SOLUTIONS TO CHAPTER 10 PROBLEMS

1. Since assembly language is specific to each machine, a port of UNIX to a new machine required rewriting the entire code in the new machine's assembly language. On the other hand, once UNIX was written in C, only a small part of the OS (e.g. device drivers for I/O devices) had to be rewritten.

2. System call interface is tightly coupled to the OS kernel. Standardizing the system call interface would have put severe restrictions (less flexibility) on the design of the OS kernel. It would also make UNIX less portable.

3. This allows Linux to use special capabilities (like language extensions) of the gcc compiler that range from providing shortcuts and simplifications to providing the compiler with hints for optimization. The main disadvantage is that there are other popular, feature-rich C compilers like LLVM that cannot be used to compile Linux. If at some future time LLVM or some other compiler becomes better than gcc in all ways, Linux will not be able to use it. This could become a serious problem.

4. The files that will be listed are: *bonefish, quacker, seahorse,* and *weasel*.

5. It prints the number of lines of the file *xyz* that contain the string "nd" in them.

6. The pipeline is as follows:

```
head –8 z | tail –1
```

The first part selects out the first eight lines of *z* and passes them to *tail*, which just writes the last one on the screen.

7. They are separate, so standard output can be redirected without affecting standard error. In a pipeline, standard output may go to another process, but standard error still writes on the terminal.

8. Each program runs in its own process, so six new processes are started.

9. Yes. The child's memory is an exact copy of the parent's, including the stack. Thus if the environment variables were on the parent's stack, they will be on the child's stack, too.

10. Since text segments are shared, only 36 KB has to be copied. The machine can copy 80 bytes per microsec, so 36 KB takes 0.46 msec. Add another 1 msec for getting into and out of the kernel, and the whole thing takes roughly 1.46 msec.

11. Linux relies on copy on write. It gives the child pointers to the parent's address space, but marks the parent's pages write-protected. When the child attempts to write into the parent's address space, a fault occurs, and a copy of the parent's page is created and allocated into the child's address space.

© Copyright 2015 Pearson Education, Inc. All Rights Reserved.

12. Negative values allow the process to have priority over all normal processes. Users cannot be trusted with such power. It is only for the superuser and then used only in critical situations.

13. The text says that the nice value is in the range −20 to +19, so the default static priority must be 120, which it is indeed. By being nice and selecting a positive nice value, a process can request to be put under lower priority.

14. Yes. It cannot run any more, so the earlier its memory goes back on the free list, the better.

15. Signals are like hardware interrupts. One example is the alarm signal, which signals the process at a specific number of seconds in the future. Another is the floating-point exception signal, which indicates division by zero or some other error. Many other signals also exist.

16. Malicious users could wreak havoc with the system if they could send signals to arbitrary unrelated processes. Nothing would stop a user from writing a program consisting of a loop that sent a signal to the process with PID *i* for all *i* from 1 to the maximum PID. Many of these processes would be unprepared for the signal and would be killed by it. If you want to kill off your own processes, that is all right, but killing off your neighbor's processes is not acceptable.

17. It would be impossible using Linux or Windows, but the Pentium hardware does make this possible. What is needed is to use the segmentation features of the hardware, which are not supported by either Linux or Windows. The operating system could be put in one or more global segments, with protected procedure calls to make system calls instead of traps. OS/2 works this way.

18. Generally, daemons run in the background doing things like printing and sending e-mail. Since people are not usually sitting on the edge of their chairs waiting for them to finish, they are given low priority, soaking up excess CPU time not needed by interactive processes.

19. A PID must be unique. Sooner or later the counter will wrap around and go back to 0. Then it will go upward to, for example, 15. If it just happens that process 15 was started months ago, but is still running, 15 cannot be assigned to a new process. Thus after a proposed PID is chosen using the counter, a search of the process table must be made to see if the PID is already in use.

