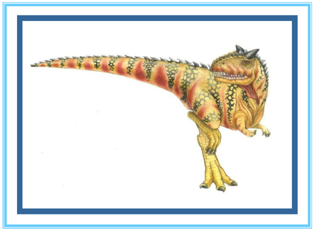
KHOA CÔNG NGHỆ THÔNG TIN & TRUYỀN THÔNG
COLLEGE OF INFORMATION & COMMUNICATION TECHNOLOGY
TRƯỜNG ĐẠI HỌC CẦN THƠ

Operating Systems

Chapter 4 CPU Scheduling



LECTURER: LAM NHUT KHANG

Slides are either from or adapted from
Operating System Concepts – 10th Ed. Silberschatz, Galvin, Gagne, 2018.

Outline

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

Outline

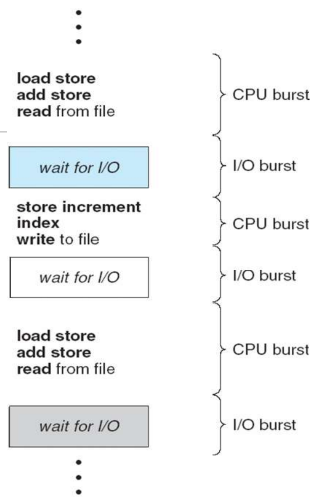
- **Basic Concepts**
- Scheduling Criteria
- Scheduling Algorithms
- Multiple-Processor Scheduling
- Operating Systems Examples
- Algorithm Evaluation

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



Alternating Sequence of CPU and I/O Bursts

CPU 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 (I/O interrupts)
2. Switches from running to ready state (Clock interrupts)
3. Switches from waiting to ready (I/O completion)
4. Terminates

Scheduling under 1 and 4 is **non-preemptive**

All other scheduling is **preemptive**

- Consider access to shared data
- Consider preemption while in kernel mode
- Consider interrupts occurring during crucial OS activities

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

Outline

➢Basic Concepts

➢**Scheduling Criteria**

➢Scheduling Algorithms

➢Multiple-Processor Scheduling

➢Operating Systems Examples

➢Algorithm Evaluation

Scheduling Criteria

➢**CPU utilization** – keep the CPU as busy as possible

→Max CPU utilization

➢**Throughput** – # of processes that complete their execution per time unit

→Max throughput

➢**Turnaround time** – amount of time to execute a particular process

→Min turnaround time

➢**Waiting time** – amount of time a process has been waiting in the ready queue

→ Min waiting 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)

→Min response time

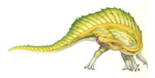
First-Come First-Served (FCFS) Scheduling

FCFS – Ex #1

Process	CPU time
P1	24
P2	3
P3	3

Suppose that the processes arrive in the order: P_1 , P_2 , P_3

1. Waiting time? Average waiting time?
2. Response time? Average response time?
3. Turnaround time? Average turnaround time?

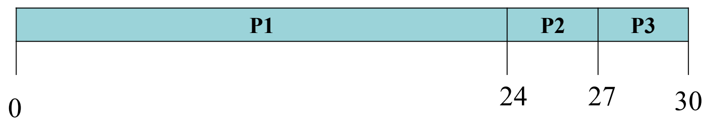


FCFS – Ex #1

Process	CPU time
P1	24
P2	3
P3	3

The processes arrive in the order: P_1 , P_2 , P_3

The Gantt Chart for the schedule is:

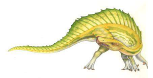


Waiting time: amount of time a process has been waiting in the ready queue

➢Waiting time for

$P_1 = 0 - 0 = 0$ $P_2 = 24 - 0 = 24$ $P_3 = 27 - 0 = 27$

➢Average waiting time $= \frac{0+24+27}{3} = 17$

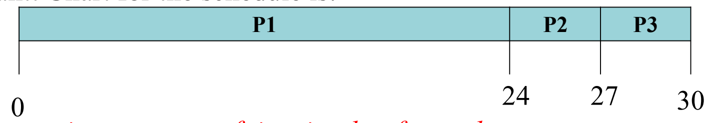


FCFS – Ex #1

Process	CPU time
P1	24
P2	3
P3	3

The processes arrive in the order: P_1 , P_2 , P_3

The Gantt Chart for the schedule is:

