THE ANSWERS

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

1. a) Consider the processes shown in Figure 1.1 with arrival time, duration as shown. In the questions below the sequence X:Y:Z indicates the priorities of X,Y,Z respectively to be 1, 2 and 3 where X,Y,Z are some permutation of A,B,C. Lower numbers represent higher priorities.

   i) Calculate the waiting time for process A under priority scheduling with preemption for each of the six cases given by priorities : A:B:C, A:C:B, B:A:C, B:C:A, C:A:B, C:B:A. (This is all possible permutations).

   [2]

   ii) Calculate the *minimum* waiting time for process A, given it may start at any time in the range 0 ms - 4 ms, with all other system parameters as shown in Figure 1.1, for the same 6 cases, under priority scheduling with preemption.

   [2]

   iii) An operating system has 10 users, each of which run both interactive and batch processes. It is desired that interactive processes should have the fastest possible response, and that batch processes should run fairly. Propose a scheduling strategy that will implement this.

   [2]

C. & B.

(i) Priorites are shown by specifying process letters in order (highest priority first).
ABC or ACB: 0
BAC: 2
CAB: 3
BCA or CBA: 5
amost everyone got this right
(ii)
BCA or CBA: 2 BAC, ABC,ACB: 0 CAB: 1
Some did not understand how to work out the arrival time for minimum delay of A - which is as late as possible allowing B & C mostly to execute before A starts.
(iii)
Multi-Level Queue with one queue for batch and one for interactive. Batch process queue runs round robin scheduling, interactive process queue runs any scheduling, interactive queue is preemptive priority scheduled relative to batch queue at higher priority than batch queue.
I was quite generous with this because there are a number of solutions which work well. But I wanted a clear strategy the interactive process tasks higher priority than the batch tasks and I wanted round-robin for the batch tasks. Some people gave two startegies but did not say how they could be combined giving higher priority to interactive - either using prority based scheduling (within which RR can fit) or via an MLQ.

   b) In the context of memory paging systems, consider the following scenario:

   - You have three available frames
   - The reference string is: 9-1-3-1-3-2-4-3-6-3

   i) Starting with empty frame contents, show the sequence of frame contents after each request and count the number of page faults for the LRU algorithm.

   [2]

   ii) Contrast the advantages and disadvantages of optimal and LRU page replacement.

   [2]

C. & B.

(i)

Ref string:	9	1	3	1	3	2	4	3	6	3
F1	9					2				3
F2		1					4			
F3			3						6	

7 page faults

This was done very well.

(ii)

LRU is simple to impelment and reasonably good at minimising page faults but in some circumstances highly non-optimal. Optimal is (by definition) optimal for minimising page faults but in genral cannot be implemented because it requires future knowledge of page access pattern.

this was also done very well

c) i) Write pseudocode showing how a single global variable can be used to force mutually exclusive access to a resource from two processes A and B.

[1]

ii) Show that Peterson's Algorithm, shown in Figure 1.2, ensures that processes A and B can never both be in their critical region at the same time.

[2]

iii) Give one reason why semaphores provide a better solution to mutual exclusion than Peterson's Algorithm.

[1]

A and B
(i)

```
var Turn = 'A';/* (or B)*/
Process A
while Turn <> 'A';
/* critical section */
Turn = 'B'

Process B
while Turn <> 'B';
/*critical section*/
Turn = 'A'
```

many people tried to use a lock variable - but that requires some extra mechanism, like a test and set instruction. Some tried semaphores but that again is more complex.
(ii)
1. Let A be the first of the two processes to enter its critical region before the overlap (if not rename to ensure this)
2. When A exits its WAIT to enter the CR either interested_B = false or Turn = 'A'
3. Case analysis:
(a) Interested_B = false. In this case B must set Turn to 'A' before it enters its WAIT. Turn cannot be reset to 'B', or Interested_A to false, until A exits its CR. therefore B cannot enter its CR till A exists its CR and in this case the proposition is proved.
(b) Turn = 'A'. In this case Process B must have set Turn to 'A' after process A set it to B. As above Turn cannot be 'B' or Interested_A false until A leaves its CR and again as above the proposition is proved. /em This was difficult. many people thought that a single path analysis showing the algrorithm behaving as required was enough - but you have to consider all possible traces of the two processes.

(iii) Semaphores are better because the busy/wait loops are replaced by suspending the process. If the wait time is long this is more efficient, freeing CPU to run some other process. /em I also allowed sempahores better because easier for multiple processes.

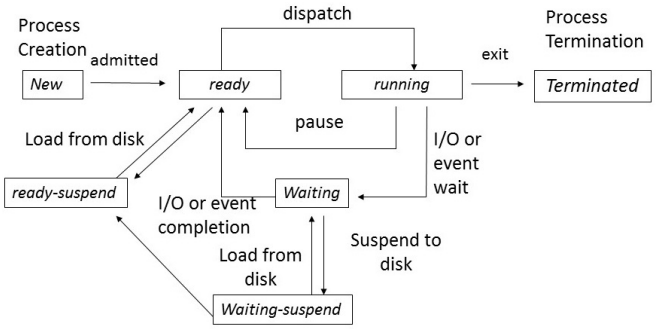
d) i) Draw a diagram showing all states and possible transitions of the 7 state process model. Which state(s) can contain a maximum number of processes at any one time, if the total number of processes is unknown. What is this number?

