



# 作業系統 **Operating System**

## Chapter 5 CPU Scheduling

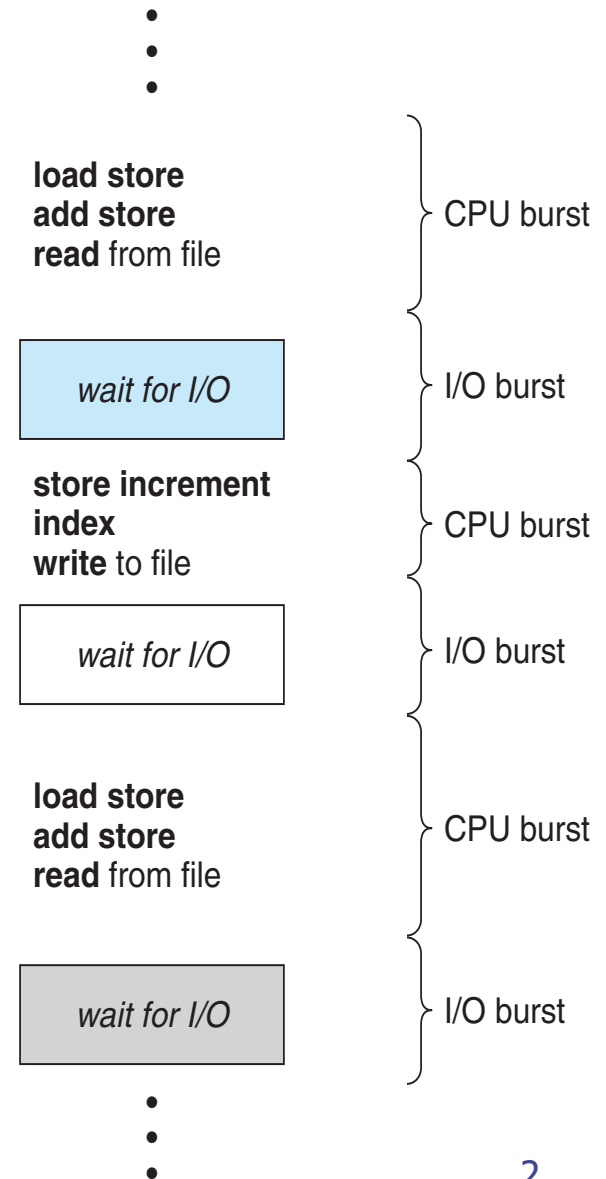


鍾武君

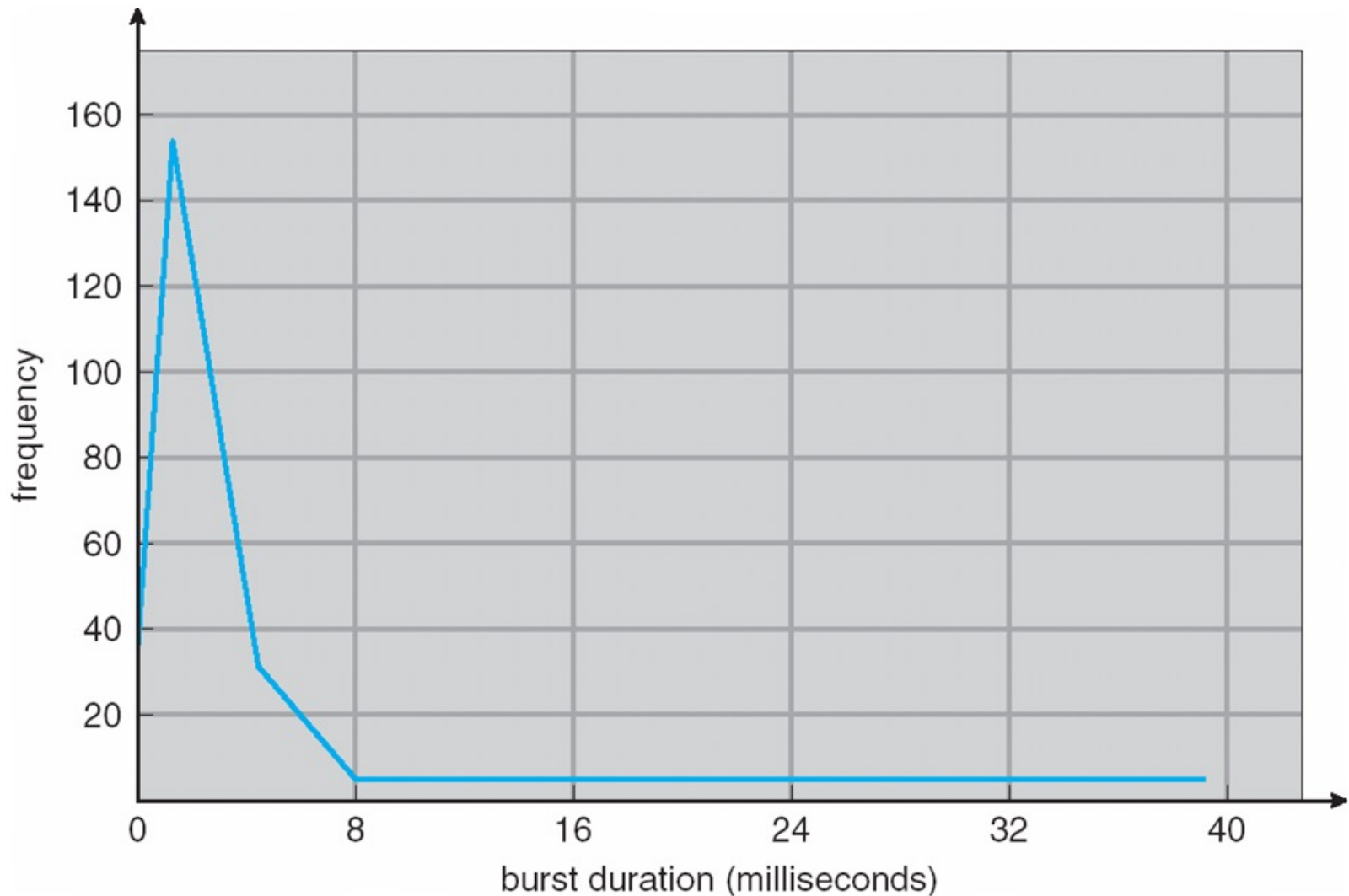
中原大學資訊工程系

# Basic Concepts

- CPU Burst
  - 使用CPU的時間
- I/O Burst
  - 使用I/O的時間
- I/O Bound Process
  - 小的CPU burst
  - Spends more of its time **doing I/O than computations**
- CPU Bound Process
  - 大的CPU burst
  - Use more of its time **doing computation than I/O**



# Histogram of CPU-burst Time



# Non-Preemptive and Preemptive (1)

- 有些排程是一旦處理程序擁有中央處理器，則必須讓處理程序執行完畢，才允許分派別的處理程序；有些則不須這麼沒有彈性
  - 不可奪取的（Non-Preemptive）
  - 可奪取的（Preemptive）

## Non-Preemptive and Preemptive (2)

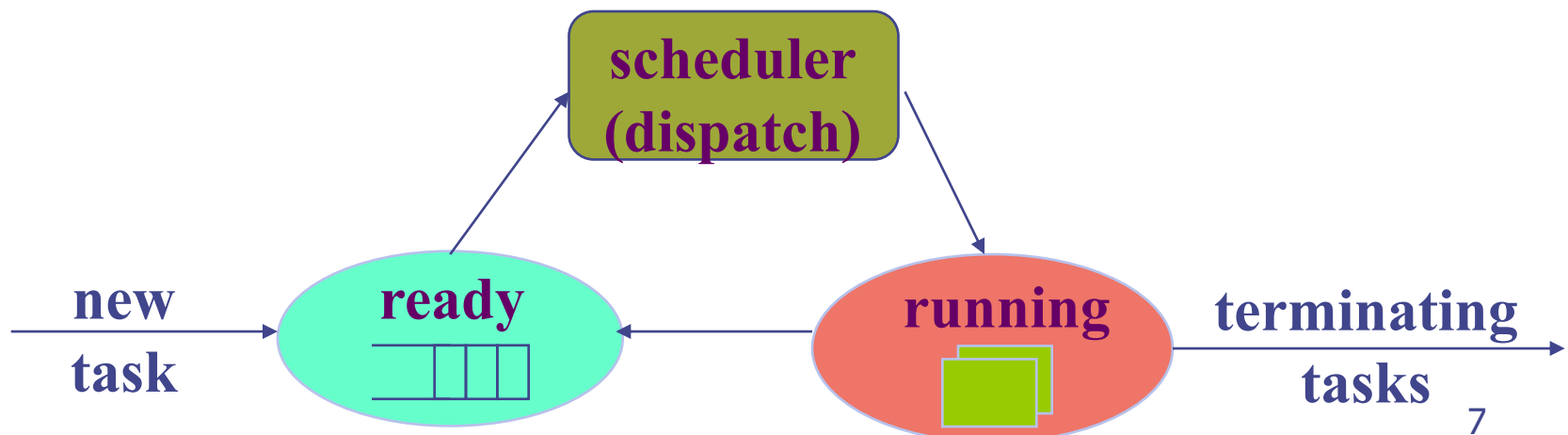
- 當處理程序生成之後，有些處理程序優先等級非常的高，一旦它佔有中央處理器之後，就不允許別的處理程序奪取中央處理器，我們稱此類處理程序為不可奪取處理程序（Non-Preemptive Process）
- 大部份處理程序均依照排程策略，允許中央處理器被奪取，我們稱這類處理程序為可奪取處理程序（Preemptive Process）

# Non-Preemptive and Preemptive (3)

- Non-Preemptive process