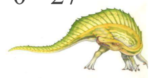


Response time: amount of time it takes from when a request was submitted until the first response is produced, not output

➢Response time for

$P_1 = 0 - 0 = 0$ $P_2 = 24 - 0 = 24$ $P_3 = 27 - 0 = 27$

➢Average response time $= \frac{0+24+27}{3} = 17$

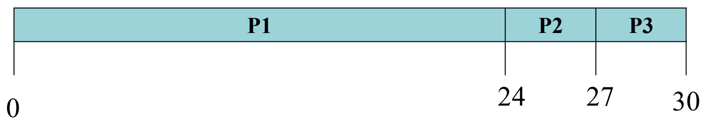


FCFS – Ex #1

Process	CPU time
P1	24
P2	3
P3	3

The processes arrive in the order: P_1 , P_2 , P_3

The Gantt Chart for the schedule is:

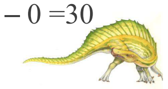


Turnaround time: amount of time to execute a particular process

➢Turnaround time for

P_1 = 24 – 0 = 24 P_2 = 27 – 0 = 27 P_3 = 30 – 0 =30

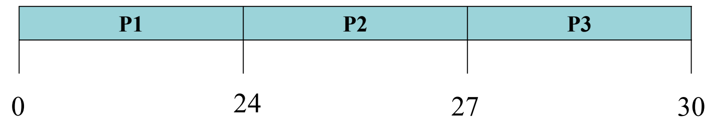
➢Average turnaround time = $\frac{24+27+30}{3} = 27$



FCFS – Ex #1

Process	CPU time
P1	24
P2	3
P3	3

Gantt chart



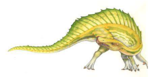
Process	Waiting time	Response time	Turnaround time
P1	0	0	24
P2	24	24	27
P3	27	27	30
Average	**17**	**17**	**27**

FCFS – Ex #2

Process	CPU time
P1	24
P2	3
P3	3

Suppose that the processes arrive in the order: $\boldsymbol{P_2}$, $\boldsymbol{P_3}$, $\boldsymbol{P_1}$

1. Waiting time? Average waiting time?
2. Response time? Average response time?
3. Turnaround time? Average turnaround time?

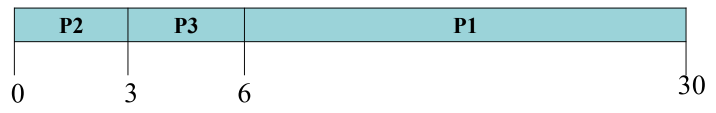


FCFS – Ex #2

Process	CPU time
P1	24
P2	3
P3	3

The processes arrive in the order: $\boldsymbol{P_2}$, $\boldsymbol{P_3}$, $\boldsymbol{P_1}$

The Gantt Chart for the schedule is:

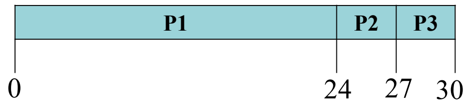


Process	Waiting time	Response time	Turnaround time
P1	6 – 0 = 6	6 – 0 = 6	30 – 0 = 30
P2	0 – 0 = 0	0 – 0 = 0	3 – 0 = 3
P3	3 – 0 = 3	3 – 0 = 3	6 – 0 = 6
Average	$\frac{6+0+3}{3} = 3$	$\frac{6+0+3}{3} = 3$	$\frac{30+3+6}{3} = 13$

FCFS

Process	CPU time
P1	24
P2	3
P3	3

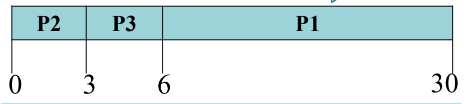
THE ORDER: P_1 , P_2 , P_3



Process	Waiting time	Response time	Turnaround time
P1	0	0	24
P2	24	24	27
P3	27	27	30
Average	17	17	27

THE ORDER: P_2, P_3, P_1

Much better



Process	Waiting time	Response time	Turnaround time
P1	6	6	30
P2	0	0	3
P3	3	3	6
Average	3	3	13

Convoy effect *- short process behind long process (Consider one CPU-bound and many I/O-bound processes)*

FCFS – Ex #3

Process	CPU time	Arrival time
P1	2	2
P2	1	0
P3	2	8
P4	4	9

1. Waiting time? Average waiting time?
2. Response time? Average response time?
3. Turnaround time? Average turnaround time?

