Computer Science 340

Operating Systems

Assignment 1, 1998

ANSWERS, NOTES, ETC.

This sheet presents an answer to the assignment. It is not in any sense ***the*** *answer; there are many ways to complete the requested task, and many of them are equally good. Some will perhaps not be quite as good, either because they are incomplete, or because they are incomprehensible. Material submitted which wasn't requested in the assignment sheet is unlikely to get any marks.*

Material intended primarily for the marker is in this typeface.

SUBMISSION 1.

Performing the task manually using a word processor :

There are many methods; here are three.

(I've used Word 6; equivalents using other word processor software are just as good.)

1 : Open a window for the summary file. Then open the three input files one by one in their own windows, and copy (or retype) the course number and other three data one by one into a line as specified in the question, with |!!tabulate!| separators between the items on a line. Always place the new line so that the courses are in numerical order.

2 : Concatenate all the old files into one single new file in alphabetical order of filename (Open file, Insert file, Insert file), then delete the bits you don't want, perhaps manually, or by using global replaces, perhaps with wild characters.

```
Repeat as many times as necessary :
        Replace up to and including "number!|" by newline;
        Find and replace newline to Assignments!|newline by |!!tabulate!|;
        Find and replace newline to Test!|newline by |!!tabulate!|;
        Find and replace newline to Examination!|newline by |!!tabulate!|;
        Find newline.
Delete to end of file.
```

3 : Like 2, but including getting the next file within the loop instead of doing it beforehand.

Note that there should be some instruction to produce a sorted file; this turns out to be significant for the answer in submission 4. (The requirement for sorting comes from the definition of the summary file based on the presentation in the handbook.) The instructions should not specify names for individual files.

(Strictly, my answer 2 is wrong; the assignment sheet doesn't say that the file names will always be constructed from the course numbers. It does say that the files are of the same format as those given, and people might take that to include the file name. That's fair enough, so accept answers which make that assumption.)

A method for using the shell :

I think this is the simplest and most straightforward solution. There has to be a loop somewhere to cycle through the parameters, and `foreach` is the easiest way. Answers which wouldn't work for an arbitrary number of files get at most half the marks. Note that only Unix features which appear in the examples provided are acceptable; I think that restricts people to simple Unix instructions and sed, grep, and sort. It doesn't include awk, perl, or tr.

The shell script :

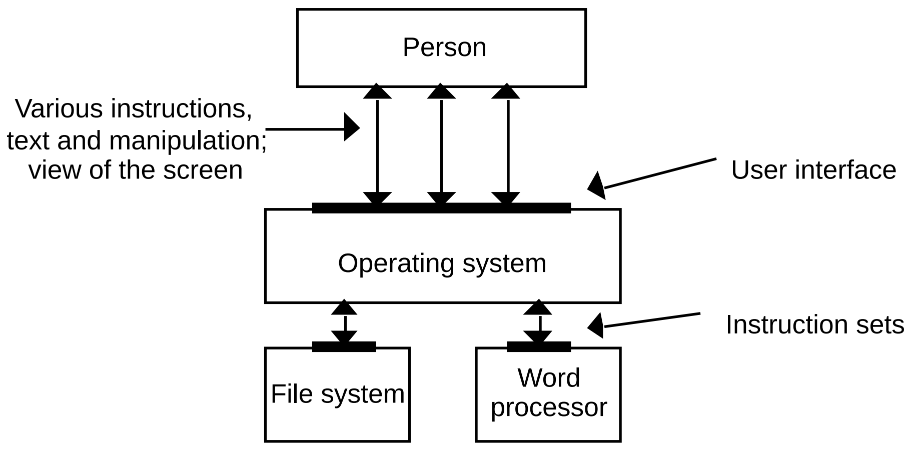
```
#
# "summary" : produces the mark allocation summary.
#
# When executed by the instruction "summary <file list>", this
# shell script converts the listed files into a summary of
# mark allocations.
#
# It is assumed that the files are all standard course files in
# the set format.

if ( $#argv == 0 ) then
     echo NO FILES ! > sumfile
else
     rm sumfile

     foreach course ( $* )

# "sed -n" suppresses the standard output.

          sed -n -f sumscript $course
          cat sumfile1 >> sumfile
     end
endif
```

