Computer Science 340

Operating Systems

FIRST TEST, 1993 : 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.

QUESTION 1.

This question was not very well done. Roughly 60% of the marks were for listing the "important points" given, the rest for more detail where appropriate. (The interpretation of "roughly" depended on circumstances. I looked for evidence that the points were appreciated, even when they weren't explicitly mentioned.) Rather few people got past the 60% level. Many also missed out most or all of at least one of the important parts of the answer for each system, so only addressed about 67% of the question. 60% of 67% is about 40%, which is typical of the marks achieved.

Look at the question. In it, I do three things :

- *first I define two sequences of actions producing computer input ({ enter text, press RETURN } and { move pointer, double-click }); the sequences are quite deliberately made rather similar;*
- *then I say that each sequence has to be converted into a "basic sequence", which I specify;*
- *then I ask for sets of operations needed to transform the inputs into the "basic sequence".*

Rather a lot of people answered a different question, which they made up for themselves. One began "For this question I assume that we are meant to expand on the given sequence". Others made similar, though usually unstated, assumptions. The question is about implementing the "user interface", which was the topic of much of the first term's work.

The point of the question is that any operating system must have a perfectly standard way of dealing with the instructions that it is given, and must somehow or other end up with something more or less like the "basic sequence". It was my intention to focus your attention on this standard procedure in these two systems, and to cause you to compare them.

I didn't ask how to execute the "basic sequence" - people who told me got no marks for it. We won't get to how to execute programmes until well on into the second term. I was sometimes able to salvage a few marks from the wreckage, but some said nothing at all about how to get the required information from the input.

The "computer input" is a sequence of electrical signals from a keyboard or mouse; see the "sequences of actions". The line of text or the identity of an icon doesn't appear by magic - it has to be found, fairly laboriously, by the operating system in both cases. As this is a question about operating systems, the details are significant. You are not expected to know what the signals are, but it is reasonable to suppose that you are aware that some signal passes from keyboard to computer each time a key is pressed, and that the mouse must produce signals to show how it has been moved, and any change in the state of its button.

At a lower level, the question is about moving information around in operating systems. (That was the point of suggesting a top-down analysis.) A lot of questions to do with operating systems can be answered by asking what information is needed to perform a task, and where it comes from. Here, the information needed is defined by the "basic sequence", and the key to the solution is that the required information is presented explicitly in the input text for the shell, but has to be found by the Macintosh system by interpreting the input signals in the context of what it knows is on the screen.

*In the question, I said "I do not want a full description of **how** all the operations are implemented ..." - but many of the comments I wrote on your answers say "How ... ?". I accept there's an incongruity, but it's resolved in the recommendation to proceed by top-down analysis until you get to anything specific to a system. I think that my specimen answers include the actions that are essential if the system is to work, and nothing specific to any particular implementation except insofar as that's implied by the question - so I don't mention wildcard characters, or marking windows as selected.*

In hindsight, it would have been better to use "identify" instead of "locate" in the "basic sequence", but as you weren't expected to go into how the sequence was executed it shouldn't have mattered.

In the shell system :

```
Repeat until an end-of-line is found :
      On each key depression,
            read the character;
            echo it to the screen;
            if special ( backspace, end-of-line etc. )
                  act appropriately
            else
                  save it.
Group characters into words.
Proceed to the "basic sequence", using the first word as the name of the editor
                                    and the rest as the argument.
```

Important points :

get the input;
analyse it;
interpret it for the "basic sequence".

My impression was that most people knew pretty well how it worked, but didn't trouble to write it down. Unfortunately, I need rather more than impressions to convert into marks.

In some cases, it wasn't obvious that people had ever thought about what goes on. They seemed to think that the system knew by telepathy it had to find vi. I'd have accepted that if they'd explained how to write the programme that does it. The convention that the first "word" of the instruction identifies the action to be performed is essential to the operation of the system.

(The notes below are for information - they're about remarks made in the answers which weren't really responsive to the question.)

