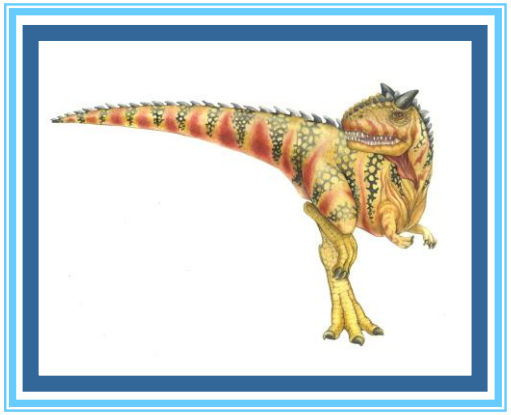
# Chapter 4 : CPU Scheduling



# Chapter 4 : CPU Scheduling

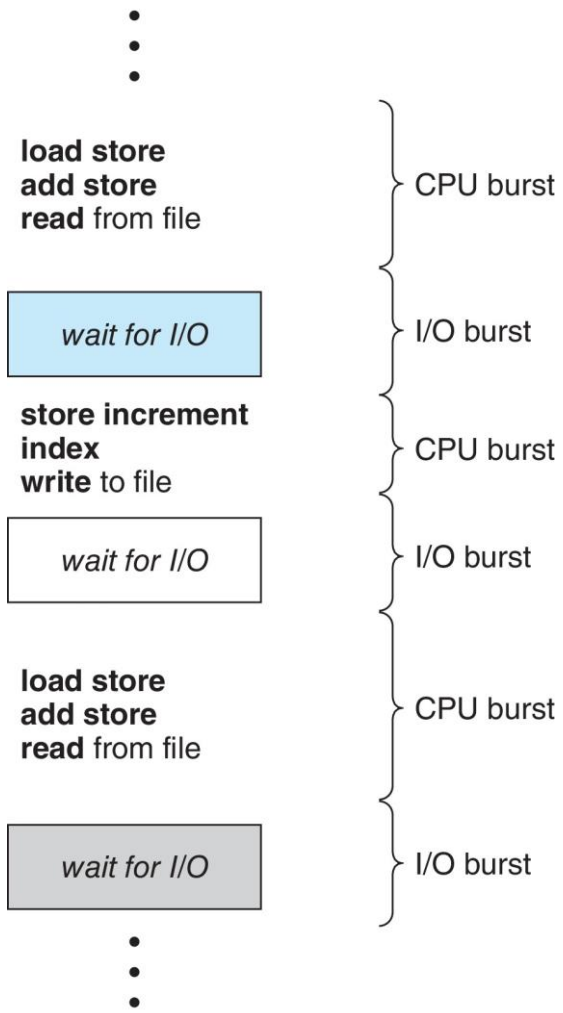
- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
- Thread Scheduling
- Multi-Processor Scheduling
- Operating Systems Example
- Algorithm Evaluation

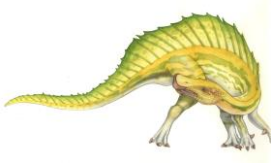
# Objectives

- Describe various CPU scheduling algorithms
- Assess CPU scheduling algorithms based on scheduling criteria
- Explain the issues related to multiprocessor and multicore scheduling
- Describe the scheduling algorithm used in the Windows operating system
- Apply modeling and simulations to evaluate CPU scheduling algorithms

# 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** followed by **I/O burst**
- CPU burst distribution is of main concern

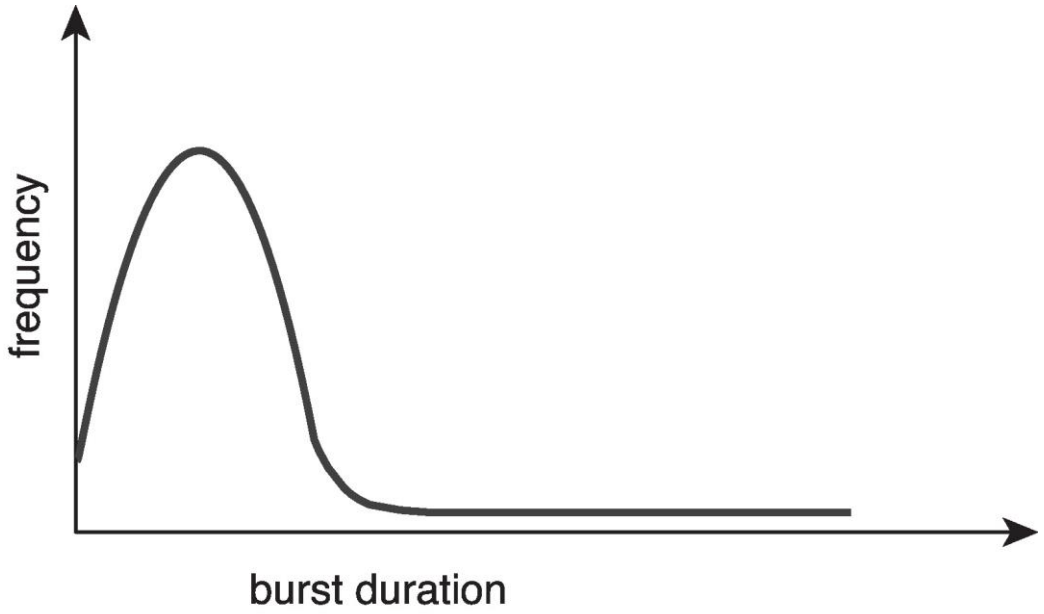


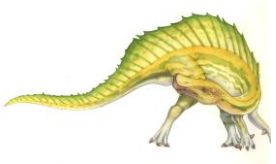



# Histogram of CPU-burst Times

Large number of short bursts

Small number of longer bursts

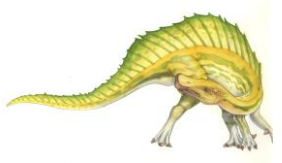


# CPU Scheduler

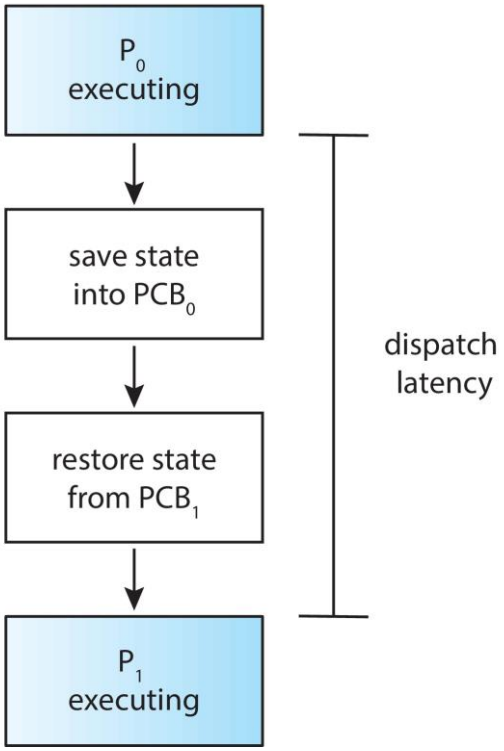
- The **CPU scheduler** selects from among the processes in ready queue, and allocates a CPU core to one of them
  - Queue may be ordered in various ways
- CPU scheduling decisions may take place when a process:
  1. Switches from running to waiting state
  2. Switches from running to ready state
  3. Switches from waiting to ready
  4. Terminates
- Scheduling under 1 and 4 is **nonpreemptive**
- All other scheduling is **preemptive**
  - Consider access to shared data
  - Consider preemption while in kernel mode
  - Consider interrupts occurring during crucial OS activities