20. When the process exits, the parent will be given the exit status of its child. The PID is needed to be able to identify the parent so the exit status can be transferred to the correct process.

© Copyright 2015 Pearson Education, Inc. All Rights Reserved.

21. A page may now be shared by all three processes in this case. In general, the copy-on-write mechanism may result in several processes sharing a page. To handle this situation, the OS has to keep a reference count for each shared page. If *p1* writes on a page shared three ways, it gets a private copy while the other two processes continue to share their copy.

22. If all of the *sharing_flags* bits are set, the clone call starts a conventional thread. If all the bits are cleared,1 the call is essentially a fork.

23. Each scheduling decision requires looking up a bitmap for the active array and searching for the first set bit in the array, which can be done in constant time, dequeuing a single task from the selected queue, again a constant-time operation, or if the bitmap value is zero, swapping the values of the active and expired lists, again a constant-time operation.

24. Clock interrupts, especially at high frequency, can eat up a fair amount of CPU time. Their main function (other than keeping track of the time of day, which can be done other ways) is to determine when to preempt a long-running process. However, under normal conditions, a process makes hundreds of system calls per second, so the kernel can check on each call if the running process has run too long. To handle the case of a completely CPU-bound process running too long, a timer can be set up to go off in, say, 1 second, just in case it makes no system calls. If there is only one process, preemption is not needed, so ticklessness is just fine.

25. The program loaded from block 0 is a maximum of 512 bytes long, so it cannot be very complicated. Loading the operating system requires understanding the file system-layout in order to find and load the operating system. Different operating systems have very different file systems; it is asking too much to expect a 512-byte program to sort all this out. Instead, the block 0 loader just fetches another loader from a fixed location on the disk partition. This program can be much longer and system specific so that it can find and load the OS.

26. With shared text, 100 KB is needed for the text. Each of the three processes needs 80 KB for its data segment and 10 KB for its stack, so the total memory needed is 370 KB. Without shared text, each program needs 190 KB, so three of them need a total of 570 KB.

27. Processes sharing a file, including the current file-pointer position, can just share an open file descriptor, without having to update anything in each other's private file-descriptor tables. At the same time, another process can access the same file through a separate open-file descriptor, obtain a different file pointer, and move through the file at its own will.

© Copyright 2015 Pearson Education, Inc. All Rights Reserved.

28. The text segment cannot change, so it never has to be paged out. If its frames are needed, they can just be abandoned. The pages can always be retrieved from the file system. The data segment must not be paged back to the executable file, because it is likely that it has changed since being brought in. Paging it back would ruin the executable file. The stack segment is not even present in the executable file.

29. Two processes could map the same file into their address spaces at the same time. This gives them a way to share physical memory. Half of the shared memory could be used as a buffer from *A* to *B* and half as a buffer from *B* to *A*. To communicate, one process writes a message to its part of the shared memory, then a signal to the other one to indicate there is a message waiting for it. The reply could use the other buffer.

30. Memory address 65,536 is file byte 0, so memory address 72,000 is byte 6464.

31. Originally, four pages worth of the file were mapped: 0, 1, 2, and 3. The call succeeds and after it is done, only pages 2 and 3 are still mapped, that is, bytes 16,384 though 32,767.

32. It is possible. For example, when the stack grows beyond the bottom page, a page fault occurs and the operating system normally assigns the next-lowest page to it. However, it the stack has bumped into the data segment, the next page cannot be allocated to the stack, so the process must be terminated because it has run out of virtual address space. Also, even if there is another page available in virtual memory, the paging area of the disk might be full, making it impossible to allocate backing store for the new page, which would also terminate the process.

33. It is possible if the two blocks are not buddies. Consider the situation of Fig. 10-17(e). Two new requests come in for eight pages each. At this point the bottom 32 pages of memory are owned by four different users, each with eight pages. Now users 1 and 2 release their pages, but users 0 and 3 hold theirs. This yields a situation with eight pages used, eight pages free, eight pages free, and eight pages used. We have two adjacent blocks of equal size that cannot be merged because they are not buddies.

