Computer Science 340

Operating Systems

FIRST TEST, 1991 : SKETCH ANSWERS

Remarks in italic type were added after the test.

I correct spelling, grammar, punctuation, and the like when I see mistakes (until I get tired). There are no marks for good spelling and grammar, but if no one ever tells you about your mistakes, you don't get a chance to improve your performance.

Give some thought to how you write down your answers. If they contain a number of separate points, lay them out in separate paragraphs, or a table; if they're in the nature of algorithms, consider pseudocode. If you don't, it's easy for points to get missed in the middle of hectares of (naturally impeccable) handwriting. I try to read it carefully, but the easier you make it for me, the more reliable the process. Of course, if you're trying to disguise a profound lack of understanding, untidy layout might help. But not much.

Lots of people missed out bits of the questions, or decided to answer questions which weren't on the test paper, and therefore denied themselves a share of the marks. It happens every year. Why ?

No, I didn't expect as much detail as I give you in the answers here - but I did expect a selection of the important bits.

QUESTION 1.

The idea of the HINT was to encourage you to work out what the system was supposed to do before you started to worry about how to do it. I didn't ask you to write it down, so I can't check on whether you used it; a few people wrote down something that could have been a response to the hint, but then apparently ignored it, and at least one wrote a description of how a conventional incremental backup system worked, which ISN'T the same as saying what it's supposed to do.

The point of an incremental (or other) backup system is to guard against accident by keeping a copy of everything off-line. Keeping "backup" copies on the same disc is of only minimal help. See the sheet "PROTECTION".

It didn't (and still doesn't) occur to me that combining the two services in the same system meant any more than running a generation-based file system and providing incremental backup. There were occasional suggestions, not amounting to clear descriptions, that systems in which old generations of files were immediately copied to backup storage were under consideration. These don't, in fact, provide either the "undo" you get from a generation system or a satisfactory backup : to "undo", you have to retrieve a copy from the backup medium, while you never get the current versions into the backup system at all.

The aim of the backup system is to restore the file system to its state at the time of the most recent backup. It must therefore record sufficient information to recreate this state. It should be designed to perform as efficiently as possible; in particular, we don't want to copy files unnecessarily. If a new version of a file has been made in a system which preserves, say, three versions, then the backup already knows about the two older versions, so all that needs to be saved is the newest version. The correct number and order of versions to be kept in the restored system can be managed by the restoring software.

(a)

To ***save*** *a file is to have its current state recorded in the system directory as a permanent file. This most commonly happens when a programme has finished making a new file, which I have assumed in my specimen answer. An alternative interpretation is that found in many editors and similar programmes, where a* ***save*** *instruction is provided which performs those operations, but keeps the file open for further amendment if need be. In neither case is* ***save*** *concerned with changes to the file data which may be made (by* ***write*** *instructions or equivalents) in normal use of the file.*

Many answers included nothing at all about backup. Perhaps not everyone read the first paragraph, where the combined backup and generations system was described.

That paragraph also speaks of a backup system as a way to protect files from accident. Just renaming a file "???.bak" doesn't help much : what happens if the disc crashes ?

Answers were usually vague, particularly as to detailed descriptions of what happens. As I went to some trouble to emphasise that the test would concentrate on how things worked, I thought that was odd.

Some answers weren't at all clear because you confused words like "file", "version", "generation", and so on. The question speaks of a "file" composed of several "versions", or "generations".

Confusion in terminology is encouraged by some systems' habit of automatically keeping ".bak" files in some circumstances; in the terms of the question, these are different versions, not backup copies. It's because of this sort of confusion that I get worked up about the terminology of computing. The confusion is avoidable in principle if all real professional computists take care with the way they use words, and ignore sales blurbs, advertising agents, large computer companies, and hackers. Compare, say, physics : terms like "work", "energy", "weight", and so on, which have all sorts of vague meanings in popular speech, are nevertheless precisely defined in their physical applications.

Alternative ways of keeping track of the versions were acceptable; for example, some people relied on renaming the files in some systematic way.

There were a number of suggestions that deletion should remove the current generation, but preserve older generations. That doesn't really seem to be very profitable : either you want to get back to where you were, or you may as well throw the whole lot away.

Not many people "discussed possibilities" : almost all gave one view of deletion, often with little or no justification.

If you claim that your operating system provides multiple file versions with undoable deletes, then you absolutely cannot start overwriting on the disc anything that might still be required. Conversely, if you want to keep file data on the disc only until you need the space, you don't have a protected system.

There was a widespread preoccupation with keeping lots of information in file names - file generations were identified by suffixes (typically, and regrettably, ".bakn"), and deleted files by prefixes. Why mix it all in with the name, thereby making it much harder to retrieve ? It's much easier to build appropriate fields into the directory structure, so that you can get at the information directly.