The sed script :

```
# Selects the required bits from a course file, and composes a line for
# the summary file in the hold space.
# Find the line with "|!|number!|", replace with the next line, move that
# into the hold space.

/|!|number!|/ {
n
h
}

# Do the same sort of thing for the other significant features, but
# append these lines to the hold space.
/|!|examination!|/ {
n
H
}
/|!|tests!|/ {
n
H
}
/|!|assignments!|/ {
n
H
}

# At the end of the file, move the hold space to the pattern space,
# replace all the internal newline characters by |!!tabulate!|, and
# write the line to sumfile1.
$ {
g
s/\n/|!!tabulate!|/g
w sumfile1
}
```

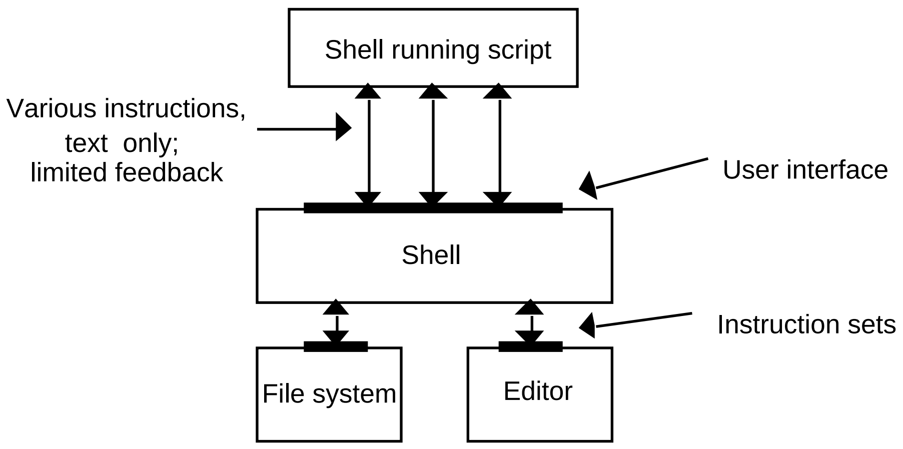
SUBMISSION 2.

For the manual method :



Some different views are possible. One is to have the person in the centre, an instruction sheet at the top, and the user interface as the means of communication between person and software. (And other changes to suit.) That's almost, but not quite, as good as the example above; it fails because people don't really get two-way interaction with instructions. It should get most of the marks. Provided that the right transactions appear in an answer and the substitutions are plausible, accept it.

For the shell script :



Similar comments; in this case, there isn't much feedback to the shell running the script, but there's a bit, so the two-way arrows are justified. Notice that there are two shells involved, which is correct.

Notice that a complete answer should have labels both for the components and for the communication channels between them. Managing the communication between things is a large part of the operating system's job.

(The significance of this sort of comparison is nicely illustrated by my experience in composing this answer. I originally had the script in the upper box, but then couldn't match any transactions with the two-way arrows. In the middleware diagram, the top box contains a programme, which is an active entity, so I had to look for a correspondingly active component in this case.)

SUBMISSION 3.

Subject	*Word processor*	*Shell script*
Overall	Easy	Harder
Instructions	Rather powerful editing instructions, apply to arbitrary ranges.	Unfamiliar instruction set; not obviously well matched to the problem. Smaller and simpler repertoire, only operate on one line at a time.
Operating	Manipulations associated with the mouse, and keyboard. Possible to manage both course files and the result file in parallel.	Only text. Strictly serial; first select the parts, then move to the result file.
Feedback	Visual feedback from the user interface; didn't have to encode just how to find the correct text.	No feedback at all - everything must be coded before execution begins.
Programme interface	Direct real-time interaction with the word processor.	Indirect action; can't watch what's happening.

I don't much care what the subjects are, provided that they identify significant differences. I do care that the answers should refer back to the labels on the diagrams, particularly those associated with the information flows. That's because it's the communication channels, not the separate programmes, that we can determine in the operating system.

SUBMISSION 4.

Performing the task manually using a word processor :

Here are modifications (if needed) for the three methods described earlier.

1 : (No change necessary.)