  - After interrupt processing is complete, it gets CPU again
  - 僅process結束或進入wait state，才可以讓別人使用
- Preemptive process
  - After interrupt processing is complete, context switch to another process
  - 除非ready queue內沒有其他process

# 分派時間 (Dispatch Latency)

- 分派程式停止某一個處理程序使用中央處理器，並分派中央處理器給另一個處理程序所需的時間，稱為分派時間 (Dispatch Latency)
- The time it takes for the dispatcher to stop one process and start another running.



# 中央處理器排程的選用標準 (1)

## Scheduling Criteria

- 儘量讓中央處理器不斷的忙碌，並有最高的中央處理器使用率（Maximum CPU Utilization）
- 使得系統有最高的產量（Maximum Throughput）
- 使得系統內最慢的反應時間是所有方法中最小的（Minimize the Maximum Response Time）
- 使得往返時間是最小的（Minimize Turnaround Time）
- 使得所有處理程序之平均等待時間是最小的（Minimize Average Waiting Time）
  - 此處的等待時間是指處理程序在備妥佇列內的時間



## 中央處理器排程的選用標準(2)

- 使得反應時間之變異數是最小的 (Minimize Variance in the Response Time)
- 使得交談使用者人數是最多的 (Maximum the Number of Interactive Users)
- 使得系統之負擔是最小的 (Minimize System Overhead)
- 使得資源被平衡的使用 (Balance Resources Used)
- 所有排程是公平的 (Be Fair)