> *Some people were much too optimistic about the intelligence of the shell - they expected it to work out whether or not arguments were required. The shell passes the arguments anyway.*
>
> *Several people thought that the shell would read characters up to a space then look for a programme with the implied name and complain if it couldn't be found. That just doesn't happen - if it did, you'd get no chance to correct errors. No analysis of the input line happens until the RETURN key is pressed.*
>
> *The system only uses the first "word" of the instruction; assuming that word boundaries are significant, it splits up the rest, but does nothing with it except to give it to the newly started programme. That's because only the programme knows how to interpret the arguments.*

In the Macintosh system :

```
On each mouse movement signal,
      amend the recorded mouse position;
      move the pointer on the screen appropriately.
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.
      compare the current pointer position with an internal screen map;
      if the pointer is within a window of any sort,
            identify the window's owner;
            if it was a double-click,
                  if the window is an icon,
                        get the required details from the information about the
                                    object corresponding to the icon,
                        execute the "basic sequence";
                  else ..........
```

Important points :

handle mouse movement and click;
find what window is selected;
get information from icon's owner.

Most people mentioned getting information on the position of the pointer (which they often called the mouse), but didn't say how to find it. It's just as important to follow the movement of the mouse as to notice its clicks. Most people ignored the mouse movement, and thereby lost a couple of marks. I did explicitly mention "move the screen pointer" in the question.

And most also assumed that determining the identity of the icon was so simple as to need no further explanation. It isn't simple; it requires that the system always keeps track of the screen pointer (see above) and maintains a map of the screen.

You can't "double-click on a file"; you can only get (comparatively) directly at an icon. The icon is not the file, despite gibberish to that effect from people who should know better; in operating systems, if nowhere else, it's important to make distinctions clear. (Compare mouse and pointer.) An icon is a pattern on the screen; just conceivably it could also be regarded as (on the Macintosh) an entry in a file's resource fork. It is merely an attribute of the file, and to find out about the file's properties starting with the icon you have to be able to get to the file itself somehow. This was the most commonly omitted step in the answers given.

There is evidence of enormous faith that icons are omniscient, omnipotent, and otherwise possess powers normally reserved for a deity. This has always (for the last millennium and a half, or so) been a problem with icons; people invest them with attributes they don't possess.

> **icon** ... *c*in*, (*Eastern Ch.*) A sacred image, picture, mosaic, or monumental figure of a holy personage, usu. regarded as endowed with miraculous attributes ...

(A.L. Hayward, J.J. Sparkes : Cassell's English Dictionary (Cassell, 19th. edn., 1962), page 580.)

Lots of people wanted the Macintosh system to check files' access rights.

Quite a number of people didn't seem to know what happens in a Macintosh system. After using them regularly for at least two years, that isn't very impressive for computer science students. There's nothing in my pseudocode answer that you can't infer for yourself from observations of what happens when you use the system and general knowledge of what computers do.

Double-click means "open", not "edit".

QUESTION 2.

NOT PART OF THE ANSWER :

The **Unix** short cut illustrated is actually a short short cut. The primary means for editing the instruction line is - reasonably enough - based on the **sed** line editor. Here's an example which does the same as the shorter version shown in the question :

```
% rm deaf*
No match.
% !!:s/f/d
rm dead*
%
```

Notice in the **Macintosh** system that double-clicking is a short cut; the long way is to select the object to be opened, then to open it by selecting **Open** from the file menu. (The "option" key works with either sort of **open** instruction.)

Throughout this question, I was more concerned that there should be an answer than that it should be the same as mine. (- with one exception : I just do not believe that either of the short cuts could in any realistic sense be described as intuitively obvious. Could you really guess either of them without a fairly strong hint ? You could just possibly find them by accident, but that's irrelevant.) There is room for opinion on these questions; there is no room for sloppy thinking.

There was some confusion about definitions. A short cut is an instruction that isn't quite standard, or bends the rules, or is otherwise out of the ordinary, but give you a quick way to do something. Many people thought that wildcard characters were short cuts in the shell language - but these are built into the system as a normal feature, and always work.

A major difficulty in marking was that very many people didn't seem able to look critically on their own preferences - "I like it" was not distinguished from "it's good for everyone". That's exactly how Unix got to be the way it is, and it won't do in modern system design. Despite that, though, very many people (not always quite the same very many) made good points in their answers, so I didn't want to give zero marks which strict interpretation dictated.

Finally, I ran out of time, and had to finish off the marking much too quickly. I don't think anyone suffered by this unseemly haste - if anything, some people have more marks than they really should.