nonpreemptive: ไม่สามารถแทรกการทำงานกลางคัน
preemptive: แทรกการทำงานกลางคัน



# Dispatcher

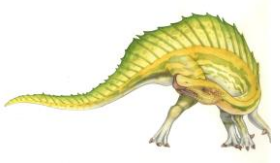
- Dispatcher module gives control of the CPU to the process selected by the short-term scheduler; this involves:
  - switching context
  - switching to user mode
  - jumping to the proper location in the user program to restart that program
- **Dispatch latency** – time it takes for the dispatcher to stop one process and start another running




Dispatcher: ตัวส่งข่าวสารไปยัง state อื่น , ตัวส่งต่อ

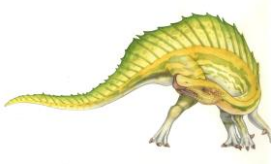
# Scheduling Criteria

- **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
  (เวลาของโพรเซสที่ถูกประมวลผลนั้น ๆ จบการทำงาน)
- **Waiting time** – amount of time a process has been waiting in the ready queue (เวลาที่โพรเซสนั้น ๆ รอคอยที่ Ready queue ก่อนจะถูกประมวลผลกับ CPU)
- **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)



# Scheduling Algorithm Optimization Criteria

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



# 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:


- Waiting time for $P_1$ = 0; $P_2$ = 24; $P_3$ = 27 mS
  $P_1$ $P_2$ $P_3$
- Average waiting time: (0 + 24 + 27)/3 = 17 mS
- Turnaround Time : $P_1$= 24 ; $P_2$= 27 ; $P_3$= 30 mS

# FCFS Scheduling (Cont.)

Suppose that the processes arrive in the order:

$$P_2 , P_3 , P_1$$

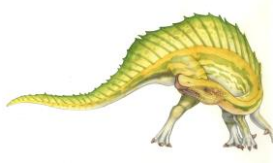
- The Gantt chart for the schedule is:


- Waiting time for $P_1 = 6; P_2 = 0; P_3 = 3$ mS

$P_1$ $P_2$ $P_3$

- Average waiting time: (6 + 0 + 3)/3 = 3 mS
- Turnaround Time : $P_1$= 30 ; $P_2$= 3 ; $P_3$= 6 mS

- Much better than previous case
- **Convoy effect** - short process behind long process
  - Consider one CPU-bound and many I/O-bound processes



# 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
- Two schemes:
  - ***nonpreemptive*** – once CPU given to the process it cannot be preempted until completes its CPU burst.
  - ***preemptive*** – if a new process arrives with CPU burst length less than remaining time of current executing process, preempt. This scheme is know as the Shortest-Remaining-Time-First (SRTF).
- 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

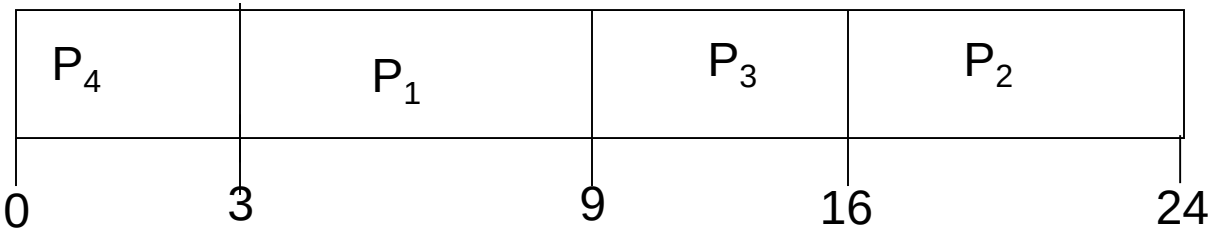
arrive : มาถึง

# Example of SJF

| Process | Burst Time |
|---|---|
| $P_1$ | 6 |
| $P_2$ | 8 |
| $P_3$ | 7 |
| $P_4$ | 3 |

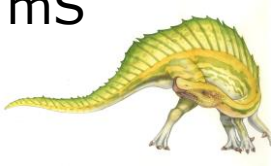
** ทุก Process มาถึงเวลาเดียวกัน

- SJF scheduling chart




$P_1$ $P_2$ $P_3$ $P_4$

- Average waiting time = (3 + 16 + 9 + 0) / 4 = 7 mS
- Turnaround Time : $P_1 = 9$; $P_2 = 24$; $P_3 = 16$; $P_4 = 3$ ; mS

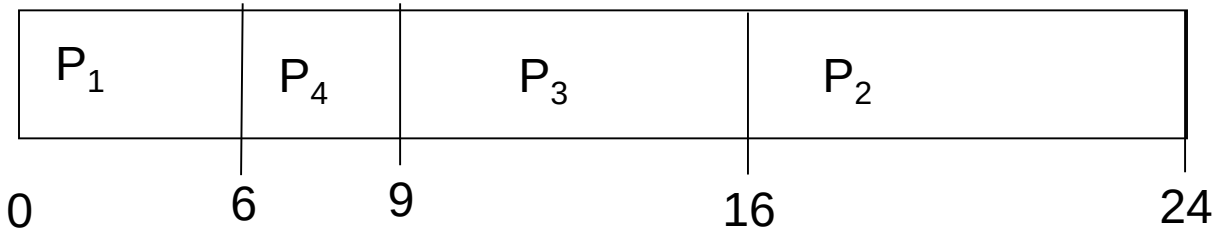


# Example of nonpreemptive SJF

| Process | Arrival Time | Burst Time |
|---|---|---|
| $P_1$ | 0.0 | 6 |
| $P_2$ | 2.0 | 8 |
| $P_3$ | 4.0 | 7 |
| $P_4$ | 5.0 | 3 |

** Process มาถึงเวลาไม่เท่ากัน

- SJF scheduling chart : แบบ **nonpreemptive** ไม่สามารถแทรกการทำงานกลางคันได้




$P_1$ $P_2$ $P_3$ $P_4$

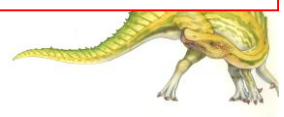
- Average waiting time =( 0 + 14 + 5 + 1) / 4 = 20/4 = 5 mS

