Computer Science 340

Operating Systems

FIRST TEST, 1992 : 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.

A general comment : There are very few "correct answers" to any questions connected with operating systems. Even if there were, the answers would probably change as the technological base extended. If all the questions had simple and straightforward answers, I could set you a multiple choice test and save myself many unexciting hours of marking. (I probably wouldn't, though, as I think anything requiring stock answers is quite inappropriate for a stage 3 course.) Because of the absence of easy answers, it's important that you have some understanding of the factors involved so that you are able to reevaluate the answers when circumstances change. This is the sort of ability which I try to get at in the tests and the examination.

I have my own opinions, of which you will doubtless be aware. I know that they are opinions, though, and do not expect you to accept them uncritically. Unless there is a clear answer, I don't much care what you believe - for example, I don't care whether you are a rabid graphical interfacer or whether you would happily consign all graphical interfaces to perdition. I do care whether you have thought about the merits of different sorts of interface, and have good reasons for your beliefs, and can explain them when requested to do so.

There were 15 marks for each question.

If you think of an example which seems to answer a question, it is a good idea to ask yourself whether it is just one example of a more general, and therefore often more significant, phenomenon. For example, that one icon can be placed on top of another one, completely hiding it, is a nuisance; the general principle that this sort of hiding will always be possible if you have "opaque" objects on the screen which can be moved about at will is more interesting if you're looking for general properties of graphical interfaces, for it suggests general solutions. (Either make the system move potentially obsuring objects a little, or make objects slightly transparent, which so far as I know is a brand new thought which has only occurred to me because I looked for the general statement. QED)

And now, a word from our sponsors, the Council for Restoring English AsI Knewit :

> *There are no marks for spelling, grammar, or punctuation, but I correct mistakes if I see any. If no one ever tells you, you may never know. I accept both English and American spelling.*
>
> *(All the following comments refer to mistakes made by at least two people, so don't take them personally.)*
>
> - *The abbreviated form of "they are" is "they're".*
> - *The abbreviated form of "you are" is "you're".*
> - *"Available" is spelt "available".*
> - *"Basically" is spelt "basically".*
> - *"Explanatory" is spelt "explanatory", just as it was on the test paper.*
> - *"Privilege" is spelt "privilege".*
> - *"Receive" is spelt "receive".*

- *"Their" means "belonging to them"; "there" means "in that place"; "they're" : see above. ("Th'air" is occasionally used in rather affected poetry to mean "the air"; I don't think that has anything to do with the test, but I add it for completeness.)*
- *"Its" means "of it"; "it's" is an abbreviated form of "it is".*

One person appeared to think it was a 310 test; another thought 320, which is even cleverer.

QUESTION 1.

"Comment" can cover all manner of things : you can enlarge on the question, distinguish between different cases, support it or attack it, analyse it, explain it, and so on. They all require a measure of thought, and that's what I wanted in the answers. I didn't always get it. It seems that a number of people accept the superiority of the graphical interface as revealed truth, and do not wish to risk confusion by thinking about it.

I didn't expect answers of anything remotely approaching the length of this specimen. I gave full marks for two or three good points, or one or two very good points, of the many which could be made. Rather less than full marks usually means that I thought you made good points, but could have worked them out more carefully. Around half marks or below usually means that I thought either that you were wrong in fact or in inference, or that you should have covered more ground in some way.

"Using the Macintosh and Unix systems as examples" isn't the same as "Consider only Macintosh and Unix systems in your answer". It's not sound argument to condemn all textual interfaces because the Unix shell language is dreadful. It's also worth bearing in mind that quite a lot of people think that the Unix shell language is marvellous.

I don't discuss menus in my answer, mainly because I didn't think of it. That's because I intended the question to be about operating system interfaces, and I hardly ever use the Finder menus. Many people did write about menus, which I accepted, for they are certainly important even if the Macintosh system doesn't make much use of them. (A passing thought : the authors of the Macintosh system have gone to some lengths to avoid nasty technical words like "file". So why does the Finder have a "File" menu ?)

In quite a number of answers to (a), it wasn't clear whether points made were supposed to be about assertion (i) or assertion (ii).

The restriction "for operating systems " means that I only have to talk about using the interface to give operating system instructions; this is not a question about GUIs generally. With both sorts of interface, the same information must somehow be conveyed to the operating system, so the major difference is in the amount of "predigestion" which can easily be handled by the system.

(a)

(i) **graphical interfaces for operating systems are more self-explanatory than textual interfaces;**

Textual interfaces : Rarely give any indication of what you can do; typically require that you know exactly what software you want to use, the names of any file(s) you wish to work on, and details of parts of the instructions which affect the way in which each operation is performed.

As nothing is presented, it is impossible to show distinctions between different sorts of operation; while some (mainly older) systems required xxx to execute the system

instruction **xxx** and `run yyy` to execute the non-system programme **yyy**, this was more of an added complication than a helpful feature.

It is easy to make a textual interface very bad indeed, by not taking care to use sensible names for instructions and files, and by hiding useful functions in obscure parameters.