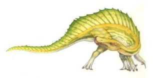
KHOA CÔNG NGHỆ THÔNG TIN & TRUYỀN THÔNG
COLLEGE OF INFORMATION & COMMUNICATION TECHNOLOGY
TRƯỜNG ĐẠI HỌC CẦN THƠ

Shortest-Job-First (SJF) Scheduling

SJF nonpreemptive – Ex #1

Process	CPU time
P1	24
P2	3
P3	3

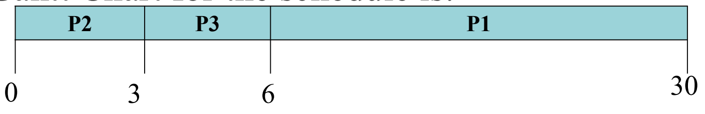
1. Waiting time? Average waiting time?
2. Response time? Average response time?
3. Turnaround time? Average turnaround time?



SJF nonpreemptive – Ex #1

Process	CPU time
P1	24
P2	3
P3	3

The Gantt Chart for the schedule is:

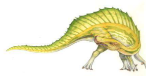


Process	Waiting time	Response time	Turnaround time
P1	6 – 0 = 6	6 – 0 = 6	30 – 0 = 30
P2	0 – 0 = 0	0 – 0 = 0	3 – 0 = 3
P3	3 – 0 = 3	3 – 0 = 3	6 – 0 = 6
Average	$\frac{6+0+3}{3} = 3$	$\frac{6+0+3}{3} = 3$	$\frac{30+3+6}{3} = 13$

SJF Scheduling

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

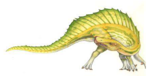
SJF is optimal – gives minimum average waiting time for a given set of processes
- The difficulty is knowing the length of the next CPU request
- Could ask the user



SJF nonpreemptive – Ex #2

Process	CPU time
P1	6
P2	8
P3	7
P4	3

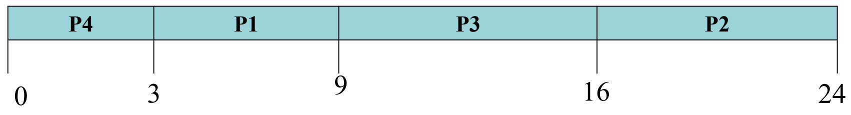
1. Waiting time? Average waiting time?
2. Response time? Average response time?
3. Turnaround time? Average turnaround time?



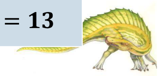
SJF nonpreemptive – Ex #2

Process	CPU time
P1	6
P2	8
P3	7
P4	3

Gantt Chart



Process	Waiting time	Response time	Turnaround time
P1	3 – 0 = 3	3 – 0 = 3	9 – 0 = 9
P2	16 – 0 = 16	16 – 0 = 16	24 – 0 = 24
P3	9 – 0 = 9	9 – 0 = 9	16 – 0 = 16
P4	0 – 0 = 0	0 – 0 = 0	3 – 0 = 3
Average	$\frac{3+16+9+0}{4} = 7$	$\frac{3+16+9+0}{4} = 7$	$\frac{9+24+16+3}{4} = 13$



SJF nonpreemptive – Ex #3

Process	CPU time	Arrival time
P1	6	0
P2	8	2
P3	7	4
P4	3	5

1. Waiting time? Average waiting time?
2. Response time? Average response time?
3. Turnaround time? Average turnaround time?

SJF nonpreemptive – Ex #4

Process	CPU time	Arrival time
P1	8	0
P2	4	1
P3	9	2
P4	5	3

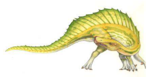
1. Waiting time? Average waiting time?
2. Response time? Average response time?
3. Turnaround time? Average turnaround time?

