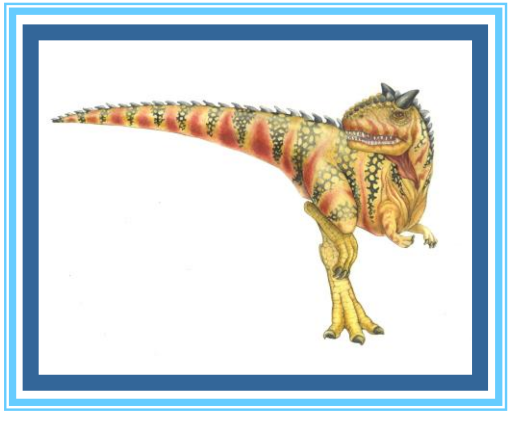
CPU Scheduling

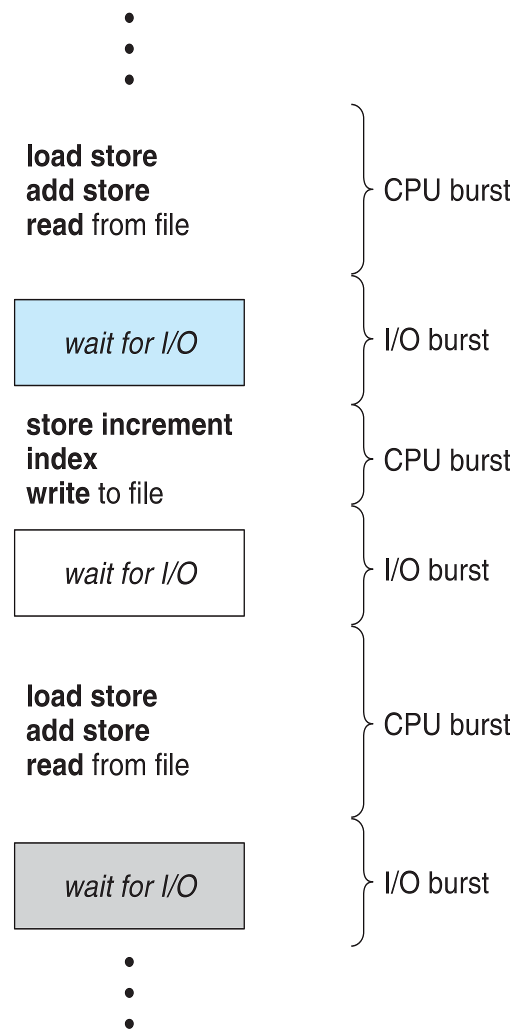


Objectives

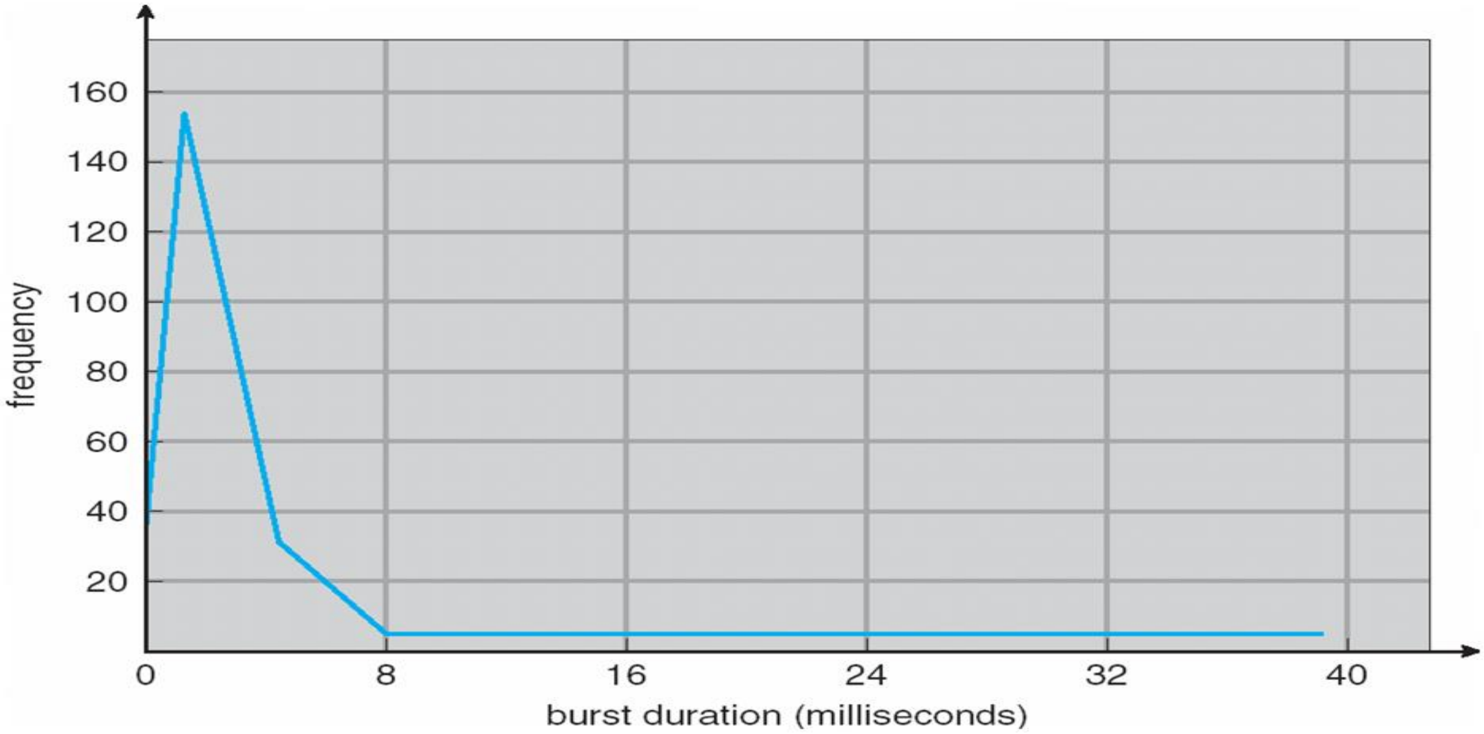
- To introduce CPU scheduling, which is the basis for multiprogrammed operating systems
    - To understand basic concepts, CPU / i/o burst cycle, cpu schedulers, dispatcher
- To discuss evaluation criteria for selecting a CPU-scheduling algorithm for a particular system
- To describe various CPU-scheduling algorithms