Graphical interfaces : Provided that there are not too many possibilities, a graphical interface can show you much of what you can do at any time by displaying icons on the screen to represent various system facilities and files which are available to you, and which you would have to request through the keyboard in a textual system. It is also possible easily to distinguish between different sorts of action - so as well as representing non-system programmes and files by icons, there can be a menu bar for system administration tasks.

It is sometimes suggested that the icons themselves can convey much useful information. (The suggestion is usually followed by a remark that "a picture is worth a thousand words", or more commonly a garbled version of that proverbial saying.) I am not persuaded that this is true. A very few of the icons I have seen have really been self-explanatory; and most of my windows rapidly fill up with identical icons representing word-processor documents. (Or, rather, they would if I ever used the icons.) It is much harder to make your own icons for files than to make your own names, and to make the icons represent their contents in any useful way requires a significant level of skill in graphic design, which few people have. (If they did, we wouldn't be continually pestered by competitions to design logograms for all manner of organisations which presumably have nothing better to do. Some pictures may be worth a thousand words, but it doesn't follow that they all are.)

There is more merit in the suggestion that you can represent useful information in the arrangement of icons on the screen. In practice, though, it doesn't seem to be done much; I don't think that any of the windows which I've seen has been laid out in any significant way. That's not to say that it wouldn't work, though.

While there is no reason why a textual interface can't be made to give you lots of information, the custom is not to. It would be perfectly straightforward to arrange a textual system to show lists of files in the current directory, classified as programmes and data, and of available system instructions as part of the tidying-up operations which go on at the end of each programme.

The textual interfaces therefore don't *have to* be any less informative than the graphical ones; the question is, why in fact *are* they ? The answer is a combination of history and economics. History contributed both an expectation that people who used computers would know a lot about them, and the teletype. The teletype was slow, and to present anything resembling a menu after each step in a session would have been intolerable. Communications lines weren't particularly fast, either, and a screenful of characters (once we got screens) could take a significant time to transmit and display. Economics said we couldn't afford to buy faster communications lines, so we put up with slow ones, and minimal text displays.

In fact, the distinction between "textual" and "graphical" interfaces is not in itself particularly significant. Even a textual interface using a display restricted to characters instead of full graphics is not greatly limited; provided the speeds of the hardware and communications are comparable, you can do just the same things with the character display as with the graphics - except that you can't have icons. If I'm correct in supposing that the icons themselves contribute rather little to the "self-explanatoriness" of the interface, then the original statement must be rejected.

A more significant distinction is that between the old purely serial displays - teletypes and their screen equivalents - where characters can only be presented on the current line (sometimes without even permitting backspaces), and displays in which any point on the screen can be

addressed at any time. Once full-screen addressing is possible, the two-dimensional screen interface can be exploited even if only characters can be displayed. Many sophisticated forms packages were developed for addressable character screens, and they worked very well indeed. (They still do.) For the rest of the question, I shall use "textual" and "graphics" to distinguish between the serial and addressable displays.

A gratifyingly large proportion of people did write "a picture is worth a thousand words", though not many were garbled. Well done ! (It was less encouraging that quite a few of those who quoted the proverb seemed to regard it as some sort of axiom - therefore, they said, a graphical interface is more self-explanatory.)

Someone remarked on the uniformity presented by the graphical interface, arguing that it made all operations easier to use, and that the conventions established helped to make the whole interface more self-explanatory. It's a good point, and the basis of ideas about the system Genie; but for a complete answer to the question you also have to ask why similar conventions couldn't be established for a textual interface.

Another point is that overall consistency is important; icons should always behave in the same way, even when displayed in windows that don't represent directories.

There seems to be a widespread opinion that you don't have to remember anything once you have a graphical interface. In fact, you may not have to remember the names of files so much, but you have to remember many more sorts of activity than you do with a textual interface - click, double click (or other mouse buttons), drag, drag to trash, selecting (text and icons), and so on.

One part of learning is easier with a graphics interface : you can learn the set of useful actions by watching someone else do it. This does make it easier to learn what it's sensible to try, and also that you won't get bitten if you do something silly. All you learn by watching someone use a text terminal is that you proceed by typing.

A good answer (only slightly paraphrased) : no, it isn't self-explanatory, but given a menu bar, some windows, and a collection of folders, then you should be able to get going with a bit of trial and error. I think that's related to another suggestion that the good thing about a graphical interface was that you didn't have to remember syntax.

A complementary point of view was put in some other answers. If you sit a real novice in front of an interface of either sort, it's fairly probable that nothing useful would happen in any reasonable time. Both of them need some learning. But I don't think that's really what "self-explanatory" means; we don't say that some written document isn't self-explanatory because it would be no help to people who can't read or don't understand English. We assume basic competence in using the medium in question.

Even a textual interface is a sort of metaphor : think of it as a "slave" metaphor. You give instructions to an agent which you expect to do exactly as instructed. That's fairly close to the basic computer, but not many basic computers can understand even the mangled human language implemented by the Unix shells. It is a metaphor; the observation that so much computer input and output looks at least comprehensible prompted many people to believe that the system could understand English, and that all you had to do was to ask it questions.

Someone - very properly - made an interesting attempt to define a criterion for assessment. The idea was to measure "self-explanatoriness" by the time it takes to "understand and use" a system, which in turn is defined as having sufficient knowledge to use the system confidently. (I would have said having acquired an adequate system Genie.) Think about it.