For each file known to the operating system, it must keep an ordered list of the positions occupied on the disc by the several versions of the file. It must also count the number of new versions saved since the last backup operation. The backup must save the correct number of newest versions if it is to be able to restore the system to its state at the time of backup.

When a new version is saved :

```
add the location of the new version to the head of the list;
add 1 to the number of new versions saved;
if the maximum number of versions is exceeded,
    remove the oldest version from the list;
    remove the oldest version itself from the disc.
```

When the owner deletes the file, the system can respond in several ways. The simplest is to implement the decision immediately, and to delete all versions; but that defeats the "undo" objective of maintaining the multiple versions. If instead the existing versions are kept, but the file marked as deleted, then the disc is likely eventually to become full of versions of deleted files, most of which are of no further interest to anyone. A sensible compromise is to mark a file as deleted, retaining all versions, but to remove the versions at the end of the current job or session.

Whatever the decision, the backup system must conform to it. If the existing versions are kept, but tagged as deleted, then the "deleted" marker, together with any new versions of the file which have been saved since the last backup, must be copied to the backup tapes. If the deletion is instead supposed to be absolute, the "deleted" marker must be copied to the backup tape, but in this instance no files need be copied, and the disc system's record of the file may be removed after the marker has been copied.

(b)

Some of you are happy to believe that the combined system is "obviously" more expensive; there are "many more" files to copy to the backup medium. You must be politician's dreams : DO WORK IT OUT ! You have all the information you need.

The question asked for "Comment" : that doesn't mean that any comment will do. I expected a comment worth half the marks for the question, but didn't always get it.

The question also asked for a <u>comparison</u> of costs, which quite a large proportion of people ignored.

There were occasional comments about the merits of file systems which were not described. Some of them might have been good, but if I don't know what you're talking about I can't assess it.

A number of people checked whether or not any files had been changed - but without describing any mechanism in (a) for recording the required information anywhere.

Some time was wasted on careful descriptions of the linked-section implementation of generations described in a sheet which I gave you. I think the question is independent of the implementation technique - which is one reason why I asked for a comparison of the costs. I'm rather surprised that you should consider this quite odd technique even though I made no explicit reference to it.

REMARK : I can't think of a clever way to manage backup with the linked files; there may be a better way, but I suspect you'd have to store the "real" file - by traversing all the links - each time. That would make restoration a lot harder. Note, too, that, if it were possible somehow to backup amendments only, to retrieve a copy of a single file that had been lost or damaged, you'd have to read right back to the first filed version. But so far as this question is concerned, I don't think it matters, as the backup cost would be the same in both cases.

FURTHER REMARK, added very late in the marking : I have just read the first answer to this question which explains clearly that the comparison considered is between backup in a system which stores complete conventional files, and backup in a generations system using tree-structured files as described in the sheet "FILE GENERATIONS". I'd considered backup costs of tree-structured files, with and without generations. That means that my assumption that the same implementation technique was used in both cases compared was wrong, so my conclusion could also be wrong. But my rightness or wrongness depends on the validity of my conjecture that the best way to backup the tree-structured files was to reconstitute them into ordinary files anyway. Therefore, if anyone cares to offer me a clear and comprehensible written account of an efficient backup and recovery technique for the tree-structured files, and undertakes to explain how it works in a 340 lecture, I'll give everyone 10 more marks for the question in acknowledgment of my fallibility. The technique should be supported by some reasonable sort of analysis of its efficiency, and it would help if it were presented in machine-readable form.

Several answers included suggestions, either explicit or by implication, that the backup device would have to be on-line permanently for the combined system. I gave the HINT in the hope that it would cause you to think about the purpose of a backup system : see the first line (on page 1) of the specimen answer.

Many people completely ignored the retrieval cost.

Some interpreted "retrieval" as getting back a copy of a single file which had been lost by accident or error. I hadn't thought of that, but it's fair enough. There are two answers, depending on what you want to retrieve. You would normally only need the latest version, in which case the answer is very like that for the complete retrieval - you'd expect that the cost would be perhaps slightly higher for the combined system, but not much. If instead you want to retrieve all versions, then you will typically have to do a lot more work.

If at most one new version of a file is saved between backup runs, there is no additional expense for the backup itself; if several new versions have been saved, several (up to the maximum number of versions) copies must be stored in backup, so some additional expense is incurred. The maximum cost is greater than the cost without multiple versions by a factor of the maximum number of versions; in practice, it is very unlikely that this limit will be approached, as it implies that the majority of the files which have been changed require the maximum number of versions saved, while the remainder require none, a very curious pattern of activity.