Basic Concepts

- The success of CPU scheduling depends on an observed property of processes:
- 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** followed by **I/O burst**
  - *Size of cpu and i/o burst may vary for different processes.*
  - *CPU burst distribution is of main concern*
  - *I/O burst size does not affect the cpu scheduling*



Histogram of CPU-burst Times



- ✓ The curve is generally characterized as exponential or hyperexponential
  - ➢ large number of short CPU bursts and
  - ➢ a small number of long CPU bursts
- ✓ An I/O-bound program typically has many short CPU bursts. A CPU-bound program might have a few long CPU bursts.

CPU Scheduler

- **Short-term scheduler** selects from among the processes in ready queue, and allocates the CPU to one of them
  - Queue may be ordered in various ways
- CPU scheduling decisions may take place when a process:
- 1. Switches from running to waiting state
    - for example, as the result of an I/O request or an invocation of wait() for the termination of a child process

  2. Switches from running to ready state
    - when an interrupt occurs

  3. Switches from waiting to ready
    - at completion of I/O

  4. Terminates

Non preemptive

- For situations 1 and 4 , no choice in terms of scheduling (a new process(if exists in ready queue) must be selected for execution) ----➔ **non preemptive/ cooperative**
- There is a choice for situations 2 and 3 ➔ **preemptive**

- Once the CPU has been allocated to a process, the process keeps the CPU until it releases the CPU
  - Either **by terminating**
  - Or **by switching to the waiting state.**
    - Ex:- used by Microsoft Windows 3.x

Problems of preemptive

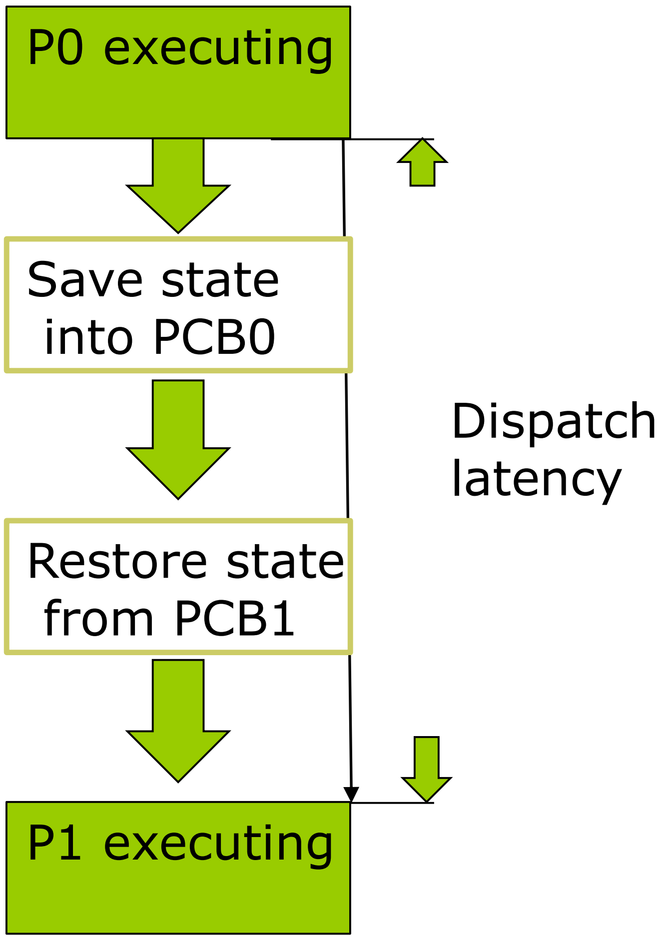
Preemptive scheduling can result in race condition when data are shared among several processes

- Consider access to shared data
- Consider preemption while in kernel mode
- Consider interrupts occurring during crucial OS activities
  - While one process is updating the data, it is preempted so that the second process can run. The second process then tries to read the data, which are in an inconsistent state

- Preemptive introduced by windows 95
- All modern OS(Windows, Mac OS X, Linux, Unix uses preemptive scheduling algorithms.