One interesting illustration that turned up a few times was assertion that the presence of the disc icons on the Macintosh interface showed you that your disc was in the machine. It would be a better illustration if it were true. Try ejecting using the Special menu.

Dialogue boxes are rather close to text.

Graphical interfaces rely less on words, and therefore less on fluent English. (I think that's one of the best arguments I've heard for graphical interfaces !)

Most people took "self-explanatory" to mean "helpful in showing what you can do", which is how I intended it to be taken. One took it to mean "effective in showing the state of the system", which isn't what I meant, but I accepted it.

But nothing is as simple as it seems : someone suggested that a textual interface was more self-explanatory because, as there was only one thing you could do with it (press a key), you couldn't go wrong, whereas there were several different possibilities with a graphical interface - press a key on the keyboard, move the mouse, press the mouse button. See remarks above about assuming basic competence.

Someone, somewhere, has done a magnificent publicity job for the Macintosh icons. Many people remarked that "you can tell what sort of file it is by looking at the icon". That's true, more or less. But why do you want to know ? All you do is click it anyway. It's much more use to me to distinguish between two files called `340 Test` *and* `340 Test Answers` *than to know that I have two Word text files.*

One or two people wrote (often following the stuff about the Xerox Star) of the icons becoming the files, or the display becoming reality. I suggest that such views are very sloppy thinking. If a few illuminated pixels on a screen which spell a name don't "become" a file, then how is it that a few illuminated pixels on a screen which form an icon do ? Computers are much too insidious for any such carelessness to be safe; there are already far too many people about who can't distinguish fact from fiction.

Another common belief is that a graphical interface somehow relieves you of the need to know how the system works. Try asking a random selection of people who use MS-DOS how it works; most of them don't know. They find their ways through the operating system bits by following recipes - "Just type these things and it will work". Note that the recipes do work; I've seen someone trying to follow a recipe on a Macintosh, and being quite lost because (I think) an icon had been moved off screen.

You have to be fair. There was a tendency in some answers to compare a Macintosh interface used by someone with some experience of it with a Unix interface used by a rank beginner.

You may care to reflect on just how self-explanatory the Macintosh interface was when you tried to read my Word and MacWrite files.

(ii) **it is easier to select an action using a graphical interface than with a textual interface.**

Textual interfaces : you usually have to specify the names of the programme and files you want to use, and perhaps additional bits which affect the details of what happens. Everything is done by poking keys. There is no physical limit on the number of actions which can be simultaneously available.

Graphical interfaces : You communicate your wishes to the computer primarily by a sequence of selection actions of some sort; any variants must be handled within the software. The selection usually requires that you use a mouse to position a pointer of some sort, then issue a signal - typically a "click". If the screen becomes too cluttered, it's hard to find the icon you want.

So which is the easier ? That depends on you, and on what you're trying to do.

You : The better you can type, the easier it is to use a textual interface. It's significant that many people move from the mouse to the keyboard alternatives once they become accustomed to the Macintosh; unfortunately the keyboard alternatives are usually combinations of one character key with one or two special keys, and have all the valuable mnemonic properties of the Unix shell instructions.

The problems posed by graphical interfaces to people with poor vision are covered in the other part of the question, but there are others. The less physical dexterity you have, the harder it is to use a graphical interface. For many people who don't have good control of what their hands do, the marvellous new graphical interfaces represented a great leap backwards. You can use a keyboard if you can get your hand to about the right position and stab (though shift keys can be awkward); you can't use a mouse effectively unless you have reasonably good "fine motor control" to control the pointer fairly precisely.

For the majority, who don't type too well, and have adequate control over their hands, the graphical interfaces undoubtedly have advantages. The screen becomes an extension to the keyboard, with some number (but probably not more than 20 or so, even on a large screen) of keys each of which selects a single action.

What you are trying to do : The magic of the graphical interface only works if the action you want is visible on the screen. If it isn't, then you have to go back to remembering where to find the required object just as you do with the textual interface. This is not an easy problem to solve. Unless you are interested in only a rather small number of activities, the screen will be too small; then you have to choose between putting too many icons on a small screen, using a window larger than the screen, and - probably best, but still losing the immediate presentation of all possibilities - imposing some sort of hierarchic classification on the activities, such as the Macintosh "folders".

Once again, the conventionally formed question is misleading. The significant distinction here is that between interfaces which require that instructions be entered in textual form through the keyboard, and those in which there are means to select an instruction already displayed (or readily available through some conventional means such as pulling down a menu) on the screen. This is commonly implemented by using a mouse, though you may find it informative to think about alternatives, such as cursor keys, joystick, light pen, touch screen, trackball, or eye-gaze detector.

Notice that not only selection by clicking is important : compare the Unix wildcard characters with the Macintosh interface's rough equivalent of drawing a box round several icons.

A good point : simple actions are easy with a graphical interface, but how do you manage more complicated things like file redirection ?

A lot of people said that selection with a graphical interface was easy, but didn't say anything about what you did if you couldn't see the icon you wanted.