**0-0** **16-2** **9-4** **6-5**

* คิด Arrival time ด้วย

- Turnaround Time = $P_1$= 6 ,$P_2$= 22 , $P_3$= 12 , $P_4$= 4 mS

6-0 24-2 16-4 9-5

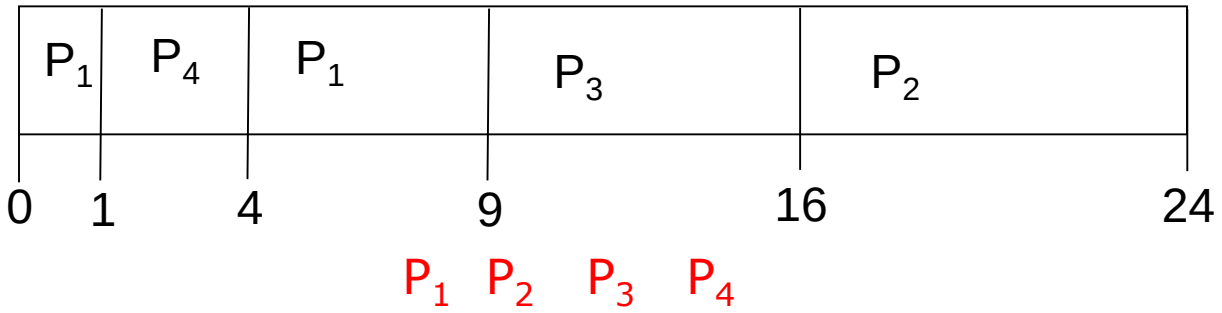


# Example of preemptive SJF

| Process | Arrival Time | Burst Time |
|---|---|---|
| $P_1$ | 0.0 | 6 |
| $P_2$ | 2.0 | 8 |
| $P_3$ | 4.0 | 7 |
| $P_4$ | 1.0 | 3 |

** Process มาถึงเวลาไม่เท่ากัน

- SJF scheduling chart : แบบ **preemptive** แทรกการทำงานกลางคันได้




- Average waiting time = (3 + 14 + 5 + 0) / 4 = 22/4 = 5.5 mS

4-1 16-2 9-4 1-1

* คิด Arrival time ด้วย

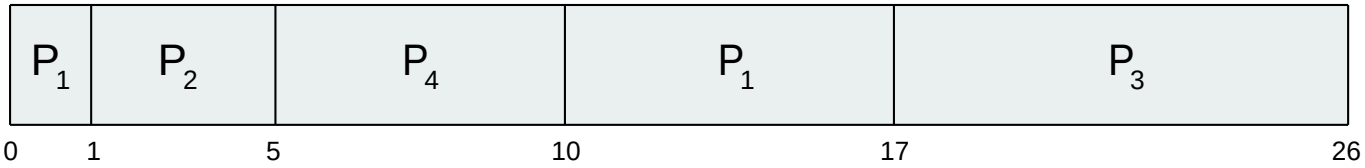
- Turnaround Time = ???

# Practice: Shortest-remaining-time-first

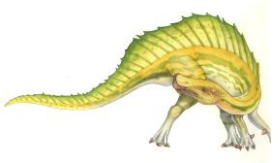
- Now we add the concepts of varying arrival times and preemption to the analysis

| 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 = ????
- Turnaround time = ???



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

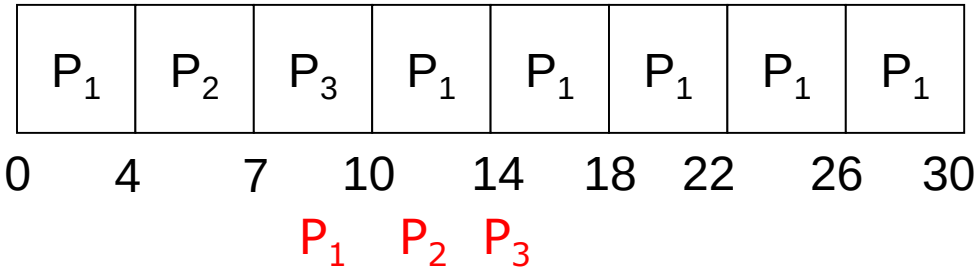
time quantum: ส่วนแบ่งเวลา

# Example of RR with Time Quantum = 4

| Process | Burst Time |
|---|---|
| $P_1$ | 24 |
| $P_2$ | 3 |
| $P_3$ | 3 |

* ทุก process มาถึงเวลาเดียวกัน

- The Gantt chart is:



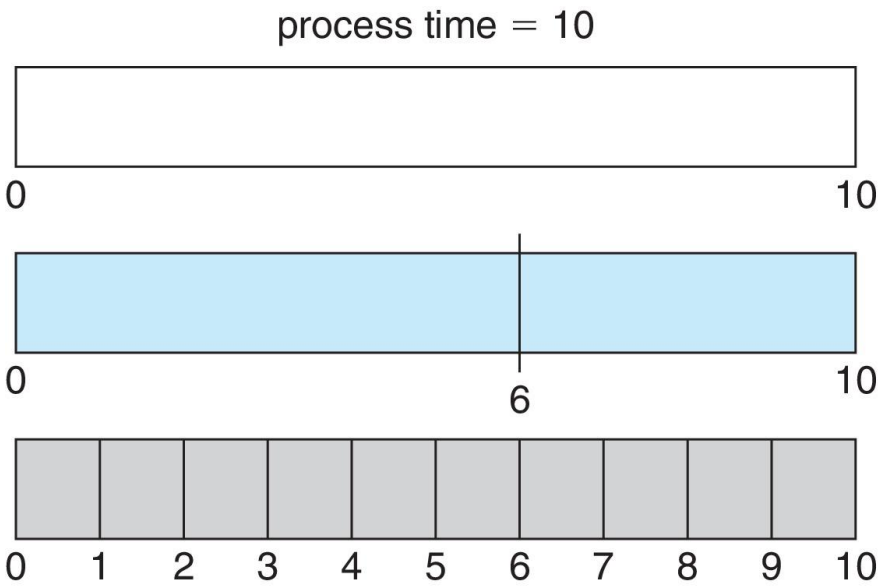

- average waiting time: ( 6 + 4 + 7 ) /3 = 5.67 mS

$P_1$ $P_2$ $P_3$

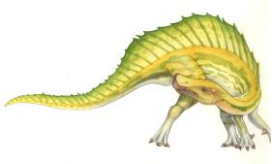
**26-4-4-4-4-4** **4** **7**

- Turnaround time : $P_1$= 30 ; $P_2$ = 7 ; $P_3$= 10 ;
- 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

