Question 1

a) Describe the Producer/Consumer problem.

(3 marks)

b) Describe the problems associated with producing a software solution to the producer/consumer problem.

(7 marks)

c) Show a possible solution to the above problem, stating any assumptions that you make.

(15 marks)

Question 2

a) Describe the four generations of computing and how operating systems developed as a result.

(12 marks)

b) 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?

(13 marks)

Question 3

a) Describe the following scheduling algorithms
   - Non Pre-Emptive, First Come, First Serve
   - Round Robin
   - Shortest Job First

(9 marks)

b) Given the following processes and burst times

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>10</td>
</tr>
<tr>
<td>P2</td>
<td>6</td>
</tr>
<tr>
<td>P3</td>
<td>22</td>
</tr>
<tr>
<td>P4</td>
<td>9</td>
</tr>
<tr>
<td>P5</td>
<td>31</td>
</tr>
<tr>
<td>P6</td>
<td>5</td>
</tr>
<tr>
<td>P7</td>
<td>19</td>
</tr>
</tbody>
</table>

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

Assume that a quantum of 8 is being used.

(12 marks)

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

(4 marks)

Question 4

a) Describe the benefits of a mono-programming operating system.

(5 marks)

b) A company, using a multi-programming operating system, has 1 megabyte of memory. The operating system occupies 250K of memory and every process that is executed also requires 250K of memory. The processes have an average I/O wait time of 80%.

The company ask you if they should invest in more memory and, if so, how much. What would you advise and why?

Would your advice change if the company said they had made a mistake and the average I/O wait time was only 20%? If so, why?

(20 marks)
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 marks)

b) Assume a computer with a memory size of 256K, initially empty. Requests are received for blocks of memory of 5K, 25K, 35K and 20K. Show how the buddy system would deal with each request, showing the memory layout at each stage and the status of the lists at the end.

(10 marks)

c) Describe and evaluate an alternative to the buddy system.

(10 marks)

---

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 attributes? Give your reasons.

(5 marks)

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 a new name?

(5 marks)

c) An operating system only allows a single directory hierarchy but allows arbitrary long filenames. Could you simulate something approximating a hierarchical file system? Give your reasons.

(5 marks)

d) When a file is removed, the blocks it occupies are simply placed back onto the free list. Can you see any problems with this? If so, how would you overcome them and what problems, if any, would now exist and how would you resolve these?

(5 marks)

e) When the UNIX filling system opens a file its i-node is copied into memory. It has been suggested, that with memory getting cheaper that if n processes open the same file then n copies of the i-node could be held in memory. Is this a good idea? Give your reasons.

(5 marks)
Here is an example of a series of Down and Up’s. We are assuming we have a semaphore called mutex (for mutual exclusion). It is initially set to 1. The subscript figure, in this example, represents the process, p, that is issuing the Down.

```c
void consumer(void) {
    semaaphore full = 0;
    typedef int semaphore;

    // we can use semaphores to ensure that only one process is in its critical section at any one time, i.e. the principle of mutual exclusion.
    // We can also use semaphores to synchronise processes. For example, the producer and consumer functions in the producer-consumer problem. Take a look at this program fragment:
    int BUFFER_SIZE = 100;
    typedef int semaphore;
    semaphore mutex = 1;
    semaphore empty = BUFFER_SIZE;
    semaphore full = 0;

    void producer(void) {
        int item;
        while(TRUE) {
            up(&empty); // increment count of empty slots
            down(&mutex); // leave critical region
            remove_item(&item); // remove item from buffer
            down(&full); // decrement full count
            enter_item(item); // put item in buffer
            down(&empty); // decrement empty count
        }
    }

    void consumer(void) {
        int item;
        while(TRUE) {
            down(&full); // decrement full count
            up(&mutex); // enter critical region
            remove_item(&item); // remove item from buffer
            up(&mutex); // leave critical region
            up(&empty); // increment count of full slots
            consuine_item(&item); // print item
        }
    }
}
```

The mutex semaphore (given the above example) should be self-explanatory.

