106

Operating Systems, First Term Exam, Memory Management, Apr. 20, 2020

Chapter 3- Chapter 6 [redacted]

Student ID [redacted]

This exam is closed book. Write your answers directly on the exam sheets. 直接作答.若答題空間不夠,請找空間作答.(注意:要標明題號)

I. Please write down the full names of the following abbreviations. Please do **not** make any explanations. (5 points)

(1) FCFS: first-come-first-served

(2) FIFO: first-in-first-out

(3) BIOS: basic input output system

(4) PCB(note: this is not for PC Board) process control block

(5) IPC: interprocess communication

II. True or false: (5 points)

(1) (F)A process in a ready queue can itself change its state to the waiting(blocked) state.

(2) (F)A process is a thread in execution.

(3) (F)An operating system's system call can only be invoked by assembly code. There is no high-level language-based program that can call's the operating system's call without writing some assembly code.

(4) (F)We can only uses hardware atomic instructions to support mutual exclusion.

(5) (F) "Starvation" means that a process is blocked by the operating system indefinitely because the process asks too much memory space.

III. Briefly answer the following questions. (40 points)

-5 (1) What is the distinction between kernel mode and user mode.

kernel mode 在執行 kernel 的 process

user mode 在執行使用者程式的 process

(2) Describe how system call works.

user-level 中的程式叫用API(此時在 user mode),API中呼叫 system call,此時切換成 kernel mode,OS尋找對應 system call 的程式碼執行,完成後回傳,再回到 user mode

(3) What is the difference between a process and a thread? Describe two benefits using threads. process 中實際在運行的是 thread。

thread 共用了 process 的程式碼與部分 data,比起複製多個 process 以執行相同的工作,複製 thread 較省空間。

多個 thread 一起做某些事(ex:矩陣乘法)的速度較快。

(4) Why it is a good idea to separate policy from mechanism? How can it be achieved in process scheduling?

policy (要做什麼)、mechanism (如何做)，分開的話，日後要修改時較有彈性。

process cheduling 中，policy 決定排程規則 (ex: FCFS, SJF, Round Robin)，mechanism 決定其運作方式 (ex: FCFS 演算法的實作)。

(5) Describe the difference between a preemptive scheduling and nonpreemptive scheduling algorithms. Which one is more suitable for timesharing systems?

preemptive：假如某 process 正在執行，有需要切換 process (ex：高 priority 的 process 進來) 時，會暫停目前 process 的運作。

nonpreemptive：不管怎樣，一定會等目前 process 完成。

preemptive scheduling 較適合，資源比較不會被占用，且操作反應較好。

(6) Name two advantages of using binary semaphores to achieve mutual exclusion among several processes over Peterson's solution.

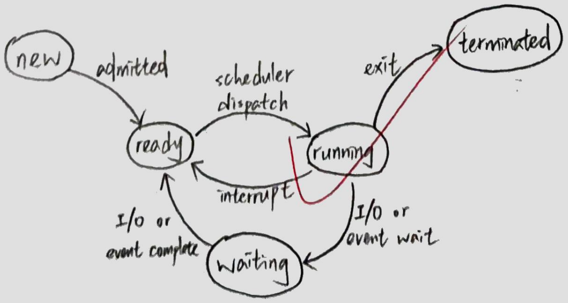
-2

Peterson's solution 需要為每個 process 建 flag (代表是否 ready)，還要 turn 代表誰在 critical solution，而 binary semaphore 只要一個變數 S。Semaphore 較簡單，也較省空間。

(7) Both WINDOWS and UNIX boost the priority of I/O-bound processes. Explain why this is a good idea.

I/O 運作速度較慢，先讓 I/O-bound processes 完成 CPU burst，然後排隊等待I/O。在它們慢慢等 I/O 時，CPU 也正在處理 CPU-bound processes，如此，CPU 與 I/O 同時保持高使用量，減少閒置時的資源浪費。

(8) Draw a process state (transition) diagram. The states are new, ready, running, terminated, and waiting. Note that you should identify the events that trigger each state transition.



