

作業說明

- 請在下列的 70 個問題中，挑選並完成其中 20 個題目。
- 其中每一章節需要完成至少 1/5 的題目(4, 2, 3, 2, 3 題)。
- 其餘六題可以自行挑選要完成哪個章節的那些題目。
- 每一題請標明章節-題號(例如 1-3, 5-4)並按照章節-題號順序作答。
- 請上傳完成的作業 pdf 檔案，可以使用手寫或打字。
- 可以參考投影片內容回答，但請不要直接複製投影片內容。
- 繳交截止時間 2023/4/6 09:10。

Chapter 1

1. ■請描述一個作業系統應該具備的基本目標(goal)?

A: OS 設計的基本目標應該要具備以下 4 種:

1. 提供一個讓 users 易於操作電腦之溝通介面。
2. 提供一個讓 users programs 易於執行之環。
3. 作為一個 resource 的管理者，協調分配 resource，使 resource 可以有效利用。
4. 作為一個監督者，監控所有 process 執行，避免 process 之有意 or 無意的破壞，使 system 有重大危害。

2. ■在電腦系統架構中，一共包含四個基本元素：硬體(Hardware)、作業系統(Operating System)、應用程式(Application)、使用者(User)，請描述這四者間的基本關係。

A: 使用者(User)對應用程式(Application)下命令，應用程式(Application)利用 system call 發中斷請求給 OS，並由 OS 分配硬體(Hardware)資源完成任務。

3. ■請解釋系統程式(System Program)和應用程式(Application Program)之間的差別。

A: 系統程式(System Program)是指附帶(ships)於 OS，但不屬於 kernel 的程式，通常被預先安裝在 OS 中，例如: Compiler、Assemble、Linking Loader、Debugger...

而應用程式(Application Program)則是與 OS 分離的程式，例如: Office、Word、Processer...

4. ■請解釋在電腦系統中，bus 的意義。

A: Bus，即匯流排，是指資料傳輸的通道，它可以讓不同的硬體元件（例如 CPU、記憶體、顯示卡、硬碟）之間透過共同的通道進行資料的傳輸。

5. ■請解釋甚麼是中斷向量表(Interrupt vector)

A: 中斷向量表(Interrupt vector)是指，在 Interrupt I/O 的運作流程中，當 I/O-operating 完成，Device controller 會發出”I/O-completed”interrupt 通知 CPU，CPU 偵測到中斷，確認中斷發生種類，會跳到中斷向量表(Interrupt vector)中所對應的 ISR 或 Interrupt handler 之 memory address 去執行 ISR；所以中斷向量表(Interrupt vector)中的欄位需要有中斷編號、ISR(或 Interrupt handler)之 memory address...

Chapter 2

1. ■請解釋何為 CLI，以及它的優點及缺點。

A: CLI(Command-Line Interface)，是指使用者操作介面以命令列的方式呈現，優點:效能較好、較精確，缺點:介面較難被接受

2. ■請解釋何為 GUI，以及它的優點及缺點。

A: GUI (Graphics User Interface)，是指使用者操作介面以圖形化的方式呈現，優點:介面較美觀、簡潔、易於使用，操作直觀，缺點:開發較費時、效能比較差

Chapter 3

- 對於一個執行中的程序，其記憶體部分可以分為 text section / Stack / Data / Heap，請分別說明這些區域中存放了何種資料。

A: text section 即 code section，存放程式碼

Stack 中存放該 process 的 local variables、parameters、return address

Data 中存放該 process 的 global variables(有初值與未設初值)

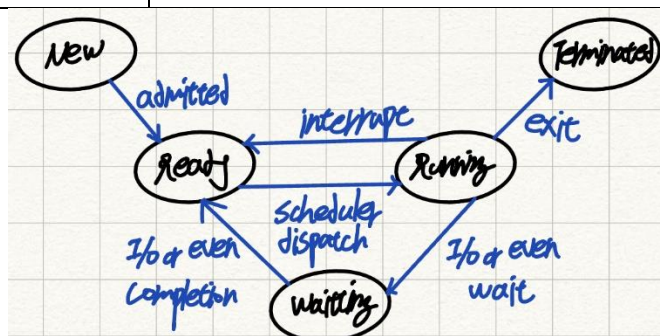
Heap 中存放向系統要求的額外配置之 Memory space(例：pointer 變數透過 malloc new 要求之空間)

