

Operating Systems Exercises week 3 – Complete answers.

Linux exercises.

16) Writing a short Shell script

Use man to find out what the head command does.

Using an editor (can be the simple editor you used in 1), write a short (two or four line) shell script which applies sort and head to a test file and will give a sorted list of the first five lines in the file.

It is a good idea to begin your script with a comment that explains what your script does. Comment lines begin with a hash (#). (You might have to experiment with the keyboard to find out which key produces the # on screen.

For example, it might be the '£' key if you have got a British keyboard.)

Remember that you will have to make your script executable, using the chmod command (come out of the editor first).

Run your script by typing its name at the prompt. If it doesn't work, go back into the editor and modify it until it does.

Answer:

Note – this will be included in the portfolio so please do not show the model answer to the students.

Here is such a shell script, complete with comments in the lines beginning with a '#'.
a short C shell script

Operating Systems Week 3 exercises
#

applies sort and head to a test file to give sorted list of
the first 5 lines in the file.

sort > temp

head -n 5 < temp

18) The uniq command

The uniq command reads the input file, comparing adjacent lines.

Investigate the manual entry for uniq. There is a flag which will precede each output line with a count of the number of times each line appears in the file.

Try out uniq with the flag referred to above on a suitable file.

NB A suitable file would contain something like the following (one word per line)

letter

letter

three

three

time

sun

star

star

star

star
twenty

Answer.

The following gives an appropriate output.

```
uniq -c < uniqtestfile
```

Tutorial questions.

1) MS-DOS provided no means of concurrent processing. Discuss the major complications that concurrent processing adds to an operating system.

Answer:

The major additional complications have to do with sharing resources between several competing processes.

- A method of time sharing must be implemented to allow each of several processes to have access to the system. This method involves the preemption of processes that do not voluntarily give up the CPU (by using a system call, for instance) and the kernel being reentrant (so more than one process may be executing kernel code concurrently).
- Processes and system resources must have protections and must be protected from each other. Any given process must be limited in the amount of memory it can use and the operations it can perform on devices like disks.
- Care must be taken in the kernel to prevent deadlocks between processes, so processes aren't waiting for each other's allocated resources.

2) Describe the differences among short-term, medium-term, and long-term scheduling.

Answer:

- Short-term (CPU scheduler) - selects from processes in memory, that process which is ready to execute, and allocates the CPU to it.
- Medium-term - used especially with time-sharing systems as an intermediate scheduling level. A swapping scheme is implemented to remove partially run programs from memory and reinstate them later to continue where they left off.
- Long-term (job scheduler) - determines which jobs are brought into memory for processing. It determines whether a job will be admitted into memory and will be allowed to compete with other processes for the CPU and other resources.

There is a difference between the schedulers in terms of frequency of their execution. The short-term must select a new process quite often. Long-term is used much less often since it handles placing jobs in the system, and may wait a while for a job to finish before it admits another one.

3) A DECSYSTEM-20 computer has multiple register sets. Describe the actions of a context switch if the new context is already loaded into one of the

register sets. What else must happen if the new context is in memory rather than a register set, and all the register sets are in use?

Answer:

The CPU current-register-set pointer is changed to point to the set containing the new context, which takes very little time. If the context is in memory, one of the contexts in a register set must be chosen and moved to memory, and the new context must be loaded from memory into the set. This process takes a little more time than on systems with one set of registers, depending on how a replacement victim is selected.

4) What is a process?

Answer.

A program in execution -consisting of program text, data area, system data area.

5) What is a PCB?

Answer.

Process control block - a data structure that holds much of the system information about a process - see lecture for list of typical items..

6) List some of the queues on a typical system.

Answer.

Ready queue, I/O device queues.

7) What is the “degree of multiprogramming?”

Answer.

Number of jobs in the current mix of running/waiting jobs.

8) What is swapping?

Answer.

Copying a process out of memory onto a fast disk, to allow space for other active processes. It will be copied back into memory when there is enough space.

9) What is spawning?

Answer.

On analogy with fish parentage, spawning is the process by which one process asks the OS to create a new process on its behalf. The original process is the parent process and the new process is the child process. The OS is the midwife (among other things).

10) Give examples of programs that might be related together as producer and consumer.

Answer.

Word would be a producer and the print spooler is a consumer. Later the print spooler becomes the producer and the printer device driver is the consumer.

11) How do you determine if a buffer is full in a circular buffer (array)?

Answer.

One scheme covered in lecture - when the buffer in-index = out-index - 1.

12) The ability of a process to create a new process is an important capability, but it has dangers. Consider the consequences of allowing a user to run the process below. Assume that fork() is a system call that creates a new child process.

```
int main() {  
  
    while (true) {  
        fork();  
    }  
}
```

a) What would the consequences of allowing such a process to run?

b) Suppose that you are an operating systems designer and have been asked to build in safeguards against such processes. Assume that you have decided that it is inappropriate to reject certain processes, and that the best approach is to place certain runtime controls on them. What controls might the operating system use to detect processes like the above at runtime?

c) would the controls that you propose hinder a process' ability to create new processes?

Answer:

a) The parent process will repeatedly spawn a new process in an infinite loop, using up increasing amounts of system resources until the system runs out and grinds to a halt, unless precautions are taken to prevent this from happening.

b) OS could require that a process specify a maximum number of processes that it might spawn (or OS might impose some such restriction). The OS would then keep a count which it incremented each time it created a new process and compared it against the maximum (and decremented each time a child terminated).

Note – this will be included in the portfolio so please do not show the model answer to the students.

13) Explain the difference and the relationship between a program and a process. Is this difference important in single-process operating systems? Why or why not?

Answer.

A program is a specification of the behaviour of the computer in the form of an algorithm. A process is the representation of a program within the operating system in order to run the program, consisting of memory for code and data and system data.

The difference is still significant even if only one process runs on a system, as the difference is between a specification and the representation of a running program (process).