Question 1

Two processes, P_0 and P_1, are to be run and they update a shared variable. This update is protected by Petersons solution to the mutual exclusion problem.

a) Show Petersons algorithm and show the truth table for the part of the algorithm which dictates if a process is allowed to enter its critical region.

(10)

b) P_0, attempts to enter its critical region. Show the state of the variables that are created/updated. Will P_0 be allowed to enter its critical region? If not, why not?

(5)

c) P_1, attempts to enter its critical region. Show the state of the variables that are created/updated. Will P_1 be allowed to enter its critical region? If not, why not?

(5)

d) P_0 leaves its critical region. What effect does this have on the variables?

(2)

e) Assume no processes are running and P_0 and P_1 try to enter their critical region at ***exactly*** the same time. What will happen?

(3)

Question 2

a) With regards to I/O design principles describe the layers of the I/O system and justify this structure.

(15)

b) Consider the following C-language I/O statement:

```
count = read(fd, buffer, nbytes);
```

where fd is an integer (called the file descriptor)
buffer is a memory address pointing into the processes data memory
nbytes is an integer indicating the number of bytes to be read.

Supposing that fd refers to a file on disk with a controller which uses DMA, describe how this statement may be handled in the layers of the I/O system?

5 marks

c) How are devices represented in the UNIX operating system? How are the drivers specified? Could the file descriptor in question (b) be referring to a terminal? What would be the difference if this were so?

5 marks

Question 3

a) Describe the following scheduling algorithms

- Non Pre-Emptive, First Come, First Serve
- Round Robin
- Shortest Job First

(9)

b) Given the following processes and burst times

Process	Burst Time
P_1	13
P_2	5
P_3	23
P_4	3
P_5	31
P_6	3
P_7	14

Calculate the average wait time when each of the above scheduling algorithms is used?

Assume that a quantum of 6 is being used.

(12)

c) Which scheduling algorithm, as an operating systems designer, would you implement?

(4)

Question 4

(a) Given a disk with 200 tracks, where track requests are received in the following order

55, 58, 39, 18, 90, 160, 150, 38, 184.

The starting position for the arm is track 100. Calculate the number of tracks crossed when the following algorithms are used

- First Come First Serve
- Shortest Seek First
- The elevator algorithm starting in the direction UP.

(15)

b) Briefly explain, in which of the following cases can the algorithms in (a) augment its performance?

- A process is reading 10,000 blocks with consecutive disk addresses.
- A process is reading 10,000 blocks with random disk addresses.
- A process is creating child processes to read 10,000 blocks with random addresses.
- Processes are communicating with each other by writing and reading blocks to disk.

(8)

c) In the last case of question (b), could the algorithm influence the synchronisation between the processes?

(2)

Question 5

a) The buddy system is a memory management scheme that uses variable sized partitions.

Explain the basic principle behind the buddy system.

(5)

b) Assume a computer with a memory size of 256K, initially empty. Requests are received for blocks of memory of 17K, 6K, 63K and 9K. Show how the buddy system would deal with each request, showing the memory layout at each stage and the status of the lists at each stage.

(8)

(c) The processes terminate in the following order; 6K, 9K, 17K and 63K. Discuss what happens as each process terminates.

(4)

d) Describe and evaluate an alternative to the buddy system

(8)

Question 6

a) Every file in a filing system has a set of attributes (read only, date created etc.). Assume a filing system allows an attribute of *temporary*, meaning the creating process only uses the file during the time it is executing and has no need for the data thereafter. Assume the process is written correctly, so that it deletes the file at the end of its execution. Do you see any reason for an operating system to have *temporary* file attribute? Give your reasons.

(5)

b) An operating system supplies system calls to allow you to COPY, DELETE and RENAME a file.
Discuss the differences between using COPY/DELETE and RENAME to give a file new name?

(5)

c) There is some debate as to what will constitute a fifth generation computer. Assume such a computer is available. What do you think will differentiate it from the computers of today?
What advances do you think need to be made in order to produce a fifth generation computer?

(15)

Question 1 – Model Answer

This question formed the basis of an exercise sheet given out in the lectures.