- 請說明關於行程的五種狀態的名稱，以及五種狀態之間的關係。

A:

state	explain
New(Create)	Process 被建立，已分得 PCB 空間，尚未 Loading 進記憶體(or 尚未取得記憶體資源)
Ready	Process 已在記憶體中，且在 Ready Queue 內，具有資格爭奪 CPU
Running	Process 取得 CPU 執行中
Wait(Block)	Process 在 Waiting Queue 中 wait for I/O completed or event occurs(不會與其他 Process 爭奪 CPU)
Exit(Terminal、Zombie、Abort)	Process 完成工作，正常結束或異常終止，此時可能其 PCB 尚未回收，因為要等其 Parent Process collect 該 subprocess 的成果後，才會回收 PCB 空間，而其他 resource 已回收

Transition	explain
Admit	當記憶體空間足夠時，可由 Long-Term Scheduler (in 批次系統中)決定將此 Job Loading 進記憶體中
Dispatch	由 Short-Term Scheduler(CPU Scheduler) 決定讓高優先權 Process 取得 CPU
Time-out	也稱 Interrupt，執行中的 Process 會因為某些 event 發生，而被迫釋放 CPU，回 Ready Queue，例：Time-out、Interrupt、高優先權 Process 到達、插隊
Wait	Wait for I/O completed 或 event occurs
I/O-completed	I/O-completed 或 event occurs
Exit	Process 完成工作或異常終止



3. ■請說明什麼是 PCB，以及一個 PCB 中會包含哪些內容。

A: PCB(Process Control Block)OS 為了管理所有 Process，會在 kernel memory 中，替每一個 Process 各自準備一個 Block(表格)，記錄 Process 之所有相關資訊。PCB 主要內容有：

1. Process ID：每個 Process 具有一個唯一的(unique)ID
2. Process state：Ready、Running、Waiting...
3. Programming Counter：內放下一個要執行的 instruction's address
4. CPU Register：紀錄使用到暫存器的值，例：Accumulator、password、Stack、Top
5. CPU 排班資訊：紀錄 Process priority、Arrived time、CPU time、quantum...
6. Memory Management information：隨 OS 記憶體管理方法不同，記錄不同資訊。例：Base/limit Register 或 Page Table 或 Segment Table
7. Accounting information：紀錄 Process 已使用多少 CPU time、使用哪些資源、還剩多少 CPU time 可用...
8. I/O status information：紀錄 Process 已發出多少 I/O-request、完成狀況如何、目前暫用哪些 I/O 資源...

Chapter 4

1. ■請解釋 Process 和 Thread 之間的關係。

A: Thread 又稱”Lightening Process”，是 OS 分配 CPU Time 的對象單位，”Process”和”Thread”之間的關係好比”可行駛的汽車”與”引擎”的關係，每個 Process 至少存在一條 Thread，好比每台可行駛的汽車至少存在一顆引擎。

Process VS Thread

Heavy Weight Process	Lightening Process
Single-Threaded Model	Multithreaded Model
是 OS 分配 Resource 的對象單位	是 OS 分配 CPU Time 的對象單位
不同 Process 不會有共享的 Memory 與其他 Resource(除非採用 shared memory)	同一 Process 內之 Thread 彼此共享 Process 之 Memory 與其他 Resource
若 Process 內的 single Thread 被 Blocked，整個 Process 亦被 Blocked	只要 Process 內尚有 available Thread 可執行，整個 Process 不會被 Blocked
Process 之 Creation、Context Switching 慢，管理成本高	Thread 之 Creation、Context Switching 快，管理成本低
對於 Multiprocessors 架構之效益發揮較差	對於 Multiprocessors 架構之效益發揮較佳
Process 無此議題	因為同一 Process 內之 Thread 彼此共享 Process Data section，所以必須對共享的 Data 提供互斥存取機制，防止 race condition

2. ■請描述衡量一個工作的可平行度的 Amdahl's Law，並說明公式中每個部分的意義。

A: $speedup \leq \frac{1}{s + \frac{1-s}{N}}$ ，其中 S 代表 serial portion，N 代表 Processing croes 數

量，此公式是在計算在 N 個 Processing croes 下，平行化程度的比例對執行

速度的影響，其中由公式得知， $\lim_{N \rightarrow \infty} \frac{1}{s + \frac{1-s}{N}} = \frac{1}{s}$ ，代表真正影響平行化效率

的因素不在 Processing croes 數量，而是在於平行化的程度多寡。

Chapter 5

1. ■請說明 CPU 排程器介入排程的四種時機點。

A: CPU Scheduler 即為 Short-Term Scheduler(短期排班器)，目的是從 Ready Queue 中，挑出一個高優先權 Process 並分派 CPU 給其執行。而 CPU Scheduler 有 4 個介入的時機點：