# 先到先服務排程法(1)

## Scheduling Scheme

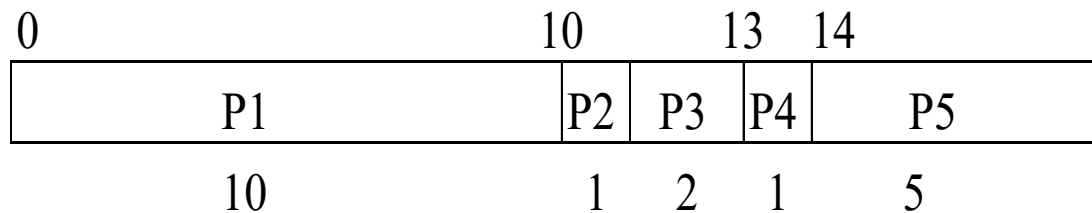
- 先到先服務（First Come First Served，簡稱為FCFS）排程法
  - 一種不可奪取的排程法
  - 不適合使用者進行交談
  - 不適用於分時系統

## 先到先服務排程法(2)

Process	CPU Burst Time
P1	10
P2	1
P3	2
P4	1
P5	5

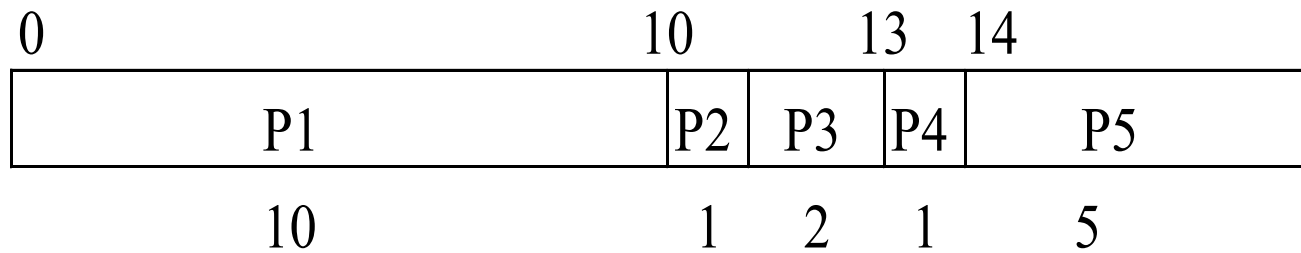
五個處理程序使用中央處理器時間

Gantt Chart



# 先到先服務排程法(3)

Gantt Chart



Process	Turnaround Time	Waiting Time
P1	10	0
P2	11	10
P3	13	11
P4	14	13
P5	19	14

先到先服務排程法之甘特圖、往返時間及等待時間

# 最短工作優先排程法 (1)

- 最短工作優先（Shortest Job First）排程法，簡稱為SJF
- 依照使用中央處理器時間由小至大依序排列至備妥佇列內，以進行中央處理器排程
  - 若處理程序有相同中央處理器時間，則依FCFS排列
- Smallest estimated run time to completion is run next
- 可被區分為不可奪取最短工作優先排程法（Non-Preemptive SJF）及可奪取最短工作優先排程法（Preemptive SJF）

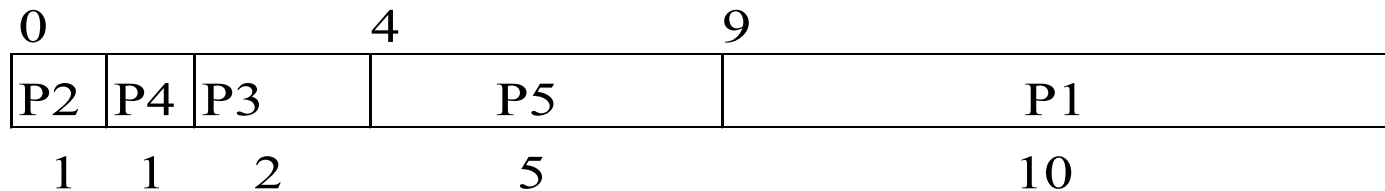
## 最短工作優先排程法 (2)

- 不可奪取最短工作優先排程法是指，在此排程法下之處理程序均為不可奪取處理程序
- 可奪取最短工作優先排程法則是，隨時注意是否有較小使用中央處理器時間之處理程序出現，若有，則將中央處理器讓給此使用中央處理器時間較小的處理程序
  - 最短剩餘時間優先 (Shortest Remaining Time First) 排程法，簡稱為SRTF
- Running process may be preempted by a new process with a **shortest estimated run time**

## 最短工作優先排程法 (3)

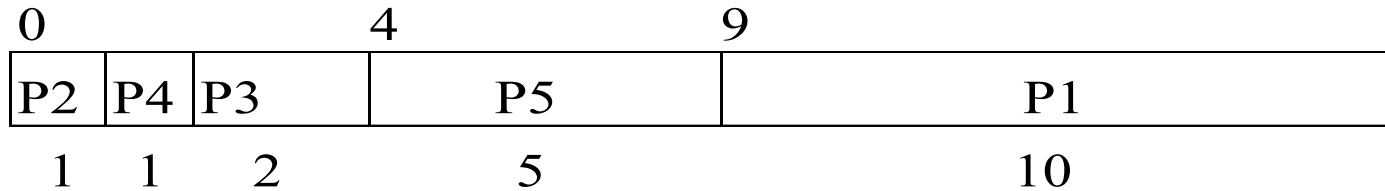
Process	CPU Burst Time
P1	10
P2	1
P3	2
P4	1
P5	5

