Computer Science 340

Operating Systems

FIRST TEST, 1994 : Answers

These are not the only possible answers, nor necessarily the best ones. They are the best I can think of under a very crude approximation to examination conditions; and they are far more extensive than I expect anyone to produce in the test itself.

Most material in *italic* type was added after marking the test.

I have corrected some spelling. I didn't correct every mistake that I saw, so don't be complacent; there are no marks for spelling, grammar, punctuation, style ....

It is fairly clear that some people have difficulty with the volume of English which they have to read. I do try to cut down the volume as much as I can, but there is a limit to how far one can go on cutting down without losing imformation. If you are having trouble, you should study the previous years' tests and examinations, and make sure that you understand them. In an examination or test paper, there is usually (almost always in a 340 examination) a certain amount of repetition of material from previous years. You can't bank on it's being exactly the same, but the style of question and vocabulary remain fairly predictable.

Please use the old questions and answers sensibly. It is very unusual to find a question exactly repeated – and particularly unusual in this first test, which has been an open-book test for many years. You still need to understand the questions and answers even if only to select the bits you copy from the old answers. (And that doesn't bother me at all : if you do understand, that's what I want, and if you don't, well, we'll find out in the final examination.) Notice that this makes it positively dangerous to rely on memorising the answers without understanding them; a slight change in the question can make a memorised answer worse than useless.

QUESTION 1.

The three parts of this question are complementary. The first part is about basic requirements for protection, the second is intended to emphasise the need to consider all parts of a protection system, while the third directs attention to different sorts of protection.

(a) Memory protection is necessary : without memory protection, any process can read from or write into the memory area currently in use by another process. There are other approaches (such as forcing all programmes to be compiled by compilers which check memory addresses and insert checking code when they can't make the checks while compiling), but they all (?) have potential loopholes. Physically preventing undesirable access by building it into the hardware will work with any software.

Supervisor mode is necessary : without supervisor mode, all instructions are equally available to the operating system and every running process. Assuming no other special features, this means that whatever measures the operating system takes to confine processes to limited areas of memory can be undone by the processes. The supervisor call instruction must therefore transfer control to a trusted process running in a different memory space. It is not absolutely essential that the transfer be combined with a switch to supervisor mode, as there are some supervisor operations which don't need privileged instructions, but it's obviously useful, and I don't know of any modern processor which separates the two.

The two features together are sufficient : memory protection instructions (such as setting base and index registers) can be made privileged, and therefore only executable in supervisor mode; and the fixed address which is the destination of the supervisor call branch can be made the

entry to the operating system's management routines, and therefore under the system's control. A normal process cannot reach memory outside its allocated area, and cannot execute any instruction which will change the allocated area.

Most people had at least some idea of the nature of memory address checking, though many didn't really use it to answer the question. Significantly fewer understood the nature of supervisor mode, or how it's conventionally used – there were several quite startling claims as to what supervisor mode could do, despite the definition in the question. Very few knew (or, at least, very few told me) why memory protection on its own is not sufficient for effective protection, but must be augmented by something in the nature of supervisor calls. If you're not sure what a supervisor call is, FIND OUT : I thought you'd covered it in 211, and in any case it's elementary knowledge about quite simple processors. There is no magic involved, and it's essential basic knowledge for operating systems.

Several people gave me descriptions of address checking and supervisor mode, and said how nice they were, but that's not what I wanted. To show that X is necessary, it isn't enough to say how good it is. You have to show that something will go wrong if X isn't there. I've been fairly fussy about this in the marking, because in protection and security this sort of argument is necessary. (Because something will go wrong if the argument isn't there.)

Someone remarked that it wasn't necessary that memory protection be implemented in hardware – which is true, provided that you can be absolutely certain that no one can avoid your software checks.

There were occasional suggestions that either address checking or supervisor calls controlled the system's memory allocation. They don't.

There were several suggestions that you could make your programme do all sorts of wonderful things when in supervisor mode. Unless someone has made a mess of the operating system, your code will never run in supervisor mode – unless you're a systems programmer.

(b) Provided that the owners of the processes can be adequately identified, much the same argument applies, with input and output instructions made privileged. To use the disc, a privileged instruction must be executed; to execute a privileged instruction, the processor must be in supervisor mode; to put the processor in supervisor mode, a supervisor call must be executed; executing a supervisor call forces a branch to the operating system code, which is (should be !) safe. There is no escape from the logic. (On processors with memory-mapped input and output, the "memory" ranges corresponding to input and output operations must not be allocated to any normal process.) Various system tables are needed to manage the disc, but these too can be protected by the protection mechanisms under discussion.

Identifying the owners is quite a different question. Simple protection methods can be used to safeguard the system's records of identifying data, but cannot be effective against people who by any means have discovered other people's identifying data (such as passwords).