2 : Concatenate all the old files into one single new file (Open file, Insert file, Insert file), then delete the bits you don't want, perhaps manually, or by using global replaces, perhaps with wild characters.

```
Repeat as many times as necessary :
    Replace up to and including "number!|" by newline;
    Find and replace newline to Assignments!|newline by |!!tabulate!|;
    Find and replace newline to Test!|newline by |!!tabulate!|;
    Find and replace newline to Examination!|newline by |!!tabulate!|;
    Find newline.
Delete to end of file.
```

Finally, inspect the result and sort the lines into numerical order by cutting and pasting if necessary.

3 : (No change necessary.)

If there was no sorting instruction in submission 1, there should be one here; if the previous instruction was very specific, it might have to be changed.

A method for using the shell :

The shell script :

```
#
# "summarysort" : produces the mark allocation summary.
#
# When executed by the instruction "summary <file list>", this
# shell script converts the listed files into a summary of
# mark allocations.
#
# It is assumed that the files are all standard course files in
# the set format.

if ( $#argv == 0 ) then
     echo NO FILES ! > sumfile
else
     rm sumfile

     foreach course ( $* )

# "sed -n" suppresses the standard output.

          sed -n -f sumscript $course
          cat sumfile1 >> sumfile
     end
endif

sort sumfile > sumfilesort
```

(The only difference is the addition of the last line.)

The sed script :

```
- is unchanged.
```

SUBMISSION 5.

The summary file using the three real course files :

```
415.333|!!tabulate!|70%|!!tabulate!|0%|!!tabulate!|30%
415.340|!!tabulate!|70%|!!tabulate!|10 %|!!tabulate!|20%
415.350|!!tabulate!|70%|!!tabulate!|10%|!!tabulate!|20%
```

The summary file using the alternative 415.340 file :

```
415.333|!!tabulate!|70%|!!tabulate!|0%|!!tabulate!|30%

415.350|!!tabulate!|70%|!!tabulate!|10%|!!tabulate!|20%
```

That's what my answer produces. They don't have to be exactly like that, but I'd expect something generally similar. If there are any spectacular deviations, have a look at the script to see why.

SUBMISSION 6.

For the MANUAL method :

Yes, it can be done. Add an overall instruction : "If a file doesn't match the recommended format, call the information coordinator for help".

That works because the information flow between the person and the user interface is very detailed, and because the active entity (the person) is very good at pattern matching and interpreting instructions. It is very easy to check whether a file matches the requirements simply by checking it on the screen.

For the SHELL SCRIPT method :

Yes, it can be done, but it's a lot harder. It would be nice if the check could be done as a by-product of the existing method, but I couldn't find a straightforward way to report errors found by sed; clearly, it would be helpful if files from which required parts were missing could be reported. Instead, it is necessary to check by other means; an obvious possibility is to use grep to ensure that the required mark-up symbols are all present, and to call for help if they're not.

The difficulty is that the channels feeding back information from the editor to the script shell are very restricted. There is no overall view of the document; given that Unix provides only character streams, the best that can be done is to apply tests appropriate to such streams, and try to cope with the results.

Accept any comment which looks as though the writer had some idea of the reasons and was trying to make sense of them.

If anyone answers that it can't be done, look carefully at the reasons. If there's a sensible explanation, give up to half the marks.

The question doesn't ask for an implementation, but just to show it's possible this script works :

```
#
# "summarycheck" : produces the mark allocation summary.
#
# When executed by the instruction "summary <file list>", this
# shell script converts the listed files into a summary of
# mark allocations.
#
# If any of the files is not a standard course file in
# the set format, the file is displayed with vi.

if ( $#argv == 0 ) then
     echo NO FILES ! > sumfile
else
     rm sumfile

     foreach course ( $* )

          grep "|!|number!|" $course
          set failed = $status

          if ( $status == 0 ) then
               grep '|!|examination!|' $course
               set failed = $status
          endif

          if ( $status == 0 ) then
               grep '|!|tests!|' $course
               set failed = $status
          endif

          if ( $status == 0 ) then
               grep '|!|assignments!|' $course
               set failed = $status
          endif

          if ( $failed == 0 ) then

# "sed -n" suppresses the standard output.

               sed -n -f sumscript $course
               cat sumfile1 >> sumfile
          else
               vi $course
          endif
     end
endif
```

FINAL NOTES, AFTER THE EVENT.