五個處理程序使用中央處理器時間



最短工作優先排程法之甘特圖

# 最短工作優先排程法 (4)



最短工作優先排程法之甘特圖

Process	Turnaround Time	Waiting Time
P1	19	9
P2	1	0
P3	4	2
P4	2	1
P5	9	4

最短工作優先排程法之甘特圖、往返時間及等待時間



# FCFS vs. SJF

Process	Turnaround Time	Waiting Time
P1	10	0
P2	11	10
P3	13	11
P4	14	13
P5	19	14

先到先服務排程法之甘特圖、往返時間及等待時間

- 平均等待時間為 9.6ms

Process	Turnaround Time	Waiting Time
P1	19	9
P2	1	0
P3	4	2
P4	2	1
P5	9	4

最短工作優先排程法之往返時間及等待時間

- 平均等待時間為 3.2ms

## 最短工作優先排程法 (5)

- 最短工作優先排程法有最小的平均等待時間，所以若以等待時間來評估方法的好壞，最短工作優先排程法是一個最佳的演算法（Optimal Algorithm）
- It is difficult to implement, because no way to predict burst time.

# 可奪取最短工作優先排程法 (6)

Process	Arrival Time	CPU Burst
P1	0	4
P2	1	2
P3	2	4

Turnaround time =  
Finish time – Arrival time

0	1	3	6
P1	P2	P1	P3

Waiting time =  
Turnaround time – CPU burst

可奪取最短工作優先排程之甘特圖

Process	Turnaround Time	Waiting Time
P1	6	2
P2	2	0
P3	8	4

可奪取最短工作優先排程之往返時間及等待時間

# 最高反應時間比率優先排程法 (1)

- 反應時間比率（Response Ratio）與等待時間及中央處理器時間有關，它的公式為：

$$\text{Response Ratio} = \frac{\text{Waiting Time} + \text{CPU Burst Time}}{\text{CPU Burst Time}}$$

- 最高反應時間比率優先（Highest Response Ratio Next）排程法，簡稱HRRN，屬於不可奪取的排程法
- 以反應時間比率當作變動優先等級來排程，計算反應時間比率愈高的處理程序優先處理

## 最高反應時間比率優先排程法 (2)

Process	Arrival Time	CPU Burst
P1	0	8
P2	1	7
P3	2	3
P4	4	5

- 甘特圖

# 優先等級排程法 (1)

- 優先等級（Priority）排程法是給予每一個處理程序優先等級，並依照優先等級將處理程序由高優先等級排至低優先等級，然後依序分派處理程序擁有中央處理器
  - 當處理程序有相同的優先等級時，則採用先到先服務方式處理
  - 若有一個較高優先等級的處理程序後來才生成，它必須排在比它優先等級低的處理程序之前，以進行排程
  - 可分為可奪取優先等級排程及不可奪取優先等級排程
- CPU is allocated to the process with the highest priority. Equal priority processes are scheduled in FCFS order.

## 優先等級排程法 (2)

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

優先等級排程法

0	1	6	16	18
P2	P5	P1	P3	P4

Process	Turnaround Time	Waiting Time
P1	16	6
P2	1	0
P3	18	16
P4	19	18
P5	6	1



優先等級排程法之甘特圖，往返時間、及等待時間

## 優先等級排程法 (3)

- 當處理程序之優先等級很低，而系統內不斷有較高優先等級之處理程序出現，則低優先等級之處理程序可能造成永遠無法被執行的問題
  - 稱此低優先等級處理程序無限懸置（Indefinite Blocking）或餓死（Starvation）
- The major problem is **Indefinite Blocking** or **Starvation** for low priority process.
- 常用解法
  - 利用時間升級（Aging）機制來解決
  - 變動優先等級（Dynamic Priority）或 浮動優先等級（Floating Priority）
  - **Aging**: Gradually increasing the priority of processes that wait in the system for a long time.



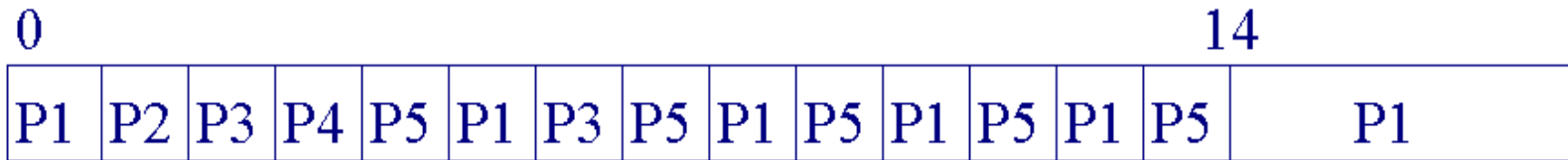
# 知更鳥式循環排程法 (1)

- 知更鳥式循環（Round Robin）排程法就是分時系統所使用之排程法
- Processes are dispatched FCFS but are given a **limited amount** of CPU time (**time slice**).
- **Preemptive**，適合 **time sharing**
- 時間片段的選擇
  - 時間片段太大  先到先服務排程法
  - 時間片段太小  系統的效率太差