Despite my warning, quite a few people put material more appropriate to question 2 here. It was usually wrong, too : not all graphical interfaces require you to select the object to be worked on rather than the programme which does the work.

A surprising number of people chose copying files as an example of an operation which is easier with a graphical interface than with a textual interface. It can in fact be quite messy; with a Macintosh, unless you're copying to a different disc, you have to select the file, choose "Duplicate" from the File menu (or enter D), drag the new icon to where it should be, then - unless you're happy with "Copy of X" - rename it. Compare "cp X d". (Exactly one person described copying correctly - but that isn't to say that others didn't know about it.)

And several people thought it "more natural" to drag icons than to give the instruction - but don't you ever ask people to do things for you ? I move things on my desktop, but I never copy then there (except on rare occasions, very approximately, and very laboriously), though I quite often ask people to copy things for me. This isn't meant as nitpicking - I knew what they meant, and marked accordingly - but just as a suggestion that things are not nearly as clear cut as some of you seem to think. Don't believe what you read in Macintosh (or Unix) books (or in lecturers' comments on tests, for that matter) without thinking about it first.

There was widespread confidence that the graphical interface is easier because you can see everything on the screen. I just don't believe it. I have an A3 screen which I tile fairly carefully to avoid accidental window overlaps, and I still don't have space to display more than a fraction of the files I need to work with. Perhaps the graphical interface is easier for people who have hardly anything to do ?

Many answers to both (i) and (ii) made the very sensible point that one's response to claims like those presented in the question depends strongly on one's background and experience. We all have a tendency to like what we know, even if it's Unix, and to regard new ideas with reserve. It is even just possible that what I fondly believe to be my unbiased and objective views on the relative merits of various sorts of interface, derived through many years of thought and careful analysis, may not be quite as unbiased and objective as I think.

(b) "Soundstation"

(i) The interface as described is far from self-explanatory - indeed, unless you move the pointer around, it conveys no information at all. Even moving the pointer may or may not convey information, depending on whether or not the pointer happens to touch in "earcon". To make it work at all, one would need a way of sounding all the available "earcons" - probably sequentially rather than simultaneously ! - to find out what and where they were.

Though a large number of *distinguishable* sounds with recognisable characteristics can be produced by exploiting musical pitch, rhythm, timbre, and so on, it is even harder to produce *self-explanatory* sounds than self-explanatory icons. It is possible to speak each item's name as its "earcon", but this can take significantly longer than is desirable. (Though it's an important secondary facility.) Grouping can still be effective.

(ii) The important differences between (human - assumed throughout) eyes and ears as devices to locate objects are that eyes are more precise, and they can attend to one object among many others simultaneously visible. Provided that the Soundstation "screen" is made large enough (it's an advantage that it doesn't have to be physical), it's easy enough to identify at least nine distinct directions on a 3 x 3 grid, and selecting one of these with a mouse-driven pointer would be reasonably straightforward.

Selection becomes more difficult as the Soundspace becomes more congested; it would be useful to be able to select a location, then to sound the "earcons" of nearby items in sequence and to select the desired item as it was sounded.

This part of the question was answered rather well. Most people found something sensible to say, and most agreed that the Soundstation isn't very impressive as decribed in the question. I think most identified some deficiency; a few went on to suggest how the deficiency could be remedied. It was clear that there was a general understanding of what the interface was supposed to do, and a sound (NOT intended as a pun) critical attitude towards the proposal.

One person in effect declined to answer the question on the grounds that it was impossible for a sighted person to judge the effectiveness of an interface designed for people who cannot see well. There is an important principle involved here, and I shall go into it in a little detail.

The basic point made is valid. I cannot tell just what it is like to be unable to see reasonably well. (I am slightly myopic, but certainly not sufficiently to get any practical idea of seriously impaired vision.) By the same token, though, I cannot tell just what it is like to be enthusiastic about graphical interfaces, and I don't feel too inhibited about discussing them. The analogy is not trivial; the point is that we are all different, and if you decline to discuss situations about which you don't have personal experience you will find little enough to talk about.

The other side of the coin is that we all have much in common : we are all human, with broadly similar emotions, desires, and thoughts, and we do not in our daily lives assume that all other people are completely unknown quantities. We accept the differences, and use our imaginations and our intellects to try to see the world from someone else's point of view if we need to. We can thereby come to some conclusions on behalf of other people. The dangerous step is then to put those conclusions into practice without seeking the opinions of the people whom they will affect; we must not forget the differences, but we must also not forget that because of them our conclusions will be approximate.

If you refuse to offer your opinion, then you are in effect depriving people without your knowledge, experience, and understanding of access to your knowledge, experience, and understanding. There are lots of people who have special needs of one sort or another and who want to use computers, but who have no idea of what's possible or of the questions which they should be asking. If they are to use computing effectively, then the people who do know - us - have to take the first step, and make suggestions. Following the foregoing remarks, we shall certainly express our suggestions by saying "Would it be a good idea if ... ?" rather than "What you need is ... .". (You may then find that they don't understand - through unfamiliarity with the technology rather than stupidity - so you have to provide a prototype before you can get any further.)