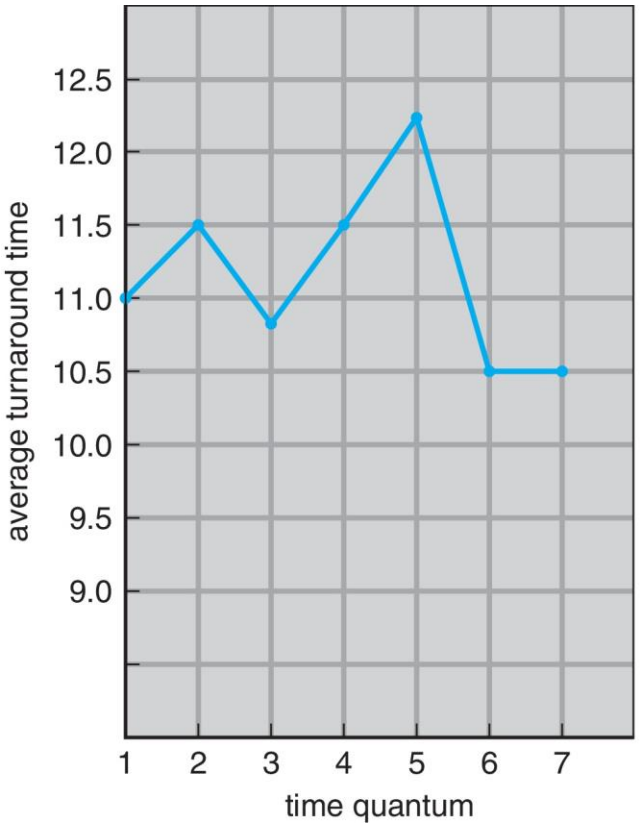
# Time Quantum and Context Switch Time




| quantum | context switches |
|---|---|
| 12 | 0 |
| 6 | 1 |
| 1 | 9 |



# Turnaround Time Varies With The Time Quantum



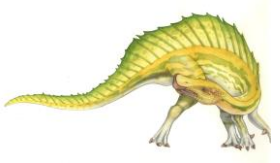

| process | time |
|---|---|
| $P_1$ | 6 |
| $P_2$ | 3 |
| $P_3$ | 1 |
| $P_4$ | 7 |

80% of CPU bursts
should be shorter than q

**Turnaround time depends on the size of quantum time**

# 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

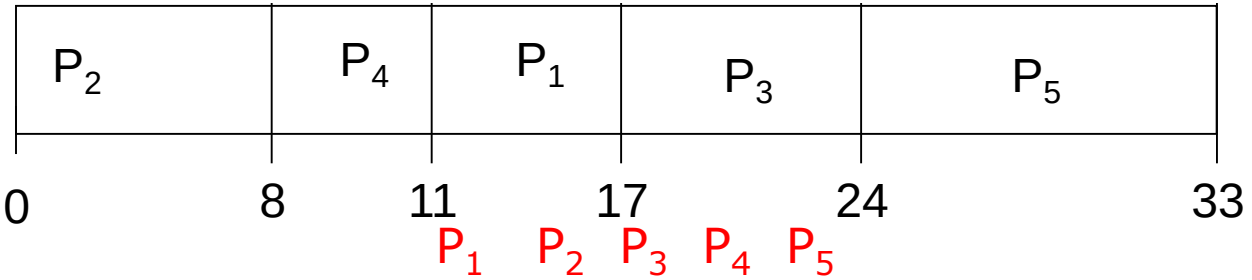


# Example of Priority Scheduling

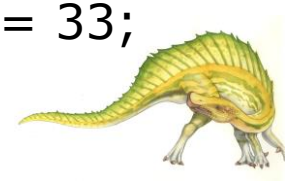
| Process | Burst Time | Priority |
|---|---|---|
| $P_1$ | 6 | 3 |
| $P_2$ | 8 | 1 |
| $P_3$ | 7 | 4 |
| $P_4$ | 3 | 2 |
| $P_5$ | 9 | 5 |

** ทุก Process มาถึงเวลาเดียวกัน

- priority scheduling chart



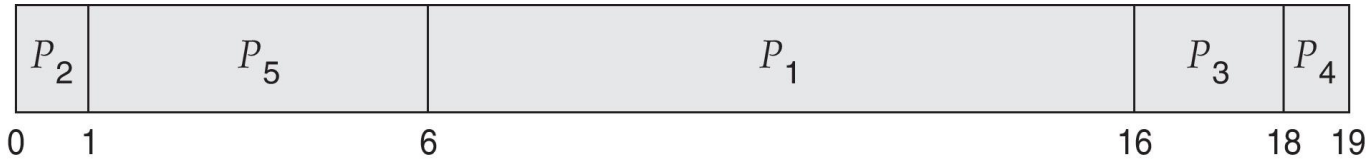

- Average waiting time = (11 + 0 + 17 + 8+ 24) / 5 = 12 mS
- Turnaround Time : $P_1 = 17$; $P_2 = 8$; $P_3 = 24$; $P_4 = 11$; $P_5 = 33$;



# Practice: 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



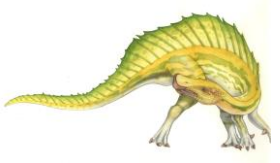

- Average waiting time = ???
- Turnaround Time : ???

# Practice: Priority Scheduling w/ 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 |

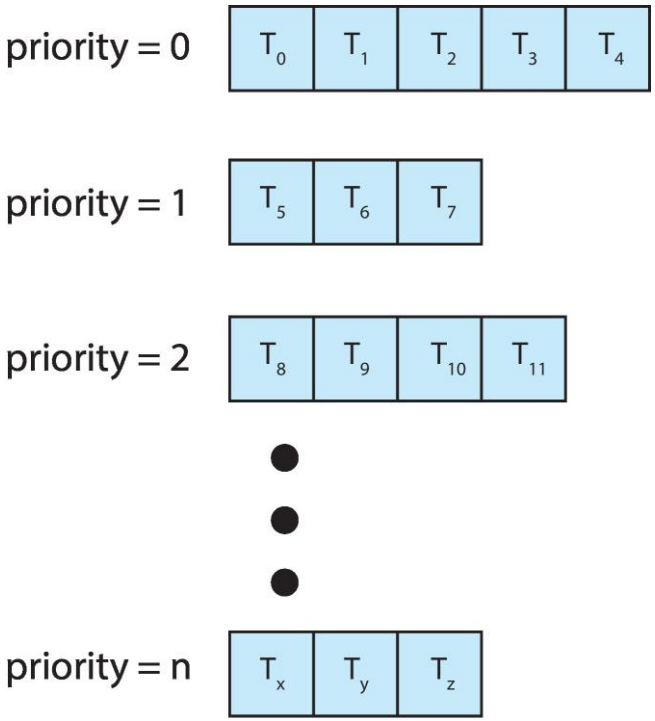
- Run the process with the highest priority. Processes with the same priority run round-robin

- จงเขียน Gantt Chart with 2 ms time quantum ???