不能太小：context switch 時間被佔滿，process 根本無法 run（或 run 極短時間）

## 知更鳥式循環排程法 (2)

- Time Slice = 1



Process	Turnaround Time	Waiting Time
P1	19	9
P2	2	1
P3	7	5
P4	4	3
P5	14	9

知更鳥式循環排程法之甘特圖、往返時間、及等待時間

# 知更鳥式循環排程法－練習

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

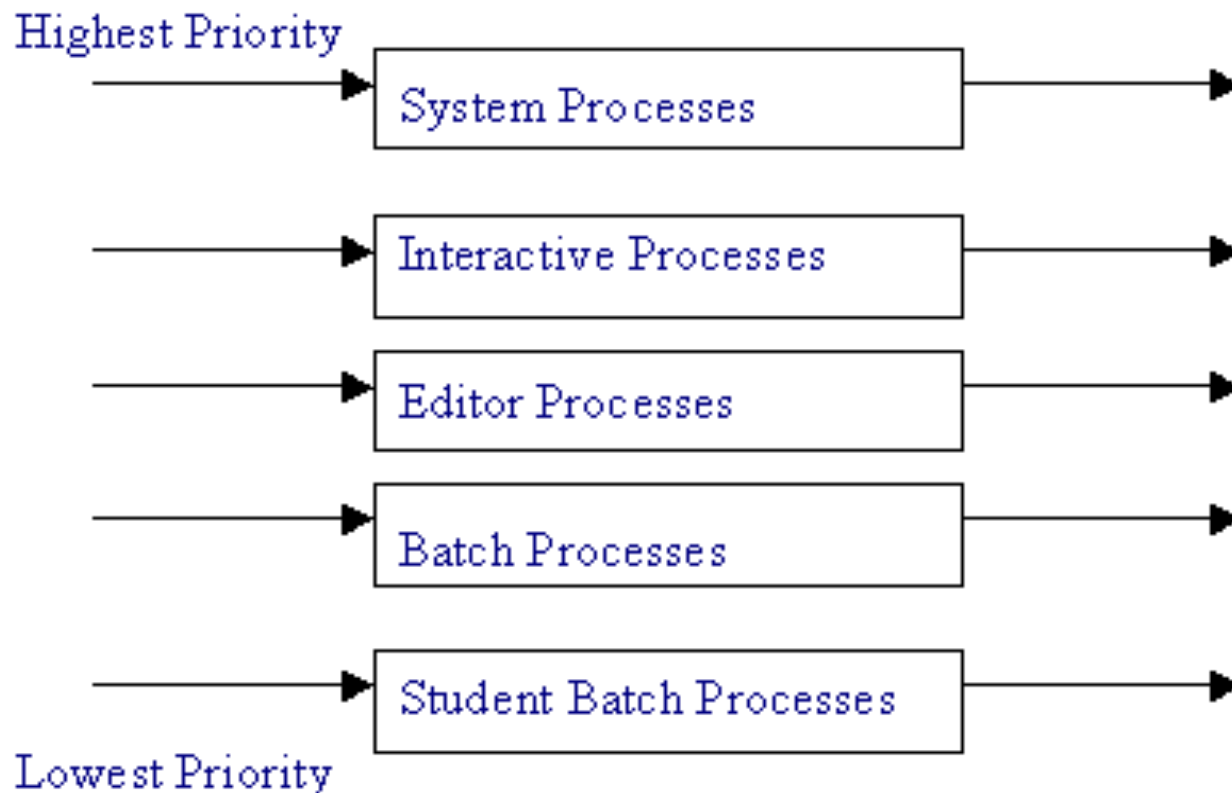
- Time Slice = 3
- 請計算平均等待時間為何？

# 多階層佇列(Multilevel Queue)排程法 (1)

- Partitions the ready queue into several separate queues.
- 當電腦系統內有許多不同優先等級編號之處理程序要執行，且亦有許多相同優先等級編號之處理程序要執行，則：
  - 最高優先等級之處理程序會先被執行
  - 若同時有多個最高優先等級之處理程序在系統內，則它們會優先在佇列內排隊等待被執行

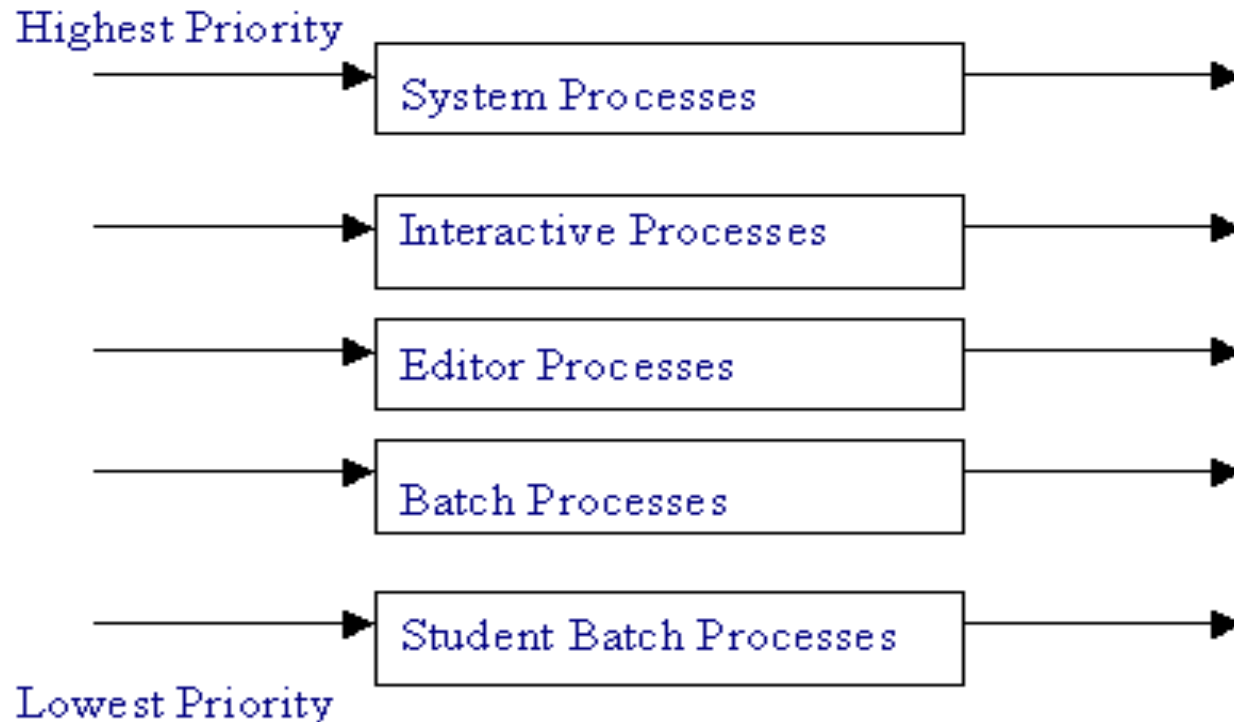
## 多階層佇列(Multilevel Queue)排程法 (2)

