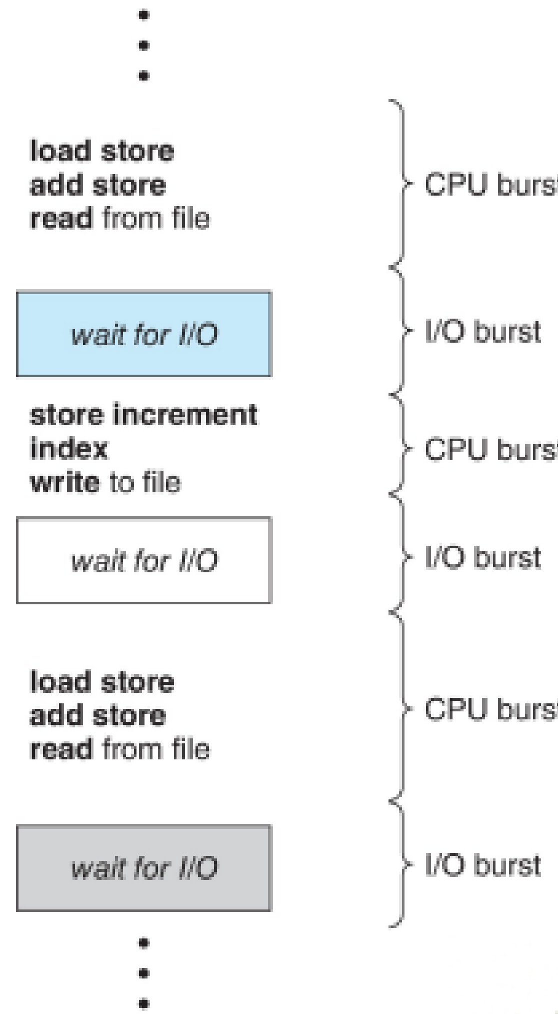
Objectives

- Scheduling Criteria
- Scheduling Algorithms

- Describe various CPU scheduling algorithms
- Assess CPU scheduling algorithms based on schedulingcriteria
- Explain the issues related to multiprocessor and multicorescheduling
- Describe various real-time scheduling algorithms
- Describe the scheduling algorithms used in the Windows, Linux, andSolaris operating systems
- Apply modeling and simulations to evaluate CPUschedulingalgorithms

Basic Concepts

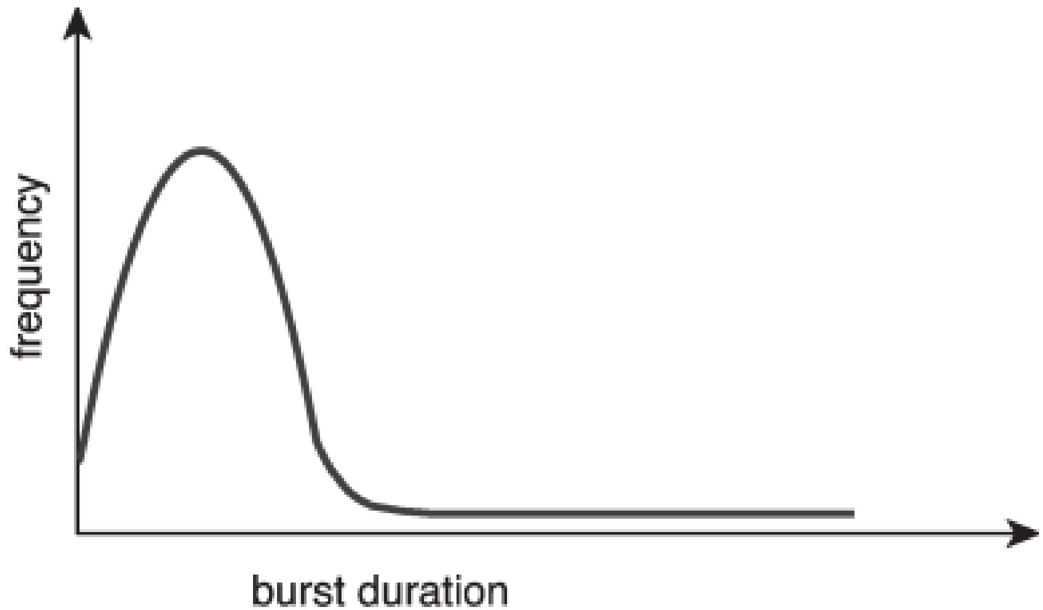
- The objective of multiprogramming is to have some process running at all times in processor, to maximize CPU utilization.
- All resources are scheduled before they are used.
- Scheduling is central to operating-system design
- Process execution consists of a cycle of CPU execution and I/O wait.
- The state of process under execution is called CPU burst and state of process under I/O request & handling is called I/O burst.
- Processes alternate between these two states.



