THE UNIVERSITY OF AUCKLAND

EXAMINATION FOR BA BSc ETC 1995

COMPUTER SCIENCE

Operating Systems

(Time allowed : THREE hours)

NOTES:
Answer SIX questions. The total mark for each question is 20; the total for the paper is 120 marks.

QUESTION 1.

(a) Describe the components of a disc file which must be managed by the operating system, and show how the file name is used to find the file on the disc. How does a Macintosh file differ from a conventional file ?

(b) Explain the ideas behind an object-oriented file system, and how they contribute to a simple user interface. Illustrate your answer by describing a possible implementation of the operations execute, print, and delete.

(c) A word-processing package includes a core programme which constructs, edits, and displays files, and a specialised printing programme which is supplied as a separate utility. Answer questions (i), (ii), and (iii) below, in each case describing classes and methods which must be defined, and the links between the various components, and explaining the reasons for the structure you suggest. (Throughout this part of the question, ignore all file operations except constructing a new file, editing a file, and printing a file.)

(i) Explain what must be done to install the package in an object-oriented file system. What information must the core programme have in order to construct a new file ?

(ii) A new version of the package is received, incorporating new versions of both core programme and printing utility which operate with a new file format. The new core programme only produces new files, though it can read old files, but the new printing utility cannot print old files. What must be done to replace the old core programme with the new version, while preserving the old printing utility to provide service for old files ?

(iii) The old printing utility eventually becomes obsolete, and must be abandoned, but it is impossible to tell whether any old files still exist. (Some might be stored off-line, on separate discs.) What should be done ?

QUESTION 2.

(a) In many systems, all input and output must be managed through a standard system call of the form

```
DOIO( fileidentifier, action, data, resultdescriptor ).
```

List a possible and reasonable set of values for the `action` parameter, explain what they mean and how they interact with the other parameters, and explain how the operating system can interpret them in different ways for different devices. Do *not* describe the operation of the device driver or interrupt handler software.

(b) A programme is using a disc file XYZ, which is open for input and output. A single one-sector buffer is available for the file. The programme executes an instruction which means "WRITE 67 TO BYTE 3376 IN FILE XYZ"

(i) What sequence of operations (including calculations peformed and decisions made by software and physical operations carried out by hardware) must be performed to get the byte safely written to the disc ?

(ii) What sort of `DOIO` calls would be issued to carry out the operations which you listed in part (i), and how would they be executed ?

QUESTION 3.

(a) Explain what is meant by a *memory model*, briefly describe an example (NOT a flat one), and explain why operating systems do not support different memory models.

(b) The programme example given on the next page is executed in a system without virtual memory. The local memory requirements of each procedure do not change while it is being executed. State the minimum memory required to run the programme under conditions (i) and (ii) below, explaining the reasoning which leads to your answers.

(i) Using a flat memory model, with all memory allocated statically before execution by a compiler which allocates each procedure's memory separately without any sharing;

(ii) Using a segmented memory model, permitting memory segments to be allocated and released during execution.

(c) For a system using a segmented memory model (condition (ii) of the previous part), identify the memory management functions which must be provided in the application programme interface and used by the programme if it is to minimise its memory requirements.

(THE EXAMPLE FOR PART (b) OF THIS QUESTION IS ON THE NEXT PAGE.)

EXAMPLE :

The programme structure is Pascal-like, with procedures nested within the programme. *Data memory* is memory required for data storage within a programme or procedure; it does not include memory for code storage. All procedure calls are listed, but no other code details.

```
Programme A;
begin
     Declare A's local data memory, size Ma;

     Procedure B ;
     begin
          Declare B's local data memory, size Mb;
          { B's code, size Cb, including :
          call Procedure C;
          call Procedure D; }
     end Procedure B;

     Procedure C ;
     begin
          Declare C's local data memory, size Mc;
          { C's code, size Cc }
     end Procedure C;

     Procedure D ;
     begin
          Declare D's local data memory, size Md;
          { D's code, size Cd. }
     end Procedure D;

     Procedure E ;
     begin
          Declare E's local data memory, size Me;
          { E's code, size Ce, including :
          call Procedure F; }
     end Procedure E;

     Procedure F ;
     begin
          Declare F's local data memory, size Mf;
          { F's code, size Cf }
     end Procedure F;

     { A's code, size Ca, including :
     call Procedure B;
     call Procedure E; }

end Programme A.
```

CONTINUED

QUESTION 4.

(a) Semaphores, locks and monitors are three synchronization primitives. Compare them in terms of complexity and safety, mention data structures required for their implementation and the operations with which they are used, and give reasons why semaphores are the most commonly available of the three.

(b) On some machines binary semaphores can be easier to implement than general (or counting) semaphores. Binary semaphores have integer values which can only hold zero or one. It is possible to implement general semaphores using binary semaphores and the following code is an attempt at this. s1 and s2 are binary semaphores. Unfortunately it doesn't work. Explain why it doesn't work and give corrected versions of the code for the operations. Also give the initial values of s1 and s2 and explain what the counter variable represents.