(a)

Simple : Neither short cut makes the system genie any simpler; neither introduces any principle by which we can more easily remember how to use the system. Both can make it easier to use the system in practice, but at the expense of a more complicated genie.

Small : Neither short cut makes the system genie any smaller. Neither is in any sense related to any other feature of the system, so the increase in size is, in a sense, as big as it can be. The Unix short cut could in principle be consistently used with the input to any system programme, so, once introduced, could convey overall benefits without additional effort, but it is hard to see how the Macintosh short cut can be similarly generalised.

Powerful : At the simplest level, neither short cut makes the system genie any more powerful : the system with the short cut can do no more than the system without it. Notice, though, that that must be true by definition - for a short cut is just a way to do something which is quicker than the "correct" way. (That answer begs a question. It amounts to saying that an aeroplane is no more powerful than a pair of boots, because both can get me to Wellington. To what extent is "power" associated with the speed with which we can complete a job ? As I can't answer that question, I've let my remarks stand, because they make an important point, but I'm not going to insist on the detail. Anyone wanting to take it further should begin by defining "power".)

EXAMPLE OF AN ALTERNATIVE - ALSO NOT PART OF THE ANSWER :

Suppose the shell offered an instruction which would bring back the line previously entered, and permit you to edit it using cursor keys, overtyping, insertions, etc., and then submit the corrected instruction. This would still be an addition to the system, so would expand the system genie, but would be easier to use than the method of the example because you would be able to see the edited instruction before submitting it. Technically, the "cognitive load" would be reduced.

WE LEARN SOMETHING NEW EVERY DAY SECTION, part 1.

After reading the comment above while checking the test paper, Robert Sheehan remarked casually that of course I knew how to do that. He was wrong - but pressing the up-arrow key does display the previous line, and pressing it again the one before that ...., and you can edit them and resubmit.

Nevertheless, the point is still made : Robert found out about this useful facility by accident. And I, of course, found out by hearsay again.

The system genie is a body of knowledge - so it's simpler if it's easier to learn, and smaller if there's less to learn. Power is something to do with how effective the genie is in practice; a more powerful genie will enable you to do more things more easily. (Or something like that - the definitions are not precise, so there's room for variation. I've already commented on the interpretation of "power".) I intended to apply the "easier to learn" and "less to learn" criteria fairly rigidly, and to take the answers for "power" as they came. I gave that up when it became clear that if I did rather few people would get any marks. Apart from that, I've given about half the marks for a simple statement which is to the point ("Neither short cut makes the system genie any simpler"), and the other half for a reason ("neither introduces any principle by which we can more easily remember how to use the system"). Simple statements which just repeat the question without indicating what the effect on the genie is ("You can edit the line and submit it again" - yes, there were some) didn't get many marks at all.

From an answer : "What is a system genie ?". It's the collection of stuff you have to know to use the system effectively. I talked about it during several lectures, and it's defined in the handouts (Requirements specification, page 20 - and enough people quoted it from there to convince me that it was accessible).

The system genie is as much about making the system easy to learn as easy to use. Unix is very easy to use - once you know it very well; its defect is that its system genie is enormous, so it's hard to learn well enough to make it easy to use.

The system genie is not the same as the system metaphor. The system metaphor (or illusion) is the model you have of the system. The system designer can try to design a system so that it projects a simple metaphor - such as that of a desktop, which is supposed to be easier to understand than a computer. The intention is to reduce the size of people's system genies, so that the system is easier to learn. If you get a good metaphor, then a lot of its properties can be comparatively "intuitively obvious", so they're easy to learn, or even can be guessed from other knowledge about the system or the object of the metaphor which it's supposed to resemble.

The reason for thinking about the system genie is to formalise the ideas behind "user-friendliness". Unfortunately, "user-friendly" is an advertising agent's word, and is therefore essentially meaningless. If it means anything, it's something like "what I happened to think was a bright idea when I wrote it". Avoid it in serious discussion.

A fairly common view was that short cuts were a good thing. If that's so, then why do we bother with the long cuts ? Why not just collect all the short cuts together and use them as the interface ? (Here comes Unix again ....)

A number of enthusiasts assumed that "^" meant the "up-arrow" cursor key. It didn't. It meant "^". Others thought I was using it as an abbreviation for a control character. Why wouldn't it just mean "^" ? If it had meant anything but "^", I would have told you. (And, no, they didn't write it down as an assumption.)

