CS-228
OPERATING SYSTEMS
Attempt 2 questions out of 3

Question 1

(a) With the aid of a state transition diagram, illustrate the transitions between the process states **running**, **ready**, **blocked**, **suspended-ready**, and **suspended-blocked**. The events causing the transitions should be clearly indicated.

Consider each of the following suggested transitions. If it is included in your state transition diagram, give an example event that would cause such a transition; otherwise, explain why such a transition is not possible.

(i) From "running" to "ready";

(ii) from "blocked" to "suspended blocked";

(iii) from "blocked" to "running";

(iv) from "suspended-ready" to "blocked";

(v) from "suspended-blocked" to "suspended-ready".

[10 marks]

(b) Consider the set of processes shown in the table below. We assume that the CPU ready queue is empty at time 0, and that the time needed for context switches is negligible.

	P_1	P_2	P_3	P_4
Arrival time (ms)	0	1	4	7
Burst time (ms)	3	5	6	4

With the aid of a Gantt chart or an equivalent, determine the average waiting time (over all four processes) for the following scheduling algorithms:

(i) First-in-first-out.

(ii) Preemptive round robin, with quantum set to 2ms. Newly arrived processes go onto the back of the queue ahead of any processes pre-empted at the same time.

(iii) A simplified multi-level feedback queue scheduling algorithm with only **two levels**. The quantum for processes in the first level is set to 2ms, and that for the second level is 4ms.

[7 marks]

(c) Explain the purpose and functionality of interrupts in the context of I/O processing. You should discuss such concepts as the interrupt controller, interrupt vectors, interrupt handlers, interrupt classes, and any other mechanisms found in the Operating System and hardware which you feel are relevant.

[8 marks]

Question 2

(a) In the context of Dijkstra's deadlock avoidance algorithm (banker's algorithm) for single-type resource management, state whether each of the following states is safe or unsafe, briefly justifying your answer in each case.

We assume that there are **10** resources in total and only **2** are free at the moment. In the tables, for each process, *loan* is the number of resources currently held by it, *max* is the maximum number of resources needed, and *claim* is the number of resources to be claimed.

STATE I	*loan*	*max*	*claim*
P_1	2	10	8
P_2	2	5	3
P_3	2	4	2
P_4	2	7	5

STATE II	*loan*	*max*	*claim*
P_1	2	6	4
P_2	4	7	3
P_3	2	7	5
P_4	0	4	4

STATE III	*loan*	*max*	*claim*
P_1	2	9	7
P_2	2	4	2
P_3	1	3	2
P_4	3	9	6

STATE IV	*loan*	*max*	*claim*
P_1	3	7	4
P_2	2	8	6
P_3	3	5	2
P_4	0	4	4

[4 marks]

(b) In the context of memory management, what is meant by the following?

(i) Fragmentation;

(ii) monoprogramming;

(iii) anticipatory paging;

(iv) relocatable loading.

[8 marks]

(c) (i) A file containing 2700KB of data is to be stored in a Unix file system with 1KB data blocks. Each indirect block can hold 256 disk addresses (i.e. 32-bits per address). What is the total number of data blocks required to store this file (including all indirect blocks but not the inode itself)? Explain your answer in detail.

(ii) Describe, with the aid of a diagram if necessary, the concepts of Networked File System (NFS, designed by Sun Microsystems).

[7 marks]

(Question 2 continued on next page.)

(d) (i) Describe in detail how events may be ordered in a distributed environment without the benefit of a common clock.

(ii) Consider three processes running concurrently on different computers, each with its own clock. During the execution, these processes exchange messages frequently, and generate reports regularly (e.g. every hour) which must be displayed on the screen at a similar time. However, as shown below, the speeds (in seconds) of the three clocks are quite inconsistent with each other. Using this example, explain how the 'time stamping' concept can be used to co-ordinate the activities of these three processes.

C1	C2	C3
0	0	0
1	4	5
2	8	10
3	12	15
4	16	20
...	...	...
16	64	80
17	68	85
18	72	90
...	...	...

[6 marks]

Question 3

(a) With demand paging memory management and the **least recently used (LRU) page replacement strategy**, the current LRU matrix (6x6) is shown below, assuming that the memory has only six page frames. If the next three page reference numbers are 6, 2, and 1 respectively, illustrate the changes to the LRU matrix step by step.

	4	**3**	**5**	**7**	**9**	**2**
4	0	1	1	1	1	1
3	0	0	1	1	1	1
5	0	0	0	1	1	0
7	0	0	0	0	0	0
9	0	0	0	1	0	0
2	0	0	1	1	1	0

Does the LRU page replacement strategy suffer from Belady's anomaly? Explain Belady's anomaly and your answer.

[7 marks]

(b) (i) Describe how a virus infects an executable program, and how an anti-virus program detects known viruses.

(ii) Describe the mechanisms used by encrypted & polymorphic viruses in order to defeat the detection methods commonly used by anti-virus programs.

[6 marks]

(c) Explain why the multi-level feedback queues scheduling algorithm used in CTSS & MULTICS is said to favour short jobs and I/O bound jobs.

[4 marks]

(Question 3 continued on next page.)

(d) Consider the *bounded buffer producer/consumer* problem: there is a buffer shared among a number of processes, some of which (producers) only add items to the buffer and the others (consumers) only take items out of the buffer. The pseudocode describing this problem is given below:

```
1   program Producer/Consumer;

2   const producer_count=10;
3   const consumer_count=10;
4   const buffer_size=100;          { maximum no. of items buffer can hold }
5   shared var num_items: integer;    { items currently in buffer - shared }
6   var i: integer;

7   concurrent_procedure Producer();
8   begin
9       repeat
10          ProduceAnItem();             { takes a random time to complete }
11          while num_items > buffer_size do nothing;            { waiting }
12          AddTheItemToBuffer();
13          num_items := num_items + 1;
14      forever
15  end;

16  concurrent_procedure Consumer();
17  begin
18      repeat
19          while num_items <= 0 do nothing;                     { waiting }
20          RemoveAnItemFromBuffer();
21          num_items := num_items - 1;
22          ConsumeTheItem();            { takes a random time to complete }
23      forever
24  end;

25  begin                                                  { main program }
26      num_items := 0;
27      concurrent_begin
28          for i := 1 to producer_count do
29              Producer();               { starts in a new process/thread }
30          for i := 1 to consumer_count do
31              Consumer();               { starts in a new process/thread }
32      concurrent_end;
33  end.
```

(i) Indicate all critical sections in the above program by giving line numbers.

(ii) Provide the program with mutual exclusion by inserting necessary semaphore operations.

(iii) Assume that producer and consumer processes cannot share any global variables (such as `num_items`) except semaphores. Using pseudocode, outline a solution to the problem with appropriate semaphore operations.

[8 marks]