So far as this test question is concerned, the system has been designed. I asked you to apply your expertise on user interfaces to the proposal. Certainly your answers would need vetting by people who could speak for the intended community of users, but they have to have something to vet. Perhaps if the people who designed the Unix shell interface had taken the trouble to do the same, they'd have produced something rather less open to criticism. (Actually, that's a bit unfair : they may well have asked around a little, but the original version was never intended for a universal clientele.)

Some people wandered from the point a little. There was concern as to whether other activities could be managed. Just to set your minds at rest :

- *Blind people have been touch typing just as well as anyone else for decades.*
- *For text output, Braille printers have been available for a long time, and speech synthesisers are now widely used.*

- *True graphical output is much harder to manage, and still experimental.*
- *It really is possible to generate a wide range of distinguishable short sound sequences. Armies and navies got along for a long time with bugle calls and bosun's whistles without insisting that all their personnel should have a degree in music.*
- *"Tone-deafness" does not seem to be a problem in practice.*

Several people thought that a textual interface with speech synthesiser would be better than the Soundstation. That was perfectly acceptableas part of the answer to the question, but it's worth making the point that you may not have the choice. If your employer expects you to use a Macintosh, what do you do ?

How do you implement menus ? It's a good question, for which I have no snappy answer. Perhaps a more realistic one would ask how you manage without menus. Do you need them ? What sort of alternative is there ? (I have no snappy answers to those questions either, but it's a sound design approach.)

Someone pointed out that the Soundstation was designed to provide the same system Genie as the conventional graphical interface.

Several people were concerned that it would be hard to associate arbitrary sounds with items of software. I haven't tried it. But I don't suppose it's very different from remembering arbitrary control and escape codes, which people seem to manage well enough; and if you want to use an editor, it may be even easier to remember that it sounds like a descending major seventh broken chord played on a zither than that it's spelt "vi".

A rather pleasing idea was to identify the current window by distinctive background music.

QUESTION 2.

(a) **Explain what is meant by ....**

The "metaphor" is the way we think (or are intended to think) about the behaviour of the system. With the "desktop metaphor", we think of the system as behaving, in some respects, like a desktop, while with the "tools metaphor" we think of a kit of tools available for use.

Metaphors have become more important as computers have become cheaper, and have therefore been used more and more by people with little or no knowledge of how the machines or the operating systems work. The theory is that it is easier to explain how to use the system if it behaves more or less like something with which people are already familiar than it is to explain the system's internal workings.

This was more or less bookwork, and most people reproduced the party line more or less adequately. A number of people went to considerable trouble to describe the desktop and tools metaphors, though I didn't ask you to, and many didn't tell me why metaphors are now thought important whereas once they weren't.

There were two common variant answers.

- *One was that the metaphors are thought important now because we can now afford to spend computer power on matters not directly related to getting the work done. That's obviously a factor, but I don't think it's the answer. Once the interactive systems came along, people who used computers were very happy with the old-fashioned interfaces, and people who didn't use computers*

accepted that they'd have to learn. Cheaper hardware was usually seen as an opportunity to run yet more terminals from your even bigger mainframe. Even when microprocessors made it possible to build clever terminals, the idea was to make the old model run better.

- *The second variant was the economic answer : we're interested in metaphors because it sells more computers. That's a rather more cynical view of the world, and I think it's also wrong. The economic factors might perhaps explain why there are such a lot of desktoppy interfaces about, but a lot of the early ideas came from work designed to produce - in effect - operating system interfaces for children. The Xerox Star and Smalltalk grew up more or less together from work on how best to use computers in education.*

I don't think that you can get away without somewhere including the idea of making computers easier to use. Of course, it may well be that you want them to be easier to use in order to sell more computers, but that's a second step in the argument.

Another answer which hadn't occurred to me before, but which is obviously correct once you think about it, was that the metaphor is now thought to be important simply because people expect it. (Which is to say that you won't sell many computers if you don't provide some fancy windowy interface.) That's the same reason why almost all cars have their controls in more or less the same positions, which is probably not too bad, and almost all English typewriter keyboards begin QWERTY, which is well known to be an inefficient design.

(b) It is appropriate to look at the text from the point of view of the authors. It is they who issue the instructions, so they are analogous to the people using the computer. They know what they wish to be done; they presumably find this form of description clear and straightforward, and a reasonable way of communicating with a system which they must for safety assume to be fairly stupid.

Not everyone understood this point. The feelings of the entity which is to carry out the instructions are irrelevant (unless you wish to start a Society for the Prevention of Cruelty to Operating Systems); the quotations are presented as natural ways to <u>give</u> instructions.

(i) The *tools* are hardly ever mentioned; the only exception is the frying pan in line 6. The *objects* (oysters, egg, breadcrumbs, dripping) are mentioned when necessary. But the primary emphasis is on the *actions* : beard and drain, dip, beat, dip, coat, put, fry. Quite a lot is implied : you are assumed to know (or to be able to work out) which tools to use - that the flour and pepper should be put on a plate or other flat surface, that you know how to beat an egg - and that you will naturally put the frying pan onto some source of heat between lines 6 and 7. Notice that in practice it doesn't matter which tools you use, provided that the action is satisfactorily completed; you can beat an egg with an egg whisk or a fork, or by shaking it in a closed jar.