This wasn't intended to be a trick question, but it did introduce a new feature. In part (a), the subjects are processes, which are under the control of the operating system – but in part (b) the subjects are the owners of the files, who are outside the system. To get the information needed for security, the system must find out who owns the files and who owns the programmes, so the identification step is essential.

A large number of people didn't seem to notice that this part of the question was also about address checking and supervisor mode.

Someone (in fact, many ones) interpreted this part in a way which I hadn't expected. They supposed that I was proposing an analogue of memory address checking applied to disc memory addresses. It's interesting, but I don't think it would do any good unless it was built into the hardware somehow. You would need additional "registers" to hold the disc address limits, which would be software (hardware disc registers would certainly count as "unusual

features" !) and these would have to be protected internally – by the combination of address checking and supervisor mode which the question asks about. The disc read and write operations would still have to to be privileged to force people to use the system procedures. That being so, there's no advantage that I can see in this method which would make it preferable to the usual disc management system. The protection is important to keep the system software safe in memory.

On the other hand, it has been done. Something rather like the proposal is used in IBM's Virtual Memory operating systems, in which every process "sees" a virtual machine, just like the real hardware machine, but with a virtual disc, which is just a limited area of the real disc. So far as I know, the address checking is done by software, but the principle is much the same. Because of that, I gave some marks for answers which took this approach, but, as I don't think any of them covered the need to keep the software safe in memory, not many.

It is clearly true that anyone who can walk off with the disc can read all the files, whatever machine-imposed protection techniques you use, but it's reasonable to suppose that a shared disc won't be readily accessible to the general public.

(c) It's half true. The answer depends on what you mean by "protection". It is correct to the extent that we are concerned to keep our information secret, but if it is possible for someone to read or intercept our data then it is quite probably possible for someone to overwrite or corrupt our data – after which, whether encrypted or not, it's not likely to be of much use.

This question is an example of the danger of trying to avoid long and complicated questions. The first script I read made the point that if the material were impossible to decode, then the owner couldn't decode it either, so the method was useless. I wasn't very generous in giving marks, because I think it's clear from the context that the impossiblility was that encountered by the intruder, not the owner. ("Impossible to decode" is presented as an argument in support of the thoroughly practicable, and widely used, technique of encryption, and applied to the value to an intruder of stolen material.)

It is also an example of the dangers of trying to make questions more comprehensible. The question is not about encryption. The real question starts at the second sentence; the first sentence is there only to show that the second sentence could make sense, which would not necessarily be obvious if the first sentence were omitted.

And it is also, apparently, an example of mistaken assumptions. I'd supposed that the general ideas of encryption would be part of the "general computing knowledge as can reasonably be expected of a stage 3 Computer Science student". I'm surprised to find that that assumption is wrong.

The effectiveness of encryption itself was questioned. It is certainly true that if there is an algorithm which can decode an encoded text, then it's possible in principle to find that algorithm. If it's a standard algorithm which is widely used and doesn't rely on an arbitrary key of some sort, then it isn't serious encryption – it's something between a toy and a crime. It certainly doesn't demonstrate that encryption techniques are unsafe, any more than the insecurity of a tent demonstrates that a castle is unsafe.

Encryption techniques rely on the fact that you can make it exceedingly difficult to find a key used by the decoding algorithm from a sample of encoded text. If your decoding key is, say 50 bits long, you will expect to have to try 2^{49} values before finding the one that works. If you can try one key value every microsecond, which would itself probably need some rather ambitious computing machinery, that will take you about 16 years. And if that's not long enough, or if your enemies can afford the machinery, (or if I've got the arithmetic wrong,) then every additional bit on the decoding key doubles the time. It may be possible to break the code, but you can make it too expensive to try.

There was mention of covert channels – if an intruder can see the file at all, then information can be passed by encoding it in the size of the file, or by other means not

depending on the contents. That's true, but it's more about security than protection, which gets us back to the second sentence in my specimen answer. (And the argument only works if the covert channel is available in an encryption system, but not available in a hardware system.)

There was some argument from a basis of morality, with a few people taking a strong line that it is bad for anything to gain access to other people's property. There's nothing to prevent you from taking that as an axiom if you so desire, but then you will presumably not want the operating system to gain access to your files, which is a bit awkward. I have nothing against moral standards, provided that you know what they are – which is, usually, guidelines which help you to preserve some deeper principle on which you base your scale of values. I suggest that the appropriate axiom for computing is that "good" means "conducive to the implementation of a properly functioning computer system" – where we have (well, I have) already defined the function of a computer system as "to do work for people as directed". That isn't quite enough to talk about protection and security, so we have to add another axiom which distinguishes legitimate work from illegitimate work – and which is curiously hard to formulate precisely, but we all know what we mean by it. (Unfortunately, though, we soon discover that we don't all mean the same thing.) My point is that there are sounder bases than arbitrary local principles for designing computer systems, and that we're more likely to design consistent systems if we always start from the same place. (If you want to quibble with these assertions, please do. The preferred medium is electronic mail.)

