[PREAMBLE & STUFF TO BE DONE]

UNIVERSITY OF LONDON

CS2630: OPERATING SYSTEMS THEORY

Time Allowed: 2 Hours

Answer FOUR questions

Question 1

a) Draw a state transition diagram for processes in an operating system. When a process is created, in which state will it be?

[4 marks]

b) Explain why a process might enter the *blocked* state and describe two mutual exclusion algorithms which rely on *busy waiting* while the process is in the blocked state. Why should busy waiting be avoided if possible?

[5 marks]

c) What is a *semaphore*? Describe the operations that can be performed on a semaphore. It is not usually possible to read the value of a semaphore without altering it - describe a method to allow processes to find out the value of a semaphore.

[8 marks]

d) Describe the *producer-consumer* problem. Give some examples of where the producer-consumer problem occurs in operating systems and show how semaphores can be used to solve the producer-consumer problem.

[8 marks]

Question 2

a) What is the purpose of a *page replacement algorithm (*PRA)? Describe the characteristics of the optimal PRA.

[4 marks]

b) One of the simplest page replacement algorithms is FIFO. Describe this algorithm and any disadvantages it has. What does *Belady's anomaly* demonstrate about the FIFO algorithm? Does *Belady's anomaly* have any practical implications for page replacement algorithms?

[6 marks]

c) Describe the *Not Recently Used* page replacement algorithm, showing the groups into which pages are placed. Justify why pages from group 0 are chosen to be replaced first.

[6 marks]

d) Explain the difference between *demand paging* and *working set paging*, and describe advantages and disadvantages of each.

[5 marks]

e) A computer has four page frames. The time of loading, time of last access and the R and M bits for each page are as shown below. (Times are in clock ticks).

Page	Loaded	Last ref.	R	M
0	125	280	0	0
1	232	263	1	0
2	118	270	1	1
3	160	280	1	1

Which page will be replaced by each of the following algorithms?

i. FIFO

ii. Second Chance

iii. Not Recently Used

iv. Least Recently Used

[4 marks]

Question 3

a) Explain why an operating system needs a *memory manager* and list some of the functions of the memory manager.

[4 marks]

b) Explain how *linked lists* can be used for memory management, and describe four possible algorithms for choosing a block of memory to allocate from the list.

[6 marks]

c) Why could the amount of memory that a computer has available have an effect on CPU utilisation? Assuming that all processes will spend the same proportion of time p, awaiting I/O, derive an equation which relates the number of processes in memory to the CPU utilisation. Assume a computer has 256 Mbytes of available memory, and that processes occupy 32Mb of memory each. Assume also that there are always processes ready to occupy all the available memory. What is the CPU utilisation if processes spend 80% of their time in I/O wait? What modifications would have the greatest impact on CPU utilisation – doubling the available memory or reducing the I/O wait proportion to 70%?

[15 marks]

Question 4

a) What is the purpose of a *scheduling algorithm*? Describe some of the aims of a scheduling algorithm and give examples of where these aims may conflict.

[8 marks]

b) Switching between processes imposes an overhead on the computer. Explain what activities the operating system must perform to achieve a context switch.

[4 marks]

c) Assume that an operating system uses a round robin scheduling algorithm. Describe the operation of this algorithm. Show how the quantum size used by the round robin algorithm affects the proportion of time used in making context switches.

[4 marks]

d) Show, with examples, how multiple, prioritised queues, with higher priority queues having longer quanta can both improve response time for interactive processes and reduce context switch overhead for CPU intensive processes.

[9 marks]

Question 5

a) Explain the meaning of a processor *interrupt* and describe how an operating system typically handles interrupts. Draw a diagram to show how control is transferred between user programs and interrupt handlers when multiple, prioritised interrupts are allowed.

[8 marks]

b) Describe a typical *direct memory access* DMA operation. Compare this interrupt driven I/O and explain which techniques is best suited to which types of I/O device.

[7 marks]

c) Describe the steps in fulfilling a request for a disk block to be read into memory, showing the source of timing delays at each step. What hardware parameters determine the performance of disk devices? Given a set of example disk read requests, and assuming a first-come-first-served scheduling algorithm show how you would use these parameters to calculate:

i. Average latency

ii. Average turnround time

iii. Overall transfer rate

[10 marks]

Question 6

a) Describe how *deadlock* can arise during resource allocation, give an example of a deadlock situation and list the four conditions that must exist for deadlock to occur.

[9 marks]

b) Describe each of the following techniques and show they prevent (or do not prevent) deadlock arising, and which, if any of the conditions above it addresses.

i. Ordered resource allocation

ii. Pre-allocation of resources

iii. The Ostrich algorithm

[9 marks]

c) Draw a resource allocation graph for the following scenario, determine the circular wait and identify the process which should be terminated by the operating system to break the deadlock.