Several people contrived to "answer" the question with out mentioning short cuts. Others didn't mention the system genie. Odd.

(b) Are the short cuts "intuitively obvious" ? - I suppose that depends on your intuition, but they certainly aren't obvious to me. Neither of them has any discernible relationship to the knowledge I have about the rest of the system (though the longer Unix short cut based on the **sed** instructions *is* connected to principles which turn up in several parts of the Unix folklore). ^...^... does suggest a replacement operation to someone accustomed to the older style of editor, but that it should apply to the previous input line isn't obvious. "Option" has no obvious association with "I don't want this any more".

Should they be "intuitively obvious" ? - Probably not. We've seen that they have no positive effects on the system genie, so are likely to make the system harder to learn if included from the beginning. Once someone has learnt the basic operations of the system, it may make sense to *complicate* the system genie to gain more efficiency, but that's another consideration.

How should people be informed of them ? - Certainly not in the "First steps in using the ... system", whatever form that takes. Ideally, they should be presented in something like an "Advanced use of the ... system" document. Such documents may exist - but I learnt of both of the short cuts by watching other people.

My intention was again half marks for simple statements, with the other half for reasons; again, I was overtaken by reality and guessed.

I had thought of adding another part to the question asking you to design intuitively obvious shortcuts for the two actions illustrated. Those who thought that they should be intuitively obvious may like to try it.

Lots of you persist in calling interfaces "intuitive". I can only suppose you mean that they have to guess what you're trying to do. I wish I had an interface which would guess (reliably) what I was trying to do, and then do it.

"Intuitively obvious" is not the same as "what I am used to". Despite several claims, it is not *intuitively obvious that P means "print".*

An interface is not easier to understand just because you don't use a character keyboard. A well-designed pocket calculator can have a very satisfactory interface even though all its instructions go through a keyboard; video recorders, which just have buttons to push, are notoriously difficult to use.

It is not intuitively obvious that pressing "option" when you open something will do what it does just because pressing "option" often causes peculiar side-effects. If you could demonstrate that using "option" always causes the same sort of side-effect, you could make a stronger case, but that certainly isn't obvious.

The point about wanting an intuitively obvious set of instructions isn't so that you can remember them (you could call such instructions mnemonic if you wanted to), but so that you can guess them without having to be taught. The Unix "up arrow" key is easy to remember as a way to get to the previous input line, but I don't think you'd ever guess it without some prompting.

Some thought that obvious short cuts would make it easier to use them by mistake. I'd have thought it would be harder - if an action could obviously have some strange effect, surely you're less likely to do it ?

Quite a large proportion of people didn't answer "Should they be obvious ?". (Perhaps they thought that the answer was obvious ...)

(c) This is a real example. This is how I found out about it :

> Wed, 27 May 92 17:17:53 +1200 Date: Wed, 27 May 92 17:17:53 +1200
> Subject: Stupid users or was that stupid shells?
>
> I have a little story to relate which you might find amusing. Looking back I guess I find it a little amusing too.
>
> Late one night a poor overworked, underpaid graduate student was cleaning up her directory. Wishing to remove a couple of files beginning with the word 'dead' [before they got rotten you understand] she typed :
>
> % rm dead(
> Too many ('s
>
> Oh well, won't take a moment to fix that:
>
> % ^(^*
>
> [don't try this at home]
>
> What I want to know is where the shell got that extra space from.
>
> We learn something new every day

(It is reproduced here with the author's permission.)

There are two obvious comments (apart from those which are obvious when it's just happened to you) :

- If this is a deliberate design feature, then it is idiotically bad. It is bad because the behaviour is inconsistent; an instruction supposed to implement a textual replacement should do exactly that, not occasionally insert a space as well.

  In fact, it isn't a deliberate design feature, and it isn't an implementation error; it's the result of an interaction between two parts of a system which is so complicated that there are few people who have an adequate system genie. I am not one of those people, and I'm indebted to Reuben Cockle for giving me the explanation in his answer to this question.

  *The two parts of the system are the shell parser, and the editor. The parser first dissects the instruction into three components, rm, dead, and (. After this point, the three are never brought together again, but if required as a string they are displayed with separating spaces. The editor then works on the dissected version, where it can replace (by * - but still have three components. The rest follows. (You can check these assertions by playing about with various text strings, looking at the history, and trying various replacements.)*

  It is still bad design, for the reason given above. A text replacement should be just that - not a complex operation in which the text is changed first. The bad design could be cured by better implementation : keep the original text string in case it's needed again until the instruction has been executed, edit the original, and reparse it.

