Operating Systems

CMP-320

Instructor: Muhammad Adeel Nisar

Lecture- CPU Scheduling

Acknowledgement

Some slides and pictures are adapted from Lecture slides / Books of

- Dr. Syed Mansoor Sarwar.
- Mr. M. Arif Butt
- Text Book - OS Concepts by Silberschatz, Galvin, and Gagne.

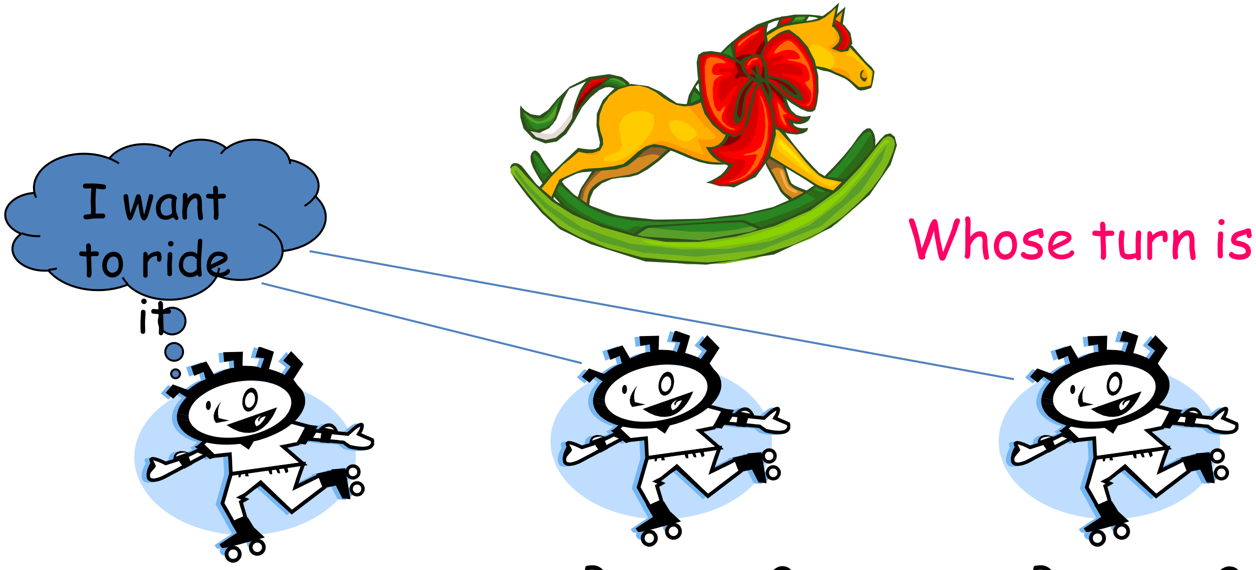
Today’s Agenda

- Review of previous Lecture
- What are CPU and IO bursts
- CPU Scheduling and Scheduling Criteria
- Scheduling Algorithms
  - FCFS
  - SJF & SRTF
  - Priority Scheduling
  - RR
  - VRR
  - MQ &MFQ
- Evaluation/Comparison of Scheduling Algorithms

Scheduling

- Deciding which process/thread should occupy a resource (CPU, disk, etc.)

(CPU (horsepower))



I want to ride it

Whose turn is it?