Dispatcher

- **Dispatcher module gives control of the CPU to the process selected by the short-term scheduler;**
- this function involves:
  - switching context
  - switching to user mode
  - jumping to the proper location in the user program to restart that program
- The dispatcher should be as fast as possible, since it is invoked during every process switch
- **Dispatch latency** – time it takes for the dispatcher to stop one process and start another running



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
- **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)
- **TAT = Exit Time - arrival Time**

  = Burst Time + Waiting Time

Scheduling Algorithm Optimization Criteria

- Max CPU utilization
- Max throughput
- Min turnaround time
- Min waiting time
- Min response time

Scheduling Algorithms

- CPU scheduling deals with the problem of deciding which of the processes in the ready queue is to be allocated the CPU.

First- Come, First-Served (FCFS) Scheduling

- **It is a non preemptive policy**

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:

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

First- Come, First-Served (FCFS) Scheduling

- **It is a non preemptive policy**

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:

- TAT for $\boldsymbol{P_1}$ **= 24;** $\boldsymbol{P_2}$ **= 27;** $\boldsymbol{P_3}$ **= 30**
- **Average TAT: (24+27+30)/3**

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: $(6 + 0 + 3)/3 = 3$
- Much better than previous case
- **Convoy effect** - short process behind long process
  - Consider one CPU-bound and many I/O-bound processes

FCFS

PID	A	B	C	D	E
AT	3	5	0	5	4
BT	4	3	2	1	3

GANTT CHART

C	Idle time	A	E	B	D	
0	2	3	7	10	13	14

PID	A	B	C	D	E
AT	3	5	0	5	4
BT	4	3	2	1	3
WT	0	5	0	8	3
TAT	4	8	2	9	6

Shortest-Job-First (SJF) Scheduling

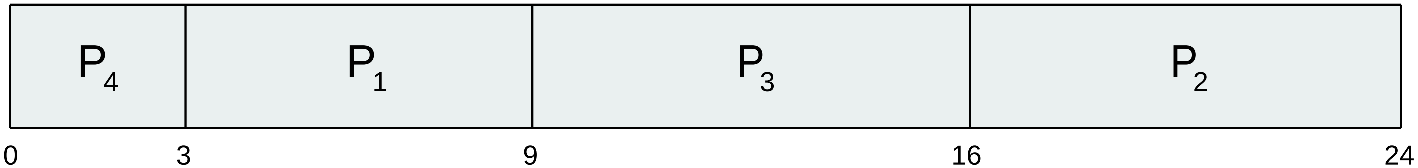
- Associate with each process the length of its next CPU burst
  - Use these lengths to schedule the process with the shortest time

- **SJF is optimal – gives minimum average waiting time for a given set of processes**
  - The difficulty is knowing the length of the next CPU request

Example of SJF(non preemptive)

PID	BT
P1	6
P2	8
P3	7
P4	3

- SJF scheduling chart



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

Determining Length of Next CPU Burst

- Preemptive version called

shortest-remaining-time-first

Example of Shortest-remaining-time-first

- Now we add the concepts of varying arrival times and preemption to the 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 =

Example of Shortest-remaining-time-first

- Now we add the concepts of varying arrival times and preemption to the 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

P_1	P_2	P_4	P_1	P_3	
0	1	5	10	17	26

- Average waiting time = [(10-0-1)+(1-1-0)+(17-2-0)+5-3-0)]/4 = 26/4 = 6.5 msec
- ***WT=FinalCUPALLOCTIME-ARRIVALTIME-PREVIOUSEXECUTIONTIME***

SJF(preemptive(SRTF))

PID	P1	P2	P3	P4
AT	0	2	4	5
BT	7	4	1	4

P1	P2	P3	P2	P4	P1

0 2 4 5 7 11 16

PID	P1	P2	P3	P4
AT	0	2	4	5
BT	7	4	1	4
WT	11-0-2=9	5-2-2=1	4-4-0=0	7-5-0=2
TAT	9+7=16	1+4=5	0+1=1	2+4=6

Practice Exercises

Suppose that the following processes arrive for execution at the times indicated. Each process will run for the amount of time listed. In answering the questions, use nonpreemptive scheduling, and base all decisions on the information you have at the time the decision must be made.

Process	Arrival Time	Burst Time
P_1	0.0	8
P_2	0.4	4
P_3	1.0	1

a. What is the average turnaround time for these processes with the FCFS scheduling algorithm?

b. What is the average turnaround time for these processes with the SJF scheduling algorithm?

c. The SJF algorithm is supposed to improve performance, but notice that we chose to run process P_1 at time 0 because we did not know that two shorter processes would arrive soon. Compute what the average turnaround time will be if the CPU is left idle for the first 1 unit and then SJF scheduling is used. Remember that processes P_1 and P_2 are waiting during this idle time, so their waiting time may increase. This algorithm could be called future-knowledge scheduling.

Practice Exercises

Process	Arrival Time	Burst Time
P_1	0.0	8
P_2	0.4	4
P_3	1.0	1

What is the average turnaround time for these processes with the FCFS scheduling algorithm?

FCFS

Practice Exercises

Process	Arrival Time	Burst Time
P_1	0.0	8
P_2	0.4	4
P_3	1.0	1

What is the average turnaround time for these processes with the SJF scheduling algorithm?