- If this is an implementation error, then it shows incompetence somewhere. A specification has not been properly checked, or is itself inconsistent somewhere.

If this isn't an implementation error, then the obvious possibility of catastrophic results suggest that at least there should be some warning. It is true that to be interrupted by a warning defeats the purpose of the short cut, but that's because it's a stupidly designed short cut.

Alternatively, we might hope that a warning would be issued by the **rm** instruction if asked to delete everything - or perhaps if asked to destroy everything by an instruction which is clearly redundant, such as that in the example.

This question was not intended as a trick; simply by comparing it with the earlier, apparently similar, example it should have been clear that something had gone dreadfully wrong. If you didn't see that, then, sorry, no marks, unless you found something else worth saying.

I accepted almost any reasonably sensible discussion.

In a few cases I had to guess whether or not people had seen the point. Any comment on the inserted space was a solid clue, and some people had obviously followed my example and assumed that the problem was obvious. Some other cases were less clear; I hope I got them right. The moral is that you should always be explicit in your answers to test and examination questions - make sure that the examiner knows that you understand the question.

"" and "(" are adjacent on many keyboards.*

QUESTION 3.

(a) The entities which might have access to the items of information from the user database are the autonomous part of the operating system (that which does things on its own during the normal operation of the computer system), the owners of the information, and various people who perform specific functions in connection with the system. Here I've distinguished between owner, system, and administrator, but more precise specifications are possible, and desirable in a full analysis; I've hinted at these in the answers.

The password :

Access by :	owner	system	administrator
Access type :	write (NOT read)	check (NOT read)	write (NOT read)
Reason :	Regular changes are desirable; only the owner should know the password; read access is not permitted to ensure that people can't find the owner's password from an unattended terminal.	The *login software* must be able to check on logging in. Without a check, the password is useless. The check should not return the password, but merely check a string.	The *user registration administrator* must be able to establish initial passwords; there must be some way to cope with forgotten passwords.

The search path :

Access by :	owner	system
Access type :	read, write	read
Reason :	There is nothing particularly sensitive about the search path, but equally nobody else need know about it. The owner must be able to set it to suit requirements.	The *file access* software must be able to read the search path to find out where to look for files; there is no reason for the system to change the search path.

The accounting information :

Access by :	owner	system	administrator
Access type :	read	read, write	read, write
Reason :	The owner can reasonably expect to find out the state of the account balance, but - for obvious reasons - should not be able to change the balance.	As resources are used, the *accounting software* must adjust the balance accordingly.	The *accounts administrator* must be able to read the balance to issue accounts, and to change the balance to reflect new allocations of funds.

Five marks for a plausible and appropriate comment under each heading; I didn't expect an answer of the same level of detail as my examples, but did expect one statement with justification. Full marks does not imply a perfect answer. Fewer than five marks implies that I thought there was something wrong with the answer given.

There was some tendency to mix the two parts of the question. Several answers said that only the operating system should have access to anything, and that people's requests should be handled through the system. Well, yes - I wasn't suggesting that you should operate directly on the disc files with tiny magnets. The operating system is there precisely to do this sort of job for you, but in many contexts it is simply a tool, and we have to decide what we want the tool to do before building it. This part of the question covers what we want; part (b) covers how to do it.

A thing called a superuser (also known as "root") appeared in several answers. Rather like an icon, it was credited with omnipotence, and was also deemed to be totally trustworthy. (Perhaps it's intuitive too.) ***PLEASE TRY TO BE CRITICAL*** *: Yes, I know Unix has an omnipotent superuser, but is that really a good idea ? Is the expert on operating systems software necessarily the right person to have unrestricted access to your bank account ? (This wasn't the only evidence of the fundamentalist position that if Unix does it, it must be right. The equally fundamentalist position that if Unix does it, it must be wrong, is equally unacceptable.)*

Despite my italics in the question, many people didn't bother to justify their answers. If something is specifically requested, write it down, even if it does seem obvious.

Many people forgot about the system in their answers, though I'd specifically asked for "who or what should have access ...".

