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。

1. (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。
5. 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.

* 1. (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.
  2. (5%) What is the turnaround time of each process for each of the scheduling algorithms in part 3a?
  3. (5%) What is the waiting time of each process for each of the scheduling algorithms in part 3a?
  4. (5%) Which of the schedulers in part 3a results in the minimal waiting time (over all processes)?

Ans:

(a)

FCFS :

**18**

**12**

**11**

**9**

**8**

**0**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| P1 | P2 | P3 | P4 | P5 |

SJR :

**18**

**10**

**4**

**2**

**1**

**0**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| P2 | P4 | P3 | P5 | P1 |

Non-preemptive priority:

**0**

**18**

**17**

**9**

**7**

**1**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| P2 | P5 | P3 | P1 | P4 |

Round-robin :

**0**

**18**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| P1 | P2 | P3 | P4 | P5 | P1 | P3 | P5 | P1 | P5 | P1 | P5 | P1 | P5 | P1 | P5 | P1 | P1 |

(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

1. (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。