Only one person noticed an enormous loophole (and thereby gained full marks for this part). I'd intended the question to be about disc file protection, carrying on from part (b), but hadn't said so. Encryption is indeed not much help if there's nothing to stop people from destroying your area of memory !

QUESTION 2.

***Please** try to be precise. The mouse position is of no interest whatever to the TUI, or to any graphical interface; the **pointer** position is of great interest.*

And please try to be consistent. In many cases, except that the handwriting was similar the different parts of the answer might have been written by different people. Examples : the process owning a window was magically known in (b) even though in (a) there was no provision for it to be stored; the current window was known in (b) but not provided for in (a); process identifiers used in (c) were matched with things that didn't exist in (a).

And, if you can, try to be reasonably efficient. It's silly to work out the current window from the pointer position every time you need it – you have to work it out in (b) (ii), so why not save it ?

This question is similar to, but not the same as, a question I set in 1991. Some people noticed the similarity; fewer noticed the difference. Answers which were obviously copied received fewer marks than they might otherwise have done. (A dead give-away was any reference to an error window. These were part of the 1991 specification, but explicitly ruled out in 1994 by the assumption that "Interface errors need not be considered".)

I use "pointer" to mean the thing on the screen that moves about when you move the mouse, and "cursor" to mean the point within a text string at which characters are inserted, or whatever. There's no standard terminology so far as I know, but that convention seems to make sense. It's a good idea to make it clear which you mean, whatever words you use.

(a) : DATA.

Data :

Current position of the pointer.

Current active window number.

List of windows, in order of "depth".

For each window :

Window number;
Edge coordinates;
Position of cursor (where to put the next character);
Number of the owning process.
Text displayed (as text, or as a screen map);

Structures :

The main structure required is a window table, which can be represented as an array of window descriptors. A reasonable description of a window descriptor is :

```
window descriptor :
      record
      window number : integer;
      top coordinate : integer;
      right coordinate : integer;
      bottom coordinate : integer;
      left coordinate : integer;
      cursor x coordinate : integer;
      cursor y coordinate : integer;
      pointer to text array : pointer;
      number of lines : integer;
      line length : integer;
      owning process number : integer;
      array index of the window behind : integer;
      array index of the window in front : integer;
      end record.

screen display : array [ 1 : maximum number of windows ] of window
                                descriptor.

text array : array [ 1 : number of lines, 1 : line length ] of character.
```

Notes :

These structures give sufficient information to implement all the operations described in the specification. The window number is necessary; there is no guarantee that the number can always be represented by the array index. The four coordinates are needed to determine where to put characters, and which windows overlap which. The cursor coordinates show where to put the next character within the window. The pointer to the text array gives access to the text currently displayed in the window; it is sensible to store this text elsewhere as different windows will have different capacities for text. The numbers of lines and the line length are redundant (they can be calculated from the window edge coordinates) but they are convenient for managing the window text. The owning process number is needed to send messages from the TUI to the process. The array indices of the window in front and behind are used to determine which windows obscure which.

Several people answered this question about data by describing what the UIMS had to do, without reference to the data structures. Others (or sometimes the same) answered this question about the screen by describing the keyboard. Why ?

(b) : BEHAVIOUR.

This part specifically asked for the actions of the UIMS on its data structures. That was not always what people put in their answers.

And it wasn't what I'd put in my answers either, but that wasn't because I had the wrong answer – it was because I had the wrong question. In trying to bend the question from its earlier form, I'd tried to make it more precise, and rather overdone it. In 1991, I'd asked "How must the TUI react to ... a character entered at the keyboard", etc., and a large proportion of the answers had completely ignored the data structures and messages. This time I made that more explicit by asking "What must the UIMS do with its data structures in order to react correctly ... a character entered at the keyboard", and thereby ruled out what the UIMS did in terms of sending messages and putting things on the screen.

But you weren't to know that, so I had to mark the question I set (once I worked out what I'd done). The result is that I've probably marked rather more leniently than I'd intended, but I think it's consistent. The leniency is not very obvios in the marks, but is nevertheless there.

(i) A character entered at the keyboard.

```
Send the character to the current window ( see SUBROUTINE below );
If the current window is not the full screen
      construct a "CharIn" message,
      send it to the current window's owning process.
```

I assumed that people would know that a mouse keeps no record of its position, and sends to the computer only an indication of the increments in its position in four directions.

Very few people told me what to do with the character buffer when the screen was full. I expected some account of that either under part (i) or under part (iii). Some of the people who did say what to do just said "scroll", giving no details – but isn't that an operation on the window's character buffer ?