Process 1

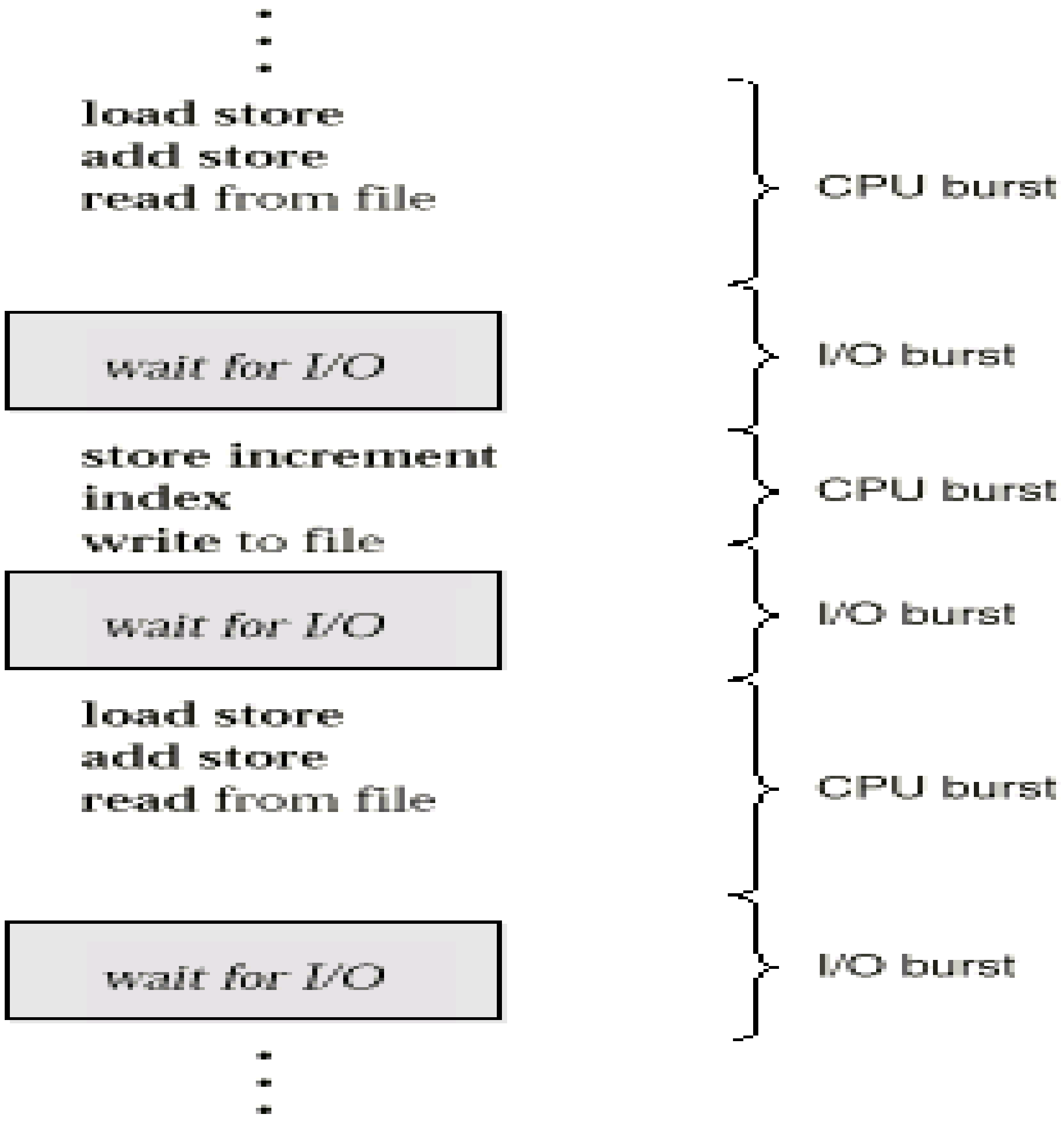
Process 2

Process 3

CPU – I/O Burst Cycle

- Process execution consists of a cycle of CPU execution and I/O wait
- Processes move back & forth between these two states
- A process execution begins with a CPU burst, followed by an I/O burst and then another CPU burst and so on
- An alternating sequence of CPU & I/O bursts is shown on next slide

Alternating Sequence of CPU and I/O Bursts



CPU Scheduler

- When CPU becomes idle the OS must select one of the processes in the Ready Queue to be executed
- **CPU Scheduler** selects a process from the processes in memory that are ready to execute (from Ready Q), and allocates the CPU to one of them
- The OS code that implements the CPU scheduling algorithm is known as CPU scheduler
- CPU scheduling decisions may take place when a process:
  1. Switches from running to waiting state
  2. Terminates
  3. Switches from running to ready state. (e.g. when time slice of a process expires or an interrupt occurs)
  4. Switches from waiting to ready state. (e.g. on completion of I/O)
  5. On arrival of a new process

Preemptive vs Non Preemptive

In **Pre-emptive scheduling** the currently running process may be interrupted and moved to the ready state by OS (forcefully)

In **Non-preemptive Scheduling**, the running process can only lose the processor voluntarily by terminating or by requesting an I/O. OR, Once CPU given to a process it cannot be preempted until the process completes its CPU burst

Dispatcher

- Dispatcher is an important component involved in CPU scheduling
- The OS code that takes the CPU away from the current process and hands it over to the newly scheduled process is known as the dispatcher
- Dispatcher gives control of the CPU to the process selected by the short-term scheduler
- Dispatcher performs following functions:
  - 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 executing another

Scheduling Criteria

- **CPU utilization** – keep the CPU as busy as possible
- **Throughput** – # of processes that complete their execution per time unit
- **Waiting time** – amount of time a process has been waiting in the ready queue
- **Turnaround time** – amount of time to execute a particular process.
  - (FinishTime – Arrival Time)
- **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

Single CPU–Scheduling Algorithms

- Batch systems
  - First Come First Serve (FCFS)
  - Shortest Job First
- Interactive Systems
  - Round Robin
  - Priority Scheduling
  - Multi Queue & Multi-level Feedback

First Come First Serve

- ***Process that requests the CPU FIRST is allocated the CPU FIRST***
- Called "FIFO", **Non-preemptive**, Used in Batch Systems
- Real life analogy: Any ticket counter

- Implementation: FIFO queues
  - A new process enters the **tail** of the queue
  - The scheduler selects from the **head** of the queue.
- Performance Metric: **Average Waiting Time** (AWT)
- Given Parameters:
  - Burst Time (in ms), Arrival Time and Order
  - Can be generalized to processes with alternate CPU and I/O bursts: blocking process goes to queue's tail

First Come First Serve (cont…)

- Simplest CPU scheduling algorithm
- Non preemptive
- The process that requests the CPU first is allocated the CPU first
- Implemented with a FIFO queue. When a process enters the Ready Queue, its PCB is linked on to the tail of the Queue. When the CPU is free it is allocated to the process at the head of the Queue
- **Limitations**
  - FCFS favor long processes as compared to short ones. (Convoy effect)
  - FCFS tends to favor processor bound processes over I/O bound processes
  - Average waiting time is often quite long
  - FCFS is non-preemptive, so it is trouble some for time sharing systems

Convoy Effect

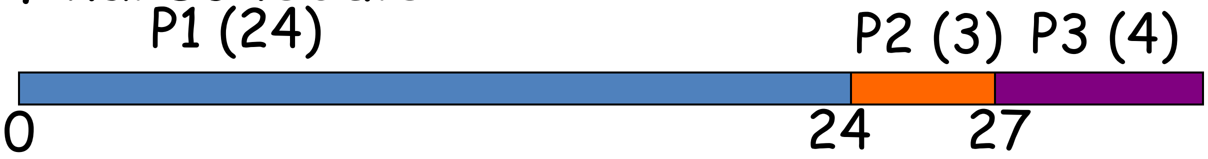
- Consider n-1 jobs in system that are I/O bound and 1 job that is CPU bound
- I/O bound jobs pass quickly through the ready queue and suspend themselves waiting for I/O
- CPU bound job arrives at head of queue and executes until complete
- I/O bound jobs rejoin ready queue and wait for CPU bound job to complete
- I/O devices idle until CPU bound job completes
- When CPU bound job completes, other processes rush to wait on I/O again
- CPU becomes idle

