Computer Science 340

Operating Systems

TEST, 1998 : Answers

I've added references to relevant places in the notes where there is material which is fairly directly relevant. There isn't always a reference to every specific part of the answer because some things you have to think about, but the basic material is there in all cases.

QUESTION 1.

(REFERENCE : *What must be done*, pages 29, 34-35, 38.)

(a) Data : information comprising the messages communicated. Can go in either direction.
Control : instructions controlling the behaviour of devices along the stream. Computer to stream.
Information : signals returned by the devices indicating state changes, errors, responses to requests, etc. Stream to computer.

That's the answer I wanted, but the last word in the question ("managed") does rather suggest that I wanted only the control and information bits. That was a mistake, but I had to accept it.

ONE person (the 133rd script I marked) simply drew the diagram, and got full marks.

Some people lost marks by describing a specific sort of stream rather than keeping the answer general. Similarly, others specified particular items of information, when I asked you to "describe the sorts of information". In such cases I looked for some mention of examples of the sorts of information I'd expected, and gave some marks accordingly if I found any, which wasn't always.

A significant proportion of the answers included descriptions of the transaction at the data communications level, which we don't do "in the first half-semester of the 340 course" (Robert covers some related material in the second half), but omitted the material relevant to the part covered by the test. (In one answer, that was mentioned as 'the "real" information', but otherwise not described.)

After reading some of the answers, I began to wonder why I hadn't heard bottoms of barrels scraping during the test.

(b) For control and information : establish a file information block, which represents the stream to the process.

Use the device information in the request ("disc") to locate the information about the device kept by the system, and get required information from there into the file information block. (Or make links.)

Then (guided by the "open" method for the device just acquired from the system) use the information given about the file ("filex") to find the correct file in the disc's file table, and save information which will be needed to use the file in the file information block.

For the data : set up buffer areas, and link them to the file information block.

A significant proportion of the answers included an approximation to the sequence of events without any indication that there was any need to get required information from anywhere – and rather few mentioned "structures set up during the operation", as specifically requested. For example, "The OS must find the device ..." (which is almost fair enough, though some mention of a device table might seem appropriate), followed by "Then it must find the filex ..." with no indication of how it knows that, or where to search.

Several people told me about the open-file table, which was welcome provided they got it right, which wasn't always.

Not very many told me about the file information block. Probably not coincidentally, not many told me how to use the "disc" parameter either. That bit just happened by magic.

I received some detailed descriptions of checking file attributes and searching directories with no indication of how the system worked it that it was a disc file. Presumably it would do the same when opening a mouse stream.

Several people interpreted the question as though the process opening the data stream were someone typing at a keyboard, and therefore told me how to parse 'open(disc, "filex")'. Processes are a good deal more likely to issue requests by making system calls, which is why the example looked like a procedure call.

A few people were very keen that the file contents should be displayed on the screen. Why ? Even more seemed to want to read the file into memory. Not even Java forces you to do that.

(c) When the file is in use, only the information from the file information block is used. Therefore, for a file which did not exist before the execution, the information can be constructed in the information block and not communicated to the device's file table.

This isn't a secret operation, or a mysterious trick; it's the way some systems work all the time, which is what the first sentence of the question says. Neither is it some way of storing a file in memory; it's a real ordinary simple straightforward disc file that just doesn't happen to have a file table entry. It has just been made by a process; it is used with the usual read and write (etc.) operations. The point of the question is that it must be managed without the process using the file needing to know; it must continue to use the ordinary file operations – therefore the stream must look the same to the process whether or not there's a directory entry. The file information block solves the problem, if there is one.

There were some peculiar answers for this one. A common answer was that the file could be located in memory. It was never explained how a set of procedures designed to read and write disc files could suddenly start working from and to memory. It seemed that most people were really talking about a data structure in memory.

When you tell a word processor to make a new "file", it doesn't make a file – it just operates on its own internal memory. From the operating system's point of view, nothing has happened. Proof : if you try to do any real file operations (such as save), it doesn't just happen as it does with a real file – you have to give some sort of name.

Yes, I know almost all processes in fact read and write using buffers in memory (and it's a pity that people who almost implied them in this part of the question didn't write them down earlier, when they'd have got more marks), but that's quite independent of the directory entry question.

This was easy marks if you knew the answer. Most people who told me about file information blocks at all knew the answer.