SJF

Process	Arrival Time	Burst Time
P_1	0.0	8
P_2	0.4	4
P_3	1.0	1

The SJF algorithm is supposed to improve performance, but notice that we chose to run process P_1 at time 0 because we did not know that two shorter processes would arrive soon. Compute what the average turnaround time will be if the CPU is left idle for the first 1 unit and then SJF scheduling is used. Remember that processes P_1 and P_2 are waiting during this idle time, so their waiting time may increase. This algorithm could be called future-knowledge scheduling.

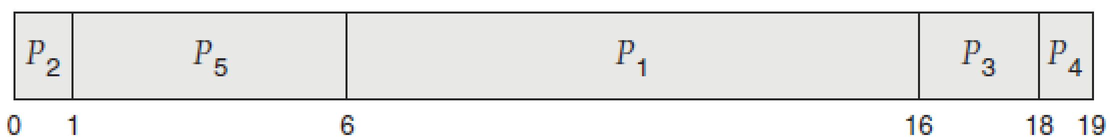
Priority Scheduling

- 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 priority scheduling where priority is the inverse of predicted next CPU burst time
- Problem ≡ **Starvation** – low priority processes may never execute
- Solution ≡ **Aging** – as time progresses increase the priority of the process

Example of Priority Scheduling

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 msec

Thread	Priority	Burst	Arrival
P_1	40	20	0
P_2	30	25	25
P_3	30	25	30
P_4	35	15	60
P_5	5	10	100
P_6	10	10	105

Thread	Priority	Burst	Arrival
P_1	40	20	0
P_2	30	25	25
P_3	30	25	30
P_4	35	15	60
P_5	5	10	100
P_6	10	10	105

PID	P1	P2	P3	P4	P5
BT	10	1	2	1	5
P*	3	1	4	5	2

PID	P1	P2	P3	P4	P5
BT	10	1	2	1	5
P*	3	1	4	5	2

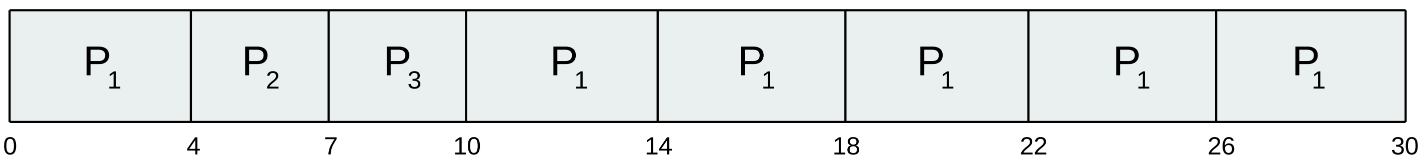
Round Robin (RR)

- Each process gets a small unit of CPU time (**time quantum** *q*), 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.
- 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 RR with Time Quantum = 4

Process	Burst Time
P_1	24
P_2	3
P_3	3

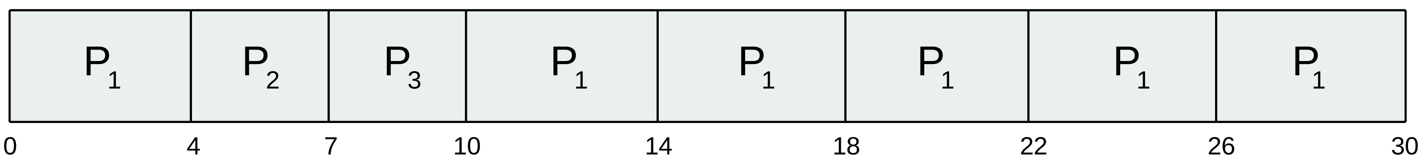
- The Gantt chart is:



Example of RR with Time Quantum = 4

Process	Burst Time
P_1	24
P_2	3
P_3	3

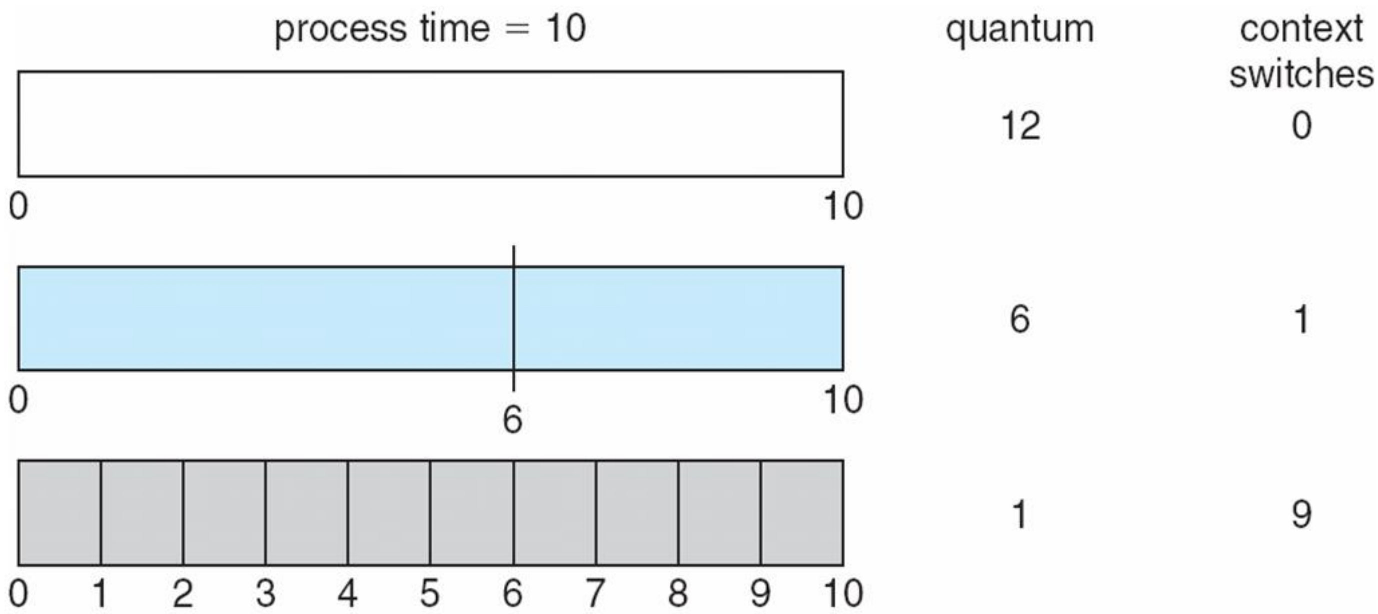
- The Gantt chart is:



- Typically, higher average turnaround than SJF, but better ***response***
- q should be large compared to context switch time
- q usually 10ms to 100ms, context switch < 10 usec

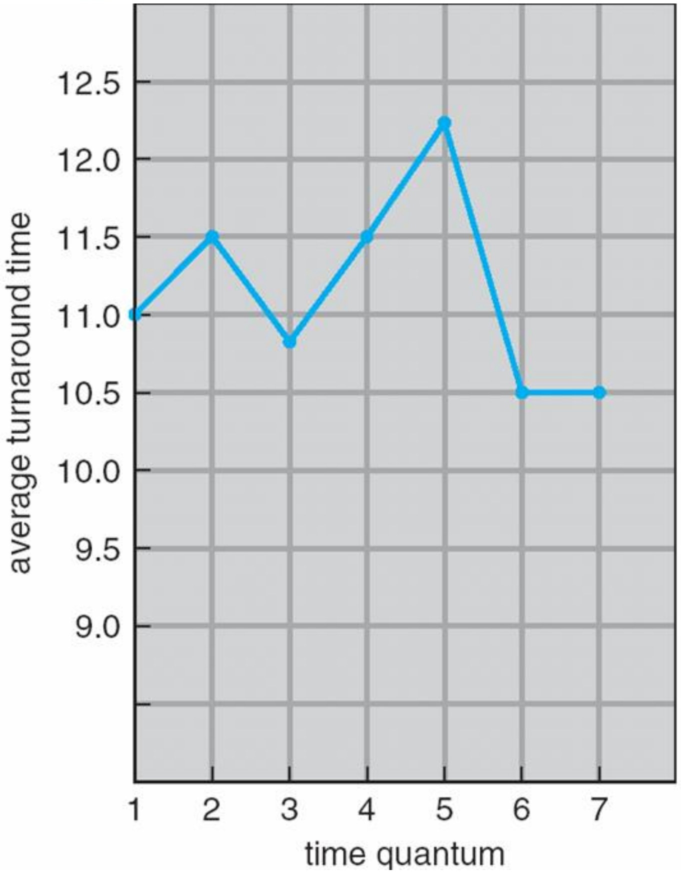
PID	P1	P2	P3	P4	P5
BT	2	1	8	4	5
P*	2	1	4	2	3

Time Quantum and Context Switch Time



- ✓ Thus, we want the time quantum to be large with respect to the context switch time.
- ✓ If the context-switch time is approximately 10 percent of the time quantum, then about 10 percent of the CPU time will be spent in context switching.
- ✓ In practice, most modern systems have time quanta ranging from 10 to 100 milliseconds. The time required for a context switch is typically less than 10 microseconds; thus, the context-switch time is a small fraction of the time quantum.