Determining Length of Next CPU Burst

- Can only estimate the length – should be similar to the previous one
  → Then pick process with shortest predicted next CPU burst
- 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. τ_{n+1} = predicted value for the next CPU burst
  3. $\alpha, 0 \le \alpha \le 1$
  4. Define : $\tau_{n+1} = \alpha\, t_n + (1 - \alpha)\tau_n.$

- Commonly, α set to ½
- Preemptive version called **shortest-remaining-time-first**



Ex. 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:

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

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

SJF preemptive – Ex #1

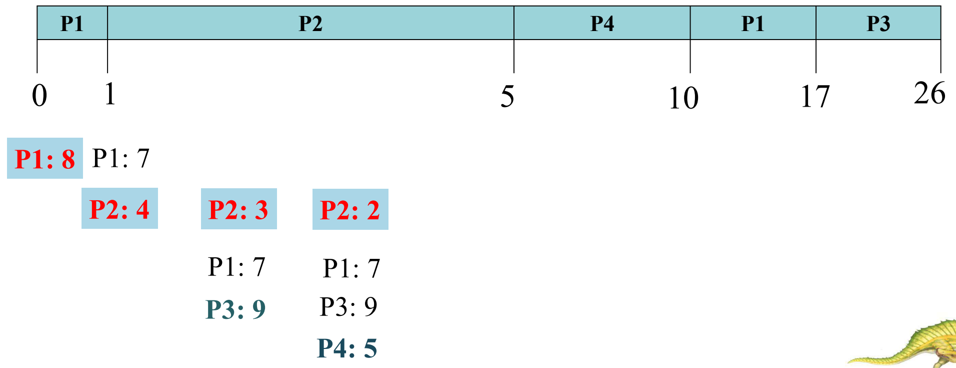
Process	CPU time	Arrival time
P1	8	0
P2	4	1
P3	9	2
P4	5	3

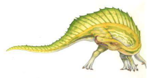
1. Waiting time? Average waiting time?
2. Response time? Average response time?
3. Turnaround time? Average turnaround time?

SJF preemptive – Ex #1

Process	CPU time	Arrival time
P1	8	0
P2	4	1
P3	9	2
P4	5	3

Gantt Chart

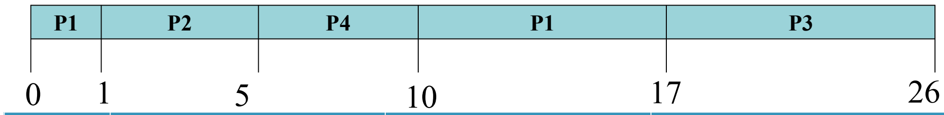




SJF preemptive – Ex #1

Process	CPU time	Arrival time
P1	8	0
P2	4	1
P3	9	2
P4	5	3

Gantt Chart



Process	Waiting time	Response time	Turnaround time
P1	(0-0)+(10-1)=9	0 – 0 = 0	17 – 0 = 17
P2	1 – 1 = 0	1 – 1 = 0	5 – 1 = 4
P3	17 – 2 = 15	17 – 2 = 15	26 – 2 = 24
P4	5 - 3 = 2	5 – 3 = 2	10 – 3 = 7
Average	$\frac{9+0+15+2}{4} = 6,5$	$\frac{0+0+15+2}{4} = 4,25$	$\frac{17+4+24+7}{4} = 13$

SJF preemptive – Ex #2

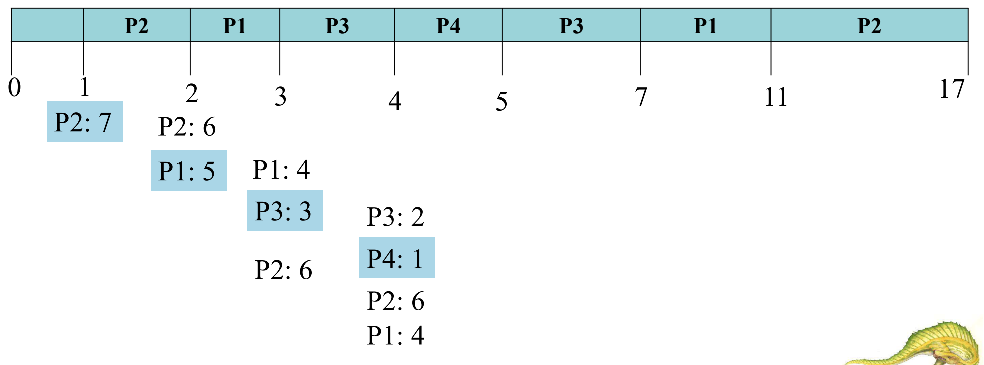
Process	CPU time	Arrival time
P1	5	2
P2	7	1
P3	3	3
P4	1	4

1. Waiting time? Average waiting time?
2. Response time? Average response time?
3. Turnaround time? Average turnaround time?