- Preemptive for different priority.
- Fixed priority for each queue.



## 多階層佇列(Multilevel Queue)排程法 (3)

- Foreground process
  - for interactive, high priority, RR scheduling.
- Background process
  - for batch, lower priority, FCFS.



## 多階層佇列(Multilevel Queue)排程法 (4)

- 在多階層佇列排程法內的處理程序一旦生成之後，就永遠固定優先等級
- 因此可能造成某些低優先等級之處理程序餓死的情形，因此在實務上並不太適用

# 多階層回饋佇列排程法

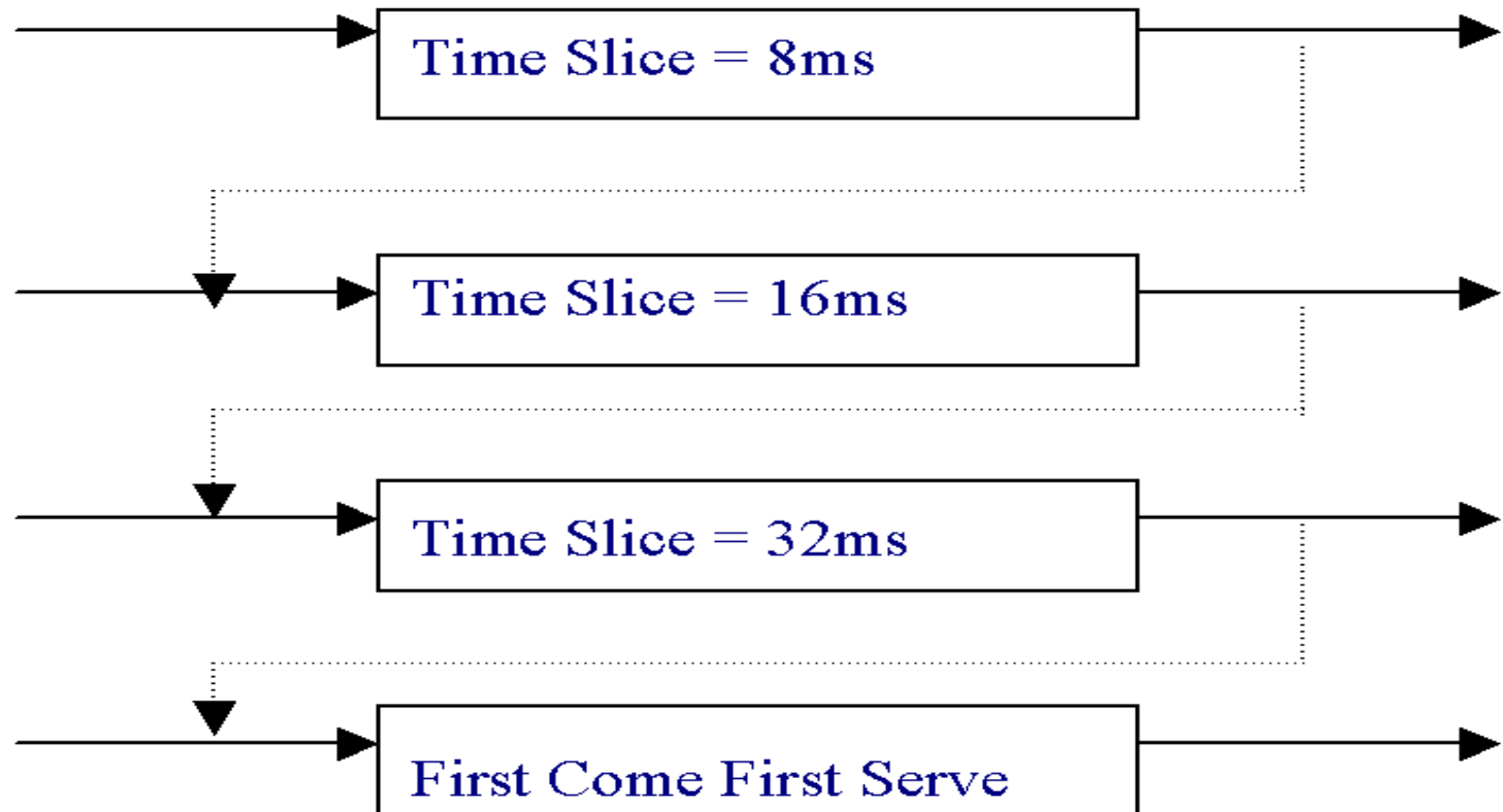
## (Multilevel Feedback Queue) (1)