Turnaround Time Varies With The Time Quantum



process	time
P_1	6
P_2	3
P_3	1
P_4	7

80% of CPU bursts
should be shorter than q

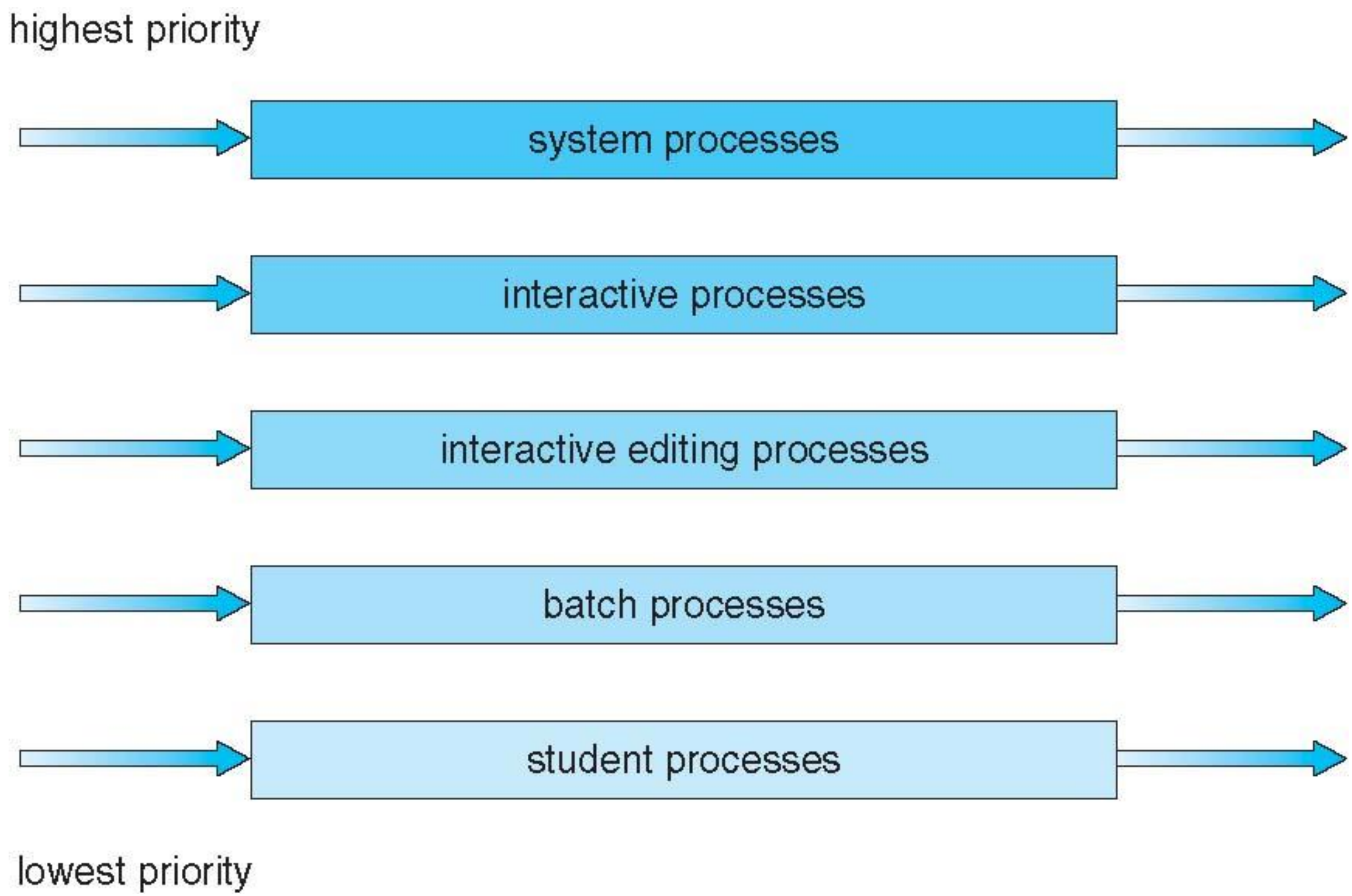
Multilevel Queue Scheduling

Another class of scheduling algorithms has been created for situations in which processes are easily classified into different groups.

Multilevel Queue Scheduling

- Ready queue is partitioned into separate queues, eg:
  - **foreground** (interactive)
  - **background** (batch)
- Process permanently in a given queue
- 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 and 20% to background in FCFS

Multilevel Queue Scheduling



Multilevel Queue Scheduling

Consider below data of four processes under multilevel queue scheduling, Q No. denotes the queue of the process

PID	Arrival Time	Burst Time	Q No.
P1	0	4	1
P2	0	3	1
P3	0	8	2
P4	10	5	1

Priority of queue 1 is greater than queue 2. Queue 1 uses RR (TQ=2) and queue 2 uses FCFS.

Draw the Gantt chart for above data and Calculate AWT, TAT

P1	P2	P1	P2	P3	P4	P3	
0	2	4	6	7	10	15	20

Multilevel Queue Scheduling

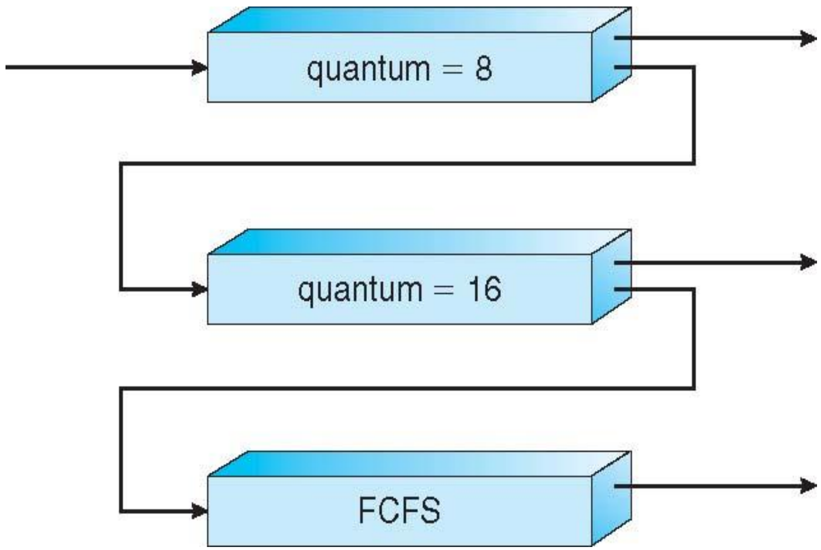
P1	P2	P1	P2	P3	P4	P3	
0	2	4	6	7	10	15	20