The cost of the recording medium actually used, like the cost of the processing, is proportional to the amount of material recorded. In practice, it is likely that a complete tape (or disc, or whatever) will be used, so it is rather unlikely that the cost will be any different.

The cost of restoration is just the cost of reading back all the files stored in the backup copies; it is (proportional to) the sum of the costs of the backup operations, and is therefore governed by the same considerations.

QUESTION 2.

This was deliberately not a question about a Macintosh; people who answered it as if it were got fewer marks than they might otherwise have done.

Several answers included references to the position of the mouse. They meant the pointer. Get it right. (See remarks above on terminology.)

I tried to make life easier for you by abolishing interface errors - but quite a few people told me about them anyway.

(a) : MESSAGES.

"The TUI displays text in windows on a screen, replacing conventional computer terminals"; "The TUI should handle all details of managing the screen display, but should otherwise be as simple as possible ...". You'd expect, then, that it would be drivable with a set of instructions quite like those used for a conventional screen, with minor modifications necessary to sort out which window was intended. There is certainly no need to transmit information about pointer position, mouse movements, or current character position.

The keyboard and mouse are essential parts of the interface.

I said you couldn't change the size of a window; I forgot to add that you couldn't move it either. "Move window" messages were therefore accepted.

I've bundled the window number, message type, and message contents together. While I think I could defend that as the best way, it isn't the only way, and I accepted message sets in which these items were separated.

A number of people wanted to buffer characters from the keyboard until a line had been built up, then to send them as a block. I don't know why. It certainly isn't "as simple as possible", and doesn't replace "conventional computer terminals". It can also be a pain to use, if you want to find out just what's happening at the terminal. I only remember one suggestion the a line buffer for each window should be included under (c).

A few answers contained suggestions that people should be able to open and close windows. What would they do with the windows once they had them ? It's a <u>programme</u>'s job to open windows, not a person's.

Several people included messages like "run a programme" or "delete a file". I thought that the meaning of "message" in the context would be clear from the last item in the "specification". I assumed that you would have some idea of what sort of information travels along the wires between a terminal and a computer; I suspect that I may have been overoptimistic.

Assumptions :

The interface need not know about anything that happens in the computer - that's the operating system's job. All it needs to know about are the windows. They are therefore identified by window numbers rather than programme numbers. Windows are numbered from 1 upwards; "window number" zero will be used for messages not associated with a window, and therefore to do with the operating system.

That messages are transmitted along conventional serial communications lines. I have therefore imagined my messages as linear streams of text, but other mechanisms are possible. For example, separate lines could be provided for window number and channel : that doesn't matter, provided that the necessary information is there.

Form of the messages :

Every message must be identifiable in two ways : by the window number (which may be zero); and by the message type, so that it can be properly dealt with. The form used here is to present the window number first, then the type (expressed for convenience as a mnemonic), then any further information needed. The order of the first two items is not important, but it is sensible to place them at the beginning of the message to facilitate easy decoding.

To the interface :

Display text :	{ window number, TextOut, text }.
Construct window :	{ 0, ConsWin, window number, position, dimensions }
Destroy window :	{ 0, DestWin, window number }

From the interface :

Send text :	{ window number, CharIn, character }

(b) : ERRORS.

I think that, while the "ability" to display an error message is useful, the machinery to make it happen is equally important, and certainly counts under "facilities". After all, the question is very much about how the TUI works.

It isn't sensible to display the message in the current window : it may have nothing to do with the activity represented by that window - or, indeed, any window represented in the TUI.

I think it's clear from the context that the "additional facilities" required are those to do with the TUI, not the computer.

An alternative to my answer is to have a permanent error window built into the TUI. That will also need its own set of messages.

Another is to add a message with which the computer can ensure that a window is brought to the front, but it needs to be augmented by some machinery to force a response.

A variety of ways of acknowledging the error were suggested : pretty well anything will do, provided that it's well defined. I chose a pointer movement because I thought it would be less likely to happen accidentally. Some suggestions were to respond by pressing the space bar, or return key, but those happen quite often during ordinary text entry anyway.

Several answers included mention of interrupts. Just what was supposed to be interrupted was rarely made clear. If it's some activity in the computer, which usually seemed to be intended, then it has nothing to do with the TUI; if it's something in the TUI, then the something is one of the new facilities, and should be described.

Yes, additional facilities are required : there must be a way for the system to make a window which, irrespective of the pointer position, cannot be obscured until it has been explicitly acknowledged, and a way to signal acknowledgment. New messages :

Assumptions :

It is acceptable for the error window to appear always in the same place with the same size;

Any movement of the pointer *into* the error window is taken as an acknowledgment of the error.

To the interface :

Construct error window :	{ 0, ErWin, window number, text }

From the interface :

Error acknowledged :	{ 0, ErAck, window number }

(c) : DATA.