“A convoy effect happens when a set of processes need to use a resource for a short time, and one process holds the resource for a long time, blocking all of the other processes. Causes poor utilization of the other resources in the system”

FCFS – Example 1

Process	Duration/B.T	Order	Arrival Time
P1	24	1	0
P2	3	2	3
P3	4	3	4

The final schedule:



P1 (24)

P2 (3) P3 (4)

0

24

27

P1 waiting time: 0-0
P2 waiting time: 24-3
P3 waiting time: 27-4

The average waiting time:
(0+21+23)/3 = 14.667

FCFS – Example 2

- Draw the graph (Gantt chart) and compute average waiting time for the following processes using **FCFS** Scheduling algorithm.

Process	**Arrival time**	**Burst Time**
P1	1	16
P2	5	3
P3	6	4
P4	9	2

SJF & SRTF Scheduling…

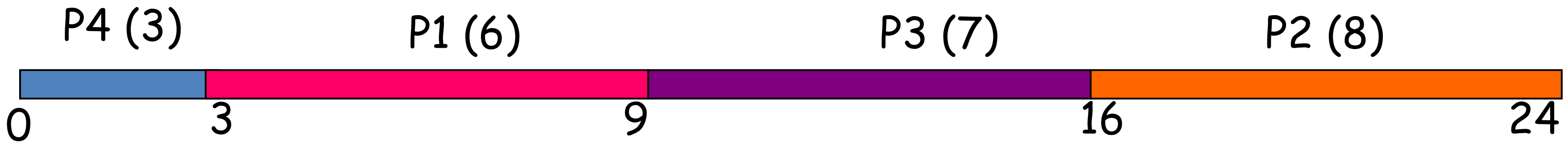
- When the CPU is available it is assigned to the process that has the smallest next CPU burst
- If two processes have the same length next CPU bursts, FCFS scheduling is used to break the tie

Comes in two flavors

- **Shortest Job First (SJF)**
  - It's a non preemptive algorithm. When a new process arrives having a shorter next CPU burst than what is left of the currently executing process, it ***allows*** the currently running process to finish its CPU burst
- **Shortest Remaining Time First (SRTF)**
  - It's a Preemptive algorithm. When a new process arrives having a shorter next CPU burst than what is left of the currently executing process, it ***preempts*** the currently running process

SJF Example 3

Process	Duration/B.T	Order	Arrival Time
P1	6	1	0
P2	8	2	0
P3	7	3	0
P4	3	4	0

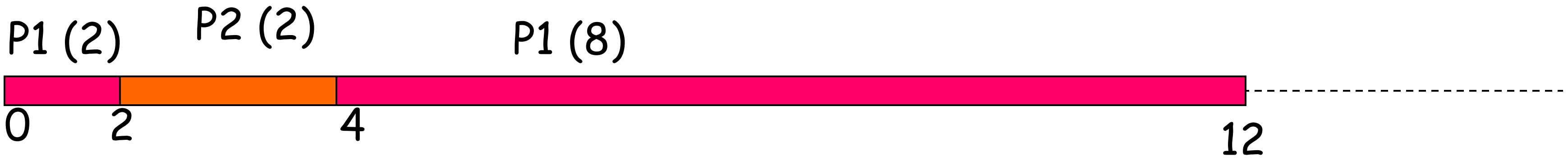


P4 waiting time: 0-0
P1 waiting time: 3-0
P3 waiting time: 9-0
P2 waiting time: 16-0

The total running time is: 24
The average waiting time (AWT):
(0+3+9+16)/4 = 7 time units

SRTF Example 4

Process	Duration	Order	Arrival Time
P1	10	1	0
P2	2	2	2



P1 waiting time: 4-2 =2
P2 waiting time: 0

The average waiting time (AWT):
(0+2)/2 = 1

Now run this using SJF!

SJF & SRTF – Example 5

Draw the graph (Gantt chart) and compute waiting time and turn around time for the following processes using **SJF** & **SRTF** Scheduling algorithm. For SJF consider no arrival time and consider all processes arrive at time 0 in sequence P1, P2, P3, P4.

Process	Arrival Time	Burst Time
P1	0	8
P2	1	4
P3	2	9
P4	3	5

NOTE: For pre emptive algos, we calculate Turn Around Time. Whenever a process enters Ready Queue more than once we use turn around time instead of waiting time.

- ❑ For non-emptive algo: Waiting time = Start time – Arrival time
- ❑ For preemptive algo: Waiting time = Finish time – Burst time – ArrivalTime
- ❑ Turn around time = Finish time – Arrival time

SJF & SRTF – Example 6

Draw the graph (Gantt chart) and compute waiting time and turn around time for the following processes using **SJF** & **SRTF** Scheduling algorithm.

Process	**Arrival Time**	**Burst Time**
P1	0	5
P2	1	2
P3	2	3
P4	3	1

SJF & SRTF – Example 7

Draw the graph (Gantt chart) and compute waiting time and turn around time for the following processes using **SJF** & **SRTF** Scheduling algorithm.

Process	**Arrival Time**	**Burst Time**
P1	0	9
P2	3	6
P3	6	2
P4	9	1

SJF & SRTF Scheduling

$100 QUESTION

How to compute the next CPU burst?