---

**Operating Systems (G53OPS) - Examination**

**Question 2 – Model Answer**

(a) Describe the four generations of computing and how operating systems developed as a result.

Three marks will be given for each generation.

The notes/lecture material for this section is given below. The important points are:

- **First Generation**
  - 1945-1955
  - No Operating System
  - Based on vacuum tubes

- **Second Generation**
  - 1955-1965
  - Had an operating system
  - Batch jobs introduced
  - Based on transistors

- **Third Generation**
  - 1965-1980 (or 1971 depending on your view)
  - Multi-programming and time-sharing possible
  - Spooling possible
  - Based on integrated circuits

- **Fourth Generation**
  - 1980 (or 1971) to present
  - Start of PC revolution so MS-DOS, UNIX etc. were developed.
  - Based on VLSI

- **First Generation (1945-1955)**

Like many developments, the first digital computer was developed due to the motivation of war. During the second World War many people were developing automatic calculating machines. For example:

- In 1944 a German engineer (Kemmlau Zuse) had developed a computer (the Z3) that designed airplanes and missiles.
- In 1943, the British had built a code breaking computer called Colossus which decoded German messages (in fact, Colossus only had a limited effect on the development of computers as it was not a general purpose computer – it could only break codes – and it was only kept secret until long after the war ended).
- In 1943, Howard H. Aiken, an engineer with IBM, had built an all-electronic calculator that created ballistics charts for the US Navy. This computer contained about 500 miles of wiring and was about half as long as a football field. Called The Harvard IBM Automatic Sequence Controlled Calculator (Mark I for short) it took between three and five seconds to do a calculation and was incapable as the sequence of calculations could not change. But it could carry out basic arithmetic as well as more complex equations.
- ENIAC (Electronic Numerical Integrator and Computer) was developed by John Mauchly and John Presper Eckert. It consisted of 18,000 vacuum tubes, 70,000 soldered resistors and five million soldered joints. It consumed so much electricity (106kW) that an entire section of Philadelphia had their lights dim whilst it was running. ENIAC was a general purpose computer that ran about 1000 faster than the Mark I.
- In 1945 John von Neumann designed the Electronic Discrete Variable Automatic Computer (EDVAC) which had a memory which held a program as well as data. In addition the CPU, allowed all computer functions to be coordinated through a single source. The UNIVAC I (Universal Automatic Computer), built by Remington Rand in 1951 was one of the first commercial computers to make use of these advances.

Three first computers filled entire rooms with thousands of vacuum tubes. Like the analytical engine they did not have an operating system, they did not even have programming language and programmers had to physically wire the computer to carry out their intended instructions. The programmers also had to book time on the computer as a programmer had to have dedicated use of the machine.

- **Second Generation (1955-1965)**

Vacuum tubes proved very unreliable and a programmer, wishing to run his program, could quite easily spend all his time searching for and replacing tubes that had blown. The mid fifties saw the development of the transistor, which, as well being smaller than vacuum tubes, were much more reliable. It now became feasible to manufacture computers that could be sold to customers willing to part with their money. Of course, the only people who could afford computers were large organisations that needed large air-conditioned rooms in which to place them.

Now, instead of programmers booking time on the machine, the computers were under the control of computer operators. Programs were submitted on punched cards that were placed onto a magnetic tape. This tape was given to the operator who ran the job through the computer and delivered the output to the expectant programmer.

As computers were so expensive methods were developed that allowed the computer to be as productive as possible. One method of doing this (which is still in use today) is the concept of batch jobs. Instead of submitting one job at a time, many jobs were placed onto a single tape and these were processed one after another by the computer. The ability to do this was seen as the first real operating system (although, as we said above, depending on your view of an operating system, much of the complexity of the hardware had been abstracted away by this time).