A few people misunderstood the question by just reading the last line, and told me about files which weren't disc files. The first line does say that the file is at least a candidate for entry into the disc file table.

And many people worried about whether other processes could use the file.

Far too many people decided that the file could be stored in the stream (or some more or less equivalent form of words). Just in case anyone doesn't immediately see how silly that is, I'll explain it : a stream is a sequence of data items in time, and once they've passed they're gone – so how can you store anything in a stream ? Can you store a drink by pouring it down the sink ?

QUESTION 2.

(REFERENCE : *History*, page 29; *Requirements specification*, pages 78-80.)

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 or performance penalties. Physically preventing undesirable access by building it into the hardware will work with any software. But memory protection alone is not enough, for processes must be able to use system services outside their own memory areas.

Supervisor mode is necessary : without supervisor mode, all instructions are equally available to the operating system and every running process, so it is not possible to prevent processes from using potentially dangerous instructions. (Examples are device management operations, such as disc reads and writes.) But supervisor mode alone is not enough, for if ordinary processes can execute the instruction used to enter supervisor mode it loses its value. 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 therefore necessary to provide both the protection and facilities required.

They are also 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.

The last paragraph above isn't strictly required for the answer, but it does make the point that both features are required to protect the memory in an acceptable way, as required by the question. Notice that if you don't have memory protection then there's no obvious way to justify having supervisor mode, and if you do have memory protection you have to explain why you also need supervisor mode.

I asked whether the two features were ***necessary***. People who simply described them, or just said they would help, got very few marks. People who just said they were important didn't answer the question either, but sometimes got a bit closer and therefore sometimes got some marks.

I asked whether ***the two features*** were necessary. One person gave a rather good part answer in support of hardware protection as opposed to alternatives (which very few others did satisfactorily), but didn't demonstrate that address checking and supervisor mode would do.

(b) In the memory protection problem, all the information required is immediately available from the information about the process maintained by the system; the base and limit registers are there, and the supervisor call itself signals the change of agent from running process to system. In the file problem, some information is not immediately available; specifically, we do not know the level of protection to be exercised, and must therefore be able to find it from the system data. Typically, we have to identify the owner of the running process, the owner of the file, and the protection level defined by these two.

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).

People generally knew about the need to work out the level of access permitted to the file (though rather few mentioned that you might have to know the file's owner). Not nearly as many explained how the supervisor mode and memory protection helped.

It is not a fact of nature that you must enter supervisor mode to use a device. It's a decision, and there's a reason for it.

Several people told me about backup and other such procedures; the question was specifically about protection against access by other people's processes. You get no marks for irrelevant information.

I didn't ask for specific protection methods. That doesn't mean that you can't mention them if you think they're relevant, but they won't do by themselves. A few answers read like lists of file protection and security methods, with no indication that the answerers knew what they were doing.

A few people thought that you could use the memory protection hardware directly with the disc files. In principle, it's possible (I think), but you'd still have to use the two methods as described to check the protection before you went ahead to set up some sort of machinery to control the disc access itself – presumably by relating disc sector addresses to virtual memory addresses in some way ?

(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.

The point of the question is that there's more to protection than keeping the information secret. so an argument based wholly on secrecy can't be adequate.

Someone commented immediately after the test that if the material really was "impossible to decode" (which is precisely what the question says), then it wouldn't be much use to the owner either. And that's perfectly true. I don't think it gave rise to any misunderstanding, but it does show how careful one has to be when setting a test. We really do worry about making the questions comprehensible; if we didn't, it would be much, much worse.

(Written slightly later than the above –) All right, all right : I should have said "impracticable". Or "impossible for anyone else to decode". Or something. You win – it does me good to be caught out sometimes. I wish you'd apply the same genius for precise and pedantic interpretation to the rest of the questions !

A significant proportion of the answers were based on the assertion that in fact everything could be decoded. That isn't true. (Proof : I offer a prize – I originally wrote "an A+ in 340 this year", but I'd have trouble getting that past Robert – to anyone who can decode the character string in quotation marks at the end of this sentence, and demonstrate that the decoding is correct in some reasonably plausible way : "x". This offer lasts only until the day of the 340 examination this year. That means you have to give me ONE string which you believe to be the plaintext of that message; a list of possibilities won't do.) But in any case that's beside the point; the question is conditional.

A subset of the answers amounted to an assertion that, even if decoding was impossible, reading should not be permitted because the material could be decoded given a longer time to work on it. Impossible means impossible.