Multilevel Feedback Queue

- 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

- Three queues:
  - Q_0 – RR with time quantum 8 milliseconds
  - Q_1 – RR time quantum 16 milliseconds
  - Q_2 – FCFS



- Scheduling
  - A new job enters queue Q_0 which is served FCFS
    - 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 FCFS and receives 16 additional milliseconds
    - If it still does not complete, it is preempted and moved to queue Q_2

Example

- Consider the following set of processes with the length of the CPU burst time given in millisecond
- The processes are assumed to have arrived in the order of P1,P2,P3,P4, P5, all at time 0.
- Draw four Gantt chart illustrating the executing of these process using FCFS, SJF, a non-preemptive priority (a smaller priority number implies a higher priority) and RR (q=1) scheduling.

PID	P1	P2	P3	P4	P5
BT	10	1	2	1	5
Priority	3	1	3	4	2

Algo.	FCFS	SJF	PRIORITY	RR
AWT				
ATT				

FCFS

PID	P1	P2	P3	P4	P5
BT	10	1	2	1	5
Priority	3	1	3	4	2

SJF

PID	P1	P2	P3	P4	P5
BT	10	1	2	1	5
Priority	3	1	3	4	2

NP Priority

PID	P1	P2	P3	P4	P5
BT	10	1	2	1	5
Priority	3	1	3	4	2

RR, q=1

PID	P1	P2	P3	P4	P5
BT	10	1	2	1	5
Priority	3	1	3	4	2

Scheduling Algorithm Optimization Criteria

- Max CPU utilization
- Max throughput
- Min turnaround time
- Min waiting time
- Min response time

Algorithm Evaluation

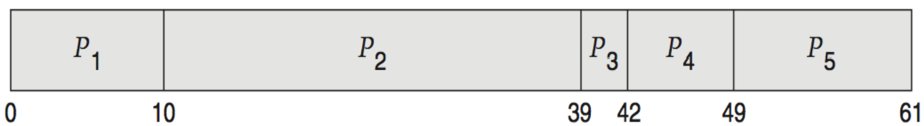
- How to select CPU-scheduling algorithm for an OS?
- Determine criteria, then evaluate algorithms

- **Deterministic modeling**
  - Type of **analytic evaluation**
  - Takes a particular predetermined workload and defines the performance of each algorithm for that workload
- Consider 5 processes arriving at time 0:

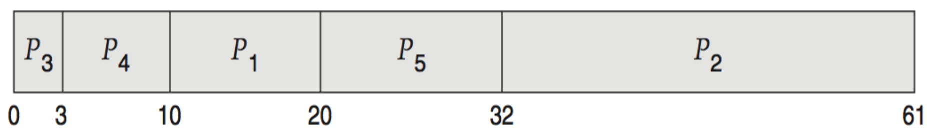
Process	Burst Time
P_1	10
P_2	29
P_3	3
P_4	7
P_5	12

Deterministic Evaluation

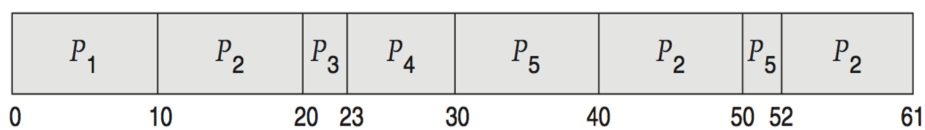
- For each algorithm, calculate **minimum average waiting time**
- Simple and fast, but requires exact numbers for input, applies only to those inputs
  - FCS is 28ms:



  - Non-preemptive SFJ is 13ms:



  - RR is 23ms:



Queueing Models

- Describes the arrival of processes, and CPU and I/O bursts probabilistically
  - Commonly exponential, and described by mean
  - Computes average throughput, utilization, waiting time, etc
- Computer system described as network of servers, each with queue of waiting processes
  - Knowing arrival rates and service rates
  - Computes utilization, average queue length, average wait time, etc

Little's Formula

- n = average queue length
- W = average waiting time in queue
- λ = average arrival rate into queue
- Little's law – in steady state, processes leaving queue must equal processes arriving, thus:

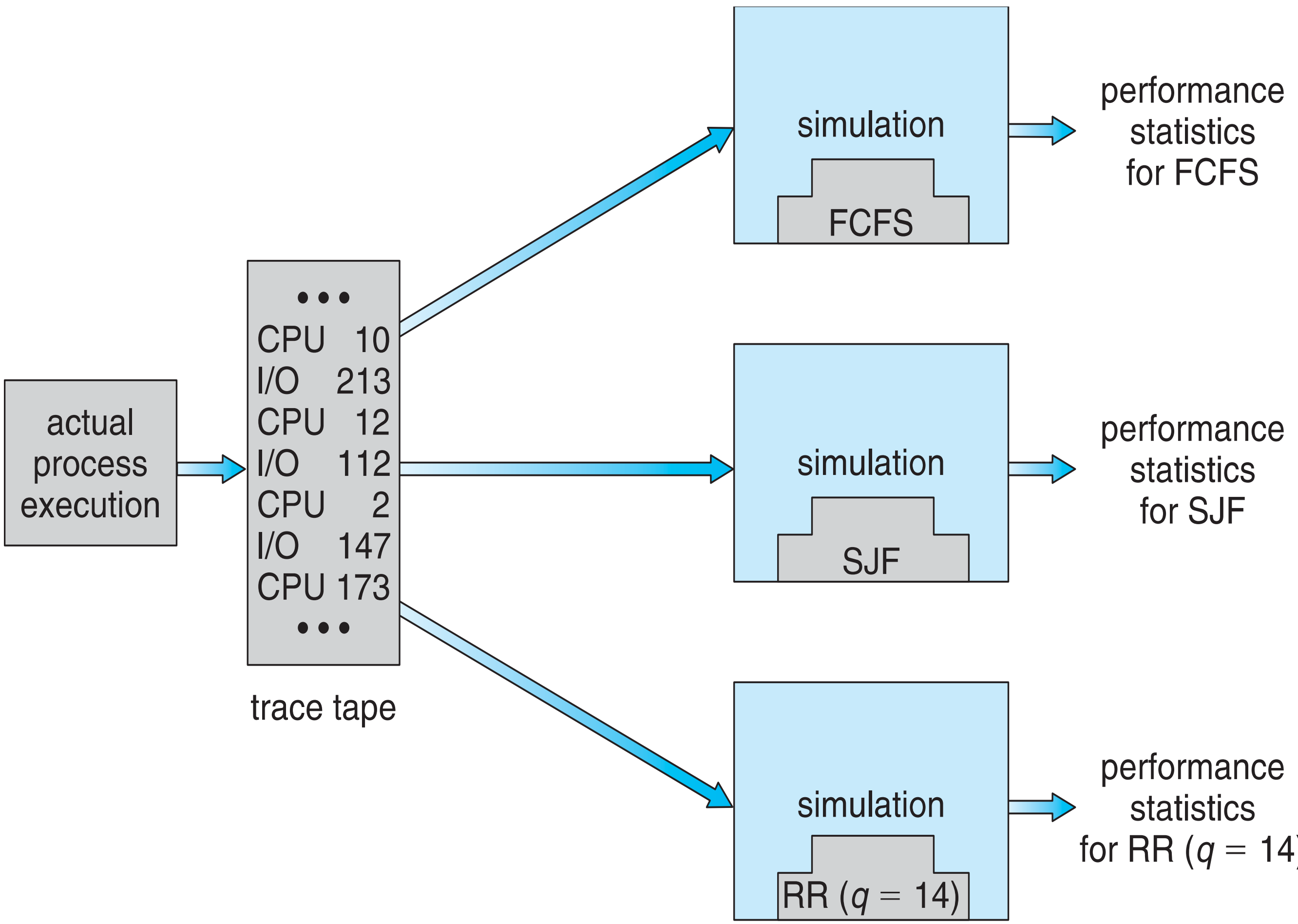
  $n = \lambda \times W$

  - Valid for any scheduling algorithm and arrival distribution
- For example, if on average 7 processes arrive per second, and normally 14 processes in queue, then average wait time per process = 2 seconds

Simulations

- Queueing models limited
- **Simulations** more accurate
  - Programmed model of computer system
  - Clock is a variable
  - Gather statistics indicating algorithm performance
  - Data to drive simulation gathered via
    - Random number generator according to probabilities
    - Distributions defined mathematically or empirically
    - Trace tapes record sequences of real events in real systems

Evaluation of CPU Schedulers by Simulation



Implementation

- Even simulations have limited accuracy
- Just implement new scheduler and test in real systems
  - High cost, high risk
  - Environments vary
- Most flexible schedulers can be modified per-site or per-system
- Or APIs to modify priorities
- But again environments vary

Consider the following set of processes, with the length of the CPU burst given in milliseconds:

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

The processes are assumed to have arrived in the order P_1, P_2, P_3, P_4, P_5, all at time 0.

a. Draw four Gantt charts that illustrate the execution of these processes using the following scheduling algorithms: FCFS, SJF, nonpreemptive priority (a larger priority number implies a higher priority), and RR (quantum = 2).

b. What is the turnaround time of each process for each of the scheduling algorithms in part a?

c. What is the waiting time of each process for each of these scheduling algorithms?

d. Which of the algorithms results in the minimum average waiting time (over all processes)?

Exercises

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

Exercises

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

Exercises

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

Exercises

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

The following processes are being scheduled using a preemptive, roundrobin scheduling algorithm. Each process is assigned a numerical priority, with a higher number indicating a higher relative priority. In addition to the processes listed below, the system also has an ***idle task*** (which consumes no CPU resources and is identified as *Pidle*). This task has priority 0 and is scheduled whenever the system has no other available processes to run. The length of a time quantum is 10 units. If a process is preempted by a higher-priority process, the preempted process is placed at the end of the queue.

Thread	Priority	Burst	Arrival
P_1	40	20	0
P_2	30	25	25
P_3	30	25	30
P_4	35	15	60
P_5	5	10	100
P_6	10	10	105

a. Show the scheduling order of the processes using a Gantt chart.
b. What is the turnaround time for each process?
c. What is the waiting time for each process?

Summary

End of Chapter 6