34. Paging to a partition allows the use of a raw device, without the overhead of using file-system data structures. To access block *n*, the operating system can calculate its disk position by just adding it to the starting block of the partition. There is no need to go through all the indirect blocks that would otherwise be needed.

35. Opening a file by a path relative to the working directory is usually more convenient for the programmer or user, since a shorter path name is needed. It is also usually much simpler and requires fewer disk accesses.

© Copyright 2015 Pearson Education, Inc. All Rights Reserved.

36. The results are as follows:

(a) The lock is granted.
(b) The lock is granted.
(c) *C* is blocked, since bytes 20 through 30 are unavailable.
(d) *A* is blocked, since bytes 20 through 25 are unavailable.
(e) *B* is blocked, since byte 8 is unavailable for exclusive locking.

At this point we now have a deadlock. None of the processes will ever be able to run again.

37. The issue arises of which process gets the lock when it becomes available. The simplest solution is to leave it undefined. This is what POSIX does because it is the easiest to implement. Another is to require the locks to be granted in the order they were requested. This approach is more work for the implementation, but prevents starvation. Still another possibility is to let processes provide a priority when asking for a lock, and use these priorities to make a choice.

38. A process will request a shared lock if it wants to read some bytes, whereas it will request an exclusive lock if it wants update some bytes. A process requesting an exclusive lock may be blocked indefinitely if there is a sequence of processes requesting shared locks. In other words, if readers always go before writers, writers could suffer from starvation.

39. The owner can read, write, and execute it, and everyone else (including the owner's group) can just read and execute it, but not write it.

40. Yes. Any block device capable of reading and writing an arbitrary block can be used to hold a file system. Even if there were no way to seek to a specific block, it is always possible to rewind the tape and then count forward to the requested block. Such a file system would not be a high-performance file system, but it would work. The author has actually done this on a PDP-11 using DECtapes and it works.

41. No. The file still has only one owner. If, for example, only the owner can write on the file, the other party cannot. Linking a file into your directory does not suddenly give you any rights you did not have before. It just creates a new path for accessing the file.

42. When the working directory is changed, using chdir, the i-node for the new working directory is fetched and kept in memory, in the i-node table. The i-node for the root directory is also there. In the user structure, pointers to both of these are maintained. When a path name has to be parsed, the first character is inspected. If it is a "/", the pointer to the root i-node is used as the starting place, otherwise the pointer to the working directory's i-node is used.

© Copyright 2015 Pearson Education, Inc. All Rights Reserved.

43. Access to the root directory's i-node does not require a disk access, so we have the following:

1. Reading the / directory to look up ''usr''.
2. Reading in the i-node for */usr*.
3. Reading the */usr* directory to look up ''ast''.
4. Reading in the i-node for */usr/ast*.
5. Reading the */usr/ast* directory to look up ''work''.
6. Reading in the i-node for */usr/ast/work*.
7. Reading the */usr/ast/work* directory to look up ''f''.
8. Reading in the i-node for */usr/ast/work/f*.

Thus, in total, eight disk accesses are needed to fetch the i-node.

44. The i-node holds 12 addresses. The single indirect block holds 256. The double indirect block leads to 65,536, and the triple indirect leads to 16,777,216, for a total of 16,843,018 blocks. This limits the maximum file size to 12 + 256 + 65,536 + 16,777,218 blocks, which is about 16 gigabytes.

45. When a file is closed, the counter of its i-node in memory is decremented. If it is greater than zero, the i-node cannot be removed from the table because the file is still open in some process. Only when the counter hits zero can the i-node be removed. Without the reference count, the system would not know when to remove the i-node from the table. Making a separate copy of the i-node each time the file was opened would not work because changes made in one copy would not be visible in the others.