the
its

Histogramof CPU-burst Times

• Many short CPU bursts and few longer CPUbursts• An I/O bound program would have many short CPUbursts• CPU bound program might have fewvery longCPUbursts



CPUScheduler

- The **short term scheduler** selects from among the processesinreadyqueue, and allocates a CPU core to one of them
  - Queue not necessarily be FIFO, may be ordered invariouswaysusing priority queue, a tree or unroders LL • Records in queues are PCBs of ready processes ▪ CPU scheduling decisions may take place when a process:

1. Switches from running to waiting state (I/Orequest, childprocesstocomplete)
2. Switches from running to ready state (Interrupt occurs) 3. Switches from waiting to ready (completion of I/O) 4. Terminates

- For situations 1 and 4, there is no choice in terms of scheduling. Anewprocess (if one exists in the ready queue) must be selectedfor execution.

- For situations 2 and 3, however, there is a choice.

Preemptive and NonpreemptiveScheduling

- When scheduling takes place only under circumstances1and4, the scheduling scheme is **nonpreemptive**.

- Otherwise, it is **preemptive**.
- **Non - Preemptive Scheduling** – once the CPUhasbeenallocated to a process, the process keeps theCPUuntil it releases the CPU either by terminating or by switchingtothewaiting state.
- Virtually all modern operating systems includingWindows,MacOS, Linux, and UNIX use preemptive schedulingalgorithms.

Preemptive SchedulingandRaceConditions

- **Preemptive Scheduling** – The process under execution, maybe released from the CPU, in

- the middle of execution due to some inconsistent stateof theprocess.
- Preemptive scheduling can result in race conditionswhendataare shared among several processes.
- Consider the case of two processes that sharedata. Whileoneprocess is updating the data, it is preempted sothat thesecondprocess can run. The second process then tries toreadthedata, which are in an inconsistent state.

Dispatcher

- Dispatcher is the module that 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

SchedulingCriteria

- **CPU utilization** – keep the CPU as busy as possible. CPUutilization can range from 0 to 100 percent. In a real system, it should range from 40 to 90 percent

- **Throughput** – # of processes that complete their executionper time unit
- **Turnaround time** – amount of time to execute a particular process;The interval from the time of submission of a process tothetimeofcompletion is the turnaround time. Turnaround time is thesumof the periods spent waiting to get into memory, waiting inthereadyqueue, executing on the CPU, and doing I/O.
- **Waiting time** – amount of time a process has been waitingintheready queue
- **Response time** – amount of time it takes fromwhen arequest wassubmitted until the first response is produced.

Scheduling AlgorithmOptimizationCriteria

- Max CPU utilization
- Max throughput

- Min turnaround time
- Min waiting time
- Min response time

First- Come, First-Served(FCFS) Scheduling

- Other names of this algorithm are:
  - First-In-First-Out (FIFO)
  - Run-to-Completion
  - Run-Until-Done
- First-Come-First-Served algorithm is the simplest schedulingalgorithm.Processes are dispatched according to their arrival timeonthereadyqueue. This algorithm is always nonpreemptive, once aprocessisassigned to CPU, it runs to completion.

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:

(24)$^{(3)\ (3)}$

- 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:

(3) (3) (24)

- 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 • Consider one CPU-bound and many I/O-bound processes

FCFS Scheduling

- **Advantages :**
- more predictable than other schemes since it offers time ▪ code for FCFS scheduling is simple to write and understand ▪ **Disadvantages:**
- Short jobs(process) may have to wait for long time ▪ Important jobs (with higher priority) have to wait ▪ cannot guarantee good response time ▪ average waiting time and turn around time is often quitelong ▪ lower CPU and device utilization.

Shortest-Job-First (SJF) Scheduling

- Associate with each process the length of its next CPUburst • Use these lengths to schedule the process with theshortest time- SJF is optimal – gives minimum average waiting time for agivensetof processes
  - The difficulty is knowing the length of the next CPUrequest • Could ask the user

Shortest-Job-First (SJF) Scheduling

- Associate with each process the length of its next CPUburst • Use these lengths to schedule the process with theshortest time
- SJF is optimal – gives minimum average waiting time for agiven set of processes
- Preemptive version called **shortest-remaining-time-first** ▪ How do we

determine the length of the next CPUburst?• Could ask the user

- Estimate

Example of SJF

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

(3) (6) (7) (8)

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

Determining Lengthof NextCPUBurst

- Can only estimate the length – should be similar to thepreviousone
  - Then pick process with shortest predicted next CPUburst
- Can be done by using the length of previous CPUbursts, usingexponential averaging