- This algorithm cannot be implemented at the level of short-term CPU scheduling. There is no way to know the length of the next CPU burst
- One approach is to try to approximate. We may not *know* the length of the next CPU burst, but we may be able to predict its value. We expect that the next CPU burst will be similar in length to the previous ones. Thus, by computing an approximation of the length of the next CPU burst, we can pick the process with the shortest predicted CPU burst

SJF & SRTF Scheduling…

Exponential Averaging

Estimation based on historical data

t_n = Actual length of n^{th} CPU burst

τ_n = Estimate for n^{th} CPU burst

τ_{n+1} = Estimate for $n+1^{st}$ CPU burst

$\alpha, 0 \leq \alpha \leq 1$

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

Exponential Averaging…

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

- Plugging in value for τ_n, we get

$\tau_{n+1} \quad = \alpha t_n + (1- \alpha)[\alpha t_{n-1} + (1- \alpha)\tau_{n-1}]$

$\tau_{n+1} \quad = \alpha t_n + (1- \alpha)\alpha t_{n-1} + (1- \alpha)^2\tau_{n-1}$

- Again plugging in value for τ_{n-1}, we get

$\tau_{n+1} \quad = \alpha t_n + (1- \alpha)\alpha t_{n-1} + (1- \alpha)^2[\alpha t_{n-2} + (1- \alpha)\tau_{n-2}]$

$\tau_{n+1} \quad = \alpha t_n + (1- \alpha)\alpha t_{n-1} + (1- \alpha)^2\ \alpha t_{n-2}\ (1- \alpha)^3\tau_{n-2}$

- Continuing like this results in

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

Exponential Averaging…

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

Lets take two extreme values of α

If α = 0

$$\tau_{n+1} = \tau_n$$

Next CPU burst estimate will be exactly equal to previous CPU burst estimate.

If α = 1

$$\tau_{n+1} = t_n$$

Next CPU burst estimate will be equal to previous actual CPU burst.

Exponential Averaging…

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

Typical value used for α is ½. With this value, our (n+1)st estimate is

$$\tau_{n+1} = t_n/2 + t_{n-1}/2^2 + t_{n-2}/2^3 + t_{n-3}/2^4 + \ldots$$

Note that as we move back in time we are giving less weight-age to the previous CPU bursts. Older histories are given exponentially less weight-age

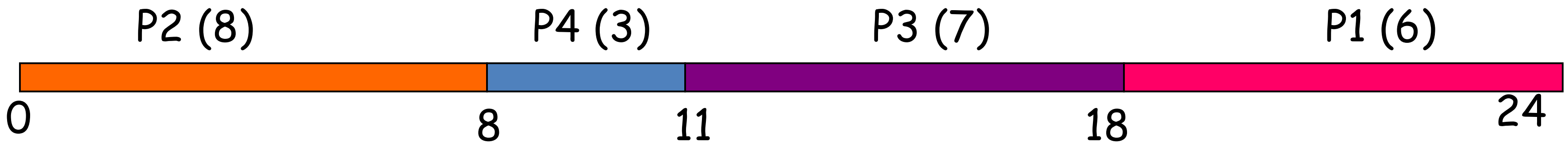
Priority Scheduling

- A priority number (integer) is associated with each process and the CPU is allocated to the process with the highest priority (smallest integer ≡ highest priority)
- A priority scheduling algorithm can be preemptive or non preemptive
  - A **Preemptive** priority scheduling algorithm will preempt the CPU if the priority of the newly arrived process is higher than the priority of the currently running process. E.g. SRTF is a priority scheduling algorithm where priority is the predicted next CPU burst time
  - A **Non preemptive** priority scheduling algorithm will simply put the new process at the tail of the Ready Queue
- Problem ≡ Starvation / Indefinite Blocking; i.e. Low priority processes may never execute
- Solution ≡ Aging; It is a technique of gradually increasing the priority of processes that wait in the system for long time

Priority Scheduling – Example 7

Lower priority # == More important

Process	Duration	Priority #	Arrival Time
P1	6	4	0
P2	8	1	0
P3	7	3	0
P4	3	2	0



P2 waiting time: 0
P4 waiting time: 8
P3 waiting time: 11
P1 waiting time: 18

The average waiting time (AWT):
(0+8+11+18)/4 = 9.25
(worse than SJF's)

Round Robin (RR) Scheduling

- RR is preemptive and designed especially for time sharing systems.
- A clock interrupt is generated at periodic intervals called time quantum or time slice.
- To implement RR scheduling, we keep the ready queue as a FIFO queue of processes. New processes are added to the tail of the ready queue. The CPU scheduler picks the first process, sets a timer to interrupt after 1 time quantum, and dispatches the process.
- One of two things will then happen. The process may have a CPU burst of less than 1 time quantum. In this case, the process itself will release the CPU voluntarily. The scheduler will then proceed to the next process in the ready queue. Otherwise, if the CPU burst of the currently running process is greater than 1 time quantum; the timer will go off and will cause an interrupt to the operating system. A context switch will be executed, and the process will be put at the tail of the ready queue. The CPU scheduler will then select the next process from the ready queue.

Round Robin (RR) Scheduling…

- The performance of the RR algorithm depends heavily on the size of the time quantum. Principal design issue in the length of the time quantum to be used are:
  - If the time quantum is very large (larger than the largest CPU burst of any process), the RR policy become the FCFS policy.
  - If the time quantum is very short, the RR approach is called processor sharing. Short processes will move through the system relatively quickly.

Round Robin (RR) Scheduling (cont..)

Limitation

- Processor bound processes tend to receive an unfair portion of processor time, which result in poor performance of I/O bound processes.

Effect of context switching on the performance of RR scheduling

- Let us assume that we have only 1 process of 10 time units. If the quantum is 12 time units, the process finishes in less than 1 time quantum, with no overhead. If the quantum is 6 time units, however, the process will require 2 quanta, resulting in a context switch. If the time quantum is 1 time unit, then 9 context switches will occur, slowing the execution of the process accordingly.
- Thus, the time quantum should be large with respect to the context switch time. If the context switch time is approximately 5 percent of the time quantum, then about 5 percent of the CPU time will be spent in context switching.