SJF preemptive – Ex #2

Process	CPU time	Arrival time
P1	5	2
P2	7	1
P3	3	3
P4	1	4

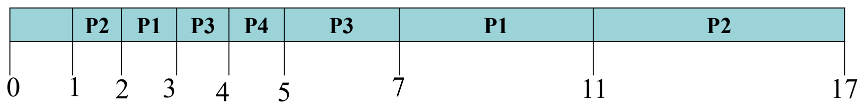
Gantt Chart



SJF preemptive – Ex #2

Process	CPU time	Arrival time
P1	5	2
P2	7	1
P3	3	3
P4	1	4

Gantt Chart



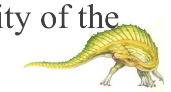
Process	Waiting time	Response time	Turnaround time
P1	(2-2)+(7-3)=0+4=4	2 – 2 = 0	11 – 2 = 9
P2	(1-1)+(11-2)=0+9=9	1 – 1 = 0	17 – 1 = 16
P3	(3-3)+(5-4)=0+1=1	3 – 3 = 0	7 – 3 = 4
P4	(4-4)=0	4 – 4 = 0	5 – 4 = 1
Average	$\frac{4+9+1+0}{4} = 3{,}5$	$\frac{0+0+0+0}{4} = 0$	$\frac{9+16+4+1}{4} = 7{,}5$

KHOA CÔNG NGHỆ THÔNG TIN & TRUYỀN THÔNG
COLLEGE OF INFORMATION & COMMUNICATION TECHNOLOGY
TRƯỜNG ĐẠI HỌC CẦN THƠ

Priority 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
  - Non-preemptive
- 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



Priority nonpreemptive – Ex #1

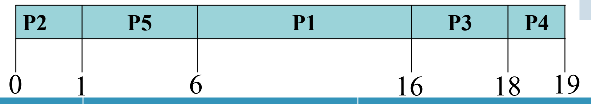
Process	CPU time	Priority
P1	10	3
P2	1	1
P3	2	4
P4	1	5
P5	5	2

1. Waiting time? Average waiting time?
2. Response time? Average response time?
3. Turnaround time? Average turnaround time?

Priority nonpreemptive – Ex #1

Process	CPU time	Priority
P1	10	3
P2	1	1
P3	2	4
P4	1	5
P5	5	2

Gantt Chart



Process	Waiting time	Response time	Turnaround time
P1	6 – 0 = 6	6 – 0 = 6	16 – 0 = 16
P2	0 – 0 = 0	0 – 0 = 0	1 – 0 = 1
P3	16 – 0 = 16	16 – 0 = 16	18 – 0 = 18
P4	18 – 0 = 18	18 – 0 = 18	19 – 0 = 19
P5	1 – 0 = 1	1 – 0 = 1	6 – 0 = 6
Average	$\frac{6+0+16+18+1}{5} = 8,2$	$\frac{6+0+16+18+1}{5} = 8,2$	$\frac{16+1+18+19+6}{5} = 12$

Priority nonpreemptive – Ex #2

Process	CPU time	Arrival time	Priority
P1	10	8	3
P2	1	9	1
P3	3	1	4
P4	1	2	5
P5	8	3	2

1. Waiting time? Average waiting time?
2. Response time? Average response time?
3. Turnaround time? Average turnaround time?

Priority nonpreemptive – Ex #3

Process	CPU time	Arrival time	Priority
P1	8	0	3
P2	3	4	1
P3	2	3	4
P4	1	2	5
P5	4	1	2

1. Waiting time? Average waiting time?
2. Response time? Average response time?
3. Turnaround time? Average turnaround time?

Priority Preemptive – Ex #1

Process	CPU time	Arrival time	Priority
P1	8	0	3
P2	3	4	1
P3	2	3	4
P4	1	2	5
P5	4	1	2

1. Waiting time? Average waiting time?
2. Response time? Average response time?
3. Turnaround time? Average turnaround time?