(ii) The "carpentry" metaphor is very similar to the "kitchen" metaphor. The only appearance of a *tool* is the reference to "cramps" in line 3, where they appear more in the role of *objects.* Again, the emphasis is on *actions*, and the tools are implied.

Two additional features are of some interest. First, lines 1 and 7 describe the current activity at a higher level - so lines 2 to 6 give the details of the action identified in line 1. Here is some evidence of hierarchic structure. This is appropriate to the much longer description of the filing cabinet (about 10 pages) than of the fried oysters (given essentially in full on the test sheet). Second, lines 9 and 10 give explanations to the "system" which executes the instructions.

We conclude that these commonplace "metaphors" emphasise the *actions* to be performed. To make them work, you have to know what actions are available; but that seems reasonable enough. You *do* know that you want to *write* a letter, or *alter* a memorandum, or *draw* a picture. This seems to me to be a very sensible approach to giving instructions.

In giving complicated instructions, it is useful to be able to identify tasks at different levels, so that the whole task may be described in terms of a hierarchy of subtasks of decreasing size but increasing detail. This suggests that an operating system interface could usefully emphasise actions rather than either objects or programmes, and provide some means for constructing subroutines.

Rather few people gave any discernible "Comment on any implications ... for operating systems design". Surely there must be some implications ! - even if only that current interfaces are perfectly adequate.

There were some interesting attempts to force the sets of instructions into one or other of the standard metaphors. Some of you must have very messy desks. There was wide divergence of views on which of the two quoted metaphors was appropriate in both the two cases - which I take as strong support for my assertion that in fact neither works too well.

There were also some misguided attempts to invent new metaphors based on the topics of cooking and woodwork. That misses the point; the metaphor isn't about the "subject matter" of the instructions, but about the forms of the instructions themselves. If you use your Macintosh for nothing but games, you don't start talking about an arcade metaphor.

You can make out a sort of case for emphasising the objects - oysters, or parts of the filing cabinet - on the grounds that they are regularly mentioned, or at least implied, in the text. Surely that's inevitable, though; you couldn't identify what had to be done without some reference to the objects ! I still maintain that the major stress in the instructions is on the action to be performed. It may be that, particularly in the cookery example, that's because the same object is subjected to a sequence of processes, so that once we have defined the object we don't need to think about it much. Perhaps if the same process were applied to a sequence of different objects, we would find a different view. One could also remark that just identifying the object simply isn't sufficient to define an operation; the egg could be beaten, or separated, or broken into the flour and pepper, or hard-boiled, while the rails could be glued in, or nailed in, or screwed in, or wedged in - and there are lots of other possibilities in both cases. One would then go on to wonder whether the object metaphor as implemented in the Macintosh might only work in such ludicrously oversimplified situations that it would have little real applicability to genuine problems.

Many people regarded actions and tools as the same thing. I've commented on different ways of beating an egg; the woodworking book describes elsewhere two different ways of gluing, and that was before the days of PVA and aerosols. If you specify an action only, then by implication you don't specify the tools to be used in making it happen.

Observe that the desktop metaphor does not necessarily imply an object-oriented view of the activities. The desktop idea is much more concerned with a way of storing and finding things than with a particular pattern of instructions. I am using a real desktop at this very moment, and it is strewn with piles of test scripts (to be marked, being marked, having been marked, with an overflow onto the floor because my desk isn't big enough - the rubbish bin is on the floor too, by the way), the test paper, my specimen answers, the computer I'm using now, a coffee mug, some boxes of discs, and a pair of scissors. It is working very well. I drag a script from the

unmarked pile to the being-marked position. It then does me no good to click on it. Nothing happens until I pick up a tool, closely resembling a ballpoint pen. In fact, the script has no "natural" action : you wrote it, I mark it, then I sort the pile, then I transcribe the marks onto a spreadsheet. An object-oriented implementation as done by Macintosh would be pretty useless; tools work well.

Two or three asserted that no metaphor was used in the examples. In the commonplace sense, that's close to true - but the same can be said about the tools metaphor, as you really do have a set of tools at your disposal which you use as required. I think it's reasonable enough to interpret "metaphor" in this question in the sense in which it is used in computing - as a name for the way we think about the system we use. Observe, too, that if the instructions in the examples are effective - from their originators' points of view - without a metaphor, then this view would presumably be a good base for a computing metaphor.

I suggested that it was "close to true" that no metaphor was used in the instruction books. It isn't quite true. If you ever have to write instructions which are to be followed by a lot of people, you very soon find out that it's no use aiming your instructions at the real people. You have to use a "metaphor" (in the computing sense) of usually well-meaning but bungling idiots, who are ready to take offence if treated as usually well-meaning but bungling idiots. (One of the nice things about working in a university is that that description can be relaxed; it's much more pleasant to deal with usually well-meaning but bungling intelligentsia, if only because the description fits all of us and we all know it.)

Someone pointed out the importance of the observations - "when smoking hot", "until the glue has set" - but didn't go on to suggest that an operating system should have some means of monitoring what processes do when they're running.