RR – Example 8

Draw the graph (Gantt chart) and compute turn around time for the following processes using **RR** Scheduling algorithm. Consider a time slice of 4 sec.

Process	Arrival Time	Burst Time
P1	0	16
P2	1	3
P3	2	3

RR – Example 9

Draw the graph (Gantt chart) and compute turn around time for the following processes using **RR** Scheduling algorithm. Consider a time slice of 3 sec

Process	**Arrival Time**	**Burst Time**
P1	0	8
P2	2	4
P3	3	2
P4	6	5

NOTE: Priority of placing a process in Ready Queue

- New Process.
- Running Process.

RR with I/O – Example 10

Draw the graph (Gantt chart) and compute turn around time for the following processes using **RR** Scheduling algorithm. Consider a time slice of 3 sec. Every even number process perform I/O after every 2 sec of its running life. I/O takes 10 seconds.

Process	Arrival Time	Burst Time
P1	0	8
P2	2	3
P3	3	5
P4	6	4

NOTE: Priority of placing a process in Ready Queue

- New Process.
- Blocked Process or I/O process.
- Running Process.

RR with I/O – Example 11

Draw the graph (Gantt chart) and compute turn around time for the following processes using **RR** Scheduling algorithm. Consider a time slice of 3 sec. Every odd number process perform I/O after every 2 sec of its running life. I/O takes 10 seconds.

Process	**Arrival Time**	**Burst Time**
P1	0	8
P2	2	3
P3	3	5
P4	6	4

Virtual Round Robin Scheduling

- In VRR there is an FCFS auxiliary Queue to which the processes are moved after being released from an I/O wait.
- When a dispatching decision is to be made, processes in the Auxiliary Queue get preference over those in the Ready Queue.
- When a process is dispatched from the Auxiliary Queue it runs for the time quantum minus the total time spent running since it was last selected from the main Ready Queue.

VRR – Example 12

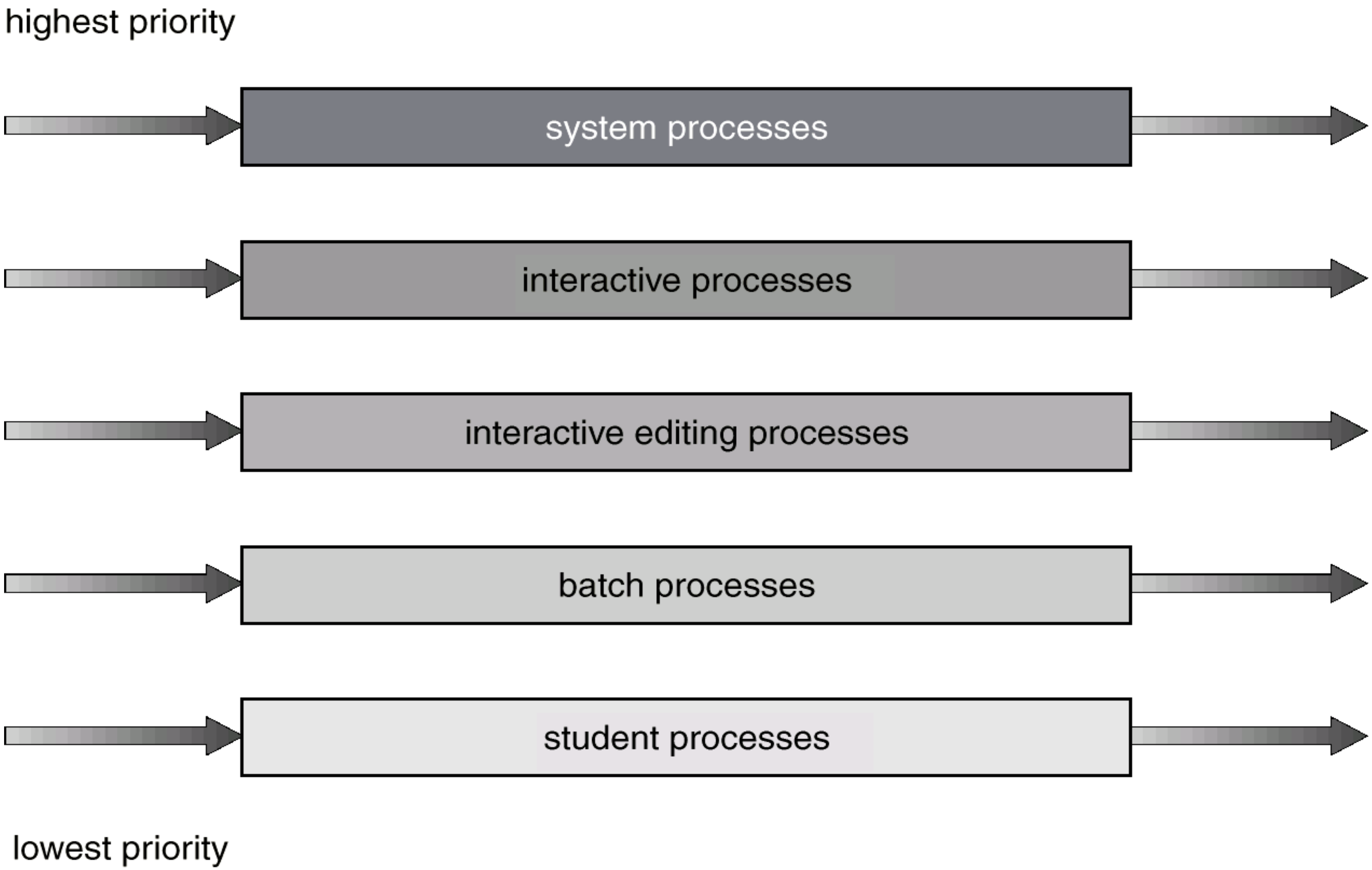
Draw the graph (Gantt chart) and compute turn around time for the following processes using **VRR** Scheduling algorithm. Consider a time slice of 3 sec. Every even number process perform I/O after every 2 sec of its running life. I/O takes 5 seconds.