Process Number	1	2	3
Holds Resource	B	A	C
Waiting for	A, C	B	B

[5 marks]

d) Explain the differences between *deadlock* and *race conditions*.

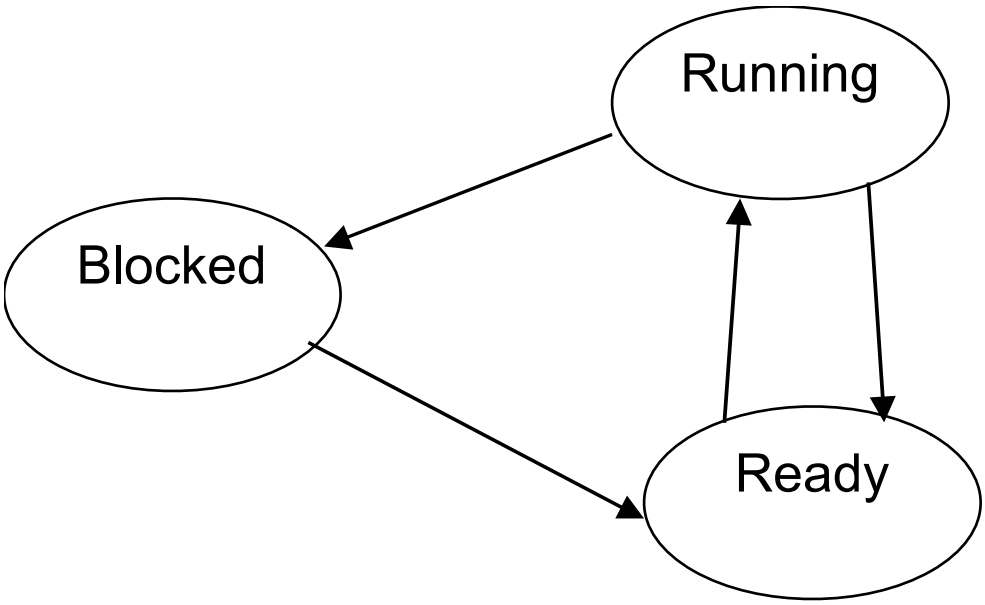
[2 marks]

(End of Question Paper)

MODEL ANSWERS

Marks awarded are shown in square brackets

1. a.



[3]

A newly created process will start in the "Ready" state [1].

1.b. A process might enter the blocked state when it requests an I/O operation and the operation takes some time to complete, for example a read from the keyboard cannot complete until the user presses a key [2]. The "Test and Set Lock" instruction TSL can be used for busy waiting, [1] as can Peterson's Algorithm [1]. Busy waiting should be avoided because it wastes CPU processing [1].

1.c. A semaphore is a type of variable, stored in memory [1]. Two operations can be performed on a semaphore, UP and DOWN [1]. DOWN tests the value of the semaphore and if it is greater than zero, decrements the semaphore and continues; if the semaphore is zero, the process is sent to sleep[2]. UP increments the semaphore and wakes up one process waiting on the semaphore [2]. To obtain the value of a semaphore we could use a separate variable as a counter. Every time a DOWN is carried out on the semaphore the counter must be decremented, and it should be incremented on every UP. [1]. Access to the counter must be protected by another semaphore to ensure mutual exclusion [1].

1.d. The producer-consumer problem is where a producer creates objects or data and places them into a buffer. A consumer will then remove these objects or data from the buffer. The problem is to ensure that the buffer does not overflow and that the consumer does not try to take something from an empty buffer [1].

The problem can be solved by using one semaphore for the producer. This is initially set to the size of the buffer [1] and the producer will do a DOWN on the semaphore every time it puts something in the buffer [1]. This will prevent the buffer from overflowing It will also do an UP on the consumer's semaphore to indicate there is data available [1].

The consumer also has a semaphore, initially set to zero [1]. The consumer will do a DOWN on this semaphore and wait until the producer does an UP [1]. The consumer also needs to do an UP on the producer's semaphore when it removes data from the buffer to tell the producer there is more space available [1].

There is also needs to be a third semaphore to ensure mutual exclusion in the use of the buffer itself [1].

2.a. A page replacement algorithm will determine which page currently in main memory should be replaced when a page fault occurs [2]. The optimal PRA would replace the page which is not going to be accessed for the longest period of time [2].

2.b. The FIFO algorithm will replace the page which has been in memory for the longest time [1]. This has the disadvantage that it takes no account of how much a page is used [1]. Belady's anomaly demonstrates that in certain circumstances having more page frames available will cause more page faults than having fewer page frames [2]. Belady's anomaly does have real practical implications as it does not demonstrate a failure of the page replacement algorithm, just an inefficiency in certain circumstances [2].