IV. Please explain the following terminology briefly. (15 points)

13

(1) kernel

作業系統的核心，負責排程、除錯…等許多任務以維持運作穩定

(2) Privileged instructions

權限較高，需要優先執行的指令

(3) heap

(4) monitor (in synchronization)

類似 class，內部變數只能由內部 procedure 修改，且一次只能執行一個 procedure，避免了 race condition。

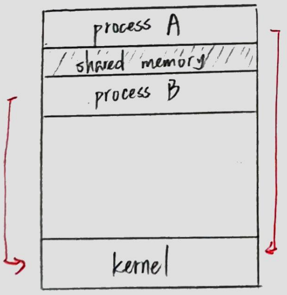
(5) thread library

負責管理 user-level-thread，在 many-to-many model 上也負責分配 LWP 給 user-level thread

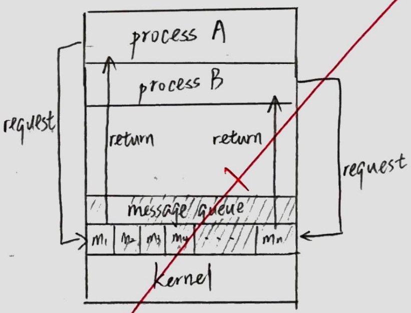
-2

V. (10 points)Describe what is a shared memory model and what is a message passing model.

Shared memory model



Message passing model



兩不同 process 需要溝通時，直接至共用的記憶體區塊存取需要的資訊。

兩不同的 process 需要互相溝通時，透過 message passing model 達成。發送者先將訊息放入 message queue，接收者再將訊息取回。

VI. (10 points)Most recently, the part of the operating system that resides in memory has been limited to a few essential modules while other functions are provided by special modules which are treated as regular applications. Please show at least two examples of those parts that (1) must reside in memory, (2) can be treated as applications.

-2

✓ (1) synchronization tools, scheduling

(2) I/O、系統附屬應用程式

VII. (10 points)What are the items of a PCB? Please list at least five items.

process number, process state, program counter, list of open files, registers

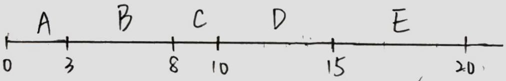
✓

VIII. (15 points) There are five processes A to E to run. Their arrival times are 0, 1, 3, 9 and 12 second, respectively. Their processing times are 3, 5, 2, 5 and 5 seconds, respectively. What is the average turnaround time using first-come-first-served, shortest-job-first, and round robin (with 1 second quantum) scheduling?

preemptive?

Process	Arrival time	Processing time
A	0	3
B	1	5
C	3	2
D	9	5
E	12	5

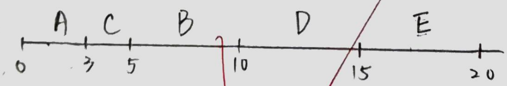
FCFS :



turnaround time = A=3, B=7, C=7, D=6, E=8

average turnaround time : (3+7+7+6+8)÷5 = 6.2 seconds

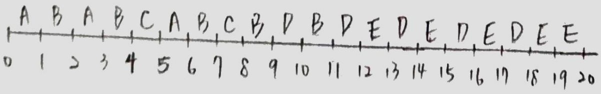
SJF :



turnaround time : A=3, B=9, C=2, D=6, E=8

average turnaround time : (3+9+2+6+8)/5 = 5.6 seconds

RR :



turnaround time : A=6, B=10, C=5, D=9, E=8

average turnaround time : (6+10+5+9+8)÷5 = 7.6 seconds

IX. (10 points) The test-and-set atomic machine instruction is defined as follows:

```
Boolean test_and_set(int i)
{
   If (i==0)     0代表解鎖
     {i =1;      1代表鎖定
      Return true;
     }
     Else return false;
}
```

```
Boolean test_and_set(int *i){
    if(*i==0){
       *i=1;
       return true;
    }
    else
       return false;
}
```

(i)Can this test-and-set instruction be used to achieve mutual exclusion? If you think it cannot, please suggest a solution. 不行！int i需要用pointer

(ii)Write a solution to critical section using the test_and_set(int i). You can reference the following code.

用我自行定義的 test_and_set(int *i)

```
do{
  while(!test_and_set(&lock));
     /*critical section*/
  lock=0;
     /*remainder section*/
}while(true);
```

The test-and-set atomic instruction in the text book is as follows:

(1)Definition:

```
boolean test_and_set (boolean *target)
    {
          boolean rv = *target;
          *target = TRUE;
          return rv:
    }
```

(2) And its solution: to critical section is used as follows:

```
do {
    while (test_and_set(&lock))
        ; /* do nothing */
              /* critical section */
    lock = false;
              /* remainder section */
} while (true);
```