Process	**Arrival Time**	**Burst Time**
P1	0	7
P2	2	4
P3	3	3
P4	5	5
P5	7	6

Multilevel Queue Scheduling

- A multilevel queue scheduling algorithm partitions the Ready Queue into several separate queues:
  - Foreground (interactive)
  - Background (batch)
- Processes are permanently assigned to a queue on entry to the system (based on some property of the process, e.g. memory size, process priority, process type).
- Processes do not move between queues.
- Each queue has its own scheduling algorithm,
  - Foreground – RR
  - Background – FCFS
- More over there must be scheduling between the queues. E.g. foreground queue may have absolute priority over back ground queue.

Multilevel Queue Scheduling



highest priority

system processes

interactive processes

interactive editing processes

batch processes

student processes

lowest priority

MQ Scheduling – Example 13

Draw the graph (Gantt chart) for the following processes using **MQ** Scheduling algorithm.

If (CPU time < 4) **then** Q1 (FCFS)

else **if** (4 <= CPU time < 7) **then** Q2 (FCFS)

else Q3 (FCFS)

Process	**Arrival Time**	**Burst Time**
P1	0	8
P2	2	2
P3	3	4
P4	5	6
P5	7	9
P6	9	3
P7	10	5

MQ Scheduling – Example 14

Draw the graph (Gantt chart) for the following processes using **MQ** Scheduling algorithm.

If (CPU time < 4) **then** Q1 (FCFS)

else **if** (4 <= CPU time < 7) **then** Q2 (RR – 3sec)

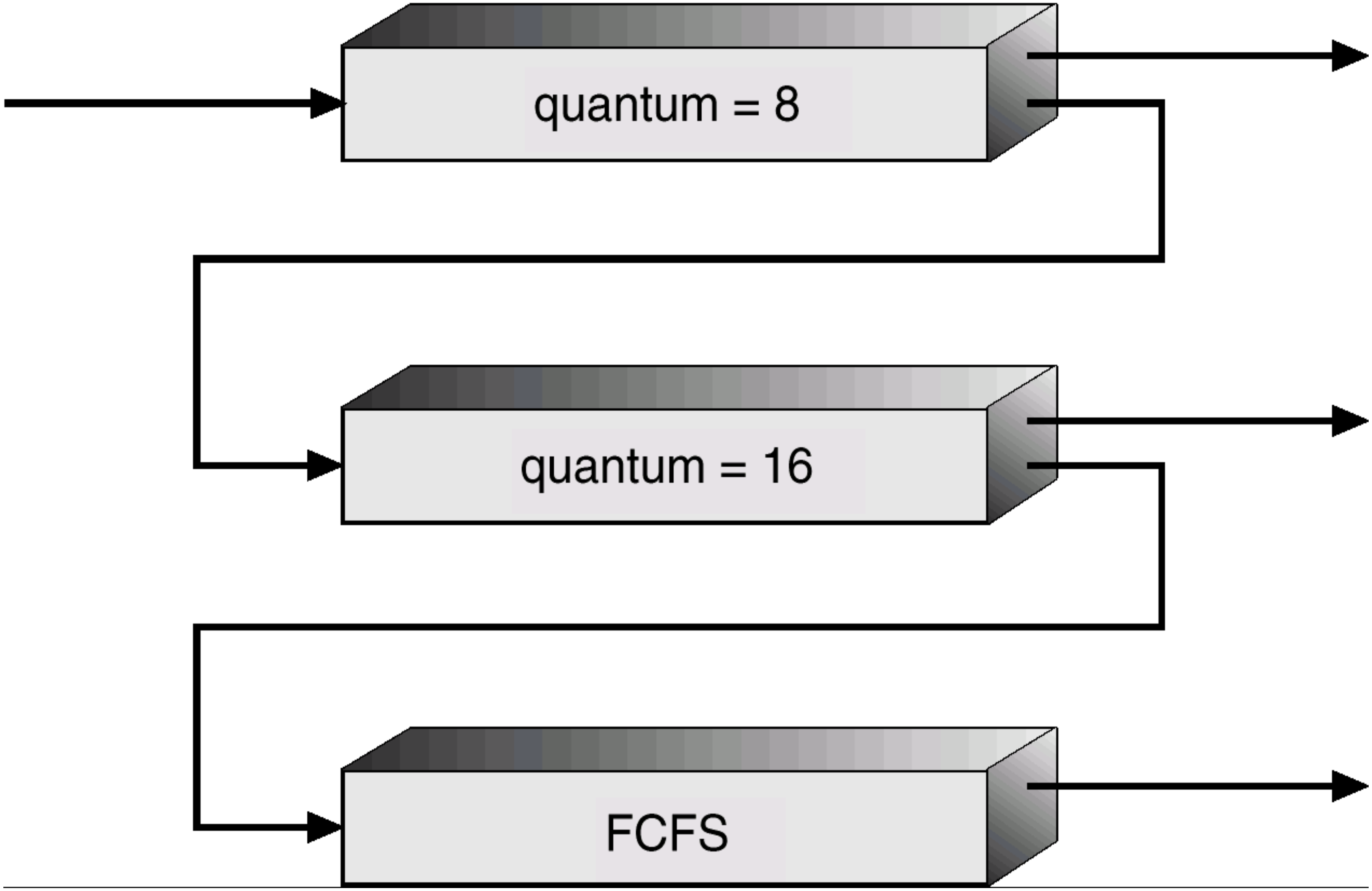
else Q3 (FCFS)