1. Running→Wait (Wait)
2. Running→Ready (Time out)
3. Wait→Ready(I/O completed、Wake up)
4. Terminat (Exit)

其中的一跟四在排程時，是不能中斷的，二跟三是可以中斷，但需要考慮到是否有 shared data、是否在 kernel mode，還有他們是不是真的能中斷。

2. ■請說明 Preemptive 和 Nonpreemptive 的主要差別。

A:

	Nonpreemptive	Preemptive
定義	除非執行中的 Process 自願放掉 CPU，其他 Process 才有機會取得 CPU，否則就只能 wait，不可逕自搶奪 CPU	執行中的 Process 有可能被迫放棄 CPU 回到 Ready Queue，將 CPU 切給別的 Process 使用
優點	<ol style="list-style-type: none">1. Context switching 次數較少2. Process 完工時間點較可以預期(predictable)3. 比較不會產生 Race condition	<ol style="list-style-type: none">1. 排班效能較佳，平均 waiting、turnaround time 較小2. 適合用在 Time-sharing System 與 Real-Time System
缺點	<ol style="list-style-type: none">1. 排班效能較差(可能有 Convoy Effect)2. 不適合用在 Time-sharing System 與 Real-Time System	<ol style="list-style-type: none">1. Context switching 次數較多2. Process 完工時間較不可預期3. 需注意 Race condition 發生

3. ■請解釋 Dispatcher 的功用及作業流程。

A: Dispatcher(排程器)，是在 CPU 排程功能中的一個重要元件，Dispatcher 是一塊 Module，由他把 CPU 的控制權授予經由 Short-Term Scheduler(短期排班器)所選出的 Process。Dispatcher 的功能包括：

1. Switching context
2. Switching to user mode
3. Jumping to the proper location in the user program to restart that program

上述 3 個工作的時間加總，即為 Dispatch latency(分派延遲)。

4. ■在 CPU 排程中，其中一種衡量的指標是 Response Time，請說明 Response Time 的定義。

A: Response Time(回應時間)即為 user(user program)input 命令/Data 給系統到 System 產生第 1 個回應的時間差。

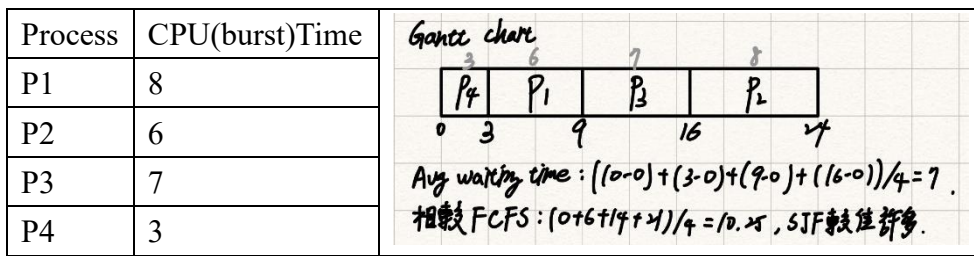
5. ■在 CPU 排程中，其中一種衡量的指標是 Waiting Time，請說明 Waiting Time 的定義。
- A: Waiting Time(等待時間) 即為 Process 花在 Ready Queue 中等待獲得 CPU 之等待時間加總。
6. ■請說明排程演算法中 FCFS(First-Come-First-Service) Scheduling 的運作方式。
- A: FCFS 的做法是：到達時間最小的 Process 會最優先取的 CPU。FCFS 的分析如下：
1. 排班效能最差，即 average waiting time、average turnaround time 最長，但製作簡單(使用 Ready Queue 即為 FCFS)
 2. 可能有"Convoy Effect"(護衛效應)
(護衛效應:許多 Process 均在等待一個需很長 CPU Time 之 Process 完成工作，才能取得 CPU，造成 average waiting time 痕長之不良現象)
 3. 公平
 4. No Starvation(沒有飢餓現象)
 5. Non-Preemptive 法則(不可插隊、不可搶先)