KHOA CÔNG NGHỆ THÔNG TIN & TRUYỀN THÔNG
COLLEGE OF INFORMATION & COMMUNICATION TECHNOLOGY
TRƯỜNG ĐẠI HỌC CẦN THƠ

Round Robin (RR) Scheduling

RR Scheduling

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

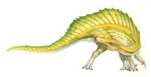
Performance

- *q* large ⇒ FIFO
- *q* small ⇒ *q* must be large with respect to context switch, otherwise overhead is too high

RR (q=4) - Ex#1

Process	CPU time
P1	24
P2	3
P3	3

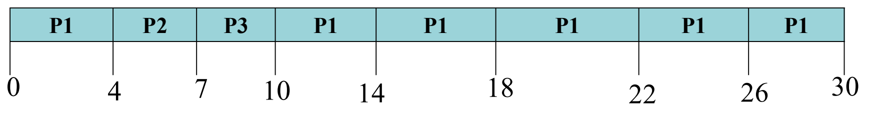
1. Waiting time? Average waiting time?
2. Response time? Average response time?
3. Turnaround time? Average turnaround time?



RR (q=4) - Ex#1

Process	CPU time
P1	24
P2	3
P3	3

Gantt Chart

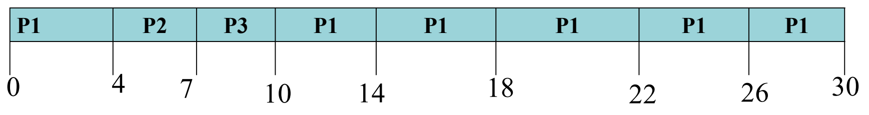


P1:24	P2:3	P3:3	P1:20	P1	P1	P1	P1	P1
P2:3	P3:3	P1:20						
P3:3	P1:20							

RR (q=4) - Ex#1

Process	CPU time
P1	24
P2	3
P3	3

Gantt Chart



Process	Waiting time	Response time	Turnaround time
P1	(0 - 0) + (10 - 4) = 6	0 – 0 = 0	30 – 0 = 30
P2	4 – 0 = 4	4 – 0 = 4	7 - 0 = 7
P3	7 - 0 = 7	7 - 0 = 7	10 – 0 = 10
Average	$\frac{6+4+7}{3}$ = **5,67**	$\frac{0+4+7}{3}$ =**3,67**	$\frac{30+7+10}{3}$ =**15,67**

RR Scheduling

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

RR (q=2) - Ex#2

Process	CPU time
P1	6
P2	5
P3	2
P4	3
P5	7

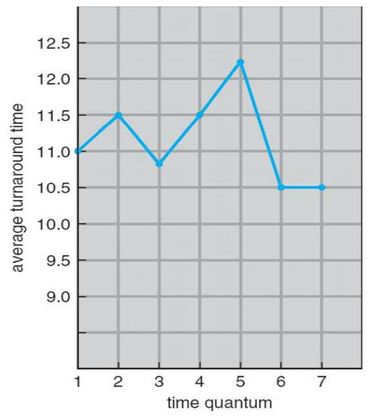
1. Waiting time? Average waiting time?
2. Response time? Average response time?
3. Turnaround time? Average turnaround time?

RR (q=3) - Ex#3

Process	CPU time	Arrival time
P1	4	0
P2	5	1
P3	2	2
P4	1	3
P5	6	4
P6	3	6

1. Waiting time? Average waiting time?
2. Response time? Average response time?
3. Turnaround time? Average turnaround time?

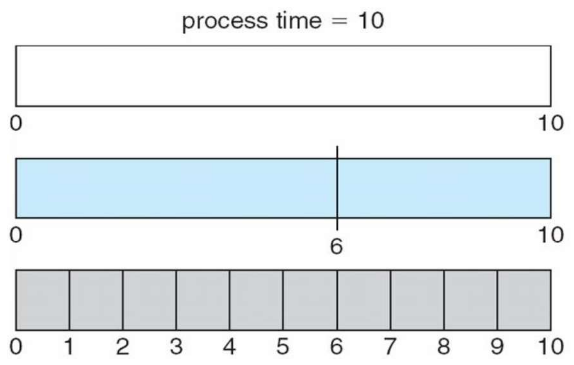
Turnaround Time Varies With The Time Quantum