(a) Petersons Solution – The algorithm

```
int No_Of_Processes;
int turn;
int interested[No_Of_Processes];

void enter_region(int process) {
    int other;
    other = 1 – process;
    interested[process] = TRUE;
    turn = process;
    while(turn == process &&  interested[other] == TRUE);
}

void leave_region(int process) {
    interested[process] = FALSE;
}
```

8 marks (pro-rata) for re-producing the algorithm.

The truth table for the while loop

	turn == process	Interested[other] = TRUE	Action
1.	F	F	F (Continue)
2.	F	T	F (Continue)
3.	T	F	F (Continue)
4.	T	T	T (Wait)

2 marks for showing the truth table

(b) P_0 tries to enter its critical region
other = 1, turn = 0
interested[0] = TRUE, interested[1] = FALSE
turn == process == TRUE && interested[other] == FALSE, therefore line 3 ***will*** be called (above truth table) and thus P_0 will be allowed to enter its critical region

5 marks, pro-rata, for how much the student re-produces

(c) P_1 tries to enter its critical region
other = 0, turn = 1
interested[0] = TRUE, interested[1] = TRUE
turn == process == TRUE && interested[other] == TRUE, therefore line 3 will be called and thus P_1 will ***NOT*** be allowed to enter its critical region

5 marks, pro-rata, for how much the student re-produces

(d) P_0 leaves its critical region. What effect does this have on the variables?
P_0 will set interested[0] to FALSE. This will allow the WHILE loop that P_1 is executing to terminate as
turn == process == TRUE && interested[0] == FALSE, thus allowing line 3 to be called and P_1 can now enter its critical region

2 marks, pro-rata, for how much the student re-produces

(e) P_0 and P_1 try to enter their critical region at *exactly* the same time.
As *other* is a local variable then both calls can set this variable without any problems. Similarly, the relevant element of the *interested* array can be set without race conditions occurring.
As *turn* is a shared variable, there is the potential of race conditions occurring. Assume P_0 sets turn to zero and it is immediately set to one by P_1 (or vice versa). Due to the WHILE statement, it does not matter which process sets the variable first (or last), only one of the processes will be allowed to enter its critical region. The other will be forced to wait. That is,

	turn == process	interested[other] = TRUE	Action
P_0	F	T	F (Continue)
P_1	T	T	T (Wait)

3 marks if the student mentions all the above. 2 marks for mentioning part of it. 1 consolation mark for stating that one process can enter its critical region and the other one will be forced to wait.

Question 2 – Model Answer

a) With regards to I/O design principles describe the layers of the I/O system and justify this structure.

Layer	Functions
User processes	Produce I/O request, formatting, spooling
Device independent software	Naming, protection, blocking, buffering, allocating
Device drivers	Setting device registers, check status
Interrupt handlers	Wake up the driver when I/O is ready
Hardware	Perform the I/O operation

6 marks for producing this table (pro-rata)

Note: Stallings distinguishes general devices, communication devices and file systems. Each has a hardware, a scheduling and controlling and a device I/O layer (driver). The top layer is always "user processes". The one layer in between is different in the three cases: "logical I/O", "Communication architecture" and three "sub-layers" for the file system. Since this was presented in class, it should be considered as a correct answer.

The layered structure is justified by the need for efficiency on the hardware side and for uniformity, simplicity and device independence on the user side. User processes need a uniform way to treat I/O devices (e.g. UNIX). The naming must be independent of the devices. Errors should be handled at the lowest level possible. Processes may produce I/O requests asynchronously, while, for efficiency, devices may be better of handling them in parallel. Some devices are dedicated, while other can be shared. The handling of dedicated devices should be transparent and uniform.
Interrupt handlers should be deeply hidden in the operating system, since treating interrupts in a correct way is tricky and can cause serious system errors when done incorrectly. The interrupt handlers may communicate with the device handlers through the use of classical synchronisation techniques such as semaphores, signals or monitors.
Device drivers contain all device dependent code. A driver can handle one type of device or a set of related devices. The driver knows how to steer the controller, knows about interleaving, tracks, cylinders,… It should offer an abstraction to the software layer above. It must accept requests at any time, but may put them in a queue. The driver may wait for the controller to finish, effectively blocking, or it may continue immediately in case it does not expect the controller to respond. In any case it has to check for possible errors.