Comparatively well done; see notes for (d).

A number of people seem to have difficulty distinguishing between data and actions.

Assumption :

The description "as with a conventional character terminal" means what it says, so we don't need to worry about typesetting details.

Data :

Current position of the pointer.

Current active window number.

How many error windows are displayed.

List of windows, in order of "depth" :

Window number;
Edge coordinates;
Position of cursor (where to put the next character);
Text displayed (as text, or as a screen map);
This is an error window.

Notes :

We need to know how many error windows are displayed because we have to handle the pointer in a different way if there is an error window; see the description below of the action to be taken when the mouse is moved.

The list must be kept in order of depth so that it is possible to determine whether a character arriving from the computer for a window other than the current window is visible or not.

We need to know whether this is an error window so that we can send an ErAck message when the pointer enters the window.

(d) : BEHAVIOUR.

Several people, sometimes after giving good answers to (c), assumed that the data structures would magically look after themselves. My main reason for including part (c), which was easier than the others, was to draw to your attention things you'd need to consider in answering part (d). The general answer is "handle messages as in (a) and manage data structures as in (c), while making sure that error procedures as in (b) are observed".

(i) A character entered at the keyboard.

The fourth "Specification" item tells you that characters entered are displayed immediately, and not by an echo from the computer. I accepted answers using echoes largely because of an inexplicable impulse of generosity, which I can no longer quite understand, but which would now take too long to go back and undo. It's true that echoes are common, and have certain advantages - receipt of the echo tells you that the computer's listening - but both techniques are common in practice.

A surprising number of people forgot to send the character to the computer.

I didn't make explicit provision for control characters (return, backspace, etc.), because it didn't occur to me; I just assumed that "display" covered whatever had to be done. Perhaps I should have been more careful.

A positively remarkable number of people said in (c) that the TUI must know which was the active window, then said in this part that it would have to find out the active window from the position of the pointer.

```
If the current window number is not zero
    Display the character on the screen at the current cursor position of the active window;
    Append the character to the text in the current window data structure;
    Adjust the cursor position;
Construct a CharIn message, and send it to the computer.
```

Notes :

I've assumed that text entered when the pointer is not in a window is directed to the operating system, and I haven't made provision to display it on the screen because - though obviously desirable - it adds considerably to the complexity of the system, and the system is required to be "as simple as possible". It also forces a change to the rules : either the text must be displayed not in a window, or the interface must make a new window without instruction from the operating system. If pushed, I'd make the interface construct a new window number zero at the current pointer position to display the text, and dispose of the window as soon as the pointer moves outside it.

(ii) A movement of the mouse.

I didn't specify that the currently active window had to be at the front of the screen; I expected you to work it out. (If it isn't, how can you tell that text you've entered is there ?)

```
Calculate the new pointer position;
If the pointer has crossed a window boundary
        If an error window is displayed
                If the pointer is now in the error window
                        Send an ErAck message
else    If the pointer is in another window
                Set the new current window number;
                Redraw the current window;
        else    Set the current window number to zero;
Display the pointer.
```

Notes :

The current window must be redrawn so that the person using the terminal can tell when the pointer crosses its boundary. (The pointer must be moving *from* the current window to another - except perhaps when an error window is visible, but I haven't defined that case precisely anyway.)

The normal redrawing operation must be suspended if an error window is displayed, or the error window might be obscured. Apart from that, it's probable that we shall want to restore the status quo after the error has been acknowledged, and it seems silly to mess it up more than we can help.

(iii) A character arriving from a programme.

There was a mistake in the question, but I don't think it should have caused any trouble. (It did, but I don't think it should have.) The words "a programme" are relics of an earlier version of the question, and should have been replaced by "the computer". The question had originally supposed that there were several programmes in the computer, each associated with one TUI window, but this was an unnecesary complication and I (almost) eliminated it. (You may be surprised to hear that I do try to set simple questions.) I'm sorry I made the mistake; but nevertheless, as the only place for "a programme" to run <u>is</u> "the computer", I'm not inclined to see it as a serious error. In fact, of course, even specifying that the TUI is connected to a computer is unnecessary, as the TUI doesn't know where its messages come from or go to, but it's a shorthand way of defining roughly the sort of thing the TUI will be expected to do.

A few people misinterpreted "character". It doesn't mean "byte" : if we know it's a character, we must have received a message (composed of bytes) and interpreted it to discover that one of the bytes in the message represents a character.

```
Append the character to the text in the appropriate window's data structure;
If the cursor position of the appropriate window is visible on the screen
        Display the character on the screen at the cursor position;
Adjust the cursor position.
```

Alan Creak,
May 1991.