例:P1、P2、P3 依序且近乎同時到達

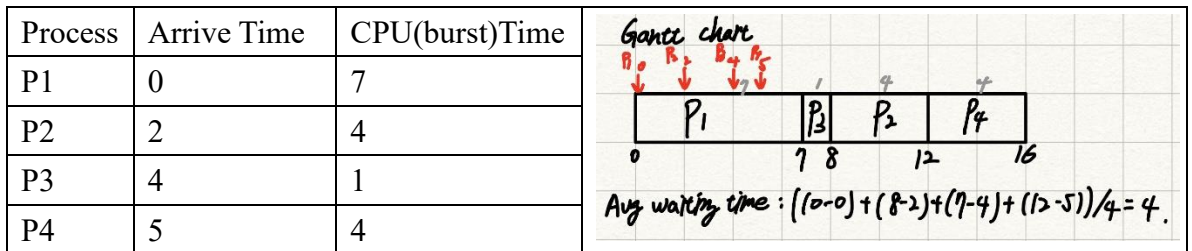
Process	CPU(burst)Time	1. Gantt chart
P1	24	<p>2. Avg waiting time: $(10-0) + (27-0) + (27-0) / 3 = 19$</p> <p>3. Avg turnaround time: $((127-0) + (27-0) + (30-0)) / 3 = 27$</p>
P2	3	
P3	3	

7. ■請說明排程演算法中 SJF(Shortest-Job-First) Scheduling 的運作方式。
- A: SJF 的做法是：具有最小時間的 CPU Time 之 Process 優先取得 CPU。SJF 的分析如下：
1. 排班效益最佳(optimal)，即 average waiting time、average turnaround time 最小(Why?因為 Short Job 所減少的 waiting time 必定大於等於 Long Job 所增加的 waiting time，使 average waiting time 為最小)
 2. 不公平，偏好 Short Job
 3. 可能會 Starvation(for Long Jobs)
 4. 可分成
 1. Nonpreemptive:以 SJF 做代表
 2. Preemptive:另外稱作 SRTF
 5. 較不適合用在 Short-Term Scheduler

例 1: P1、P2、P3、P4 依序且近乎同時到達



例 2: P1、P2、P3、P4 不同時到達



8. ■ 在估計一個行程所需的執行時間時，可以使用指數平均(Exponential Averaging)的方式進行，請描述指數平均的計算方式。

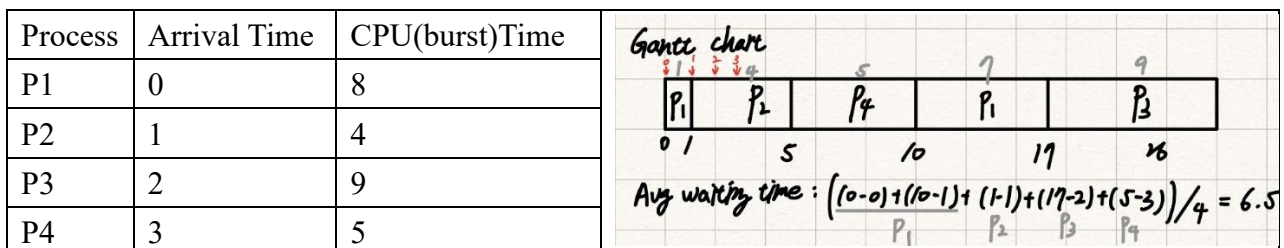
A: $\tau_{n+1} = t_n + (1 - \alpha) * \tau_n$ (下次預估值 = 本次實際值 + (1-加權值) * 本次預估值)，其中 $0 \leq \alpha \leq 1$

9. ■ 請說明排程演算法中 SRF(Shortest-Remaining-First) Scheduling 的運作方式。

A: SRTF(Shortest-Remaining Time First) 即為 "Preemptive SJF" 法則，由剩餘的 CPU burst Time 最小者取得 CPU，也就是若新到達之 Process，其 CPU burst Time 小於目前執行中 Process 的剩餘 CPU Time，則新到達之 Process 可以插隊(Preemptive)執行。SRTF 的分析如下：

1. 與 SJF 相比，SRTF 之平均 waiting、turnaround time 會較小，但 Context Switching 的負擔較大
2. 不公平、偏好 Short Running-Time Job
3. 會有 Starvation
4. Preemptive 法則

例：



10. ■請說明排程演算法中 RR(Round-Robin) Scheduling 的運作方式。

A: OS 會設定一個可參數化的 CPU Time Quantum(or Slice)，當 Process 取的 CPU 執行後，若未能在此 Quantum 內完成工作，則 Timer 會發出 Time-out Interrupt 通知 OS，OS 會牆破此 Process 放掉 CPU 且回 Ready Queue 中，等待下一輪再取得 CPU 執行，每一輪之中，Process 是以 FCFS 排班方式取得 CPU。RR 的分析如下：