- 此排程法中允許處理程序依時間升級機制，**提升至較高優先等級或降低至較低等級佇列中**
- The multilevel feedback queue scheduling allows a process to **move between queues**.
- 原則上：
  - **屬於系統之處理程序**可以固定於高優先等級，而**批次作業處理程序**可以固定於低優先等級，且它們不必使用時間升級機制；
  - **其餘處理程序**則依時間升級機制（Aging）游走於各個佇列中



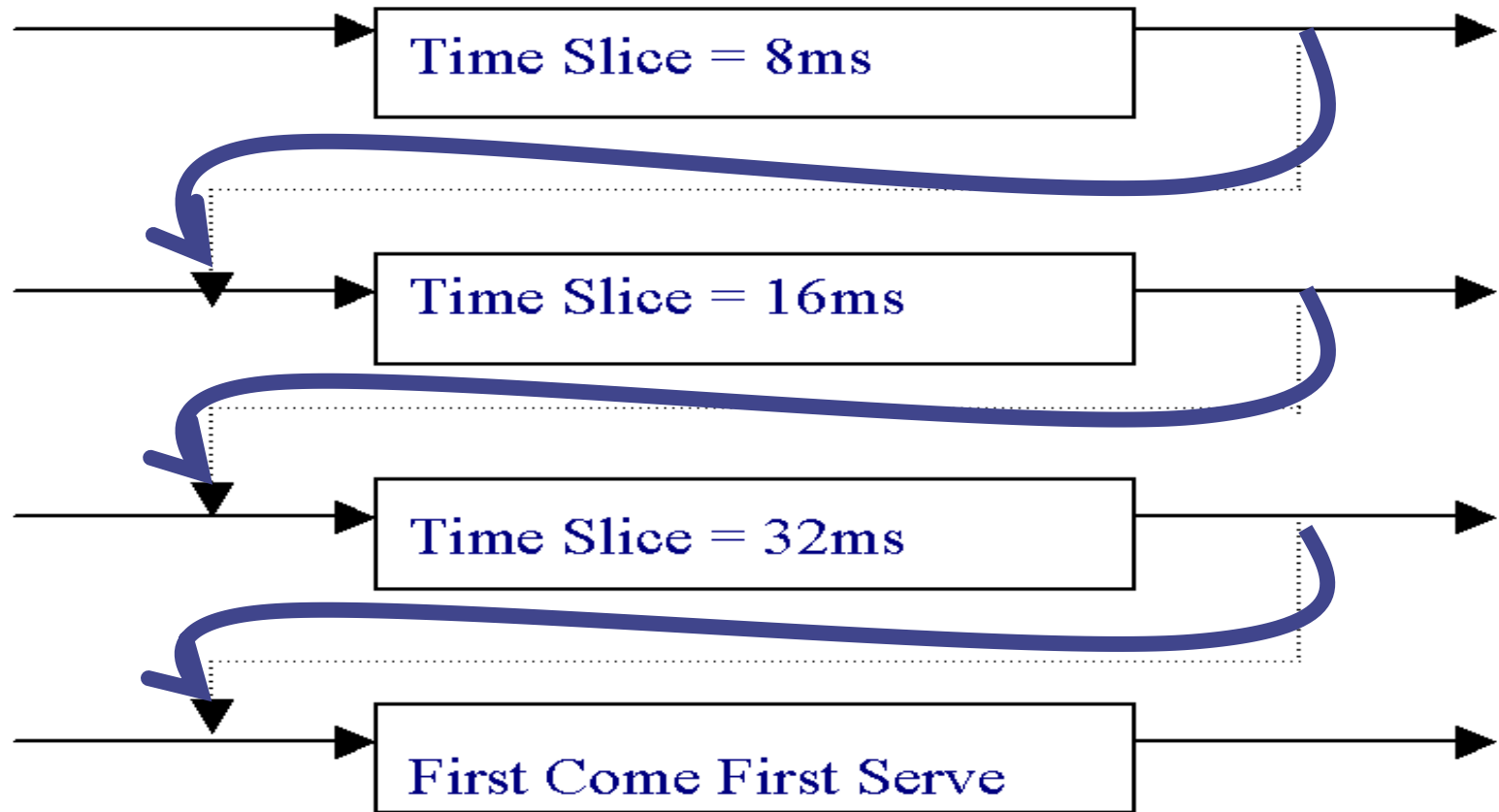
# 多階層回饋佇列排程法

## (Multilevel Feedback Queue) (2)



# 多階層回饋佇列排程法 (Multilevel Feedback Queue) (3)

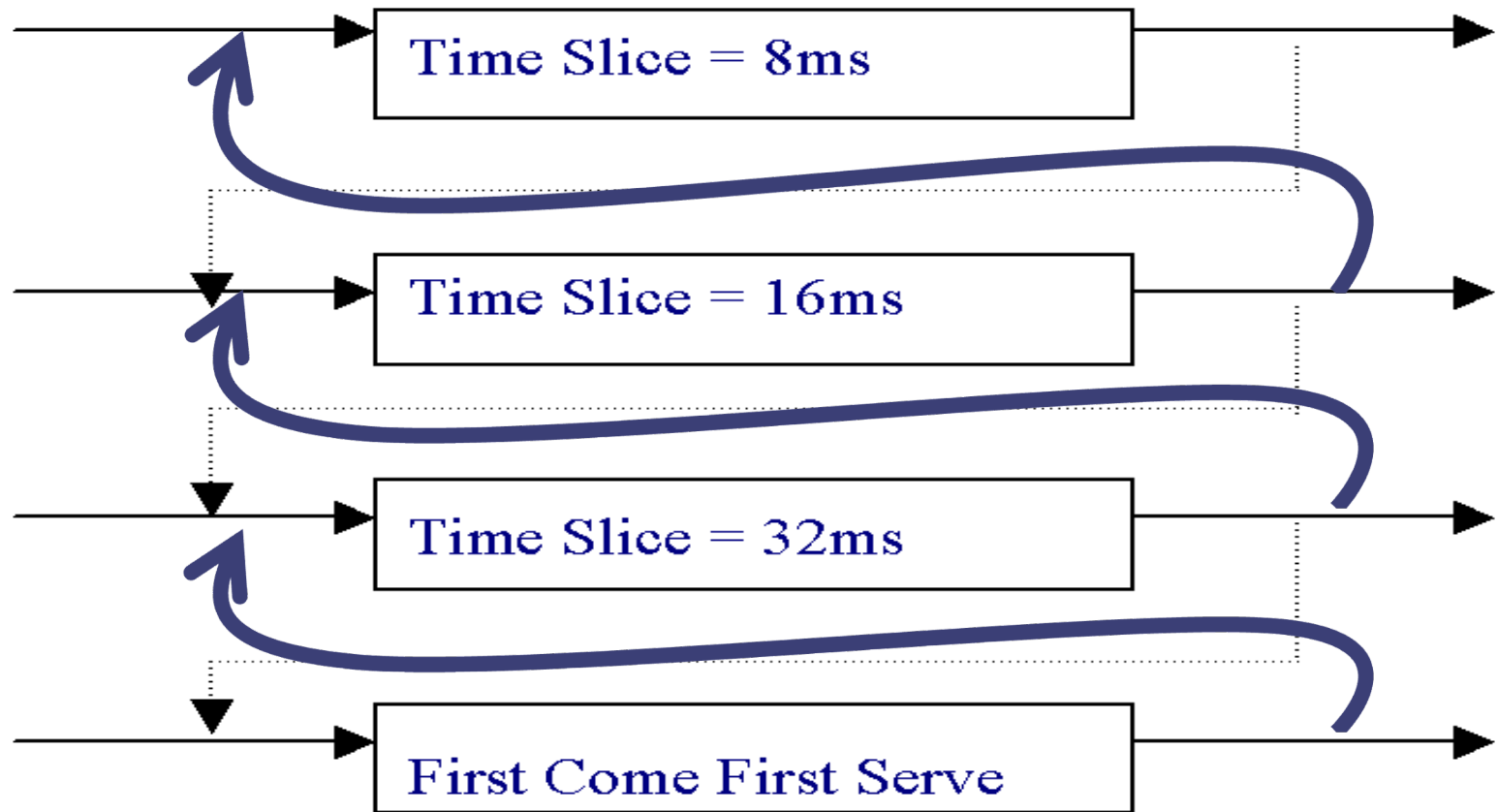
Run  $K$  次後，降級



# 多階層回饋佇列排程法

## (Multilevel Feedback Queue) (4)

waiting time > threshold , 升級



# 多階層回饋佇列排程法

## (Multilevel Feedback Queue) (5)

- 多階層回饋佇列排程法的特性
  - 可奪取排程法
  - 處理程序不會有餓死的情形發生
  - 中央處理器限制處理程序 (CPU Bound Process)  
(use too much CPU time process)
    - ◆ Small time slice，且隨著時間而降低優先等級
  - 輸出/輸入限制處理程序 (I/O Bound Process)  
(interactive, wait too long process)
    - ◆ Large time slice，且隨著時間提升優先等級

# 優先等級(Priority)

- 先到先服務排程法是以處理程序進入備妥佇列的時間先後次序，當作優先等級
- 不可奪取最短工作優先排程法是以預估處理程序使用中央處理器時間大小，當作優先等級
- 可奪取最短工作優先排程法是以處理程序剩餘使用中央處理器時間大小，當作優先等級
- 最高反應時間比率優先排程法是依其反應時間比率，當作優先等級