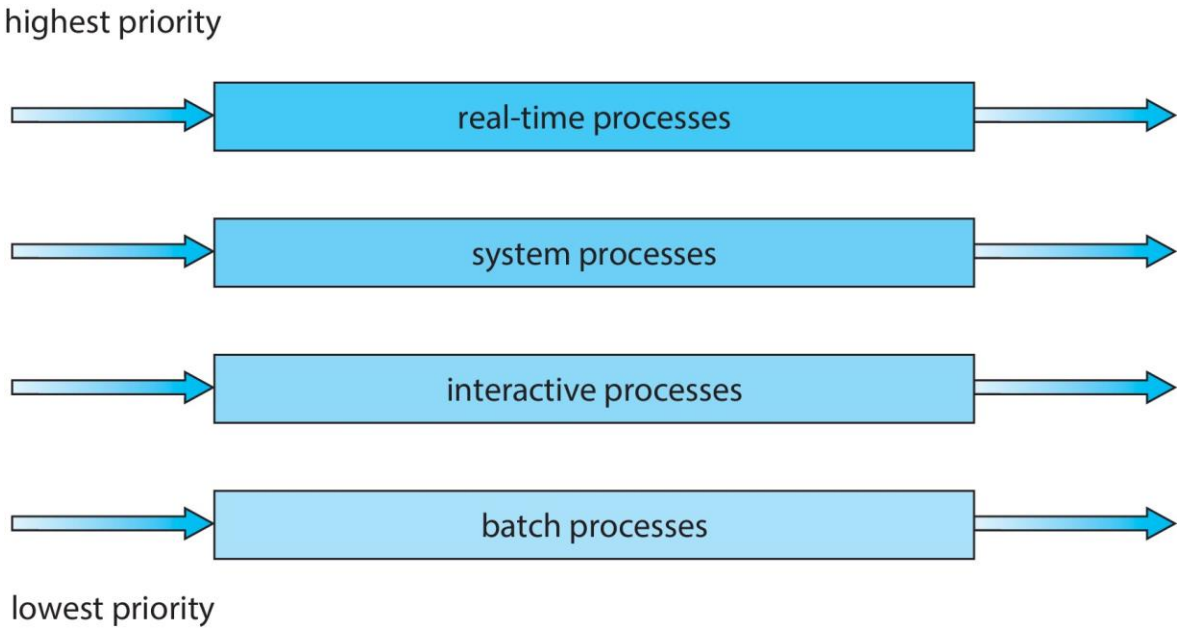
# Multilevel Queue

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




# Multilevel Queue

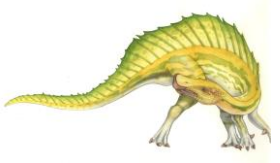
- Prioritization based upon process type




# Multilevel Feedback Queue

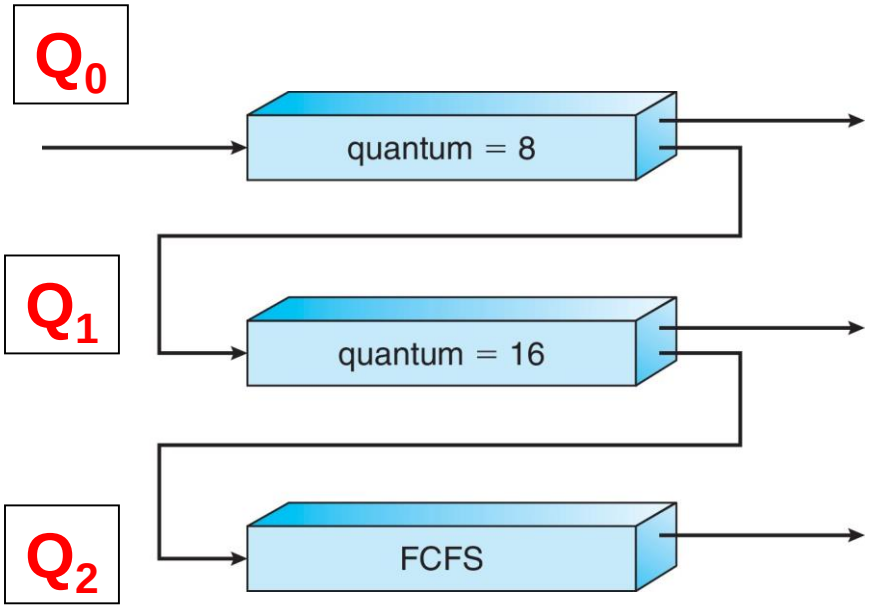
- A process can move between the various queues; aging can be implemented this way
- Multilevel-feedback-queue scheduler defined by the following parameters:
  - number of queues
  - scheduling algorithms for each queue
  - method used to determine when to upgrade a process
  - method used to determine when to demote a process
  - method used to determine which queue a process will enter when that process needs service

aging : เพิ่มศักดิ์ขึ้น
demote: ลดศักดิ์ลง

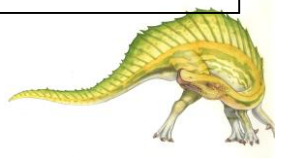


# Example of Multilevel Feedback Queue

- 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 $Q_2$




Queue ถัดมาไม่สามารถเริ่มทำงานได้ หาก Queue ก่อนหน้าทำงานไม่เสร็จ หรือ ยังไม่ empty หรือ ยังไม่หมดช่วงเวลาที่กำหนดให้ (time quantum)



# Thread Scheduling

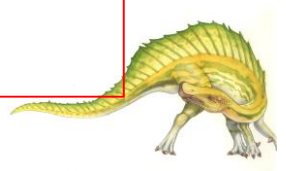
- Distinction between user-level and kernel-level threads
- When threads supported, threads scheduled, not processes
- Many-to-one and many-to-many models, thread library schedules user-level threads to run on LWP
  - Known as **process-contention scope (PCS)** since scheduling competition is within the process
  - Typically done via priority set by programmer
- Kernel thread scheduled onto available CPU is **system-contention scope (SCS)** – competition among all threads in system

LWP: Light-Weight Process เป็นโพรเซสที่ทำงานอยู่ในส่วนของ user space
ในกรณีที่มี LWP หลายตัวจะมีการใช้ทรัพยากรร่วมกันกับโพรเซสหลักด้วย

# Multiple-Processor Scheduling

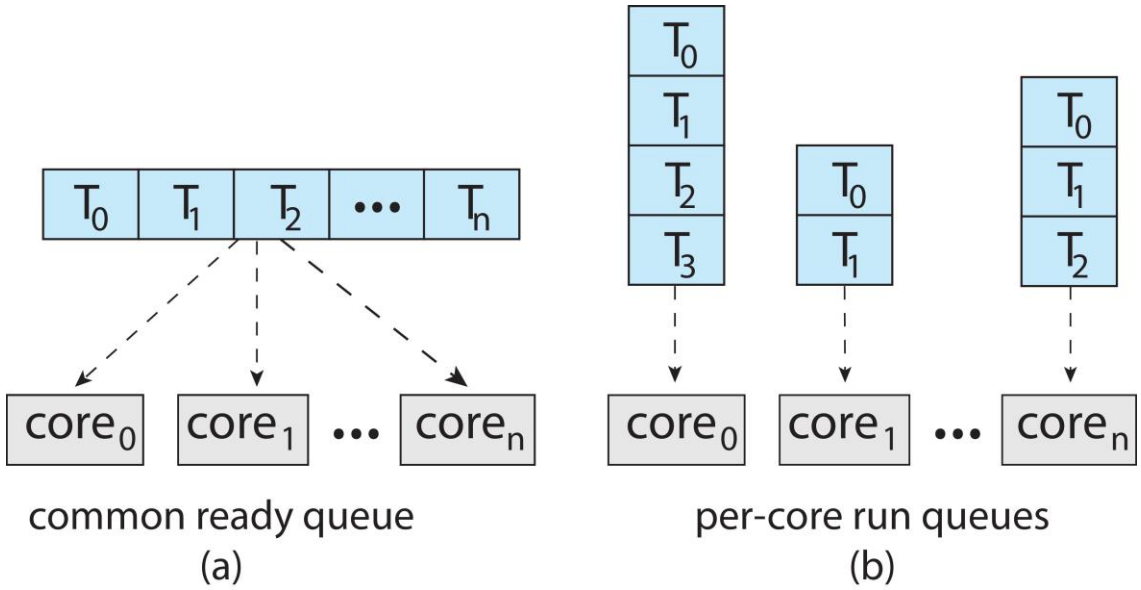
- CPU scheduling more complex when multiple CPUs are available
- Multiprocess may be any one of the following architectures:
  - Multicore CPUs
  - Multithreaded cores
  - NUMA systems
  - Heterogeneous multiprocessing

