學號:_____ 姓名:_____ 系級:_____

國立中正大學資訊工程學系大三 A 班
九十二學年度第一學期作業系統期中考試 答案卷

(1) A variable is shared among different processes and can be accessed simultaneously by several processes, subject to the following constraint: “The sum of all PIDs of the processes currently accessing the variable must be less than 200.” Write a monitor to coordinate access to the variable. (20%)

```
monitor coordinate_access {

   enum { request, grant, release} state[n];

   var pid: integer;

   var pid_sum : integer;

   var self[n] : condition;

   void request_var(int i) {

        state[i] = request;

        test(i);

        if(state[i] != grant)

           self[i].wait;

   }

   void release_var(int i) {

        state[i] = release;

        pid_sum -= self[i].pid;

        for(int j = 0; j < n; j++)

           if( j != i && state[j] == request)

              test(j);

   }
```

```
        void test(int i) {
            if(pid_sum + self[i].pid < 200) {
                state[i] = grant;
                pid_sum += self[i].pid;
                self[i].signal;
            }
        }
        void init() {
            pid_sum = 0;
        }
}
coordinate_access.request_var(i);
        ……
        access
        ……
coordinate_access.release_var(i);
```

(2) Three jobs 1, 2, 3 have arrival times, burst times, and priorities as follows. Smaller numbers imply higher priorities.

Job	Arrival Time	Burst Time	Priority
1	0	7	9
2	3	2	3
3	2	4	6

a) Draw Gantt charts and determine the average waiting time for FCFS (First Come First Served) scheduling algorithm. (5 %)

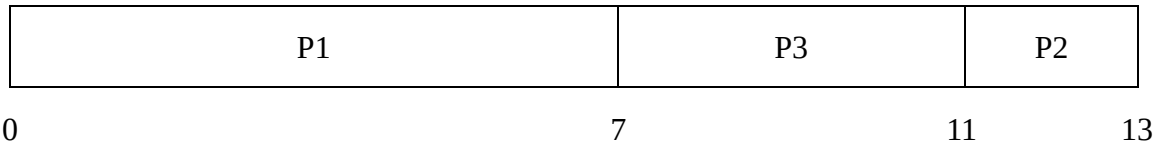
b) Draw Gantt charts and determine the average waiting time for RR (Round Robin) scheduling algorithm with a quantum time of 4 units. (5 %)

c) Draw Gantt charts and determine the average waiting time for PP (Preemptive Priority) scheduling, where smaller numbers are of higher priorities. (5 %)

d) Draw Gantt charts and determine the average waiting time for non-preemptive SJF (Shortest Job First) scheduling algorithm. (5 %)

Sol:

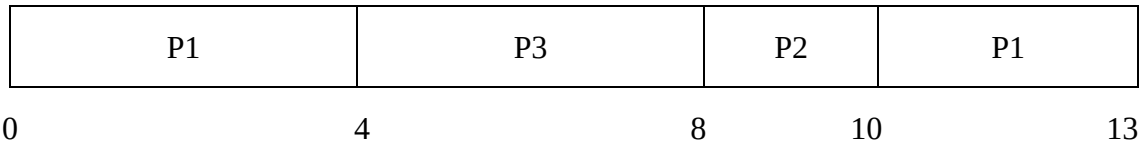
(a) FCFS (First Come First Served)



Average waiting = (0 + (11 – 3) + (7 – 2)) / 3 = 13 / 3 = 4.33 units

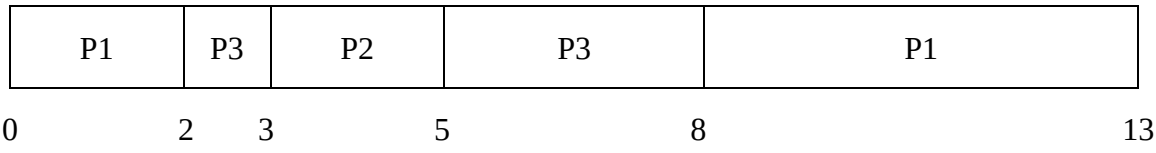
job1 job2 job3

(b) RR (Round Robin)



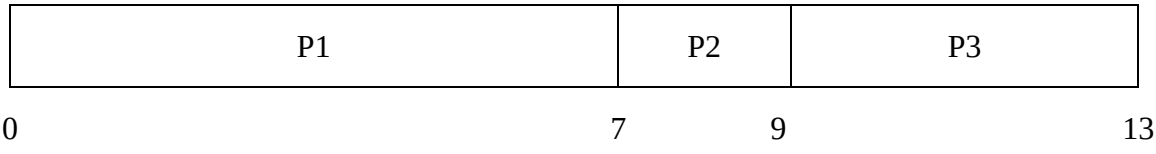
Average waiting = ((10 – 4) + (4 – 3) + (8 – 2)) / 3 = 13 / 3 = 4.33 units

(c) PP (Preemptive Priority)



Average waiting = ((8 – 2) + (3 – 3) + (5 – 3)) / 3 = 8 / 3 = 2.67 units

(d) non-preemptive SJF (Shortest Job First)



Average waiting = (0 + (7 – 3) + (9 – 2)) / 3 = 11 /3 = 3.67 units

(3) Describe two problems that can occur in a priority-based scheduling policy. How can they be solved? (10 %)

sol

(a):

Problem: Starvation - low priority processes may never execute.