- **Third Generation (1965-1980)**

The third generation of computers is characterised by the use of Integrated Circuits as a replacement for transistors. This allowed computer manufacturers to build systems that users could upgrade as necessary. IBM, at this time introduced its System/360 range and ICL introduced its 1900 range (this would later be updated to the 2900 range, the 3900 range and the SX range, which is still in use today).

Up until this time, computers were single tasking. The third generation saw the start of multiprogramming. That is, the computer could give the illusion of running more than one task at a time. Being able to do this allowed the CPU to be used much more effectively. When one job had to wait for an I/O request, another program could use the CPU.

The concept of multiprogramming led to a need for a more complex operating system. One was now needed that could schedule tasks and deal with all the problems that this brings (which we will be looking at in some detail later in the course).

In implementing multiprogramming, the system was confined by the amount of physical memory that was available (unlike today where we have the concept of virtual memory).

Another feature of third generation machines was that they implemented spooling. This allowed reading of punch cards onto disc as soon as they were brought into the computer room. This eliminated the need to store the jobs on tape, with all the problems this brings.

Similarly, the output from jobs could also be stored to disc, thus allowing programs that produced output to run at the speed of the disc, and not the printer.
Operating Systems (G53OPS) - Examination

Although, compared to first and second generation machines, third generation machines were far superior but they did have a downside. Up until this point programmers were used to giving their job to an operator (in the case of second generation machines) and watching it run (often through the computer room door – which the operator kept closed but allowed the programmers to press their nose up against the glass). The turnaround of the jobs was fairly fast.

Now, this changed. With the introduction of batch processing the turnaround could be hours if not days. This problem led to the concept of time sharing. This allowed programmers to access the computer from a terminal and work in an interactive manner.

Obviously, with the advent of multiprogramming, spoiling and time sharing, operating systems had to become a lot more complex in order to deal with all these issues.

Fourth Generation (1980-present)
The late seventies saw the development of Large Scale Integration (LSI). This led directly to the development of the personal computer (PC). These computers (originally) designed to be single user, highly interactive and provide graphic capability.

One of the requirements for the original PC produced by IBM was an operating system and, in what is probably regarded as the deal of the century, Bill Gates supplied MS-DOS on which he built his fortune. In addition, mainly on non-Intel processors, the UNIX operating system was being used.

It is still (largely) true today that there are mainframe operating systems (such as VME which runs on ICL mainframes) and PC operating systems (such as MS-Windows and UNIX), although the edges are starting to blur. For example, you can run a version of UNIX on an ICL’s mainframes and, similarly, ICL were planning to make a version of VME that could be run on a PC.

b) 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 you 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 mimics humans. Just being able to understand and understand the human world and carry out reasoning 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.

Question 3 – Model Answer

a) Describe the following scheduling algorithms

3 marks available for each algorithm

• 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 easy to implement. When a process becomes ready it is added to the tail of the queue. When the CPU becomes free the process at the head of the queue is removed, moved to a running state and executed. The processes would execute in the order they arrived. Therefore, the processes would complete in the order they arrived.

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.

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 the average waiting time can be 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.

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst Time</th>
<th>Wait Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>P2</td>
<td>23</td>
<td>19</td>
</tr>
<tr>
<td>P3</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>P4</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>P5</td>
<td>60</td>
<td>69</td>
</tr>
<tr>
<td>P6</td>
<td>51</td>
<td>50</td>
</tr>
<tr>
<td>P7</td>
<td>38</td>
<td>37</td>
</tr>
</tbody>
</table>

The average wait time is calculated as

\[ \frac{\sum W T}{N} \]

\[ \frac{0 + 19 + 7 + 9 + 69 + 50 + 37}{7} = 343 / 7 \]

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.