Someone remarked that there were a lot of technical terms in both passages, and wondered if the readers would have access to explanatory notes such as I gave on page 3 of the test. Bearing in mind that the reader is the analogue of the operating system, that's just a requirement that the system must know how to do all it will be asked to do. Put the other way round, that's to say that it's only sensible to ask an operating system to do the things it knows how to do. Both the "operating systems" in the question are very specialised indeed : one knows how to do cooking, and one knows how to do woodwork. In practice, all operating systems offer rather different facilities, which is one of the reasons for the (belated) interest in standards such as Posix. (As a matter of interest, the woodwork book defines all the terms I defined except "squareness" and "in.", which is fair enough, but it's hard to find any explanatory material at all in the cookery book.)

One good point was that the obvious usefulness of the lists of steps in both cases suggested that a metaphor which included scripts ("command files") would be useful.

A remarkable number of people (or a number of remarkable people ?) asserted that the woodwork example followed the "noun-verb" pattern. Where ? The only evidence I can see is in lines 9 and 10, which are not instructions anyway.

One person, making a number of assumptions of dubious reliability, suggested that the examples "might suggest the O.S.'s be provided with two, separate, gender-specific User Interfaces". Two comments :

- *While not endorsing that opinion, one could make a reasonable case for supplying different interfaces for people with different personality types. People are measurably different in the ways they think about problems, and might well prefer different metaphors.*

- *Curious, though, that the prefix "Mac-" means "son of", and the suffix "-ix" is sometimes used as the feminine version of "-or".*

(The next two paragraphs are an extended comment which hasn't much to do with the question, but which you may find interesting.)

If I am correct in my contention that an "action" metaphor is the natural way to give instructions, why is the "tools" metaphor common ? I suggest that it is, at least to some extent, an accident. The intention with the early system languages was to prescribe actions, not tools; that's why they were commonly called "command languages", with jobs specified as sequences of "commands" (which I would prefer to call "instructions"). Many of these were certainly not the names of programmes - "`// JOB`" meant "start a new job" to the IBM1130 Monitor system, while "`COMPILE <filename> WITH <compilername>`" meant what it said to the Burroughs B6700 MCP. Even when a programme was to be executed, its name was commonly not the primary instruction - "`// XEQ <name>`" and "`RUN <name>`" were used in the two systems mentioned. The only way to do anything in a computer is to execute a programme, though, so the semantic distance between the name of a programme and the task it performs is very small, and it became common to use the name of a programme as a synonym for the instruction to execute it. Indeed, this was regarded as a valuable attribute of a system - you could "extend the range of system instructions" by writing your own programmes. Even then, a distinction could sometimes be observed between the instruction and the tool. The best example I know comes from the DEC Tops-10 system, where, "`edit x`" meant "use the programme **edit** on the existing file **x**" while "`create x`" meant "use the programme **edit** to make the new file **x**"; the same programme was used in both cases, but the appropriate tests for existence of the file **x** were applied and different cases reported as errors. The distinction between tool and action has perhaps since become blurred, but I think there's a strong case for questioning the existence of the "tools metaphor", except in the minds of people who don't understand what they're doing.

A final interesting point is that in both cases additional information is available to the "system". The cookery book is not written for experts; but it is presumably assumed that if you don't know how to beat an egg there will be someone not too far away who does. (An off-line "help" system is available - on page 2 : "Housewives ! send your cooking problems to Elizabeth Edmonds ... A fully qualified cookery expert will be pleased to advise and assist in solving your household cooking problems.") Similarly, as I remarked above, the last two lines of the carpentry example presented information on why certain instructions were given. Clearly, this information is there, or offered, because the "system" is human, but there are two parallels with operating systems. First, fairly practically, but only tenuously connected with the examples, there is the principle that if you want an operating system to be more helpful, you have to give it the information it needs to do the additional work. Second, following the examples, but much more speculatively, a clever operating system given additional information such as the overall aims of a job or reasons for including certain steps is likely to be in a much better position to carry out intelligent fault diagnosis and, perhaps, correction or even prevention.

QUESTION 3.

I was surprised that this question was so unpopular - and on the whole so badly done. I'd expected most people to be rather happy about a "real" operating systems topic after the comparatively waffly stuff about history and people. If the answers I received had been better, I'd have assumed it was just shortage of time, but that didn't seem to be so.

The NOTE at the beginning of the question was intended to define things rather precisely. The first part cut out clever tricks with multiple processors and networks; the second said that there was no secret information; the third defined security as an internal matter, concerned only with memory access. I slipped a little on the third, because I didn't state that "memory" meant primary memory - files are

also a sort of memory. I think the intention was clear from the usage of "memory" elsewhere in the question, and also because the majority obviously accepted the meaning which I intended, but I gave some marks to answers which said the "memory protection only" system was insecure because people could get at each other's files.

To demonstrate security, it isn't enough to say that you can be good if you try; you have to show that you can't be bad however hard you try. ("Good" and "bad" are here used in the purely superficial sense of successfully achieving, or not achieving, certain aims; this is not the most appropriate context for a dissertation on the reality and nature of sin.)

Far too many people assumed that the question was preceded by a phrase something like "Using a processor just like yours at home with the minimum modification needed to make it fit the question ...". Well, it wasn't.