General Wait

```
wait( s1 );
counter := counter – 1;
if counter < 0 then
    wait( s2 );
signal( s1 );
```

General Signal

```
wait( s1 );
counter := counter + 1;
if counter <= 0 then
    signal( s2 );
signal( s1 );
```

(c) Different synchronization primitives handle some problems better than others. Message passing can be used to provide a very simple solution to the producer / consumer problem. Give reasons why this problem should be so easy to solve with message passing compared with other solutions, such as a semaphore solution to the same problem. Keep in mind that there may be multiple producers and consumers.

QUESTION 5.

(a) Explain what is meant by the terms *access matrix*, *capability*, and *access control list*. Explain how capabilities and access control lists define parts of the access matrix.

(b) Describe one type of capability system. In your description :

(i) Explain how the capabilities themselves are represented in the computer system, and how they are kept secure.

(ii) Identify the capability operations which must be provided, and show how they can be implemented.

(c) Many secure systems provide both capabilities and access control lists. Explain why it is useful to have both techniques available.

QUESTION 6.

(a) A common method of implementing services over distributed computing environments is with remote procedure calls. Briefly explain what a remote procedure call is and how it works; refer to client and server processes and the role of stub procedures.

(b) Another way to provide distributed services is with direct interprocess communication. i.e The client process sends a message to the server process with a `send` system call. The server process was waiting for a request with a `receive` system call. Eventually the server sends the result back to the client.

Describe the advantages and disadvantages a remote procedure call system has compared to a system which requires programs to make explicit sends and receives. In your answer compare the use of the systems and the implementations.

(c) A new distributed operating system is being implemented with many of the services provided via remote procedure calls. A remote procedure call to access a specialised mathematical processor is going to look like this :

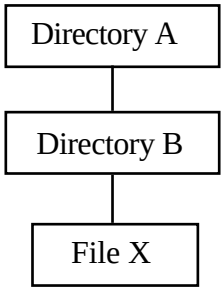
```
MathProc( operation, outAddress, outBytes, result )
```

where `operation` identifies the operation to be performed on the special processor, `outAddress` is the address of a section of memory of size `outBytes` in the requesting process where the parameters for the operation have been placed, `result` is the variable to hold the result of the operations.

Describe how the remote procedure call would deal with the parameters. Explain any difficulties.

QUESTION 7.

The Unix, Macintosh, and MS-DOS (and therefore MS Windows) file systems share a very similar structure, but are customarily used in quite different ways. Consider this structure of directories A and B, and the file X :



Suppose that the working directory is initially Directory A; in a GUI system, Directory A's window is open.

The Unix instructions listed below are executed in the order given.

(Unix)	*(MS-DOS)*
cd B	*chdir B*
cp X Y	*copy X Y*
mv Z ..	*(no equivalent)*
mv X ..	*(no equivalent)*
*rm **	*del **

(MS-DOS equivalents are given, where possible, for information only.)

(i) List the operations carried out by the file system for the efficient execution of each instruction.

(ii) Comment on the corresponding GUI (Macintosh or Windows) operation, explaining (where appropriate) how the information needed to identify the instruction is acquired.

NOTES :

`cd` *is a shell instruction to change the working directory;*	`chdir` *(or* `cd` *) is an instruction to change the working directory;*
`cp` *is a shell instruction to copy a file;*	`copy` *is an instruction to copy a file;*
`mv` *is a shell instruction to move or rename a file;*	*there is no instruction to move a file to another directory; the* `rename` *instruction is not relevant;*
`rm` *is a shell instruction to delete a file (* not *a directory);*	`del` *is an instruction to delete a file (* not *a directory);*
`..` *is a convention to denote the parent directory;*	
** is a "wild character" meaning (in the example) "all files in the directory".*	
The search path contains only the current directory.	

QUESTION 8.

(a) Describe the sequence of actions which happens after a clock interrupt in a system using a single-queue round-robin dispatcher.

(b) In a multiple-queue dispatcher system, each queue has a name, and its behaviour is described by three parameters :

- The relative frequency with which the queue is selected for dispatching, in the range 1 to 100, with 100 representing the highest frequency.
- The length of the time-slice, in the range 1 to 100 milliseconds.
- The name of a "destination queue" – the queue into which processes from this queue will be placed if they are still executing at the end of their time slices.

For example, a dispatcher consisting of a single queue called "Initial", with each process dispatched from the queue having a timeslice of 10 milliseconds, and processes which complete their timeslices returned to the same queue, might be described as

Initial : { frequency 5, timeslice 10, destination Initial }

A second queue with frequency 10 would be selected for dispatching twice as frequently as "Initial".

Whenever a process is made *ready* it is placed in a queue called **Initial**, which must always exist; there may be any number of other queues. When a running process completes its time slice, the clock interrupt handler moves it into the destination queue of its current queue. After that, or when a running process releases the processor, the dispatcher selects the occupied queue which should be run next, taking account of the queue dispatching frequencies, and moves the process at the head of that queue into the *running* state.

Show how you could set up the dispatcher to implement, separately, each of the specifications below :

(i) Any process which is still running at the end of a timeslice is given the same share of the processing time as processes newly made runnable, but with only one tenth of the number of context switches during its time of execution;

(ii) Any process which is still running at the end of a timeslice is penalised by receiving only one tenth of the share of the processing time received by processes newly made runnable, but also with only one tenth of the number of context switches during its time of execution.

In both cases, explain why you believe that your answer satisfies the specification.

(c) Explain the significance of the attempts to reduce the context-switching frequency for processes which are interrupted by the clock.