THE ANSWERS

A: Analysis, B: Bookwork, C: Computed Example, D: Design

1. a) Define a *context switch* to be whenever an unfinished job is suspended. Note that when one job finishes and another starts or restarts, it is not a context switch.

   i) In a system with n jobs, assuming no blocking, state what is the maximum number of context switches possible, or whether there is no upper bound, for each of the following scheduling methods: First-come-first-served scheduling, priority scheduling with pre-emption, round-robin scheduling.

   [3]

   ii) In an operating system scheduler design, the aim is to provide a waiting time of no more than 10ms to n interactive jobs from users. Where this 10ms limit cannot be met, the aim is to reduce the worst-case waiting time for any job. Describe, with reasons, the scheduling method you would use for interactive jobs, if context-switch time is 0.

   [2]

   iii) Suppose in the scheduler design described in (ii), it is known that no interactive job lasts for more than 5ms, and the context-switch time is also 5ms. You may assume that each interactive job overlaps with at most one interactive job from each of the $n-1$ other users. State, with reasons, what scheduing method would then be optimal.

   [2]

A.

(i) FCFS: 0, Priority with pre-emption: $n-1$, RR: no upper bound.

(ii) RR scheduling. Because in the worst case of long interactive jobs colliding it will be fairest and hence reduce the worst-case time.

(iii) FCFS scheduling without pre-emption. In this case context-switches add more to waiting-time than can be gained by pre-emption so we do not want this. FCFS is then the fairest available algorithm.
This question was done quite well. Many people though SJF scheduling would be good for (iii), but that is not true for thois case where we want to minimise the maximum (not average) waiting time.
Some said that RR scheudling with ¿=5ms time slice would do for (iii) which is true, and got full marks also.

b) i) In an operating system the following *atomic* OS API calls are provided to user processes:

- *Load-and-add*. Return the value of a memory location, then add a fixed signed integer to the location.
- *Remove-from-queue-and-make-ready*. Remove the front process from a queue and make its state *ready-to-run*.
- *Add-to-queue-and-suspend*. Add the currently running process to the front of a queue and suspend it, making its state *waiting*.

Using these OS calls, define a semaphore data structure and write pseudocode for three user functions that correctly implement the semaphore *init*, *wait*, and *signal* operations.

[3]

ii) What change, if any, would be needed to make your implementation a *strong semaphore* where waiting processes are allowed to access the critical region in first-come-first-served order.

[2]

iii) Four processes share a printer resource which requires a mutually exclusive critical region of code to implement spooling. Explain how to use semaphore operations as in (i) to ensure that critical regions in all four processes are mutually exclusive. Detail what in your code would change if there were two available printers and therefore at most two processes could enter their critical regions simultaneously.

[2]

(i) D. (ii) A. (iii) B.

(i) Semaphore = { waiters: Queue; count: int }

```
init( S, n) {
// set semaphore S.count to n and S.waiters to empty
}

wait(S) {
x := Load-and-decrement(S.count)
If x <= 0 then add-to-queue-and-suspend
}

signal(S)
x := Load-and increment(S.count)
If x < 0 Remove-from-queue-and-make-ready()
}
```

This question was quite difficult. Many people did not understand they were supposed to write the definitions of init/wait/signal. Many used count incrememnt and decrement without the atomic operations! Surprisingly many failed to correctly define init - often using the atomic chnage operation instead of just setting the variable.

(ii) For a strong semaphore the *Add-to-queue* operation would need to add to the end of the queue, or the Remove-from-queue take from end, so that this correctly implemented first-come-first-served waiting.

This was done well by almost everyone. A few did not indicate the implementation.

(iii)

```
init(1)

The code around each critical region must be:

Wait(S)
// critical region
Signal(S)
```

A few people did not specify initialisation, or said this was init(0).

The *init* parameter would need to be set to 2 instead of 1 for two printers

A few people incorrectly thought this needed Peterson's algorithm, or two semaphores.

c) Describe page table and base-and-limit-register memory allocation methods. Define interior and exterior fragmentation, and state how these apply to each memory allocation method.

[2]

B.

Page table allocation divides memory into a large number of fixed-size frames. It assigns a set of memory pages to each process, each of which is mapped to a single arbitrary memory frame. Limit and bound memory allocation assigns a contiguous region of memory to each process as determined by limit and bound registers. Interior fragmentation is unusable memory within that allocated to each process. This applies particularly to page allocation where memory allocated is quantised by page size. Exterior fragmentation is unusable memory not allocated to processes. In this case because allocated memory must be contiguous, regions of free memory smaller than the required memory for a new process cannot be used.

d) Contrast the merits of using threads or processes to implement concurrent processing needed by a single user.

[2]

B.

Threads share memory, file, etc resources and so are more efficient. Processes provide memory protection and so errors in one program do not affect others.

e) State what factors determine the performance of a virtual memory system, and under what circumstances thrashing might occur.

[2]

B.

VM system performance is governed by:
(1) speed of secondary storage
(2) performance of page replacement algorithm
(3) whether thrashing occurs, see below

Thrashing occurs when the sum of working set sizes of active processes is larger than the available RAM

Question 1. has a total of 20 marks.