One person didn't send the character to the screen, on the grounds that it was up to the process in the computer to make that decision so that passwords could be hidden. This is an excellent point to make, but is unfortunately ruled out by the second item in the TUI's specification.

(ii) A movement of the mouse.

```
Calculate the new pointer position;
If the new position is on the screen :
      Store the new position;
      If the pointer has crossed a window boundary
            Set the new current window number;
            If the current window isn't the full screen
                  Reorder the front-to-back window list,
                  Make sure that the current window is fully displayed;
      Display the pointer.
```

If the active window is the full screen, parts of it may be obscured; you can't just display the entered character without testing.

The "depth" order of the windows must be maintained somehow. It must be represented in the data structures, and it must be kept up to date in this part of the system.

(iii) A character arriving from a process.

```
Send the character to the window specified in the message ( see
                                SUBROUTINE below );
```

SUBROUTINE to display characters. The character and the window number are parameters.

```
With the appropriate window descriptor :
     If the cursor is at the end of the line
          If the cursor is on the last line
               Move all lines in the text array up one, throwing away
                         the old first line;
               Adjust the cursor y coordinate.
          Move the cursor to the beginning of the next line.
     Place the character at the cursor position.
     Advance the cursor to the next position.
     If any visible part of the window has been changed ( search forward
                         through the windows for possible obscuring
                         windows )
          Redraw visible changed parts of the window.
```

This part was ambiguous. I had assumed that incoming messages would have been decoded, so that the "character arriving" was to be displayed. The answer above corresponds to that case. The alternative interpretation is that the character is part of a message under construction. I didn't notice the ambiguity until I'd marked quite a number of the scripts, so I may have missed some. The alternative answer follows; if you think that your answer should be marked according to the alternative, but hasn't been, bring it to me, please.

```
If the character is not end-of-text
     Append the character to the message being constructed
otherwise
     Identify the window number, process number, and message type;
     Place these components, and any subsequent message text, in a
                         standard data structure;
     Execute a procedure which deals with messages of the type found;
     Empty the message buffer.
```

(I have assumed that messages follow the pattern I describe in part (c).)

(c) : MESSAGES.

Assumptions :

The TUI need not know about anything that happens in the processes. All it needs to know about are the windows and who owns them. As a process may own several windows, they are identified by window numbers rather than process numbers. Windows are numbered from 1 upwards, with the numbers allocated by the UIMS when the windows are first made. The full screen is window number 1.

The messages are transmitted along conventional serial communications lines. I have therefore imagined my messages as linear streams of text.

Form of the messages :

Every message received by the TUI must be identifiable in two ways : by the window number, and by the message type, so that it can be properly dealt with. Messages sent from the TUI must also carry the process number, so that they can be directed to the correct destination. The specifications do not require a process number on all incoming messages (there's no restriction on which windows a process may send messages to), but as some

require a process number I've included the process number in all messages for consistency, giving a standard message structure which is easy to interpret.

I assume a conventional message interpreter which works sequentially through the string, acquiring information as it goes. Dependencies between items of information are therefore ordered so that the interpreter always has the information it needs to continue the interpretation. The standard structure is :

<window number> <process number> <message type> [<further information>]

The first three items are always present; they are independent, so their order is unimportant. The <further information> is necessary in some cases, but by the time the message interpreter reaches that point the message type is known and the information can be handled appropriately.

What messages ?

The processes must be able to construct windows, destroy windows, and send text to windows; the TUI must be able to send on to processes the text which has been typed at the keyboard. The processes must therefore know the window numbers, so when a new window is constructed by the TUI it must return the window number to the process which requested the window. (An alternative would be to let the processes define the numbers for their own windows – but then a process wishing to display a message in another window would have to give both the process number and the window number. The TUI-defined number also simplifies the TUI data structures.)

Five messages are therefore needed.

Function	window number	process number	message type	further information
To the interface				
Display text	window number	process number	TextOut	text.
Construct window	0	process number	NewWin	edge coordinates
Destroy window	window number	process number	DestWin	
From the interface				
Window constructed	window number	process number	ConsWin	
Send text	window number	process number	CharIn	character

*I asked you to **design** a set of messages. Many answers just contained a list of items which could (more or less) plausibly be message types. Without any explanation, these usually made little sense, so there was correspondingly little evidence of design, and – also correspondingly – little reason to give marks. In several cases, there was evidence of design by reference to the 1991 answers, as messages contained curious components not readily interpretable in terms of the 1994 question.*

Several people suggested messages which were not required by the specification. As the question explicitly states that the UIMS should be "as simple as possible", I didn't accept these unless they were explicitly justified in the answer.

Except for the "Display text" message, the process number is required. (There is nothing in the specification to prevent a process displaying a message on another process's window. That may well be silly, but it's part of making the TUI as simple as possible.)

Alan Creak,
May, 1994.