process	time
P_1	6
P_2	3
P_3	1
P_4	7

80% of CPU bursts should be shorter than *q*

Time Quantum and Context Switch Time



quantum	context switches
12	0
6	1
1	9

Multilevel Queue

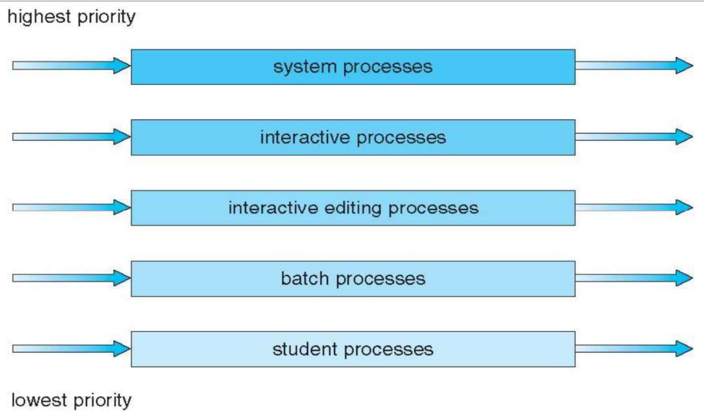
Ready queue is partitioned into separate queues, eg:
- foreground (interactive)
- background (batch)

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
- 20% to background in FCFS

Multilevel Queue Scheduling



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

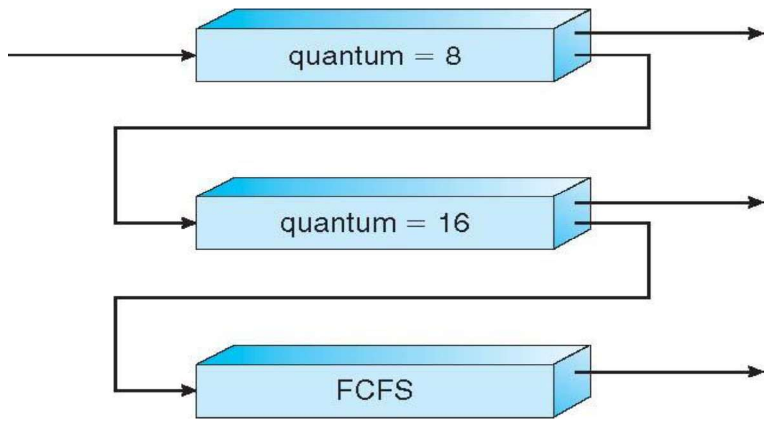
Three queues:

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

Scheduling

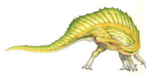
- A new job enters queue Q_0 which is served FCFS
  - When it gains CPU, job receives 8 milliseconds
  - If it does not finish in 8 milliseconds, job is moved to queue Q_1
- At Q_1 job is again served FCFS and receives 16 additional milliseconds
  - If it still does not complete, it is preempted and moved to queue Q_2

Multilevel Feedback Queues



Outline

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



Multiple-Processor Scheduling

CPU scheduling more complex when multiple CPUs are available

Homogeneous processors within a multiprocessor

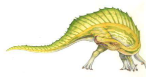
Asymmetric multiprocessing – only one processor accesses the system data structures, alleviating the need for data sharing

Symmetric multiprocessing (SMP) – each processor is self-scheduling, all processes in common ready queue, or each has its own private queue of ready processes

- Currently, most common

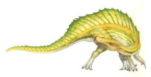
Processor affinity – process has affinity for processor on which it is currently running

- **soft affinity**
- **hard affinity**
- Variations including **processor sets**



Outline

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

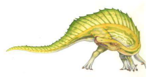


Operating System Examples

Windows XP scheduling

Solaris scheduling

Linux scheduling



Windows Scheduling

Windows uses priority-based preemptive scheduling

Highest-priority thread runs next

Dispatcher is scheduler

Thread runs until (1) blocks, (2) uses time slice, (3) preempted by higher-priority thread

Real-time threads can preempt non-real-time

32-level priority scheme

Variable class is 1-15, **real-time class** is 16-31

Priority 0 is memory-management thread

Queue for each priority

If no run-able thread, runs **idle thread**

Windows Priority Classes

Win32 API identifies several priority classes to which a process can belong

- REALTIME_PRIORITY_CLASS, HIGH_PRIORITY_CLASS, ABOVE_NORMAL_PRIORITY_CLASS,NORMAL_PRIORITY_CLASS, BELOW_NORMAL_PRIORITY_CLASS, IDLE_PRIORITY_CLASS
- All are variable except REALTIME