I looked at quite a selection of the submissions for one reason or another, and marked a sample. Here are some comments which I hope are more about general principles than details.

GENERAL COMMENT.

My biggest surprise, and the main reason for the disarray and confusion, was the enormous variety of answers, particularly in the shell scripts. Almost all those I saw were grossly inefficient, despite my saying both "produce a reasonably efficient result" and "Aim rather for reasonable efficiency and comprehensibility" in the assignment. (Comprehensibility was not an obvious feature of most scripts I saw.)

My impression is that many people (NOT all, so this might not apply to you) didn't try to *design* a solution using the facilities available; they thrashed about, and took the first vaguely appropriate thing they saw. This is both incompetent and unprofessional, and if that's the way you did your assignment I wouldn't want to employ you. I didn't expect it of stage 3 students.

SUBMISSION 1.

Most people wrote shell scripts which would cater for an arbitrary set of file names (perhaps because I said they had to work with "summary course*"); not nearly as many wrote manual instructions which would cater for an arbitrary set of file names (even though I said "repeat the task beginning with a different set of files"). Why ? You're trying to do the same job in both cases.

A script which starts and stops 10 or 15 processes (usually for sed, grep, or cat) for each file is not very satisfactory when you can do it with two process starts - see my answer. And that's apart from the sheer silliness of running the same file repeatedly through sed, picking out one line at a time. Isn't it obvious that if you're doing that you might as well pick all the bits you want in the same run ?

SUBMISSIONS 2 AND 3.

These were coupled, though hardly anyone seemed to notice, by the definition of Submission 3 as "A comparison of the two methods, based on the differences between your diagrams". The intention was to encourage you to think about how you could present the figures in order to draw out the points you wanted to use in the comparison. I thought it was quite a clever trick, because in doing it myself I learnt quite a lot about the similarities and differences between the system.

Incidentally, I was quite surprised to find that quite a significant proportion of the answers I saw included comments wholly favouring the shell script method. That's true enough for a long run or for something you're going to repeat many times, but for a short run I'd use a word processor without a doubt - debugging the scripts takes a long time.

Alan Creak,
October, 1998.

Markers' response to 415.340 Operating Systems Assignment 1

Submission 1 :

a) Most people submitted very specific lists of instructions that specified exact file names, and would only work for the three files given. The example for submission 1 in the "What level of answer is appropriate?" section seems a bit misleading; ie encouraging people to use exact file names. Typical mark was about 6/10

b) The shell scripts were usually well written, and many obtained 10/10

Submission 2 :

Generally well done. Most people who read the question got full or close to full marks here.

A good question to test the students' understanding. Students who went wrong on this question need to study harder. Otherwise, they may have a big problem in the Exam.

Submission 3 :

Many people made only low-level comparisons between the two methods; eg "opening a file" was a popular one. Some instruction to consider more general, high-level properties of the two methods could have helped. Some people did really well here, but the typical mark was about 6/10

Submission 4 :

a) This was as per submission 1; the mistakes people made above were repeated here, the most common being to name specific files and only deal with those given as an example. Some emphasis on the word 'example' might have encouraged people to come up with a more general list of instructions. The typical mark here was usually the same as for submission 1.

b) Most people got full marks here by just adding a call to 'sort' to their script.

This question was the easiest. Students went wrong mainly because they misunderstood the question.

Submission 5 :

Fine. Only a few cases where marks were lost, generally from not reading the question.

Submission 6 :

a) This question was not handled well by most people. By far the most common problem was a very brief, vague answer. Opinion on what constituted a 'short answer' varied; the most common being about two sentences. The most common mistake appeared to be not understanding the question before attempting to answer it. Most common mark was about 5/10

b) This one was handled similarly to (a); The most significant problems were caused by not reading and understanding the question. Average mark was about 5/10

General comment :

In general, ESL students seemed to be a bit disadvantaged by this assignment. The main reason for this appeared to be the amount of reading required. Some emphasis (eg bold type) on certain parts of the assignment might have helped these students to understand exactly what was required.