Process	**Arrival Time**	**Burst Time**
P1	0	8
P2	2	2
P3	3	4
P4	5	6
P5	7	9
P6	9	3
P7	10	5

Multilevel Feedback Queue (MFQ) Scheduling

- Multilevel feed back queue scheduling allows a process to move between various queues.
- Processes that uses too much CPU time is moved to a lower priority queue thus leaving the interactive and I/O bound processes in the higher priority queue.
- Similarly Aging can be done to prevent starvation i.e. the processes that waits too long in the lower priority queue are moved to a higher priority queue.
- Parameters of a Multilevel-feedback-queue scheduler are:
  - 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

Multilevel Feedback Queues



MFQ Scheduling – Example 15

Draw the graph (Gantt chart) for the following processes using **MFQ** Scheduling algorithm.

Q1 - RR - 8 sec

Q2 - RR - 16 sec

Q3 - FCFS

Process	**Arrival Time**	**Burst Time**
P1	1	10
P2	5	8
P3	8	22
P4	12	16
P5	15	30

UNIX System-V Scheduling Algorithm

- Multilevel feedback priority queues with round robin within each queue.
- Time Quantum = 1 second
- Priorities are divided into two groups/bands:
  - Kernel Group
  - User Group
- In decreasing order of priority, the bands are:
  - Swapper
  - Block I/O device control processes
  - File manipulation
  - Character I/O device control processes
  - User processes

UNIX System-V Scheduling Algorithm

- The UNIX scheduling algorithm uses a simple formula to assign a priority value to every process in the system that is ready to run
- The priority value for every process in the system is recalculated every second. Recalculating priority values of all the processes every second causes process priorities to change dynamically
- When it is time for scheduling, the CPU is given to the process with the smallest priority number
- The formula used to compute the priority value is:

Priority Value = Threshold priority + Nice value + (Recent CPU usage)/2

- **Threshold priority** is an integer usually having value of 40 or 60
- **Nice** is a positive integer with default value of 10 (-20 to 19)
- ***CPU* usage** is the number of clock ticks that the process has used CPU
- When a system tries to run too many high priority jobs at the same time, computer response time deteriorates. The **nice** command issued to adjust scheduling priorities to avoid this problem.

```
#  nice   [-inc]  command   [args]
```

Evaluation of Scheduling Algorithms

- **Deterministic Modeling**
- **Queuing Model**
- **Simulation**
- **Implementation**

Evaluation of Scheduling Algorithms…

Deterministic Modeling

- Consider a certain number of processes with their arrival time and next CPU bursts. Evaluate for various scheduling algorithms and compare the waiting time and turnaround time.

- **Characteristics**
  - Use Gantt chart.
  - Simple and fast.
  - Requires exact input to carry out the comparisons.
  - Performance figures may not be true in general.

Evaluation of Scheduling Algorithms…

Queuing Modeling

- Computer system viewed as a network of queues and servers, such as ready queue, I/O queue, event queues, CPUs, I/O device controllers, etc.
- **Input:** Arrival and service rates
- **Output:** CPU utilization, average queue length, average waiting time, …
- **Little's Formula**

» $$n = \lambda * W$$

where

n = average queue length

λ = average arrival rate

W = average waiting time in a queue

Evaluation of Scheduling Algorithms…

Queuing Modeling…

Let the average job arrival rate be 0.5

Algorithm	Average Wait Time $W=t_w$	Average Queue Length(n)
FCFS	4.6	2.3
SJF	3.6	1.8
SRTF	3.2	1.6
RR (q=1)	7.0	3.5
RR (q=4)	6.0	3.0