Homogeneous : แบบเดียวกัน เช่น cpu เป็น Intel เหมือนกัน
Heterogeneous : หลายแบบ เช่น cpu เป็น Intel, AMD, ultra spark



# Multiple-Processor Scheduling

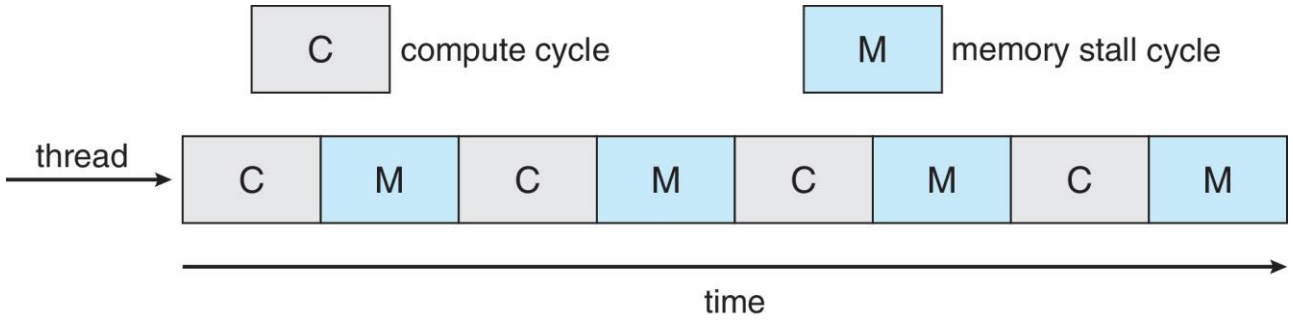
- Symmetric multiprocessing (SMP) is where each processor is self scheduling.
- All threads may be in a common ready queue (a)
- Each processor may have its own private queue of threads (b)

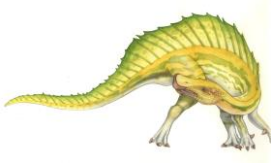



SMP: Symmetric Multiprocessing คือการทำงานที่แต่ละ Processor มีการจัดตาราง scheduling ของตัวเอง

# Multicore Processors

- Recent trend to place multiple processor cores on same physical chip
- Faster and consumes less power
- Multiple threads per core also growing
  - Takes advantage of memory stall to make progress on another thread while memory retrieve happens

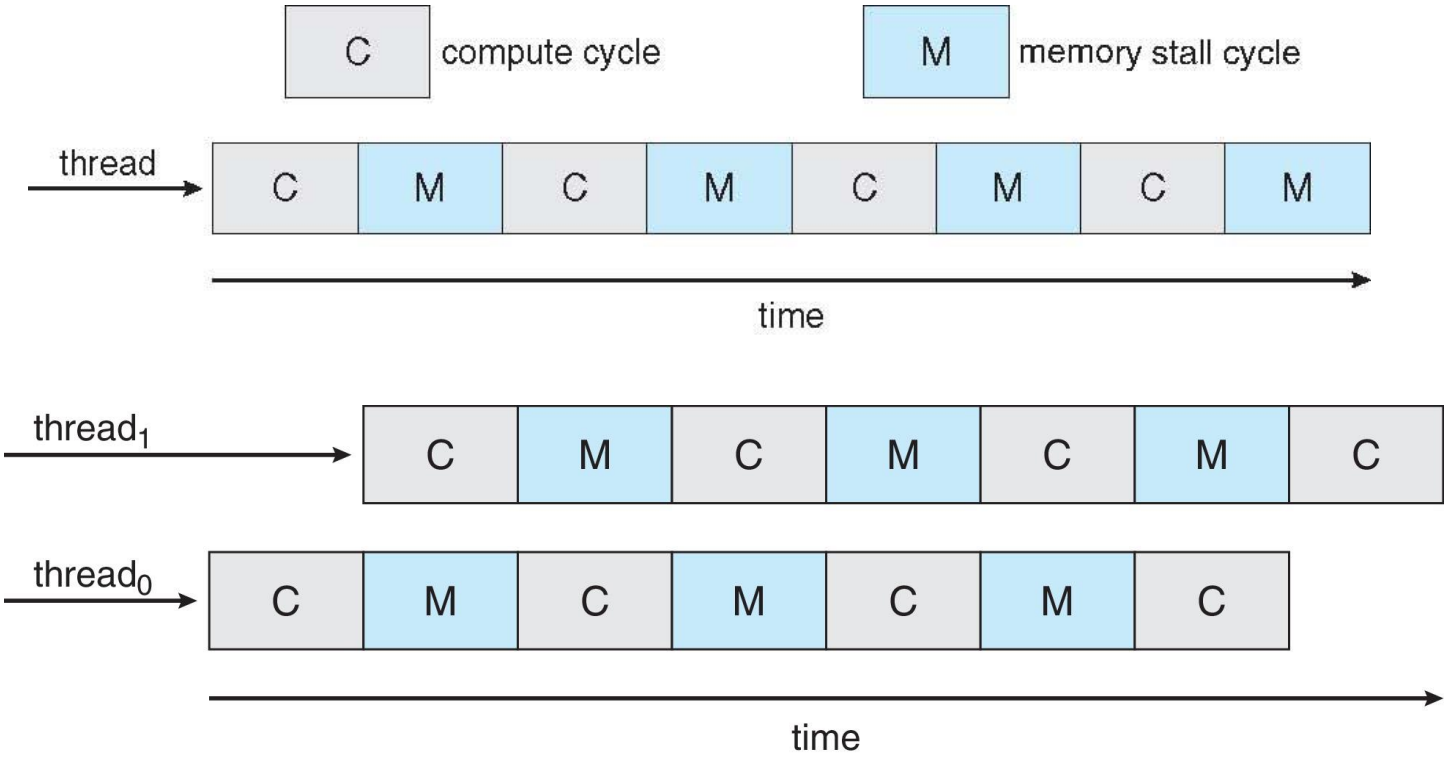


# Multithreaded Multicore System

Each core has > 1 hardware threads.