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst Time</th>
<th>Wait Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>P2</td>
<td>23</td>
<td>19</td>
</tr>
<tr>
<td>P3</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>P4</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>P5</td>
<td>60</td>
<td>69</td>
</tr>
<tr>
<td>P6</td>
<td>51</td>
<td>50</td>
</tr>
<tr>
<td>P7</td>
<td>38</td>
<td>37</td>
</tr>
</tbody>
</table>

The average wait time is calculated as

\[ \frac{\sum W T}{N} \]

\[ \frac{0 + 19 + 7 + 9 + 69 + 50 + 37}{7} = 343 / 7 \]
Shortest Job First
This scheme, simply executes the process using the burst time as the priority. That is

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst Time</th>
<th>Wait Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td>P2</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>P3</td>
<td>9</td>
<td>5</td>
</tr>
<tr>
<td>P4</td>
<td>10</td>
<td>18</td>
</tr>
<tr>
<td>P5</td>
<td>19</td>
<td>28</td>
</tr>
<tr>
<td>P6</td>
<td>22</td>
<td>4</td>
</tr>
<tr>
<td>P7</td>
<td>31</td>
<td>70</td>
</tr>
</tbody>
</table>

The average wait time is calculated as 
\[ \text{Wait Time / No Of Processes} = \frac{(0 + 3 + 9 + 18 + 28 + 47 + 70)}{7} = \frac{175}{7} = 25.00 \]

e) 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.

---

Operating Systems (G53OPS) - Examination

CPU Utilisation = 1 - \(\rho^2\)

The formula is given in the question. The following graph shows this formula being used (the spreadsheet that produced this graph was available from the web site for this course as well as being discussed in the lectures and on the handouts). Graph paper will be supplied in the exam so that the students can replicate the relevant parts of this graph, although a table of figures would also be acceptable.

Graph of CPU Utilisation vs. Degree of Multiprogramming

Using the above formula we can derive the following figures

<table>
<thead>
<tr>
<th>IO Wait Time = 80%</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>No of Processes</td>
<td>CPU Utilization</td>
</tr>
<tr>
<td>3</td>
<td>0.45</td>
</tr>
<tr>
<td>5</td>
<td>0.65</td>
</tr>
<tr>
<td>11</td>
<td>0.90</td>
</tr>
<tr>
<td>15</td>
<td>0.96</td>
</tr>
</tbody>
</table>

Initially, we run three processes. If we add another megabyte we increase the CPU utilization from 49% to 70%. This, I would argue, is worth doing. Adding another megabyte would result in 91% utilization. I would also argue that this is worth doing. One more megabyte (allowing 15 processes to run) would give us 96% utilization. I would advise the company to add another two megabytes of memory (giving 91% utilization).

---

Operating Systems (G53OPS) - Examination

Question 4 – Model Answer

a) Describe the benefits of a mono-programming operating system.

It is easy to give advantages for a multiprogramming environment but this question specifically wants the student to consider the more simple monoprogramming environment.

Some of the advantages described in the course are given below; although the student may suggest their own.

1. We do not have to worry about race conditions
2. The operating system does not have to concern itself with keeping separate processes in memory – with all the problems that this entails (e.g. ensuring processes do not overwrite another processes address space, having to swap processes to disc when physical memory is in short supply etc.)
3. Scheduling is easy. We simply start a process and execute it to completion
4. No CPU time is wasted on overheads associated with multiprogramming environments (e.g. context switching)

b) A company, using a multi-programming operating system, has 1 megabyte of memory etc....

This question centres around a simple model that was presented that allows the students to calculate the CPU utilization. I do not specifically ask for CPU utilization in the question, so the student could give intuitive (and maybe vague) answers. They will not be awarded marks for this, unless they are very well presented and argued.