A thread within a given priority class has a relative priority

- TIME_CRITICAL, HIGHEST, ABOVE_NORMAL, NORMAL, BELOW_NORMAL, LOWEST, IDLE

Priority class and relative priority combine to give numeric priority

Base priority is NORMAL within the class

If quantum expires, priority lowered, but never below base

If wait occurs, priority boosted depending on what was waited for

Foreground window given 3x priority boost

Solaris

Priority-based scheduling

Six classes available

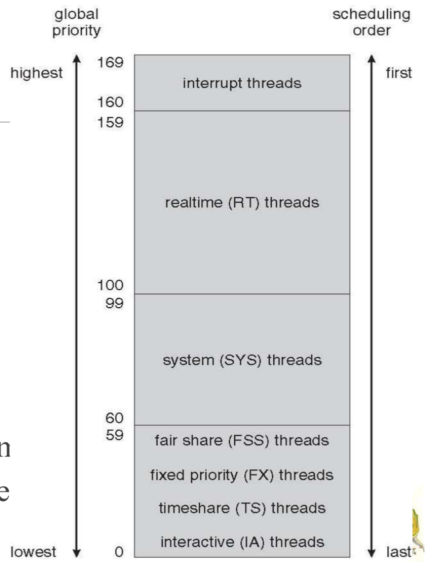
- Time sharing (default)
- Interactive
- Real time
- System
- Fair Share
- Fixed priority

Given thread can be in one class at a time

Each class has its own scheduling algorithn

Time sharing is multi-level feedback queue

- Loadable table configurable by sysadmin



Linux Scheduling

Constant order *O*(1) scheduling time

Preemptive, priority based

Two priority ranges: time-sharing and real-time

Real-time range from 0 to 99 and **nice** value from 100 to 140

Map into global priority with numerically lower values indicating higher priority

Higher priority gets larger q

Task run-able as long as time left in time slice (**active**)

If no time left (**expired**), not run-able until all other tasks use their slices

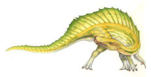
All run-able tasks tracked in per-CPU **run-queue** data structure

- Two priority arrays (active, expired)
- Tasks indexed by priority
- When no more active, arrays are exchanged

Linux Scheduling (Cont.)

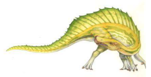
Real-time scheduling according to POSIX.1b
- Real-time tasks have static priorities

All other tasks dynamic based on *nice* value plus or minus 5
- Interactivity of task determines plus or minus
  - More interactive -> more minus
- Priority recalculated when task expired
- This exchanging arrays implements adjusted priorities



Outline

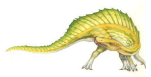
- ➢Basic Concepts
- ➢Scheduling Criteria
- ➢Scheduling Algorithms
- ➢Multiple-Processor Scheduling
- ➢Operating Systems Examples
- ➢**Algorithm Evaluation**



Algorithm Evaluation

How to select CPU-scheduling algorithm for an OS?

Determine criteria, then evaluate algorithms

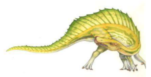


Deterministic modeling

Type of analytic evaluation

Takes a particular predetermined workload and defines the performance of each algorithm for that workload

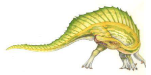
Adv. & Disadv.



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

$$n = \lambda \text{ x } W$$

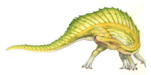
- Valid for any scheduling algorithm and arrival distribution

For example, if on average 7 processes arrive per second, and normally 14 processes in queue, then average wait time per process = 2 seconds

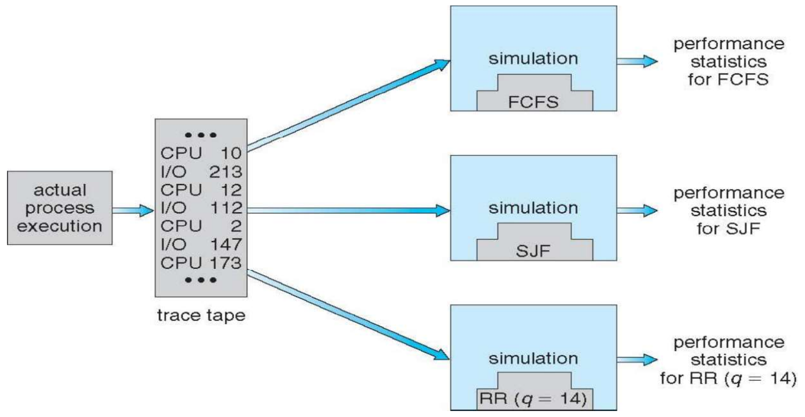
Simulations

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