SOLUTIONS TO CHAPTER 1 PROBLEMS

1. An operating system must provide the users with an extended machine, and it must manage the I/O devices and other system resources. To some extent, these are different functions.

2. Obviously, there are a lot of possible answers. Here are some.
Mainframe operating system: Claims processing in an insurance company.
Server operating system: Speech-to-text conversion service for Siri.
Multiprocessor operating system: Video editing and rendering.
Personal computer operating system: Word processing application.
Handheld computer operating system: Context-aware recommendation system.
Embedded operating system: Programming a DVD recorder for recording TV.
Sensor-node operating system: Monitoring temperature in a wilderness area.
Real-time operating system: Air traffic control system.
Smart-card operating system: Electronic payment.

3. In a timesharing system, multiple users can access and perform computations on a computing system simultaneously using their own terminals. Multiprogramming systems allow a user to run multiple programs simultaneously. All timesharing systems are multiprogramming systems but not all multiprogramming systems are timesharing systems since a multiprogramming system may run on a PC with only one user.

4. Empirical evidence shows that memory access exhibits the principle of locality of reference, where if one location is read then the probability of accessing nearby locations next is very high, particularly the following memory locations. So, by caching an entire cache line, the probability of a cache hit next is increased. Also, modern hardware can do a block transfer of 32 or 64 bytes into a cache line much faster than reading the same data as individual words.

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% busy. This of course assumes the major delay is the wait while data are copied. A CPU could do other work if the I/O were slow for other reasons (arriving on a serial line, for instance).

6. Access to I/O devices (e.g., a printer) is typically restricted for different users. Some users may be allowed to print as many pages as they like, some users may not be allowed to print at all, while some users may be limited to printing only a certain number of pages. These restrictions are set by system administrators based on some policies. Such policies need to be enforced so that user-level programs cannot interfere with them.

7. It is still alive. For example, Intel makes Core i3, i5, and i7 CPUs with a variety of different properties including speed and power consumption. All of these machines are architecturally compatible. They differ only in price and performance, which is the essence of the family idea.

8. A 25×80 character monochrome text screen requires a 2000-byte buffer. The 1200×900 pixel 24-bit color bitmap requires 3,240,000 bytes. In 1980 these two options would have cost $10 and $15,820, respectively. For current prices, check on how much RAM currently costs, probably pennies per MB.

9. Consider fairness and real time. Fairness requires that each process be allocated its resources in a fair way, with no process getting more than its fair share. On the other hand, real time requires that resources be allocated based on the times when different processes must complete their execution. A real-time process may get a disproportionate share of the resources.

10. Most modern CPUs provide two modes of execution: kernel mode and user mode. The CPU can execute every instruction in its instruction set and use every feature of the hardware when executing in kernel mode. However, it can execute only a subset of instructions and use only subset of features when executing in the user mode. Having two modes allows designers to run user programs in user mode and thus deny them access to critical instructions.

11. Number of heads = 255 GB / (65536*255*512) = 16
Number of platters = 16/2 = 8
The time for a read operation to complete is seek time + rotational latency + transfer time. The seek time is 11 ms, the rotational latency is 7 ms and the transfer time is 4 ms, so the average transfer takes 22 msec.

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

13. It may take 20, 25 or 30 msec to complete the execution of these programs depending on how the operating system schedules them. If *P0* and *P1* are scheduled on the same CPU and *P2* is scheduled on the other CPU, it will take 20 msec. If *P0* and *P2* are scheduled on the same CPU and *P1* is scheduled on the other CPU, it will take 25 msec. If *P1* and *P2* are scheduled on the same CPU and *P0* is scheduled on the other CPU, it will take 30 msec. If all three are on the same CPU, it will take 35 msec.

14. Every nanosecond one instruction emerges from the pipeline. This means the machine is executing 1 billion instructions per second. It does not matter at all how many stages the pipeline has. A 10-stage pipeline with 1 nsec per stage would also execute 1 billion instructions per second. All that matters is how often a finished instruction pops out the end of the pipeline.

15. Average access time =
0.95×1 nsec (word is in the cache)
$+ 0.05 \times 0.99 \times 10$ nsec (word is in RAM, but not in the cache)
$+ 0.05 \times 0.01 \times 10{,}000{,}000$ nsec (word on disk only)
$= 5001.445$ nsec
$= 5.001445$ μsec

16. Maybe. If the caller gets control back and immediately overwrites the data, when the write finally occurs, the wrong data will be written. However, if the driver first copies the data to a private buffer before returning, then the caller can be allowed to continue immediately. Another possibility is to allow the caller to continue and give it a signal when the buffer may be reused, but this is tricky and error prone.