An ingenious argument : if access to the disc is not controlled, someone could read the decrypting procedure. But in that case the conditions of the question wouldn't hold.

QUESTION 3.

(REFERENCE : *Requirements specification*, page 27.)

The moral of this question is :

READ THE QUESTION BEFORE YOU TRY TO ANSWER IT.

People who knew what they were doing did very well on this question, and many got full marks. (In fact, far too many got full marks. I'm very happy when people get full marks, but if there are too many it means that I've pitched the question badly. The object of a test isn't to give you, or me, warm fuzzy feelings, but to discriminate. I suppose tests will soon be prohibited.) (Quite a number got more than full marks, because I allocated 4 marks for each part but marked the question out of 20; I didn't carry the overflow forward to other questions.) People who didn't know what they were doing often gave the right sort of information, but scattered among the parts in a not obviously comprehensible way; I gave marks where it made sense, but a lot were lost by answering the wrong questions.

This isn't just my nastiness. The point is that simply knowing what happens isn't enough; anyone can learn a sequence of instructions. It is important to be able to use that basic knowledge to answer specific questions, and if you can't do that then One must wonder whether you understand the material.

A number of people didn't do very well in this question because they tried to answer it at a higher level than I intended. I think the clues are there : the level at which the description is given is the keystroke level, it has to be a level at which "the underlying operating systems are broadly similar", I ask for "basic operations". And there was the reference above, which was almost directly parallel to the question. It would also have helped to read the whole question; people who took this route typically found themselves with nothing left to say after (c) and (d).

A small number of people had given ***both*** answers to the questions which showed that they understood the point of the questions ***and*** separate good descriptions of the processes in both cases. On the principle that I was trying to assess your understanding, I gave some credit for both parts in these cases, taking care not to give marks for the same point twice. People who had simply written down all they knew about interpreting instructions under parts (a) and (b) didn't do so well. People who just gave me a memory dump with no attempt to identify parts of the question got at most half marks.

I guess from the evidence at my disposal that the main difficulty was that many people didn't read the question before starting to answer it. Therefore they guessed an answer for (a), which forced them to make an odd response to (b), by which time they'd given a lot of the information which should have been later and had to go to strange lengths to find something else to write. Those who had noticed what had happened and referred me back to material written earlier got marks where appropriate.

A few people wrote that they found the question confusing. That was a disappointment, because I'd gone to considerable trouble to spell out exactly what I wanted in great detail specifically to avoid confusion. Many people did give just what I was expecting, so it wasn't a complete failure.

(a) Apart from the task of working out what instruction was given, what operations must the system carry out in order to complete the operation ?

Locate the editor;
locate the file to be edited;
start the editor, giving it the file to be edited as a parameter.

Yes, the order and details are different in the two cases, but the same operations have to be carried out. And the system does find the file in both cases, even if the editor has to ask it to in Unix.

People who gave two different sets of operations presumably didn't see the bit which said "the same operations must be executed by the operating system in both cases" : ***please read the questions***. I didn't want all the detail that some of you gave, and tried to phrase the question to cut it out.

The "basic operations" are those which must be done somehow to perform the required action whatever the details of the implementation. You don't ***need*** to implement processes; you don't ***need*** to worry about protection and security; you don't ***need*** to have particular directory structures.

(b) State the information which the system must deduce from the input it receives in order to carry out the operations.

The identity, and thence the location, of the editor;
the identity, and thence the location, of the file to be edited.

One answer was "Operating systems don't deduce". My dictionary defines "deduce" as "To draw a conclusion by reasoning; to infer; to trace down step by step; to trace the descent from". That sounds to me like an algorithm. In logic, deduction (unlike, for example, induction and abduction) is following the strict rules of classical logic, which is just what computers do. And you might note our comments in ***HOW TO DO IT***, page 78.

There was another rash of people giving different answers for the two systems, which was inevitable if they'd given two answers to (a). I tried to make sense of it.

A few people said that the system had to deduce what operation to perform. The programme you load determines the operation, not the system.

"State the information ...." doesn't mean "tell me how to work out the information ....". It emphatically doesn't mean "tell me how to work out the information without telling me what it is", but some people did precisely that.

(c) For the Unix system, describe the signals which reach the system when you use the interface as described above.