2.c. The Not Recently Used algorithm requires two bits of information to be stored for each page, an R bit set every time the page is read [1], and which is cleared on every clock tick; and an M bit set if the page is modified [1]. Pages are grouped as follows:

Group 0: R = 0, M = 0

Group 1: R = 0, M = 1

Group 2: R = 1, M = 0

Group 3: R = 1, M = 1 [2]

Pages are chosen from group 0 first because these pages have not been read since the last clock tick and have not been modified so are not currently being used [2].

2.d. In a demand paging system pages are loaded as soon as they are required, i.e. when a page fault is generated [1]. This is simple to implement and minimises disk traffic [1]. In a system using the working set model, the operating system attempts to pre-load all of the most heavily used pages of a

process [1]. This may reduce the number of page faults, as the page will already by in memory [1] but may also increase disk traffic by loading pages not required [1].

2.e.i. page 2 [1]

2.e.ii. page 0 [1]

2.e.iii page 0 [1]

2.e.iv. page 1 [1]

3.a. An operating system needs a memory manager to manage the memory that is allocated to processes [1]. It must track which memory is allocated to which process [1], allocate memory to new processes [1] and decide which pages to swap to disk if required [1].

3.b. A linked list can be used to manage memory by representing each block of memory as an item in the list [1]. Blocks can either be free or allocated [1], and the list should be ordered by address [1]. Adjacent free blocks must be merged into a single block. Blocks can be chosen by any of the following algorithms:

First fit – first block from the front of the list that is sufficiently large [1]

Next fit – next block from most recently allocated block that is sufficiently large [1]

Best fit – the block that is nearest in size to that required [1].

Worst fit – the largest available block [1].

3.c. There more memory that is available then the more processes that can be held simultaneously in memory. If there are more processes in memory then it is more likely that there is always at least one process ready to run. If there are always processes ready to run then the CPU utilisation can be maximised [3].

Assume there are n processes currently in memory then the CPU utilisation will be $1 - p^n$. [3]

n = 256 / 32 = 8

p = 0.8

CPU utilisation = $1 - 0.8^8 = 83\%$ [3]

Given double memory –

n = 512 / 32 = 16

CPU utilisation = $1 - 0.8^{16} = 97\%$ [3]

Given I/O wait of 70%

$p = 0.7$

CPU utilisation = $1 - 0.7^8 = 94\%$ [3]

4.a. A scheduling algorithm must decide which process to run when there are several processes in the ready state [1]. A scheduling algorithm may try to achieve the following aims:

Fairness - make sure each process gets its fair share of the CPU [1]

Efficiency - keep the CPU busy 100 percent of the time [1]

Response time - minimise response time for interactive users [1]

Turnaround - minimise the time batch users must wait for output [1]

Throughput - maximise the number of jobs processed per hour [1]

Efficiency conflicts with fairness as context switches necessary to give all process a share of the CPU are overhead and reduce CPU efficiency [1]. Response time may conflict with fairness as interactive processes may get more than their share of processor in order to give a sufficient response time [1].

4.b. During a context switch the operating system must save information about the currently executing process [1]. This will include the value of the program counter [1] and the contents of other registers [1]. The corresponding information already stored about the new process must then be loaded into the program counter and registers [1].

4.c. In a round robin scheduling algorithm every process gets an equal length of time on the processor and each process is handled in turn [2]. The shorter the length of time on the processor the more frequently the operating system must make a context switch and hence the greater the proportion of time that is used making context switches [2].

4.d. In a prioritised, multi-queue system, processes are given a particular priority and all processes with the same priority are organised into a round-robin queue [1]. The highest priority queue runs most often, however processes will only run for one quantum of time [1]. The priority of the process will then reduce to that of the next group [1] – i.e. the lower the priority the longer the quantum of time. New processes, and those recently unblocked from an I/O wait are given a high priority [1], and if these are interactive processes then they will typically complete their processing (e.g. echoing a typed character) during the one quantum of time they are given [1]. Because they are high priority they will be selected as the process to run quickly, thus achieving the required rapid response time for interactive processes [1]. A CPU intensive process will also start with a high priority, however it will not complete processing in one quantum. It will then be moved to a lower priority queue where it will not run

so often but when it does run it will run for a longer period of time [1]. Each time the priority is reduced the period of time on the processor will double. A CPU intensive process will thus run only after higher priority interactive tasks have run but will run for longer when it does get on the processor [1]. The total number of context switches that will be required by the CPU intensive process will thus be much less than in a round robin queue where each process gets the same quantum of time on the processor [1].

5.a. An interrupt is an electrical signal to the processor that a particular event has occurred [1]. When an interrupt is detected the processor will store the program counter and register values of the currently executing process and call an interrupt handler [2]. The interrupt handler will do what ever processing is necessary in response to the interrupt and will then reload the original program [1].

