1. (10%) Explain what memory-mapped I/O is and how it works

   Memory-mapped I/O（MMIO）將 I/O 的 Register 映射到 memory space 上。當 CPU 操作這些 memory space 時，系統就會將這些操作送往 I/O。透過 Memory-mapped I/O 就不需要使用不同的 address space。

   與 Memory-mapped I/O 相對的就是 Port-mapped I/O，他使用了不同的 address space，使用者必須使用不同的 instruction 去操控 I/O，且其 read write 使用的 bus 會與 memory 分開。

   當 CPU access 一個 memory 的地址是對應到其中一個 I/O register，memory controller 會產生一個 control signal 來與 I/O 做溝通， 因此可以直接透過一般的 memory 指令來操作 I/O。

2. (10%) Explain what DMA is and how it works

   DMA（Direct Memory Access）是一個 specialized 的硬體設備，可以直接在協助設備間的資料傳輸而不用使用 CPU，因此能夠使傳輸更為有效率。

   在一般的 I/O 操作中，CPU 會負責發起 I/O 到 memory 或從 memory 到 I/O 設備之間的傳輸。 這包含 CPU 讀取或寫入 data 到 I/O ，及 I/O 和 memory 之間傳輸數據。 此過程很慢且效率極低。

   使用 DMA，CPU 可以將數據傳輸任務交給 DMA controller。 DMA controller 可以直接 access memory 和 I/O 並在它們之間傳輸數據，且不需要 CPU 的參與。 這樣可以大大提高數據傳輸的效率，減輕 CPU 的負擔。

   DMA 傳輸涉及的基本步驟如下：
   1) CPU 設定 DMA controller，提供 source address、destination address、transfer size 等資訊。
   2) DMA controller 與 CPU 要求 system bus 的使用權限
   3) 當 DMA controller 取得控制後，它就會直接從 I/O 或 memory 進行讀寫。
   4) 傳輸完成後，DMA controller 會向 CPU 發送完成的信號。

3. Consider the following set of processes, with the length of the CPU-burst time given in milliseconds:

Process	Burst Time	Priority
P1	8	4
P2	1	1
P3	2	3
P4	1	5
P5	6	2

   The processes are assumed to have arrived in the order P1, P2, P3, P4, P5, all at time 0.
   (a) (5%) Draw four Gantt charts illustrating the execution of these processes using FCFS, SJF, a non-preemptive priority (a smaller priority number implies a higher priority), and RR (quantum = 1) scheduling.
   (b) (5%) What is the turnaround time of each process for each of the scheduling algorithms

in part 3a?

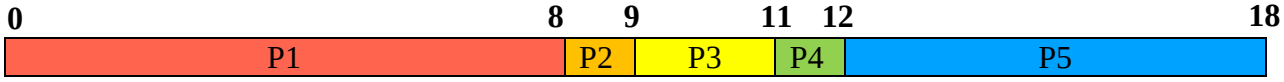
(c) (5%) What is the waiting time of each process for each of the scheduling algorithms in part 3a?

(d) (5%) Which of the schedulers in part 3a results in the minimal waiting time (over all processes)?

Ans:

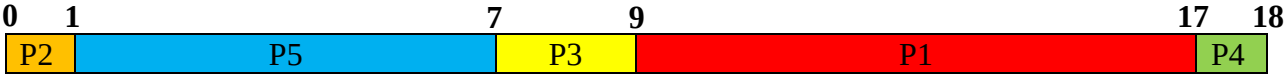
(a)

FCFS :



SJR :

Non-preemptive priority:



Round-robin :

(b)

	P1	P2	P3	P4	P5	Average
FCFS	8	9	11	12	18	11.6
SJR	18	2	4	2	10	7.2
Non-preemptive priority	17	1	9	18	7	10.4
Round Robin	18	2	7	4	16	9.4

(c)

	P1	P2	P3	P4	P5	Average
FCFS	0	8	9	11	12	8
SJR	10	0	2	1	4	3.4
Non-preemptive priority	9	0	7	17	1	6.8
Round Robin	4+2+1+1+1+1 =10	1	2+3=5	3	4+2+1+1+1+1=10	5.8

(d) SJF

4. (10%) A UNIX process has two parts—the user part and the kernel part. Is the kernel part like a subroutine and a coroutine? Why?

   Subroutine : 又稱文 procedure 或 function，代表可以執行特定任務的程式碼區塊，其他程式可以透過呼叫來執行此區塊。當呼叫 subroutine 時，控制權就會移轉到 subroutine 直到任務完成後就會返回原本呼叫的地點。Subroutine 通常用於 encapsulate 頻繁使用的程式碼，使得期可以重複使用及更好的維護。

   Coroutine : 一種特殊的 Subroutine，允許 non-preemptive multitasking，也就是說它可以在執行時，於特定的地方暫停並保存其 state，然後執行完其他的程式再繼續執行此 function (可以中斷及繼續執行的函式呼叫)。coroutine 常用於實現 cooperative multitasking，不需要使用 multithreading 的全部功能，就可以讓程式執行多個任務。

   以 user 使用 library 來呼叫底層 system call 的角度來看它比較像 subroutine，因為它是先設定好一些參數，並發出 trap 來將 control 交給 kernel code，kernel code 會執行 interrupt handler，最後再將資料回傳給 user。