[2]

ii) A disk interrupt happens and an I/O request from process A is completed. What possible state transition(s) within the 7 state model can process A make as the result of this?

[1]

A and B
(i)



The running state can have at most n processes (for an n CPU system). All other states can have any number of processes.
some people did not choose the running state. Other states may have some resource-determined limit but it is not fixed. I accepted (and expected) the single-CPU case of n=1. The 7 state model was bookwork which most remembered but a few forgot the "new process" state.
(ii)
waiting $--> $ ready
waiting-suspend $--> $ ready-suspend
a common mistake was thinking waiting-suspend -¿ waiting or forgetting the waiting-suspend transition

e) Show a block diagram of the typical hardware to implement page-based memory allocation. With reference to this diagram describe the sequence of operations required to implement a memory read by a user process of a location within a page which is swapped out of main memory before the start of the memory read.

[3]

B.

This was a difficult question since you were not taught specific hardware except at block level, and that had various possibilities. However you should have been able to state the sequence of operations, and which are hardware or software. Many used TLB hardware which was fine, I also accepted a full (non-cached) hardware table as a block (precise gates etc not required).

The common mistake was not delaing with the sequence of operation for the one case I asked, one where the page was not in physical memory and therefore the initial hardware page to frame translation lookup would fail and initiate a software-guided page allocation and disk to main memory transfer.

The sequence is:

i. The CPU issues a memory reference, and the CPU address is looked up in a hardware unit (either TLB or hardware page table) and the lookup shows that the the page is not in memory.

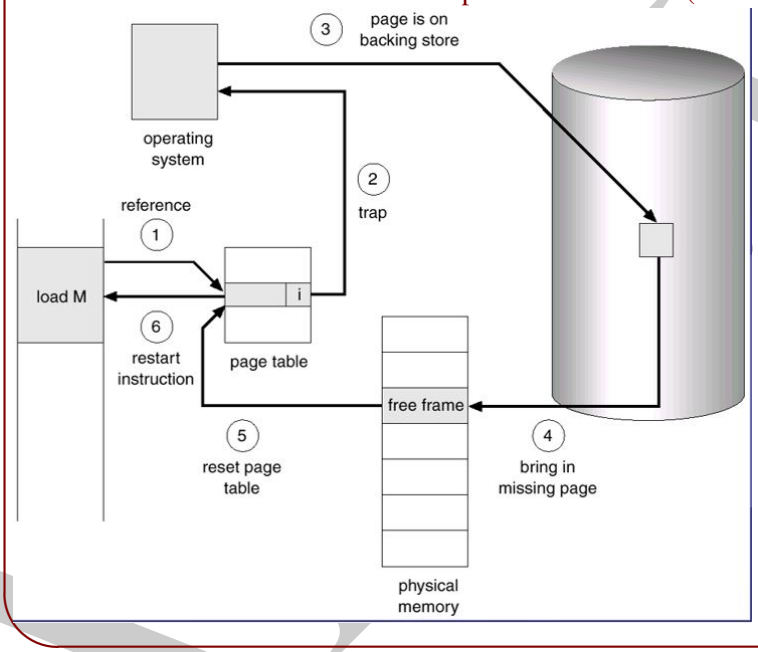
ii. There is a hardware initiated trap to protected mode code

iii. Software then initiates page transfer from storage device to physical memory

iv. software allocates a new frame of physical memory to the required page and updates the TLB or page table and stores the new page data is stored in the newly allocated frame.

v. the instruction that caused the page error is rerun and will not run to completion.

The exact refinement used in these steps does not matter (see for example the steps listed in the picture).



Question 1. has a total of 20 marks.

Process	arrival time (ms)	duration (ms)
A	0	4
B	1	2
C	2	3

Figure 1.1: Three processes.

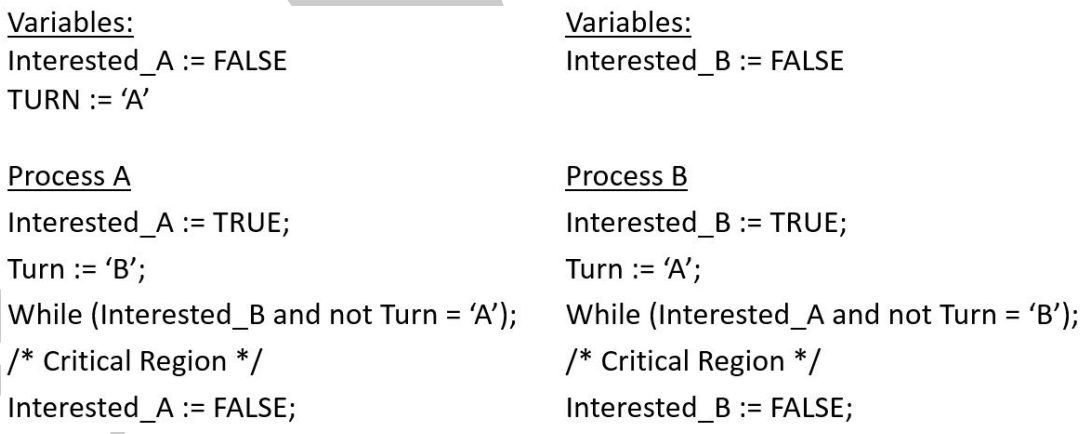


Figure 1.2: Peterson's Algorithm.