Device independent I/O software will perform the following tasks:

Provide uniform interfaces for the device drivers
Do the naming
Handle device protection
Produce device independent block sizes
Buffer
Memory management on block devices
Manage dedicated devices
Handle errors

As an example of naming, one can refer to the minor and major device numbers in UNIX, and the representation of devices in the /dev directory. (see later)
Protection depends on the system. Personal systems (MS-DOS) need less protection than mainframes. UNIX takes a way between with the rwx scheme.
Disks may differ in sector size. The OS should offer one block size and the device independent software must do the translation. This is related to the concept of buffering. Buffers are used with block and character devices to bridge processing speed differences.
The assignment of blocks, and the management of free blocks does not depend on the device, and is done in this layer.
Dedicated devices must be lockable. This can be implemented through the OPEN system call which locks the device or produces an error when it is occupied. CLOSE will free the devices.
Although errors are mostly treated by the device drivers, some errors may be passed to the upper layers.
In the user processes I/O is handled by library routines translating the I/O requests into system calls. Spooling systems must be considered as a user level solution for dedicated devices.

9 marks for this discussion (pro-rata)

b) Consider the following C-language I/O statement:

```
count = read(fd, buffer, nbytes);
```

where fd is an integer (called the file descriptor)
buffer is a memory address pointing into the processes data memory
nbytes is an integer indicating the number of bytes to be read.

Supposing that fd refers to a file on disk with a controller which uses DMA, describe how this statement may be handled in the layers of the I/O system?

"read" is a library routine which is linked to the program. It will produce a system call to read nbytes from the file into the buffer. This system call will be treated at the device independent software level to produce one or more I/O requests for blocks to be read. It will select a buffer and pass it to the device driver. The device driver will initiate a hardware I/O. It will pass the buffer address to the DMA module in the

controller. The device driver will then block. The controller will perform the I/O operation on the disk, reading the requested sectors and checking for errors. Once the operation is finished, it will produce an interrupt. The device driver will then pass the results to the device independent software which will copy the requested number of bytes to the original buffer.

5 marks, pro-rata, depending on how much of this the student covers

c) How are devices represented in the UNIX operating system? How are the drivers specified? Could the file descriptor in question (b) be referring to a terminal? What would be the difference if this were so?

In UNIX devices are mounted into the filesystem.

1 mark

The drivers are specified by their major and minor numbers.

1 mark

Since devices look like files, the fd could refer to a terminal. In this case the device independent software would apply buffering strategies for terminals which may be "raw" (character wise) or "cooked" (line wise). Other device drivers would be involved.

3 marks, pro-rata

Question 3 – Model Answer

a) Describe the following scheduling algorithms

3 marks available for each algorithm, pro-rata

- **Non Pre-Emptive, First Come, First Serve**

An obvious scheduling algorithm is to execute the processes in the order they arrive and to execute them to completion. In fact, this simply implements a non-preemptive scheduling algorithm.
It is an easy algorithm to implement. When a process becomes ready it is added to the tail of ready queue. This is achieved by adding the Process Control Block (PCB) to the queue.
When the CPU becomes free the process at the head of the queue is removed, moved to a running state and allowed to use the CPU until it is completed.
The problem with FCFS is that the average waiting time can be long.

- **Round Robin**

The processes to be run are held in a queue and the scheduler takes the first job off the front of the queue and assigns it to the CPU (so far the same as FCFS).
In addition, there is a unit of time defined (called a ***quantum***). Once the process has used up a quantum the process is preempted and a context switch occurs. The process which was using the processor is placed at the back of the ready queue and the process at the head of the queue is assigned to the CPU.
Of course, instead of being preempted the process could complete before using its quantum. This would mean a new process could start earlier and the completed process would not be placed at the end of the queue (it would either finish completely or move to a blocked state whilst it waited for some interrupt, for example I/O).

The average waiting time using RR can be quite long.

- **Shortest Job First**

Using the SJF algorithm, each process is tagged with the length of its next CPU burst. The processes are then scheduled by selecting the shortest job first.