If one thread has a memory stall, switch to another thread!

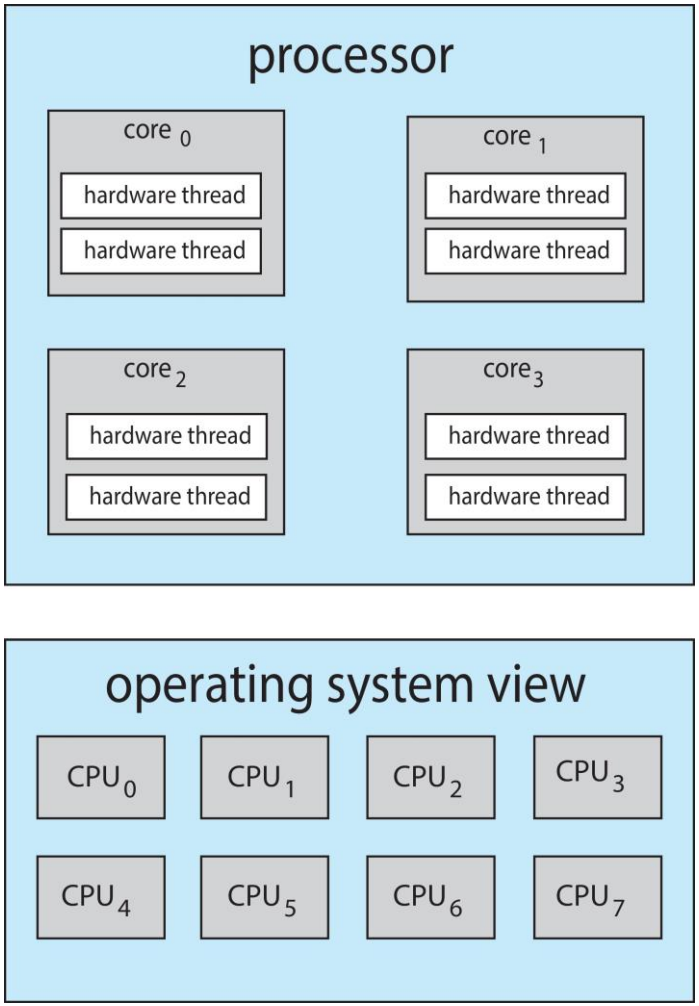


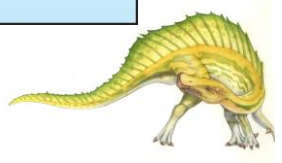

memory stall cycle : ช่วงเวลาที่ cpu ต้องรอการนำข้อมูลที่ไม่ได้อยู่ในหน่วยความจำให้ถูกload มาไว้ในหน่วยความจำ

# Multithreaded Multicore System

- **Chip-multithreading** (CMT) assigns each core multiple hardware threads. (Intel refers to this as **hyperthreading**.)

- On a quad-core system with 2 hardware threads per core, the operating system sees 8 logical processors.



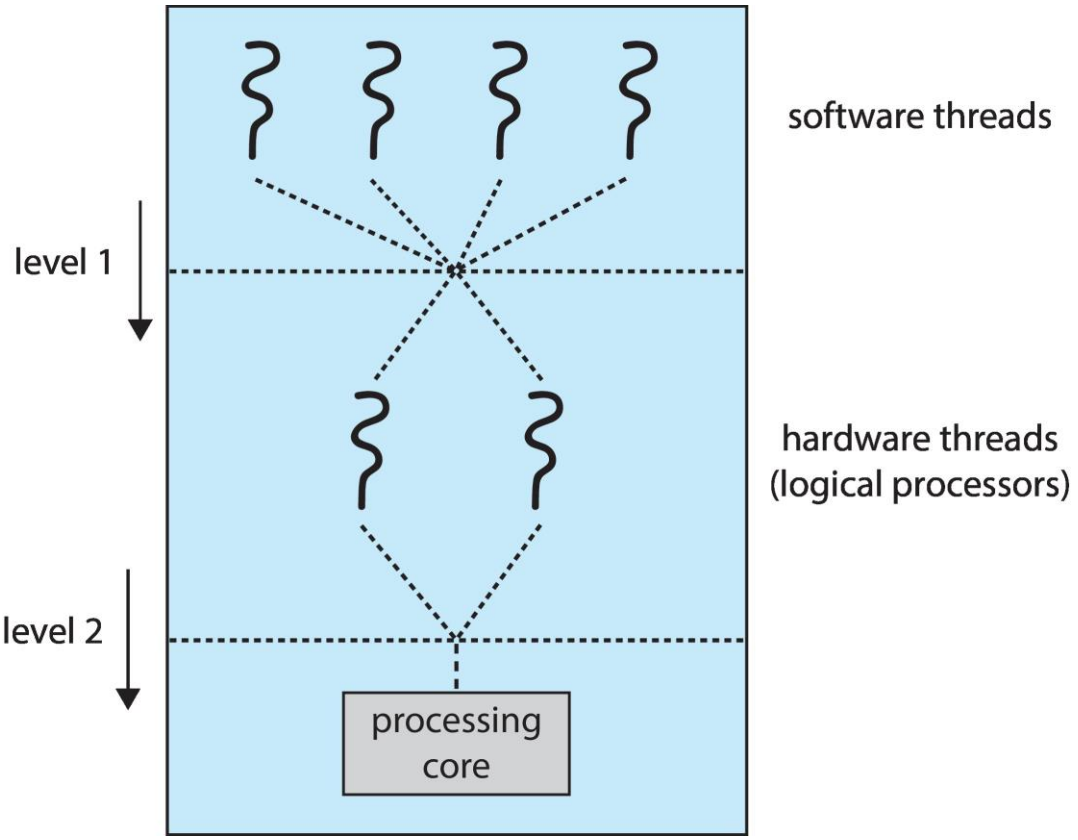



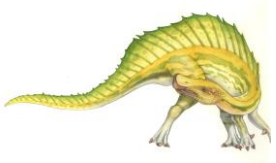
# Multithreaded Multicore System

- Two levels of scheduling:

1. The operating system deciding which software thread to run on a logical CPU

2. How each core decides which hardware thread to run on the physical core.



# Multiple-Processor Scheduling – Load Balancing

- If SMP, need to keep all CPUs loaded for efficiency
- **Load balancing** attempts to keep workload evenly distributed
- **Push migration** – periodic task checks load on each processor, and if found pushes task from overloaded CPU to other CPUs
- **Pull migration** – idle processors pulls waiting task from busy processor

SMP: Symmetric Multiprocessing คือการทำงานที่แต่ละ Processor มีการจัดตาราง scheduling ของตัวเอง

# Operating System Example

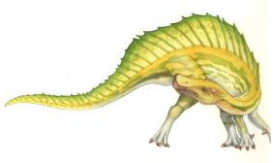
- Windows scheduling

# 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**

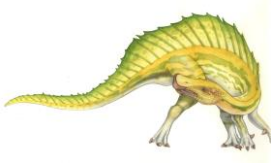
# Windows Priority Classes

- 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



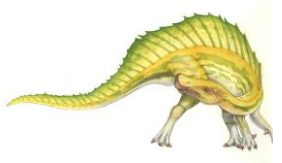
# Windows Priority Classes (Cont.)