Keyboard interrupts, each identifying the key depressed. (Commonly two for each operation, one for key-down and one for key-up, but that's not necessary for the question.)

Several people said that the vi instruction reaches the system, or something more or less equivalent to that – for example, a miraculous appearance of a full input buffer. ***Something*** has to work it out from the keystrokes.

(d) For the Macintosh system, describe the signals which reach the system when you use the interface as described above.

Mouse button interrupts (down and up).

Mouse movement interrupts (forward, backward, left, right).

The mouse cannot send its coordinates – it has no way to work them out. It can only send changes. It could in principle work out when it had a double-click, but it hardly seems worth building a mouse with enough electronics to do it when it's so easy to do in the computer; so far as I know, the mouse just reports the clicks.

Miracles here too – the system mysteriously knows where the pointer is on the screen.

(e) For the Unix system, explain how the information can be derived from the signals received.

On each key depression,
    read the character;
    echo it to the screen; –- *Not necessary for the answer.*
    if end-of-line
        group characters into words;
        perform the instruction. –- *The first word is the name of the editor (vi), and the rest is given to the editor as an argument; the editor will eventually present this to the file system to identify the file to be used.*
    else
        append it to text saved in a buffer.

In this part and the next, I expected that the information concerned would be the information you defined in (b). People who came up with new sorts of information got few marks.

I didn't worry about the details of how the file was found, so long as it was mentioned somewhere; many people thought that the system would look for the file, and I accepted that.

(f) For the Macintosh system, explain how the information can be derived from the signals received.

On each mouse movement signal,
amend the recorded mouse position;
move the pointer on the screen appropriately. –- *Not necessary for the answer.*

On each mouse click signal
wait for <a short time> to see whether there's going to be another click;
if so, it's a double-click. –- *So it means "open".*
compare the current pointer position with an internal screen map;
if the pointer is within a window of any sort,
identify the object corresponding to the window;
if it was a double-click,
if the window is an icon,
get the *name of the file* from the information in the object corresponding to the icon,
get the *name of the editor* from the attributes of the file,
perform the instruction –- *Start the editor if necessary, send it a message (an "Apple event") instructing it to load the named file*;
else ..... –- *Not part of the question.*

QUESTION 4.

(REFERENCE : *What must be done*, page 59, *How to do it*, pages 33-35 – note particularly the last paragraph on page 35.)

This was an easy one. I thought. Trivially simple data structures, stage II or even stage I. Wrong again.

You are not entitled to change the question. If you do, you are liable to get no marks, no matter how ingenious your answer. The first answer I marked introduced a "memory table" and a miraculous check to see whether memory was available. The second introduced a "memory map", despite the explicit statement in the question that "There is no system memory map". The third introduced a sort of virtual memory system. The fourth said that "there MUST be a table of valued pointers", and ignored the headers completely. The fifth was a bit better, but allocated "segments" which wrapped round the end of the memory area back to the beginning – no, I didn't say you couldn't, but it's a bit optimistic to assume that you can, and a segment is supposed to be a contiguous address space. The sixth was even more optimistic, and allocated "segments" in disjoint parts. The seventh had another miraculous operator, this time called "look". The eighth – ah, the eighth ! Nowhere near right, but at least it answered my question. At last.

Perhaps the oddest was the student who started by explaining (correctly) how to release a segment, then described in considerable detail a means of identifying empty memory areas which was inconsistent with that answer, then gave a pretty good approximation to my answer below, which ***is*** consistent with his answer for **release**, with a comment that "We do not consider it necessary to retain segments after they have been cleared ...". That was the student who didn't put his name on the answer script. In fact, it was a very good script, among the highest marks.

First

Implementing **get** : the pointer points to the first word following the segment most recently allocated. I have simplified the description of the address calculations by not taking account of the space occupied by the header; appropriate adjustments should be made to get it all right. The only complication is that if the available space is too big by an amount that is insufficient to hold a new header, a slightly larger area will be allocated.

1 : Remember this address.

2 : If the pointer is at the end of the available memory, return it to the beginning. The pointer now points to the beginning of a segment.

3 : If the segment is occupied, use the segment length to move the pointer to the beginning of the next segment, and go to 10.

4 : The segment is available. If it is big enough to satisfy the request, go to 7.

5 : The segment is available, but too small. If the next segment is occupied, change the current address to the address of the next segment, and go to END.

6 : The next segment is available. Add the length of the next segment to the length of this segment, and go to 4.