In fact, the SJF algorithm is provably optimal with regard to the average waiting time. And, intuitively, this is the case as shorter jobs add less to the average time, thus giving a shorter average.

The problem is we do not know the burst time of a process before it starts.

For some systems (notably batch systems) we can make fairly accurate estimates but for interactive processes it is not so easy.

b) Given the following processes and burst times etc.

4 marks available for each algorithm. Full marks will be awarded for showing all workings.

FCFS
The processes would execute in the order they arrived. Therefore, the processes would execute as follows, with the wait times shown.

Process	Burst Time	Wait Time
P_1	13	0
P_2	5	13
P_3	23	18
P_4	3	41
P_5	31	44
P_6	3	75
P_7	14	78

The average wait time is calculated as ∑WaitTime / No Of Processes
= (0 + 13 + 18 + 41 + 44 + 75 + 78) / 7
= 269 / 7
= **32.43**

Round Robin
This scheme allocates the processes to the CPU in the order they arrived, but only allows them to execute for a fixed period of time (quantum = 8). Then the process is pre-empted and placed at the end of the queue.

This leads to the following wait times for each process

Process	Wait Time
P_1	47
P_2	6
P_3	56
P_4	17
P_5	61
P_6	26
P_7	60

I would expect the students to give more detail than this (as we did in the lecture exercises), to show they have applied the algorithm correctly. This, in essence, shows the wait time at each stage of the process. If the students do this, but give the correct answer, they should be given credit for applying the algorithm (half marks).
However, if they do not show the working, but get the correct answer, then they should get ***full*** marks as they ***must*** have applied the algorithm to get the correct answer.

Summing all the wait times and dividing by the number of processes, gives us the average wait time. That is

(47 + 6 + 56 + 17 + 61 + 26 + 60) = 273 / 7
= **39.00**

Shortest Job First
This scheme, simply executes the process using the burst time as the priority. That is

Process	Burst Time	Wait Time
P_6	3	0
P_4	3	3
P_2	5	6
P_1	13	11
P_7	14	24
P_3	23	38
P_5	31	61

The average wait time is calculated as ∑WaitTime / No Of Processes
= (0 + 3 + 6 + 11 + 24 + 38 + 61) / 7
= 143 / 7
= **20.43**

c) Which scheduling algorithm, as an operating systems designer, would you implement?

This is an opportunity for the student to give their views on scheduling algorithms. As we discussed in the lectures there is no ideal scheduling algorithm, there are always trade offs and compromises.

No marks will be awarded for saying shortest job first (SJF) would be implemented (as this is not possible), but an algorithm that estimates the burst time, so that SJF can be partially emulated would get some marks.

I would also give marks for saying that multi-level feedback queue scheduling would be a good choice as, by varying the parameters to this algorithm, it is possible to emulate all the other algorithms we considered. But the student should also say that even this is not ideal as vast amounts of testing and guesswork would still be needed. All implementing this algorithm does is give you the flexibility to try various algorithms.

Many other answers are possible and the marks will be awarded on the basis of their argument and how they defend it.

Question 4 – Model Answer

a) Given a disk with 200 tracks, where track requests are received in the following order etc.

FCFS	#crossings	SSF	#crossings	Elevator	#crossings
100		100		100	
55	45	90	10	150	50
58	3	58	32	160	10
39	19	55	3	184	24
18	21	39	16	90	94
90	72	38	1	58	32
160	70	18	20	55	3
150	10	150	132	39	16
38	112	160	10	38	1
184	146	184	24	18	20
	498		**248**		**250**

5 marks for each for each case (=15)

b) Briefly explain, in which of the following cases can the algorithms in (a) augment its performance?

1. **A process is reading 10,000 blocks with consecutive disk addresses.**
2. **A process is reading 10,000 blocks with random disk addresses.**
3. **A process is creating child processes to read 10,000 blocks with random addresses.**
4. **Processes are communicating with each other by writing and reading blocks to disk.**

The performance of the algorithms is influenced by the way in which the requests are initiated.

1. Consecutive production of I/O requests will make the process block during each request, waiting before launching the following request. Although the arm will be well positioned before each request, since the addresses are consecutive, the algorithms cannot function since they obtain the requests one by one.
2. Since the addresses are random, this operation will cause more arm movements. But since the I/O requests are produced one after the other, the algorithms still cannot operate.