The marks for this question will be awarded as follows

<table>
<thead>
<tr>
<th>Area of answer</th>
<th>Marks Awarded</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specifying formula</td>
<td>3</td>
</tr>
<tr>
<td>Producing the correct figures or graph for the various I/O wait times and number of processes</td>
<td>7</td>
</tr>
<tr>
<td>Arguing how much memory should be added for the case where I/O wait time = 80%</td>
<td>5</td>
</tr>
<tr>
<td>Arguing how much memory should be added for the case where I/O wait time = 20%</td>
<td>5</td>
</tr>
</tbody>
</table>

Assume that that a process spends \(p\) percent of its time waiting for I/O. With \(n\) processes in memory the probability that all \(n\) processes are waiting for I/O (meaning the CPU is idle) is \(\rho^n\). The CPU utilisation is then given by

\[ CPU\ Utilisation = 1 - \rho^n \]
**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, ..., where w 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 on each list and all other lists will be empty.

b) Assume a computer with a memory size of 256K, initially empty. Requests are received for blocks of memory of 5K, 25K, 35K and 20K. Show how the buddy system would deal with each request, showing the memory layout at each stage and the status of the lists at the end.

After allocating the processes the memory will look like this. The student needs to show the memory allocation at each stage.

![Memory Allocation Diagram]

<table>
<thead>
<tr>
<th>List No.</th>
<th>Block Size</th>
<th>Entries (Start Position)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5K</td>
<td>1024</td>
</tr>
<tr>
<td></td>
<td>25K</td>
<td>16384</td>
</tr>
<tr>
<td></td>
<td>35K</td>
<td>19660</td>
</tr>
<tr>
<td></td>
<td>20K</td>
<td>24576</td>
</tr>
</tbody>
</table>

After allocating all the processes, what would be the effect of the 25K process terminating and returning its memory?

3 of the 10 available marks for part b will be awarded for answering this part of the question.

The effect of the 25K process terminating is that the memory is occupied (32K) is added to the 32K free list. The memory cannot be merged at this point as the free memory next to it (its buddy) would only add up to 56K. It would need the returning of the 8K process to give 64K of memory to combine lists.

c) 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.

![Bit Map Diagram]

<table>
<thead>
<tr>
<th>Allocation Units</th>
<th>Bit Map</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
</tr>
</tbody>
</table>

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.

| P | 0 | I | H | 1 | 3 | P | 4 | 3 | H | 7 | 1 | P | 5 | 1 |

Each entry in the list holds the following data
- P or H : for Process or Hole
- Starting segment address
- Length of the memory segment
- 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.

**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 large 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 been shown that best fit performs worse than first fit as a 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**

All of the questions below were not explicitly covered in the lectures, although enough information was given to enable the students to answer the questions. In addition, if the student has read the course textbook (or simply has some experience in using an operating system) then the questions are not that difficult.

**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) An operating system only allows a single directory hierarchy but allows arbitrary long filenames. Could you simulate something approximating a hierarchical file system? Give your reasons.**

If you allow files to be called (for example)...

---

**Operating Systems (G53OPS) - Examination**

course/op/ohp

where “/” are allowable characters in file names (which they would be as they would no longer be invalid as they are not being used as directory separators), then by using the various wildcard characters (e.g. “*” and “?”) you can search (copy – and perform other operations) for files in a very similar way to a hierarchical filing system.

d) When a file is removed, the blocks it occupies are simply placed back onto the free list. Can you see any problems with this? If so, how would you overcome them and what problems, if any, would now exist and how would you resolve these?

The main problem is that the data still exists in the block and, somebody with the correct tools, can access that data.

One solution is to erase the data in the block at the time the file is deleted but this has the disadvantage of affecting performance.

A compromise solution could be to have a file attribute which marks the data as **sensitive**.

If this attribute is set then the data in the blocks is deleted when the file is deleted. If the attribute is not set then the data in the blocks is not deleted.

e) When the UNIX filling system opens a file its i-node is copied into memory. It has been suggested, that with memory getting cheaper that if a processes open the same file then a copies of the i-node could be held in memory. Is this a good idea? Give your reasons.

No, it is not a good idea (unless all i-nodes were read only). We could (and probably would) arrive at the situation where processes updated its own i-node and then eventually the process would read the i-node back to disc. If process x wrote back its i-node just after process y then the updates to the i-node by process x would be lost.

The student would have to come up with a very good reason to argue why it is a good idea.