1. Time-sharing System 採用、是一個可參數化的法則
2. 公平(FCFS)
3. NO Starvation
4. Preemptive 法則(Running to Ready)
5. RR Scheduling 效益取決於 Time Quantum 大小之制定

11. ■請說明排程演算法中 RR(Round-Robin)中的參數 q 如何影響排程。

A: 參數 q 即為 Quantum 值的大小，若 q=無窮大，則 RR 會退化成 FCFS 法則，排班效能較差。若 q=極小值，則 Context switching 太頻繁，CPU utilization 趨近於 0。依經驗法則，若 Quantum 值能讓 80%的 Jobs 在 Quantum 內完工，效能較佳。

12. ■請說明排程演算法中 PR(Priority) Scheduling 的運作方式。

A: Priority Scheduling 作法是讓具有 Highest Priority 之 Process 優先取得 CPU，若多個 Process 權值相同，以 FCFS 為準。Priority Scheduling 的分析如下：

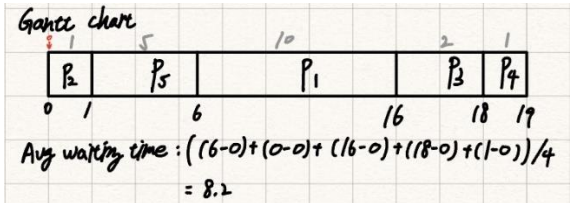
1. 是一個具參數化的法則，即給予不同的 Priority 高低定義，可展現出不同的排班行為。例：

Priority 定義	行為	包含於關係
Arrived time 越小，優先權越大	FCFS	FCFS \subset Priority
CPUtime 越小，優先權越大	SJF	SJF \subset Priority
剩餘時間越小，優先權越大	SRTF	SRTF \subset Priority

2. 不公平
3. 會有 Starvation(可用 Aging 解決)
4. 分為 NonPreemptive 與 Preemptive 2 種

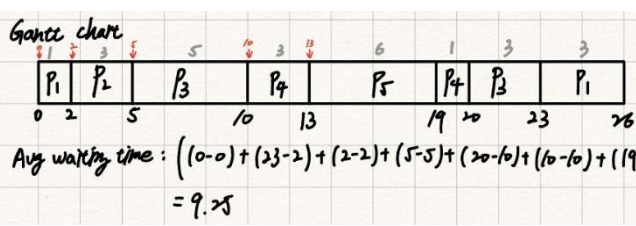
例 1:依序且近乎同時到達(越小的 Priority Number 有越高 Priority)

Process	CPU(burst)Time	Priority Number
P1	10	3
P2	1	1
P3	2	3
P4	1	4
P5	5	2



例 2:

Process	Arrival Time	Priority Number	CPU(burst)Time
P1	0	5	5
P2	2	2	3
P3	5	4	8
P4	10	3	4
P5	13	1	6



13. ■請說明 Multiple-Level-Feedback Queue 的運作方式。

A:與 Multilevel Queue 相似，差別只在允許 Process 在不同 Queue 之間移動，所以可以採取類似”Aging 技術”(優先權升高)：每隔一段時間，就將 Process 全部往上移一層 Queue，所以在有限的時間之後，Lowest Priority Process 會上升到 highest Priority Process

14. ■在負載平衡(load balancing)中可以進行的操作分為 Pull-migration 和 Push-migration，請解釋兩者的差異。

A:實施 Load Balance 有兩個調節機制：

1. Push-migration(Top-down):

Kernel 會定期確認各 CPU 的 Work load，將 loading 重的 CPU 之 Ready Queue 內的一需 Process 到 idle(或負擔較輕的)CPU 之 Ready Queue。

2. Pull-migration:

當 CPU idle(loading 低)，會發出請求給 Kernel，請 Kernel 拉一些 Process 給該 CPU 執行。

由上述可知 Pull-migration 和 Push-migration 的差異在於 CPU 是主動還是被動的去要求 Process。

15. ■在負載平衡(load balancing)中，親和性(affinity)會影響到負載平衡的操作，請說明為什麼。

A:Process Affinity 是指在 Multiprocessors System 中，當 Process 已決定某 CPU 上執行，則在他執行的過程中，盡量不要將他轉移到其他 CPU 上執行(除非有必要，像是 Processor BAD、load balancing...)，避免 CPU 內之 cache 等內容要複製(repopulation 重新布置)且刪去(elimination)之影響工作效能。可分為 1.Hard Affinity:規定 Process 不可轉移 ;2.Soft Affinity:盡可能不要轉移，若有需要仍可轉移。