3. The child processes will launch parallel I/O operations. The algorithms will reorder the random set of addresses so that better performance is obtained.
4. Again processes are working in parallel on related block addresses. The algorithms will reorder the requests.

2 marks for each for each case (=8)

c) In the last case of question (b), could the algorithm influence the synchronisation between the processes?

Since, in the last case, the algorithms do a reordering, the I/O requests will not be performed in the order that they are produced. This may influence the synchronisation between the processes.

2 marks

Question 5 – Model Answer

a) The buddy system is a memory management scheme that uses variable sized partitions etc….

If we keep a list of holes (in memory) sorted by their size, we can make allocation to processes very fast as we only need to search down the list until we find a hole that is big enough.
The problem is that when a process ends the maintenance of the list is complicated. In particular, merging adjacent holes is difficult as the entire list has to be searched in order to find its neighbours.

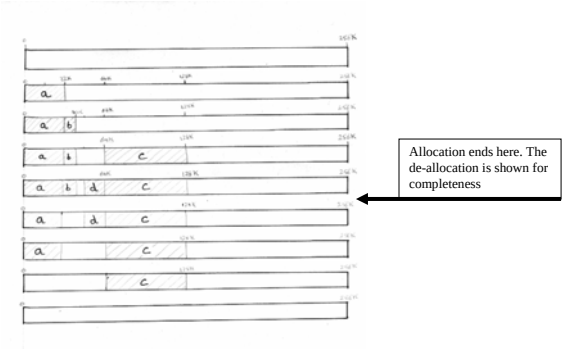
The ***Buddy System*** is a memory allocation that works on the basis of using binary numbers as these are fast for computers to manipulate.

Lists are maintained which stores lists of free memory blocks of sizes 1, 2, 4, 8,…, *n*, where *n* is the size of the memory (in bytes). This means that for a 256K memory we require 19 lists.
If we assume we have 256K of memory and it is all unused then there will be one entry in the 256K list; and all other lists will be empty.

5 marks, pro-rata

b) Assume a computer with a memory size of 256K, initially empty. Requests are received for blocks of memory of 17K, 6K, 63K and 9K. Show how the buddy system would deal with each request, showing the memory layout at each stage and the status of the lists at each stage.

The memory allocation will look like this (students only need to show down to the arrow).



1 mark for each process allocated correctly

The lists, at each stage, will look like this

List No.	Block Size	Start	17K	6K	63K	9K
1	1					
2	2					
3	4					
4	8					
5	16					
6	32					
7	64					
8	128					
9	256					
10	512					
11	1024 (1K)					
12	2048 (2K)					
13	4096 (4K)					
14	8192 (8K)			40K	40K	40K
15	16384 (16K)			48K	48K	
16	32768 (32K)		32K			
17	65536 (64K)		64K	64K		
18	131072 (128K)		128K	128K	128K	128K
19	262144 (256K)	0				

Note : The list entries are memory addresses where a block of that size starts (e.g. initially there is one 256K block starting at address zero). The student should specify the addresses ***not*** just the number of blocks of memory of that size.
Each list could, in theory, hold more than one element but, in this case, this never happens.

1 mark for each list specified correctly where memory is allocated (i.e. no marks for showing the initial state).

(c) The processes terminate in the following order; 6K, 9K, 17K and 63K. Discuss what happens as each process terminates.

The effect of each of these is described below and the changing lists are also shown. The student can show either – but a description would be better.

- When the 6K process is returned (b in the above diagram), the 8K slot of memory is returned. This returns a block of memory from 32K to 40K. Checking the 8K list, it is found that there is an adjacent block of memory, which can be merged into a 16K block. Therefore the result is that the two 8K blocks are merged into a 16K block and this is added to the list.
- When the 9K block is returned (d in diagram) this releases a 16K block of memory from 48K to 64K. Checking the 16K list there is a block from 32K (to 48K). As the two 16K blocks are consecutive it allows these two blocks to be merged into a 32K block.
- The 17K block (a in diagram) returns a 32K block of memory from 0K to 32K. This can be merged with the block from 32K to 64K, giving a block in the 64K list.
- The final release of memory (64K, c in diagram) allows two 64K blocks to be merged into a 128K block and then two 128K blocks to be merged to return to a position where the memory is empty and there is only one list entry in the 256K block starting at 0K.