Password :

> *I carelessly assumed that only login passwords were important; most of you did the same. In fact, it makes hardly any difference to the answer.*
>
> *Several people wrote about using passwords to protect access to files, not about access to passwords.*

Search path :

> *I received the impression that several people didn't know what a search path was. (It's an ordered list of directories to be searched for required files.) It isn't in the handouts, though it's in the DOING THINGS WITH UNIX notes I gave out in the Unix lecture; I suppose that's why I thought you'd have found out about it when using Unix in 211. I certainly mentioned it a couple of times in lectures. Apologies to those who didn't know.*
>
> *Quit a number of answers seemed to interpret "search path" as what I would call "pathname" - for example, A/B/C/D/E from question 2. That seemed to make some sense of their answers, but not much, because the question of access to the "search path" is then hard to understand.*

Several people wrote about permitting access to directories, not about access to the search path itself.

There is no reason why the system should be able to change a search path. My search path is a statement of what I think I want to look at; if the owners of the directories disagree, then it's up to them to protect the directories against my access. What I want and what they want are quite different things, and neither is under the system's control - it's therefore sensible to keep them quite separate.

Search paths are not completely innocuous. If I can change your search path, I can make you execute a naughty programme in my directory with the same name as one of the system programmes.

My assertion that there is no reason for the system to change a search path is a little oversimplified. Most search paths involve (usually implicitly) a system part and a user's part, where the system part is usually provided automatically when a new user is registered. A reorganisation of the system files may then make it advisable to change the system part in some way; but, as the order of items in the search path is under the owner's control, this may be hard for the system to do.

Accounting information :

A few people didn't seem to know what accounting information was. I certainly mentioned that in lectures, it appears from time to time in the handouts (for example, Requirements specification, page 20), and it's in the textbook.

Despite the question's focus on the user database, several people discussed the company's accounts.

(b) A general answer to this part of the question is appropriate, as the same considerations apply to each of the items of information. All the items of information are essential in one way or another for the proper functioning of the system, and it is appropriate to accord them hardware protection at least at the level of supervisor calls.

As the protection modes required are different in each case, it is probably sensible to keep the items in different files, to ensure that there are no loopholes through which one could - for example - open a file to change the search path and, by subverting the software, contrive to change the account balance. (On the other hand, as there is already supervisor call protection, the software should be protected, so this degree of caution may not be warranted.)

The type of protection required is independent of the subject, as all those who use the system should have the same access to the information; it is dependent on the object, as different sorts of protection are required for different items. An object-based protection scheme is therefore appropriate.

As all subjects are treated similarly, there is no need to distinguish between different protection levels for different subjects, so a scheme on the level of file protection codes should be sufficient.

There was a notable lack of justification. Perhaps you thought it was obvious that passwords should be kept in a file of their own - but it isn't : Unix mixes password information with other items of the user database. Etc. Justification was quite important for this part of the question, as there are quite a number of more or less plausible answers, and unless you give reasons for your choice I can't tell whether you know what you're doing or just made a lucky guess.

I occasionally caught an impression that some people thought that "it works" counts as justification. It doesn't. Ideally, you need "it's best", but sometimes (such as in a test) that's unrealistic. At least, though, there should be some sort of demonstration that the method meets the requirements to show how or why it works.

Password :

It is certainly necessary to impose some subject-based check on attempts to gain access to information on individual passwords; it is not satisfactory to assume that the owner of the session's login password is still sitting at the terminal. My answer is restricted to precautions necessary to guard access to the password file, and probably shouldn't be. (Observe that you can apply the same argument to access to search paths; that hadn't occurred to me before.) Some people who mentioned subject-based checks may not have received credit for it because they didn't justify their answer.

Many people recommended encryption. That's certainly a sensible thing to do, but doesn't affect the access rights and their implementation.

Search path :

Several answers advocated the Unix solution of holding search paths as files within each person's directory. This is quite a good mechanism; it satisfies the owner-only access requirement neatly. (The same protection is effected in my solution through the supervisor call access, which identifies the subject attempting access and allows control.) Its main disadvantages are that each access may require a file system search, and that the search path file is vulnerable to mistakes - such as accidental deletion.

QUESTION 4.

Only one person attempted question 4. The answer seemed fairly appropriate for question 3, so I marked it under that heading.

Alan Creak,
May, 1993.