1. actual lengthof CPUburst t_n n^{th} =

2. predictedvaluefor thenext CPUburst τ_{n+1} =

3. $\alpha, 0 \le \alpha \le 1$

4. Define:

- Commonly, α set to ½

Prediction of the Length of theNext CPUBurst

Examples of Exponential Averaging

- $\alpha = 0$
  - $\tau_{n+1} = \tau_n$
  - Recent history does not count
- $\alpha = 1$
  - $\tau_{n+1} = \alpha\, t_n$
  - 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 $(1 - \alpha)$ are less than or equal to 1, eachsuccessiveterm has less weight than its predecessor

Example of Shortest-remaining-time-first

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

Process	*Arrival* Time	Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

- *Preemptive* SJF Gantt Chart

- Average waiting time = [(10-1)+(1-1)+(17-2)+(5-3)]/4 =26/4=6.5

Round Robin(RR)

- Each process gets a small unit of CPU time (**time quantum** q), usually 10-100 milliseconds. After this time has elapsed, theprocess is preempted and added to the end of the ready queue.
- If there are n processes in the ready queue and the timequantumis q, then each process gets $1/n$ of the CPU time in chunksof 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 RRwithTimeQuantum=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 10 milliseconds to 100 milliseconds,
  - Context switch < 10 microseconds

Time QuantumandContext SwitchTime

ries WithTheTimeQuantum

80%of CPUbursts
should be shorter thanq

Priority Scheduling

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

Example of PriorityScheduling

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

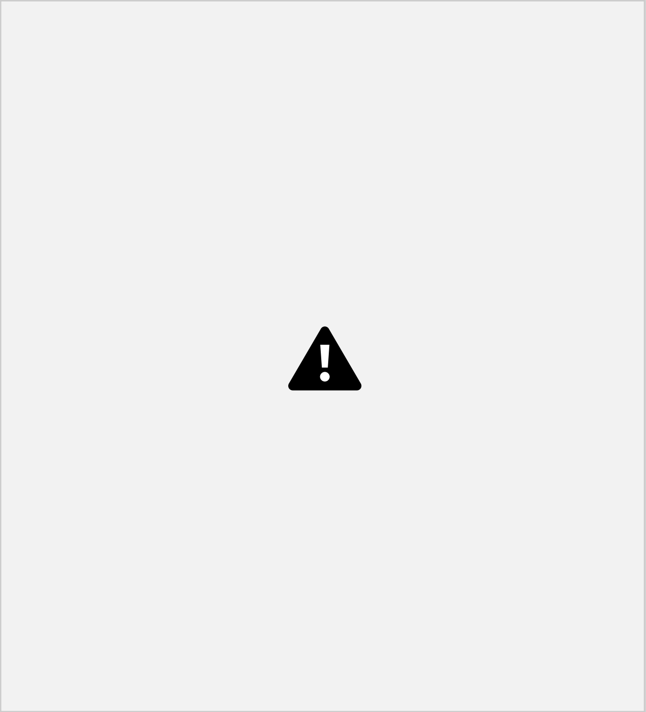
Priority Schedulingw/ Round-Robin

Process	Burst Time	Priority
P_1	4	3
P_2	5	2
P_3	8	2
P_4	7	1
P_5	3	3

- Run the process with the highest priority. Processes withthesamepriority run round-robin
- Gantt Chart with time quantum = 2

Multilevel Queue

- With priority scheduling, have separate queues for eachpriority.
- Schedule the process in the highest-priority queue!



Multilevel Queue

- Prioritization based upon process type

Multilevel FeedbackQueue

- A process can move between the various queues.
- Multilevel-feedback-queue scheduler defined by the followingparameters:
  - 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
- Aging can be implemented using multilevel feedback queue

Example of Multilevel FeedbackQueue

- Three queues:
  - Q_0 – RR with time quantum 8 milliseconds
  - Q_1 – RR time quantum 16 milliseconds
  - Q_2 – FCFS
- Scheduling
  - A new process enters queue Q_0 which is served in RR
    - When it gains CPU, the process receives 8 milliseconds
    - If it does not finish in 8 milliseconds, the

process is moved to queue Q_1

- At Q_1 job is again served in RR and receives 16 additional milliseconds
  - 4 If it still does not complete, it is preempted and moved to queue Q_2

Endof Chapter5a

Example of Multilevel FeedbackQueue

- Three queues:
  - Q_0 – RR with time quantum 8 milliseconds
  - Q_1 – RR time quantum 16 milliseconds
  - Q_2 – FCFS
- Scheduling
  - A new process enters queue Q_0 which is served RR
    - When it gains CPU, the process receives 8 milliseconds
    - If it does not finish in 8 milliseconds, it is moved to queue Q_1
  - At Q_1 the process is again served RR and receives 16 additional milliseconds
    - If it still does not complete, it is preempted and moved to queue Q_2