List No.	Block Size	Initial	6K	9K	17K	63K
1	1					
2	2					
3	4					
4	8					
5	16					
6	32					
7	64					
8	128					
9	256					
10	512					
11	1024 (1K)					
12	2048 (2K)					
13	4096 (4K)					
14	8192 (8K)	40K				
15	16384 (16K)		32K			
16	32768 (32K)			32K		

17	65536 (64K)				0K	
18	131072 (128K)	128K	128K	128K	128K	
19	262144 (256K)					0K

1 mark for each correct answer

d) Describe and evaluate an alternative to the buddy system

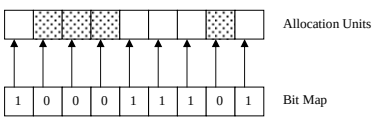
Two alternatives were presented in the lectures. These were managing memory with bit maps and managing memory with linked lists.

I would only expect a brief discussion of one of these methods. The notes below give sample answers; although I would not expect the student to go into as much detail (certainly for linked lists) – just explain the basic principle of one of the schemes.

I would expect a brief evaluation with another scheme (probably the buddy system), giving an evaluation of the scheme they have chosen to describe.

Memory Usage with Bit Maps

Under this scheme the memory is divided into allocation units and each allocation unit has a corresponding bit in a bit map. If the bit is zero, the memory is free. If the bit in the bit map is one, then the memory is currently being used.
This scheme can be shown as follows.

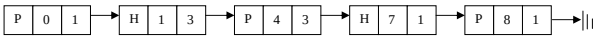


The main decision with this scheme is the size of the allocation unit. The smaller the allocation unit, the larger the bit map has to be. But, if we choose a larger allocation unit, we could waste memory as we may not use all the space allocated in each allocation unit.

The other problem with a bit map memory scheme is when we need to allocate memory to a process. Assume the allocation size is 4 bytes. If a process requests 256 bytes of memory, we must search the bit map for 64 consecutive zeroes. This is a slow operation and for this reason bit maps are not often used.

Memory Usage with Linked Lists

Free and allocated memory can be represented as a linked list. The memory shown above as a bit map can be represented as linked list as follows.



Each entry in the list holds the following data

- P or H : for Process or Hole
- Starting segment address
- The length of the memory segment
- The next pointer is not shown but assumed to be present

In the list above, processes follow holes and vice versa (with the exception of the start and the end of the list). But, it does not have to be this way. It is possible that two processes can be next to each other and we need to keep them as separate elements in the list so that if one process ends we only return the memory for that process.
Consecutive holes, on the other hand, can always be merged into a single list entry.

This leads to the following observations when we a process terminates and we return the memory.

A terminating process can have four combinations of neighbours (we'll ignore the start and the end of the list to simplify the discussion).
If X is the terminating process the four combinations are

- In the first option we simply have to replace the P by an H, other than that the list remains the same.
- In the second option we merge two list entries into one and make the list one entry shorter.
- Option three is effectively the same as option 2.
- For the last option we merge three entries into one and the list becomes two entries shorter.

In order to implement this scheme it is normally better to have a doubly linked list so that we have access to the previous entry.

When we need to allocate memory, storing the list in segment address order allows us to implement various strategies.

First Fit : This algorithm searches along the list looking for the first segment that is large enough to accommodate the process. The segment is then split into a hole and a process. This method is fast as the first available hole that is large enough to accommodate the process is used.

Best Fit : Best fit searches the entire list and uses the smallest hole that is large enough to accommodate the process. The idea is that it is better not to split up a larger hole that might be needed later.

Best fit is slower than first fit as it must search the entire list every time. It has also be shown that best fit performs worse than first fit as it tends to leave lots of small gaps.

Worst Fit : As best fit leaves many small, useless holes it might be a good idea to always use the largest hole available. The idea is that splitting a large hole into two will leave a large enough hole to be useful.
It has been shown that this algorithm is no very good either.