7 : The segment will satisfy the request. Mark it occupied. If the size is exactly correct, go to 9.

8 : The segment is too big. Set its size to the size required, and make a new segment header at the end of the required area, setting its size appropriately and marking it available.

9 : Return success, and a pointer to the current segment.

10 : If the current address is the saved address, there is no memory space which will satisfy the request; return failure. Otherwise, go to 2.

I think the answer has to be something very like that. Obviously you can reorder it in various ways, but the algorithm itself isn't negotiable. There has to be provision for coalescence somewhere, or it isn't a practical memory manager.

The question explicitly says "explain how". If you didn't, you didn't get marks. Common phrases were "it checks", "it looks to see", "it finds". How does it check ? Where does it look ? Where does it find ?

I don't see how it's possible to describe what happens without referring to the headers, but some people tried.

I didn't say in the question that you begin with all memory in a single vacant segment. I thought that was obvious, and even if it wasn't it was implied by the definition of a header with a "not in use" state, and the definition of **release**. I still think so. You can't reliably distinguish between vacant and occupied areas unless you do it that way, so far as I know, without spending quite a lot of time clearing released memory to zero (or anything) when it's released – not necessarily a bad idea – and far more time searching all the way through all the vacant memory looking at every addressable unit every time you come to the gap – which is a very bad idea.

Neither did I say that the sizes specified included the headers, which is what I intended and what I assumed in my answer. Most of you assumed the same, but a few didn't, which was fair enough. In fact, it doesn't matter one way or the other, but the example turns out a bit different if you allow an additional memory unit for the headers.

"Cyclic" means you repeatedly search through memory from start to finish looking for a space; it does not mean that you can allocate a segment which wraps round from end to beginning of the space. How would you use it ?

People who don't know what segments are could usefully find out.

Implementing **release** :

> The pointer given points to a segment header; mark it available.

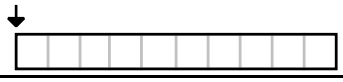
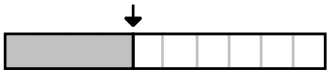
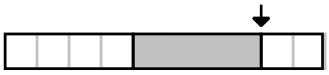
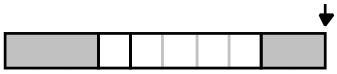
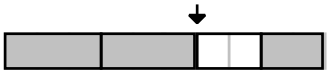
That's easy – but you have to say it.

Quite a number of people did say it. It was sometimes the only mention of the segment header in an answer. There doesn't seem to be much point in marking a segment free and then never looking at the header again.

Other people couldn't even manage to paraphrase the definition of **release** from the question, and defined their own version of **release**, which was wrong.

then

The example : the pointer position Ptr is the value before the operation is executed. Diagrams were not specifically requested, but they make the points economically.

Operation	*Ptr*	*Map (after)*	*Description*
(*Initial state*)			All memory is one free segment, length 10.
S1 = get(4);	0		Room to allocate 4 in this segment; a new segment header is constructed at the end.
S2 = get(4);	4		Same as above.
release(S1);	8		First segment free, pointer not moved.
S3 = get(3);	8		No room for 3 in current segment; no following segment, so restart; room for 3, so allocate as above.
S4 = get(2);	3		No room for 2 in this segment, go to next; this segment occupied, go to next; room to allocate 2 here.
release(S2);	10		Free the second segment – note that the vacant segments are not coalesced.
S5 = get(3).	10		No segment here, go to beginning; this segment occupied, go to next; this one too small, but next is vacant, so coalesce; now there's room.

Diagrams without explanation get very few marks, even if the sequence is right. And the explanation has to match your answer to the first part. If you didn't give a comprehensible algorithm in the first part, I checked your answer against mine.

Diagrams without the pointer marked got even fewer marks.

My reason for insisting on an explanation is that if you've even the slightest idea of what ought to happen then it's trivially easy to get the right sequence of events by drawing, or even just imagining, a diagram. Almost everybody did that, however bizarre the rest of their answers. But the memory manager can't do that – all it can see is a word (byte, whatever) at the address it's saved, and a description of how it works has to be given at something close to that level.

Generally, diagram or not, I expected three things : a description of successive states of the memory, correct according to the sequence specified in the question and your algorithm (if I could understand it); a statement of where the cyclic first fit pointer was at each step; and a brief comment explaining why each step was as it was.