It was not particularly encouraging to see how many people assumed as a matter of course that you could cause an operating system to go wrong in one way or another by simple means - passing silly parameters, interrupts, etc. You may like to learn by heart : "Any operating system which breaks down under any circumstances is incompetently designed or written". I remark that it's bad design to try to implement any sort of function using hardware that isn't capable of giving proper support.

(a) (i) **In a system with memory address checking but no supervisor call :**

If there is no supervisor mode, then there can be no special instruction. In this case, any instruction which reloads the base and limit registers must be available to all programmes, which would clearly nullify the security and answer the question. To proceed, therefore, I assert that there can be no such instruction. If the system is to be shared, then, each programme must be permanently associated with its own base and limit registers, and process switching becomes essentially a matter of directing the processor to use a different set of memory registers (together with a little necessary housekeeping).

Each programme is thereby constrained to run within some defined area of memory, and has no way out. (If it has, then security is going to be impossible anyway.) It therefore cannot rely on system facilities for memory management, input and output, and so on; so each programme must contain its own procedures for these purposes. As there is no supervisor mode, that's easy enough - in effect, each programme must contain its own monitor system, which isn't very far from the way a single-user monitor system works anyway. There may be interesting consequences for the input and output devices if several programmes try to use them at the same time, but that isn't our problem.

If the system is to remain shared, there must be some mechanism for a programme to inform the operating system when it has finished. This need be no more than a signal containing only information on the identity of the programme; the operating system can then carry out the necessary administration. The operating system will presumably have its own base and limit registers set to give access to the whole of the memory, so can load new programmes as required.

I conclude that a secure shared system is possible, but other hardware facilities - multiple base and limit registers - are necessary.

(This is something like the IBM virtual machine technique, but the analogy is not perfect.)

It is worth remarking that the absence of an instruction to alter the base and limit registers does not mean that they can't be changed; they can be made part of the memory accessible to the operating system. In the IBM 1130, the three index registers were implemented as words 1, 2, and 3 in the memory. I'm not suggesting that it's a good idea,

but we've been forced to describe a rather peculiar computer, so it may as well be a little peculiarer.

I gave some marks to people who got as far as noticing that anybody could set memory limits, and concluded that security was impossible. The appropriate conclusion is either security is impossible or people mustn't be able to change memory limits - which is the reasoning underlying my specimen answer.

A full answer must somehow address the problem that with no supervisor call everyone can do everything.

Each programme must have some system code loaded into its partition, which it can mess about with as much as it wants - but provided that the memory access protection still works, that won't do them much good.

Lots of people seem to think that a branch instruction doesn't require access to memory. Where do they think the code comes from ?

One suggestion was to forbid people to write their own programmes. That could certainly be made to work, but I don't think it really qualifies as a proper implementation of a protection system for a computer. If I were buying an operating system, I certainly wouldn't think much of it.

(ii) **In a system with supervisor call but no memory address checking :**

Insecure. Overwrite the destination of the supervisor call with your own code, and do whatever you want.

Several people didn't know what supervisor mode was. It's mentioned in the notes I distributed. It's also mentioned in the textbook, but it's in chapter 15 which I find I haven't mentioned in my notes anywhere. But didn't you cover it in 211 ?

Someone pointed out that security is still possible if you execute everything by interpretation - in effect, building yourself a software virtual machine which performs all the checks that the hardware machine doesn't.

(b) There are probably lots of ways; here's one example. Use the perfect compiler to write an assembler producing "imperfect" code which can generate any address you want. The compiler, being careful, is unlikely to permit you to branch to an array or to overwrite any part of the programme, so execute the code by writing it to a disc file formatted correctly as an executable programme, which can then be run in the ordinary way.

I slipped up again. The last sentence of the question should have read "Show how you can write a programme which, while running in one partition, can gain access to any part of the computer's memory". Most people took it that way; I apologise to the few who didn't. As the question was about security, I gave marks for any answer which described a way of doing something more or less illegal under either interpretation of the question. I remark that the triviality of the answers to the "other" question - "Specify a different partition for each programme" - might have given you a hint that there was something amiss.

One answer suggested using something like a virus, thereby getting access through other programmes. It doesn't guarantee to get everywhere, but it's certainly uncomfortable.

Several people must have thought that I intended the "rusted" to be put back; their compilers were certainly full of holes. There seems to be a deplorable lack of understanding of what compilers can and can't reasonably do. I think it's reasonable

to suppose that a trusted compiler is one which won't do anything obviously stupid - such as :

- *Implementing C.*
- *Implementing VAX assembler instructions.*
- *Letting you specify the range of the partition - and particularly letting you specify the range as the whole of memory.*
- *Compiling code which will branch to the value of a variable interpreted as an address.*
- *Allowing a programme to overwrite its own code.*
- *Executing a system call which lets you refer to any address. (Why would there be such a call anyway ?)*

Others didn't take seriously the statement that access was only through the compiler. This ruled out solutions like :

- *Import a code file from another computer.*
- *Change the programme or the compiler by "poking". (No self-respecting system would let you do that anyway. It's for toys, not computers. I agree that it's sad that there are so many big toys around.)*

Alan Creak,
May, 1992.