- If wait occurs, priority boosted depending on what was waited for
- Foreground window given 3x priority boost
- Windows 7 added **user-mode scheduling** (**UMS**)
  - Applications create and manage threads independent of kernel
  - For large number of threads, much more efficient
  - UMS schedulers come from programming language libraries like C++ **Concurrent Runtime** (ConcRT) framework



# Windows Priorities

| | real-time | high | above normal | normal | below normal | idle priority |
|---|---|---|---|---|---|---|
| time-critical | 31 | 15 | 15 | 15 | 15 | 15 |
| highest | 26 | 15 | 12 | 10 | 8 | 6 |
| above normal | 25 | 14 | 11 | 9 | 7 | 5 |
| normal | 24 | 13 | 10 | 8 | 6 | 4 |
| below normal | 23 | 12 | 9 | 7 | 5 | 3 |
| lowest | 22 | 11 | 8 | 6 | 4 | 2 |
| idle | 16 | 1 | 1 | 1 | 1 | 1 |



# Algorithm Evaluation

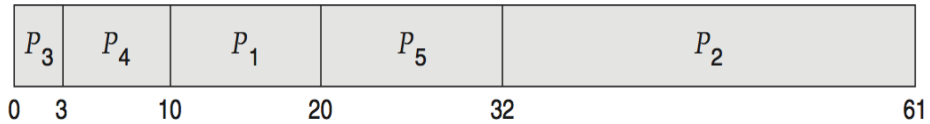
- How to select CPU-scheduling algorithm for an OS?
- Determine criteria, then evaluate algorithms
- **Deterministic modeling**
  - Type of **analytic evaluation**
  - Takes a particular predetermined workload and defines the performance of each algorithm for that workload
- Consider 5 processes arriving at time 0:

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

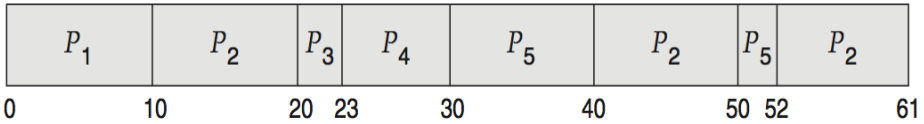
# Deterministic Evaluation

- 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:

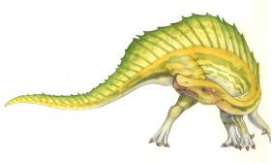



**การพิจารณาเปรียบเทียบ Scheduling Algorithm ที่ดีที่สุดสำหรับโพรเซสกลุ่มนี้ จะพิจารณาจากค่า average waiting time ของ Algorithm ที่มีค่าน้อยที่สุด**

**ดังนั้นในกรณีนี้ Algorithm Non-preemptive SJF ที่มีค่า 13 mS จึงดีที่สุด**

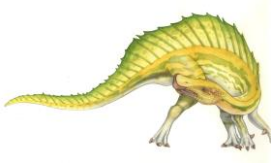
# Queueing Models

- 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

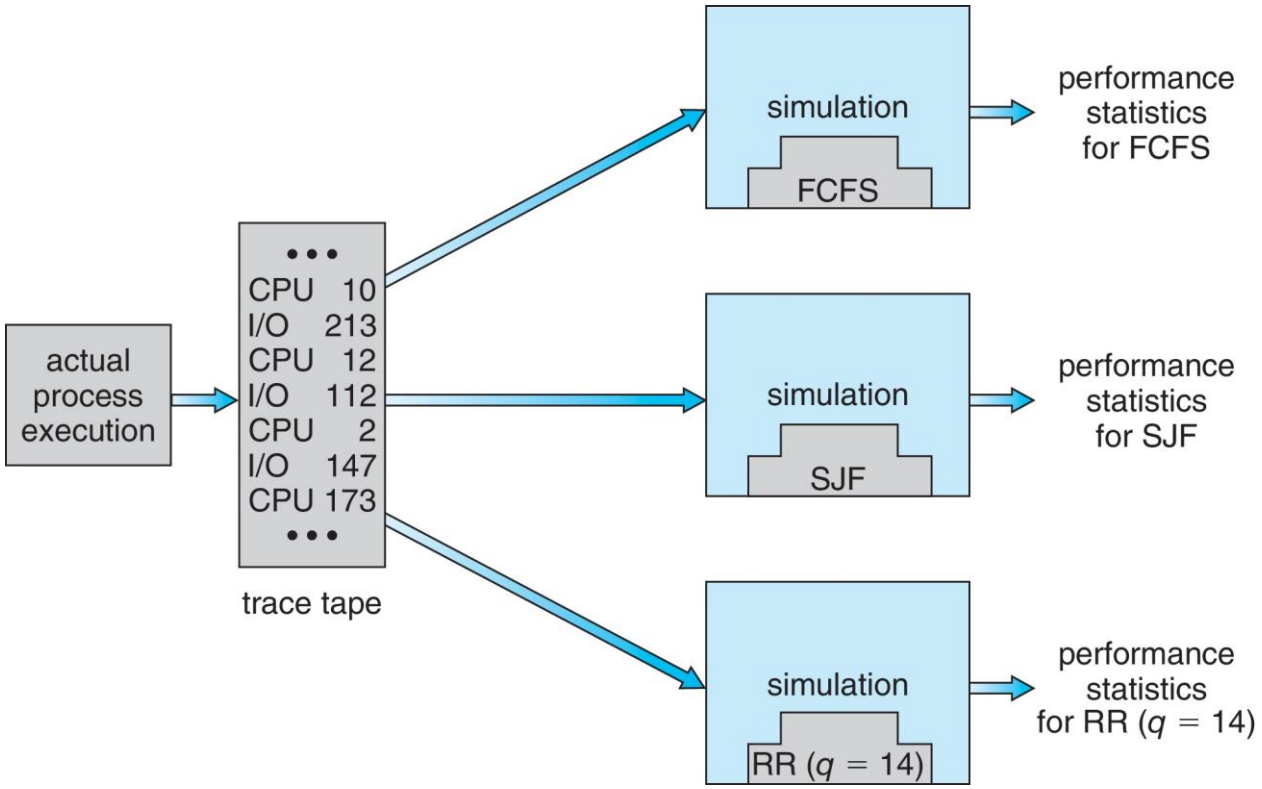


# Simulations

- 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

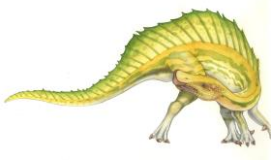


# Evaluation of CPU Schedulers by Simulation




# 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



# End of Chapter 4