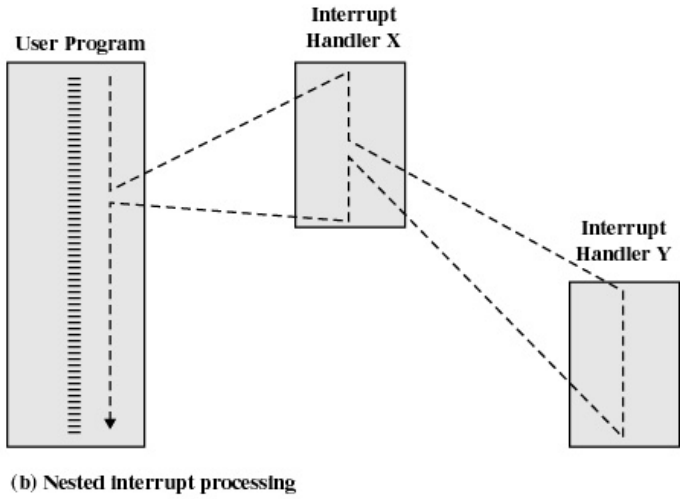


Figure 1.12 Transfer of Control with Multiple Interrupts [4]

5.b. In a DMA operation the processor programs the DMA controller with the address, size and destination of the data to be transferred [1]. The DMA controller will then manage the transfer of data directly, using the system bus [1], without the data having to go through the processor [1]. In interrupt driven IO a device with an item of data available will interrupt the processor [1], the interrupt handler will move the item of data into memory [1]. Interrupt driven IO is best suited to IO devices with small volumes of data available infrequently; DMA is best suited to moving large volumes of data in complete blocks[2].

5.c. For a disk block to be read into memory the operating system must first wait for the device to become available. This is the first source of delay [1]. The disk device may have multiple channels available and we may also have to wait

for one of the channels to become available [1]. We must then wait for the disk read head to move across the disk to the required track, this is a seek wait and a further source of delay [1]. We must then wait for the disk to rotate so that the required block is below the read head, this is another delay [1]. Finally, once the data has been read off the disk it must be transferred into memory, the final source of delay [1]. Disk performance is determined by the track to track head movement time [1] and the rotational speed [1]. We can use these values, given a set of read requests, and assuming first-come-first-served scheduling we can calculate:

Average latency: calculate and sum for each read the number of tracks that the head must move and hence the seek time, add on half the rotational delay, divide by the number of reads [1]

Average turnround: calculate and sum for each read the time taken to fulfil the request, this includes all of the delays above, divide by the number of reads [1]

Overall transfer rate: Total amount of transferred divided by the total time taken [1].

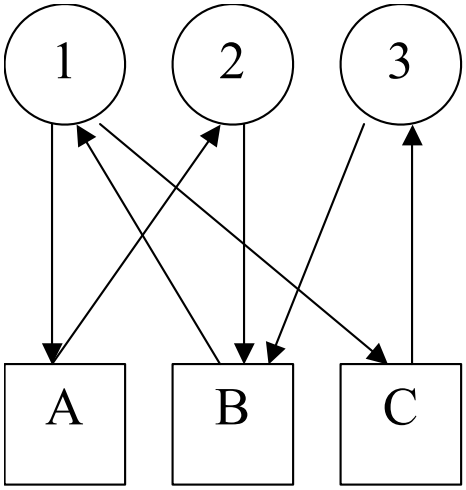
6.a. Deadlock can arise when two processes are competing for the same resources, or trying to communicate with each other [1] and neither can continue until the other one does [2]. For example process A holds resource 1 and requires resource 2, while process B holds resource 2 and requires resource 1 [2]. Deadlock can only arise if only one process can use the resource at the same time [1], if processes holding resources can attempt to request further resources [1], if resources cannot be pre-empted from processes that hold them [1], and that a circular wait between processes must exist [1].

6.b.i. In ordered resource allocation, all resources are assigned numbers and processes can only request further resources that are numbered higher than any resource it currently holds. This prevents the circular wait condition [3].

6.b.ii. With pre-allocation of resources all of the resources that a process will require must be requested before the process is allowed to begin. If one or more of the resources is not available the process will sleep until all are available. This breaks the condition that processes holding resources can request further resources [3].

6.b.iii. The Ostrich Algorithm is to ignore the problem and hope that it does not arise. This is because there is no efficient way to avoid deadlock and it may occur very rarely. The ostrich algorithm does not address any of the deadlock conditions [3].

6.c.



Processes 1 and 2 are in a circular wait for resources A and B. Processes 1 and 3 are in a circular wait for resources B and C. Therefore process 1 should be terminated [5].

6.d. Deadlock is where two processes hold resources that the other is trying to obtain. A race condition is where two processes both believe that they have exclusive access to the *same* shared resource [2].