17. A trap instruction switches the execution mode of a CPU from the user mode to the kernel mode. This instruction allows a user program to invoke functions in the operating system kernel.

18. The process table is needed to store the state of a process that is currently suspended, either ready or blocked. Modern personal computer systems have dozens of processes running even when the user is doing nothing and no programs are open. They are checking for updates, loading email, and many other things, On a UNIX system, use the *ps -a* command to see them. On a Windows system, use the task manager.

19. Mounting a file system makes any files already in the mount-point directory inaccessible, so mount points are normally empty. However, a system administrator might want to copy some of the most important files normally located in the mounted directory to the mount point so they could be found in their normal path in an emergency when the mounted device was being repaired.

20. Fork can fail if there are no free slots left in the process table (and possibly if there is no memory or swap space left). Exec can fail if the file name given does not exist or is not a valid executable file. Unlink can fail if the file to be unlinked does not exist or the calling process does not have the authority to unlink it.

21. Time multiplexing: CPU, network card, printer, keyboard.
Space multiplexing: memory, disk.
Both: display.

22. If the call fails, for example because *fd* is incorrect, it can return −1. It can also fail because the disk is full and it is not possible to write the number of bytes requested. On a correct termination, it always returns *nbytes*.

23. It contains the bytes: 1, 5, 9, 2.

24. Time to retrieve the file =
1 * 50 ms (Time to move the arm over track 50)
+ 5 ms (Time for the first sector to rotate under the head)
+ 10/200 * 1000 ms (Read 10 MB)
= 105 ms

25. 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.

26. System calls do not really have names, other than in a documentation sense. When the library procedure *read* traps to the kernel, it puts the number of the system call in a register or on the stack. This number is used to index into a table. There is really no name used anywhere. On the other hand, the name of the library procedure is very important, since that is what appears in the program.

27. This allows an executable program to be loaded in different parts of the machine's memory in different runs. Also, it enables program size to exceed the size of the machine's memory.

28. As far as program logic is concerned, it does not matter whether a call to a library procedure results in a system call. But if performance is an issue, if a task can be accomplished without a system call the program will run faster. Every system call involves overhead time in switching from the user context to the kernel context. Furthermore, on a multiuser system the operating system may schedule another process to run when a system call completes, further slowing the progress in real time of a calling process.

29. Several UNIX calls have no counterpart in the Win32 API:

Link: a Win32 program cannot refer to a file by an alternative name or see it in more than one directory. Also, attempting to create a link is a convenient way to test for and create a lock on a file.

Mount and umount: a Windows program cannot make assumptions about standard path names because on systems with multiple disk drives the drive-name part of the path may be different.

Chmod: Windows uses access control lists.

Kill: Windows programmers cannot kill a misbehaving program that is not cooperating.

30. Every system architecture has its own set of instructions that it can execute. Thus a Pentium cannot execute SPARC programs and a SPARC cannot execute Pentium programs. Also, different architectures differ in bus architecture used (such as VME, ISA, PCI, MCA, SBus, ...) as well as the word size of the CPU (usually 32 or 64 bit). Because of these differences in hardware, it is not feasible to build an operating system that is completely portable. A highly portable operating system will consist of two high-level layers—-a machine-dependent layer and a machine-independent layer. The machine-dependent layer addresses the specifics of the hardware and must be implemented separately for every architecture. This layer provides a uniform interface on which the machine-independent layer is built. The machine-independent layer has to be implemented only once. To be highly portable, the size of the machine-dependent layer must be kept as small as possible.

31. Separation of policy and mechanism allows OS designers to implement a small number of basic primitives in the kernel. These primitives are simplified, because they are not dependent of any specific policy. They can then be used to implement more complex mechanisms and policies at the user level.

32. The virtualization layer introduces increased memory usage and processor overhead as well as increased performance overhead.

33. The conversions are straightforward:

(a) A nanoyear is $10^{-9} \times 365 \times 24 \times 3600 = 31.536$ msec.
(b) 1 meter
(c) There are 2^{50} bytes, which is 1,099,511,627,776 bytes.
(d) It is 6×10^{24} kg or 6×10^{27} g.