Solution Aging – as time progresses increase the priority of the process.

(b)

Problem: Priority inversion

Low priority task is running and holds a shared resource
High priority task need that shared resource
High priority task waits for low priority task to release resource
Middle priority task activates, preempts low priority task, and runs!!!
High priority task is waiting for a long time, even forever!

Solution:將 low-priority 的優先順序調成和 high-priority 一樣

(4) (a) In what different ways can a thread be cancelled? Explain their pros and cons. What are cancellation points in the Pthread library? (5 %)

(b) A multiprocessing system may have numerous threads and thread creation/cancellation incurs significant system overhead when the number of threads becomes large. How can we solve this problem? Explain in details. (5 %)

sol:

(a)

1.

Asynchronous cancellation:

One thread immediately terminates the target thread.

Deferred cancellation:

The target thread can periodically check if it should terminate, allowing the target thread an opportunity to terminate itself in an orderly fashion.

2.

The Pthread library allows a thread to check if it should be cancelled at a point when it can safely be cancelled.

(b)

Use thread pools -- create a number of threads at process startup and place them

into a pool. When a server receives a request, it awakens a thread from the pool.

Once the thread completes its service, it returns to the pool awaiting more work.

The benefits are time-saving and limiting the number of existent threads.

(5) Explain what is context switching between processes. If context switch incurs overhead, why does an OS waste time doing context switching? Context switching can be accelerated by hardware registers, how does Sun UltraSPARC CPU support faster context switching? (10 %)

Sol:

當 context switching 發生時，kernel 必需先將目前舊的 process 之狀態儲存起來，再將新的 process 之前的狀態載入。

爲了能讓 CPU 可以同時執行多個應用程式，提昇 CPU 的使用率。

Sun UltraSPARC 提供多組暫存器，在作 context switching 時，只要簡單的將指標指到目前所要的這組暫存器。若是目前活動中的 process 數目超過暫存器的組數，則系統會先將暫存器的資料拷貝到記憶體中，或是從記憶體中拷貝出來。

(6) Explain the following terms in one sentence each. (10 %)

(a) microkernel
(b) bootstrap loader
(c) critical region
(d) cache coherency
(e) hard real-time systems
(f) asymmetric multiprocessing
(g) multiprogramming
(h) synchronization
(i) medium-term scheduling

(j) parameter marshalling

Sol:

(a) microkernel

以模組化設計的方式移除 kernel 中非必要的元件，使得 kernel 變的較小。

(b) bootstrap loader

在開機時負責載入作業系統，並執行之。

(c) critical region

可確保 region 所宣告的變數，在同一時間只有符合 region 條件內的一段敘述可以去存取。

(d) cache coherency

當一個 cache 中的變數 A 被改變後，所有其它 cache 中的變數 A 也要能夠反映這個變化。

(e) hard real-time systems

必需在給定的時間內完成所負責的關鍵性任務。

(f) asymmetric multiprocessing

在 multiple-processor 的系統中，由 master processor 負責工作排程並分配工作給 slave processor。

(g) multiprogramming

在記憶體中同時置入多個 job，使 CPU 總是有一個 job 可以執行。

(h) synchronization

確保有共用相同資料的數個 process，其運作順序不會造成資料的不一致。

(i) medium-term scheduling

可將 memory 或是 CPU 中的 process swap out，或是將被 swap out 的 process 重新 swap in 到 memory 中。

(j) parameter marshalling

在 RPC 中將參數重新包裝成另一種可以在網路上傳送的形式。

學號:_____ 姓名:_____ 系級:_____

(7) A manufacturing system, consisting of 4 processes, is to be automated by a program which contains the following 5 tasks: (20 %)

- **TAKE1**, **TAKE2**, **TAKE3, TAKE4**:
  to take manufacturing parts in respective processes,
- **ASSEMBLE**:
  to assemble the parts after taking 2 parts.

The following sequence of the tasks must be created by a suitable synchronization with semaphores:

- The assemble-task should be executed after each **even-numbered** take-task.
- The sequence of the take-tasks is given by the number in their names. TAKE1 is therefore the first task.
- The sequence given above should be repeated cyclically.
- The program of each task should be executed completely, before another task can be started.

Questions:

(a) Write down the names of the tasks in the desired sequence.

(b) Insert into each task the semaphore-operations required to guarantee the desired task sequence.

(c) With which values do the semaphore variables have to be initialized?

sol:

(a) TAKE1->TAKE2->ASSEMBLE->TAKE3->TAKE4->ASSEMBLE->TAKE1->……

(b)

TAKE1( ){
P(mutex);
P(T1);
…….
V(T2);
}
	TAKE2( ){
	P(T2);
	…….
	V(A);
	V(T3);
	}

TAKE3( ){
P(mutex);
P(T3);
…….
V(T4);
}
	TAKE4( ){
	P(T4);
	……
	V(A);
	V(T1);
	}

```
ASSEMBLE (){
P(A);
……
V(mutex);
}
```

(c)

mut