OPERATING SYSTEMS:
DESIGN AND IMPLEMENTATION

THIRD EDITION

PROBLEM SOLUTIONS

ANDREW S. TANENBAUM
Vrije Universiteit
Amsterdam, The Netherlands

ALBERT S. WOODHULL
Amherst, Massachusetts

PRENTICE HALL
UPPER SADDLE RIVER, NJ 07458

SOLUTIONS TO CHAPTER 1 PROBLEMS

1. An operating system must provide the users with an extended (i.e., virtual) machine, and it must manage the I/O devices and other system resources.

2. In kernel mode, every machine instruction is allowed, as is access to all the I/O devices. In user mode, many sensitive instructions are prohibited. Operating systems use these two modes to encapsulate user programs. Running user programs in user mode keeps them from doing I/O and prevents them from interfering with each other and with the kernel.

3. Multiprogramming is the rapid switching of the CPU between multiple processes in memory. It is commonly used to keep the CPU busy while one or more processes are doing I/O.

4. Input spooling is the technique of reading in jobs, for example, from cards, onto the disk, so that when the currently executing processes are finished, there will be work waiting for the CPU. Output spooling consists of first copying printable files to disk before printing them, rather than printing directly as the output is generated. Input spooling on a personal computer is not very likely, but output spooling is.

5. The prime reason for multiprogramming is to give the CPU something to do while waiting for I/O to complete. If there is no DMA, the CPU is fully occupied doing I/O, so there is nothing to be gained (at least in terms of CPU utilization) by multiprogramming. No matter how much I/O a program does, the CPU will be 100 percent busy. This of course assumes the major delay is the wait while data is copied. A CPU could do other work if the I/O were slow for other reasons (arriving on a serial line, for instance).

6. Second generation computers did not have the necessary hardware to protect the operating system from malicious user programs.

7. Choices (a), (c), and (d) should be restricted to kernel mode.

8. Personal computer systems are always interactive, often with only a single user. Mainframe systems nearly always emphasize batch or timesharing with many users. Protection is much more of an issue on mainframe systems, as is efficient use of all resources.

9. Arguments for closed source are that the company can vet the programmers, establish programming standards, and enforce a development and testing methodology. The main arguments for open source is that many more people look at the code, so there is a form of peer review and the odds of a bug slipping in are much smaller with so much more inspection.

10. The file will be executed.

11. It is often essential to have someone who can do things that are normally forbidden. For example, a user starts up a job that generates an infinite amount of output. The user then logs out and goes on a three-week vacation to London. Sooner or later the disk will fill up, and the superuser will have to manually kill the process and remove the output file. Many other such examples exist.

12. Any file can easily be named using its absolute path. Thus getting rid of working directories and relative paths would only be a minor inconvenience. The other way around is also possible, but trickier. In principle if the working directory is, say, */home/ast/projects/research/proj1* one could refer to the password file as *../../../../etc/passwd*, but it is very clumsy. This would not be a practical way of working.

13. The process table is needed to store the state of a process that is currently suspended, either ready or blocked. It is not needed in a single process system because the single process is never suspended.

14. Block special files consist of numbered blocks, each of which can be read or written independently of all the other ones. It is possible to seek to any block and start reading or writing. This is not possible with character special files.

15. The read works normally. User 2's directory entry contains a pointer to the i-node of the file, and the reference count in the i-node was incremented when user 2 linked to it. So the reference count will be nonzero and the file itself will not be removed when user 1 removes his directory entry for it. Only when all directory entries for a file have been removed will its i-node and data actually vanish.

16. No, they are not so essential. In the absence of pipes, program 1 could write its output to a file and program 2 could read the file. While this is less efficient than using a pipe between them, and uses unnecessary disk space, in most circumstances it would work adequately.

17. The display command and reponse for a stereo or camera is similar to the shell. It is a graphical command interface to the device.

18. Windows has a call spawn that creates a new process and starts a specific program in it. It is effectively a combination of fork and exec.

19. If an ordinary user could set the root directory anywhere in the tree, he could create a file *etc/passwd* in his home directory, and then make that the root directory. He could then execute some command, such as *su* or *login* that reads the password file, and trick the system into using his password file, instead of the real one.

20. The getpid, getuid, getgid, and getpgrp, calls just extract a word from the process table and return it. They will execute very quickly. They are all equally fast.