46. By maintaining per-CPU runqueues, scheduling decisions can be made locally, without executing expensive synchronization mechanisms to always access, and update, a shared runqueue. Also, it is more likely that all relevant memory pages will still be in the cache1 if we schedule a thread on the same CPU where it already executed.

47. One disadvantage is security, as a loadable module may contain bugs and exploits. Another disadvantage is that the kernel virtual address space may become fragmented as more and more modules are loaded.

48. By forcing the contents of the modified file out onto the disk every 30 sec, damage done by a crash is limited to 30 sec. If *pdflush* did not run, a process might write a file, then exit with the full contents of the file still in the cache. In fact, the user might then log out and go home with the file still in the cache. An hour later the system might crash and lose the file, still only in the cache and not on disk. The next day we would not have a happy user.

49. All it has to do is set the link count to 1, since only one directory entry references the i-node.

© Copyright 2015 Pearson Education, Inc. All Rights Reserved.

50. It is generally getpid, getuid, getgid, or something like that. All they do is fetch one integer from a known place and return it. Every other call does more.

51. The file is simply removed. This is the normal way (actually, the only way) to remove a file.

52. A 1.44-MB floppy disk can hold 1440 blocks of raw data. The boot block, superblock, group descriptor block, block bitmap, and i-node bitmap of an ext2 file system each use one block. If 8192 128-byte i-nodes are created, these i-nodes will occupy another 1024 blocks, leaving only 411 blocks unused. At least one block is needed for the root directory, leaving space for 410 blocks of file data. Actually the Linux *mkfs* program is smart enough not to make more i-nodes than can possibly be used, so the inefficiency is not this bad. By default 184 i-nodes occupying 23 blocks will be created. However, because of the overhead of the ext2 file system, Linux normally uses the MINIX 1 file system on floppy disks and other small devices.

53. 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. Other such examples exist.

54. Probably someone had the file open when the professor changed the permissions. The professor should have deleted the file and then put another copy into the public directory. Also, he should use a better method for distributing files, such as a Web page, but that is beyond the scope of this exercise.

55. If, say, superuser rights are given to another user with the fsuid system call, that user can access superuser files, but will not be able to send signals, kill processes, or perform other operations which require superuser privileges.

56. (a) Most of the files are listed as zero bytes in size and contain a large amount of information. This is because these files do not exist on disk. The system retrieves the information from the actual process as needed.
(b) Most of the time and date settings reflect the current time and date. This is due to the fact that the information is just retrieved and, in many cases, it is constantly updated.
(c) Most of the files are read only. This is because the information they provide is related to the process and cannot be changed by users.

57. The activity will need to save the user's current scroll position and zoom level to be able to return showing the same part of the web page as it had been. It will also need to save any data the user had input in to form fields so they will not be lost. It does not need to save anything it had downloaded to display the web page (HTML, images, JavaScript, etc); this can always be re-fetched and

© Copyright 2015 Pearson Education, Inc. All Rights Reserved.

parsed when the activity is restored, and usually that data will still be in the browser's local on-disk cache.

58. You need to keep the device running while you are reading and storing the downloaded data, so the networking code should hold a wake lock during this time. (Extra credit: the application will also need to have requested the *internet* permission, as implied by Fig. 10-63.)

59. All of your threads need to be started after forking. The new process will only have the forking thread running in it; if you had already started other threads, they would not be in the new process, and would leave the process in a poorly defined state.

60. You know that the caller has in some way been explicitly handed a reference to your original object. However you do not know if the caller is the same process that you originally sent the object to—that process may have itself given the object to another process.

61. As more RAM is needed, the out-of-memory killer will start killing process in order from least necessary to most. From the order of operations we have been given, we can determine that we have processes running with these out-of-memory levels:

1. browser: FOREGROUND
2. email: SERVICE
3. launcher: HOME
4. camera: CACHED

The out-of-memory killer will thus start at the bottom and work up, first killing the camera process, then launcher, and finally e-mail.

© Copyright 2015 Pearson Education, Inc. All Rights Reserved.