- 知更鳥式循環排程法則是以處理程序在備妥佇列的次序，當作優先等級

# 系統效能評估 (1)

- 定量模式 (Deterministic Modeling)
  - 事先設定好負載量的標準，並就現有環境提供確實的數據資料，然後利用這些數據資料評量各個演算法的效能，而得到對演算法效能的評比
  - Taken a particular **predetermined workload** and defines the performance of each algorithm for that workload.
  - 不一定可以適用在各個系統內，但對特定環境內的特定行為，大都可以反應出來

# 系統效能評估 (2)

- 佇列模式 (Queuing Modeling)
  - There is no static set of processes (and time) to use for deterministic modeling.
  - 使用佇列理論 (Queuing Theory) ，例如用 Little's formula 來解析評估

Little's formula 為  $n = \lambda \times \omega$

- ◆  $n$  代表在佇列內排隊的平均長度
- ◆  $\omega$  代表每個處理程序在佇列內的平均等待時間
- ◆  $\lambda$  代表新的處理程序平均出生率

# 系統效能評估 (3)

- 模擬法（Simulations）
  - 所謂模擬法就是撰寫模擬程式，並利用例如亂數產生器（Random Number Generator）產生所需資料，然後將此模擬資料送入模擬程式，以進行效能評估



# 參考資料

- 鍾斌賢,曾煜棋,顏春煌,”作業系統”,空中大學
- A. Silberschatz and P. B. Galvin, “Operating System Concepts”, 10th Edition. 新月
- Reference Books
  - A. N. Tanenbaum, "Modern Operating Systems", Prentice Hall.
  - A. Silberschatz, P. Galvin, and G. Gagne,”Applied Operating System Concepts”, John Wiley & Sons, Inc.