21. The system calls can collectively use 500 million instructions/sec. If each call takes 1000 instructions, up to 500,000 system calls/sec are possible while consuming only half the CPU.

22. No, unlink removes any file, whether it be for a regular file or a special file.

23. When a user program writes on a file, the data does not really go to the disk. It goes to the buffer cache. The *update* program issues SYNC calls every 30 seconds to force the dirty blocks in the cache onto the disk, in order to limit the potential damage that a system crash could cause.

24. No. What is the point of asking for a signal after a certain number of seconds if you are going to tell the system not to deliver it to you?

25. Yes it can, especially if the system is a message passing system.

26. When a user program executes a kernel-mode instruction or does something else that is not allowed in user mode, the machine *must* trap to report the attempt. The early Pentiums often ignored such instructions. This made them impossible to fully virtualize and run an arbitrary unmodified operating system in user mode.

SOLUTIONS TO CHAPTER 2 PROBLEMS

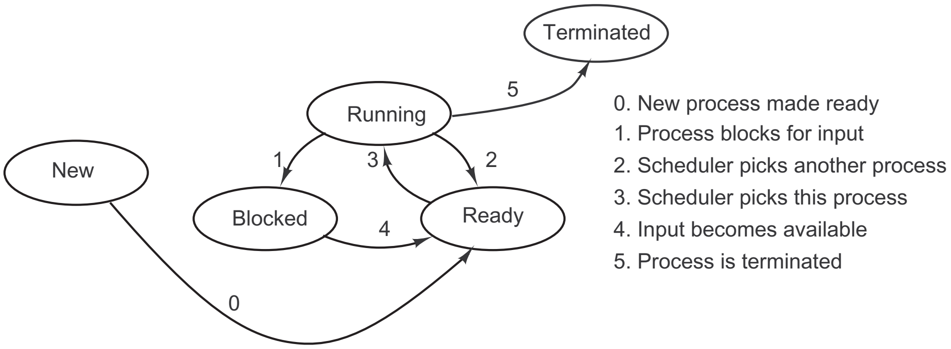
1. It is central because there is so much parallel or pseudoparallel activity—multiple user processes and I/O devices running at once. The multiprogramming model allows this activity to be described and modeled better.

2. The states are running, blocked and ready. The running state means the process has the CPU and is executing. The blocked state means that the process cannot run because it is waiting for an external event to occur, such as a message or completion of I/O. The ready state means that the process wants to run and is just waiting until the CPU is available.

3. You could have a register containing a pointer to the current process table entry. When I/O completed, the CPU would store the current machine state in the current process table entry. Then it would go to the interrupt vector for the interrupting device and fetch a pointer to another process table entry (the service procedure). This process would then be started up.

4. Generally, high level languages do not allow one the kind of access to CPU hardware that is required. For instance, an interrupt handler may be required to enable and disable the interrupt servicing a particular device, or to manipulate data within a process’ stack area. Also, interrupt service routines must execute as rapidly as possible.

5. The figure looks like this


Terminated
Running
New
Blocked
Ready
0
1
2
3
4
5
0. New process made ready
1. Process blocks for input
2. Scheduler picks another process
3. Scheduler picks this process
4. Input becomes available
5. Process is terminated

6. It would be difficult, if not impossible, to keep the file system consistent using the model in part (a) of the figure. Suppose that a client process sends a request to server process 1 to update a file. This process updates the cache entry in its memory. Shortly thereafter, another client process sends a request to server 2 to read that file. Unfortunately, if the file is also cached there, server 2, in its innocence, will return obsolete data. If the first process writes the file through to the disk after caching it, and server 2 checks the disk on every read to see if its cached copy is up-to-date, the system can be made to work, but it is precisely all these disk accesses that the caching system is trying to avoid.

7. A process is a grouping of resources: an address space, open files, signal handlers, and one or more threads. A thread is just an execution unit.

8. Each thread calls procedures on its own, so it must have its own stack for the local variables, return addresses, and so on.

9. A race condition is a situation in which two (or more) process are about to perform some action. Depending on the exact timing, one or other goes first. If one of the processes goes first, everything works, but if another one goes first, a fatal error occurs.

10. One person calls up a travel agent to find about price and availability. Then he calls the other person for approval. When he calls back, the seats are gone.

11. A possible shell script might be:

```
if [ ! –f numbers ]; echo 0 > numbers; fi
count=0
while (test $count != 200 )
do
  count=‘expr $count + 1 ‘
```