These three algorithms can all be speeded up if we maintain two lists; one for processes and one for holes. This allows the allocation of memory to a process to be speeded up as we only have to search the hole list. The downside is that list maintenance is complicated. If we allocate a hole to a process we have to move the list entry from one list to another.

However, maintaining two lists allow us to introduce another optimisation. If we hold the hole list in size order (rather than segment address order) we can make the best fit algorithm stop as soon as it finds a hole that is large enough. In fact, first fit and best fit effectively become the same algorithm.

The Quick Fit algorithm takes a different approach to those we have considered so far. Separate lists are maintained for some of the common memory sizes that are requested. For example, we could have a list for holes of 4K, a list for holes of size 8K etc. One list can be kept for large holes or holes which do not fit into any of the other lists.
Quick fit allows a hole of the right size to be found very quickly, but it suffers in that there is even more list maintenance.

Question 6 – Model Answer

The first two parts of this question (a and b) were not explicitly covered in the lectures and, answering them, could show evidence of reading around the subject. However, anybody with even limited experience of using an operating system should be able to provide some form of an answer. The questions are not difficult.

Part (c) really is an opportunity for the student to give their own views.

a) Every file in a filing system has a set of attributes (read only, date created etc.). Assume a filing system allows an attribute of *temporary*, meaning the creating process only uses the file during the time it is executing and has no need for the data thereafter.
Assume the process is written correctly, so that it deletes the file at the end of its execution. Do you see any reason for an operating system to have *temporary* file attribute? Give your reasons.

The main reason for the attribute is when a process terminates abnormally, or if the system crashes. Under these circumstances the temporary file would not be deleted. However, by checking the temporary attribute of all files the operating system is able to delete those files are marked as temporary, thus keeping the filing system "tidy."
Under normal circumstances, the attribute, is not needed.
Other reasons could be that the OS could decide to place all temporary files in a certain location – allowing the programmer to simply create a temporary file without having to concern him/herself with the location details.

b) An operating system supplies system calls to allow you to COPY, DELETE and RENAME a file.
Discuss the differences between using COPY/DELETE and RENAME to give a file new name?

I would expect most students to say that there is a performance impact in using copy/delete as the entire file is copied. If you use rename then only the index entry has to be changed.
Limited marks will be give for this, with the rest of the marks being given for the students other arguments – for example…

Perhaps a not so obvious reason, is that if you copy a file you create a brand new file and some of the attributes will change (for example, date created). If you rename a file the, date created attribute, for example, would not be changed.

c) There is some debate as to what will constitute a fifth generation computer. Assume such a computer is available. What do you think will differentiate it from the computers of today?
What advances do you think need to be made in order to produce a fifth generation computer?

This question is really up to the student to provide a convincing argument as to what they think. The lectures notes are given below. For simply re-producing that I will award half the marks for the question. I am really looking for the student to provide their own, original, thoughts.

Whatever answer they give, I would expect them to make the point that each generation of computing has been hardware driven. Is this going to be the case for the next generation?

If you look through the descriptions of the computer generations you will notice that each have been influenced by new hardware that was developed (vacuum tubes, transistors, integrated circuits and LSI). The fifth generation of computers may be the first that breaks with this tradition and the advances in software will be as important as advances in hardware.
One view of what will define a fifth generation computer is one that is able to interact with humans in a way that is natural to us. No longer will we use mice and keyboards but we will be able to talk to computers in the same way that we communicate with each other. In addition, we will be able to talk in any language and the computer will have the ability to convert to any other language.
Computers will also be able to reason in a way that imitates humans.

Just being able to accept (and understand!) the spoken word and carry out reasoning on that data requires many things to come together before we have a fifth generation computer. For example, advances need to be made in AI (Artificial Intelligence) so that the computer can mimic human reasoning. It is also likely that computers will need to be more powerful. Maybe parallel processing will be required. Maybe a computer based on a non-silicon substance may be needed to fulfill that requirement (as silicon has a theoretical limit as to how fast it can go).

This is one view of what will make a fifth generation computer. At the moment, as we do not have any, it is difficult to provide a reliable definition.

15 marks available to award based on the students argument and justification