Evaluation of Scheduling Algorithms…

Queuing Modeling…

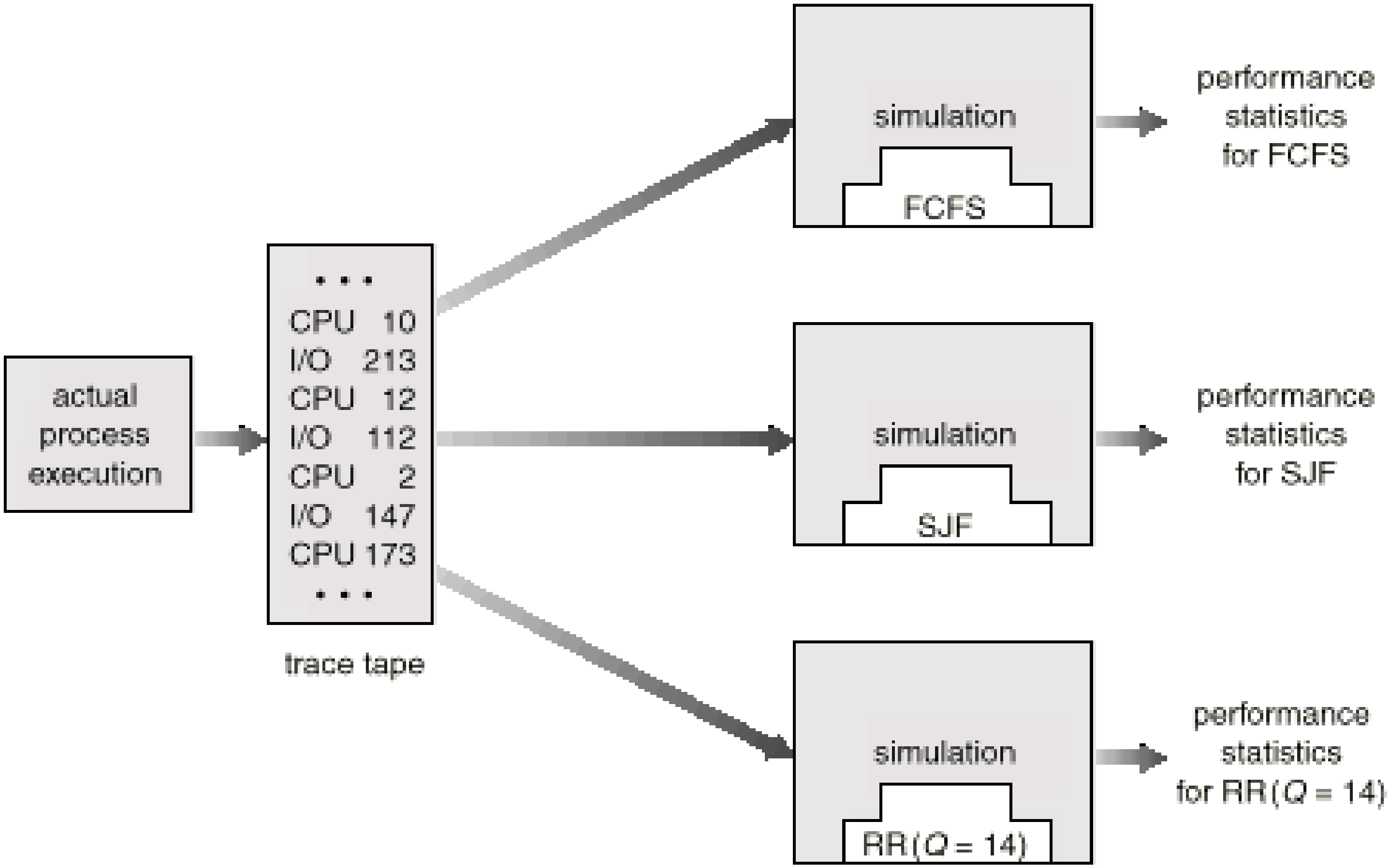
- Complicated mathematics
- Distributions (Poisson, uniform, exponential, etc) for the arrival and departure rates can be difficult to work with
- Assumptions may not be accurate
- Approximation of the real system

Evaluation of Scheduling Algorithms…

Simulation

- Programming model for the computer system
- Workload random number generator, or by collecting data from the actual system.
- Characteristics
  - Expensive: hours of programming and execution time
  - May be erroneous because of the assumptions about distributions
  - generated by assuming some distribution and a

Evaluation of Scheduling Algorithms…

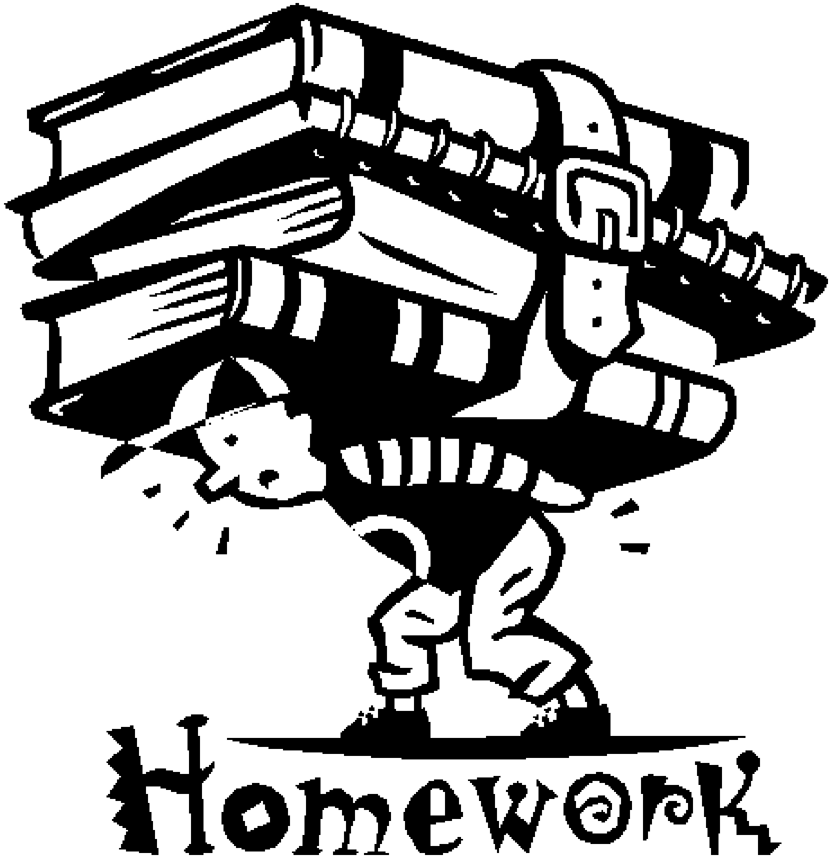


Evaluation of Scheduling Algorithms…

Implementation

- If you have UNIX Kernel source code, use VMware change the code for scheduling, compile the Kernel again and test.
- Characteristics
  - Best
  - Most expensive
  - Good option due to Open Source kernels such as Linux

COMING SOON



SCHEDULING ALGORITHMS

We’re done for now, but Todo’s for you after this lecture…

- Go through the slides and Book Sections: 5.1 to 5.3, 5.7
- Practice sample problems discussed and solved in class.
- Assignment for Scheduling Algorithms will be uploaded soon, start doing it at your earliest to get it done by the given dead line.

If you have problems visit me in counseling hours. . . .