Chapter 1

1 What are the two main functions of an operating system?
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 Nine different types of operating systems are described. Give a list of applications for each of these systems (one per operating systems type).
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 What is the difference between timesharing and multiprogramming systems?
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 To use cache memory, main memory is divided into cache lines, typically 32 or 64 bytes long. An entire cache line is cached at once. What is the advantage of caching an entire line instead of a single byte or word at a time?

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 On early computers, every byte of data read or written was handled by the CPU (i.e., there was no DMA). What implications does this have for multiprogramming?

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 Instructions related to accessing I/O devices are typically privileged instructions, that is, they can be executed in kernel mode but not in user mode. Give a reason why these instructions are privileged.

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 The family-of-computers idea was introduced in the 1960s with the IBM System/360 mainframes. Is this idea now dead as a doornail or does it live on?

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 One reason GUIs were initially slow to be adopted was the cost of the hardware needed to support them. How much video RAM is needed to support a 25-line × 80-row character monochrome text screen? How much for a 1200 × 900-pixel 24-bit color bitmap? What was the cost of this RAM at 1980 prices ($5/KB)? How much is it now?

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 There are several design goals in building an operating system, for example, resource utilization, timeliness, robustness, and so on. Give an example of two design goals that may contradict one another.

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 realtime process may get a disproportionate share of the resources.

10 What is the difference between kernel and user mode? Explain how having two distinct modes aids in designing an operating system.

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 A 255-GB disk has 65,536 cylinders with 255 sectors per track and 512 bytes per sector. How many platters and heads does this disk have? Assuming an average cylinder seek time of 11 ms, average rotational delay of 7 msec and reading rate of 100 MB/sec, calculate the average time it will take to read 400 KB from one sector.

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 Which of the following instructions should be allowed only in kernel mode?
<u>(a) Disable all interrupts.</u>
(b) Read the time-of-day clock.
<u>(c) Set the time-of-day clock.</u>

(d) Change the memory map

13 Consider a system that has two CPUs, each CPU having two threads (hyperthreading). Suppose three programs, P0, P1, and P2, are started with run times of 5, 10 and 20 msec, respectively. How long will it take to complete the execution of these programs? Assume that all three programs are 100% CPU bound, do not block during execution, and do not change CPUs once assigned.

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 A computer has a pipeline with four stages. Each stage takes the same time to do its work, namely, 1 nsec. How many instructions per second can this machine execute?

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 Consider a computer system that has cache memory, main memory (RAM) and disk, and an operating system that uses virtual memory. It takes 1 nsec to access a word from the cache, 10 nsec to access a word from the RAM, and 10 ms to access a word from the disk. If the cache hit rate is 95% and main memory hit rate (after a cache miss) is 99%, what is the average time to access a word?

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 When a user program makes a system call to read or write a disk file, it provides an indication of which file it wants, a pointer to the data buffer, and the count. Control is then transferred to the operating system, which calls the appropriate driver. Suppose that the driver starts the disk and terminates until an interrupt occurs. In the case of reading from the disk, obviously the caller will have to be blocked (because there are no data for it). What about the case of writing to the disk? Need the caller be blocked awaiting completion of the disk transfer?

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 What is a trap instruction? Explain its use in operating systems.

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 Why is the process table needed in a timesharing system? Is it also needed in personal computer systems running UNIX or Windows with a single user?

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 Is there any reason why you might want to mount a file system on a nonempty directory? If so, what is it?

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 For each of the following system calls give a condition that causes it to fail: fork, exec, and unlink.

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 What type of multiplexing (time, space, or both) can be used for sharing the following resources: CPU, memory, disk, network card, printer, keyboard, and display?

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

22 Can the
count = write(fd, buffer, nbytes);
call return any value in count other than nbytes? If so, why?

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 A file whose file descriptor is fd contains the following sequence of bytes: 3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5. The following system calls are made:
lseek(fd, 3, SEEK SET);
read(fd, &buffer, 4);
where the lseek call makes a seek to byte 3 of the file. What does buffer contain after the read has completed?

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

24 Suppose that a 10-MB file is stored on a disk on the same track (track 50) in consecutive sectors. The disk arm is currently situated over track number 100. How long will it take to retrieve this file from the disk? Assume that it takes about 1 ms to move the arm from one cylinder to the next and about 5 ms for the sector where the beginning of the file is stored to rotate under the head. Also, assume that reading occurs at a rate of 200 MB/s

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 What is the essential difference between a block special file and a character special file?

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 In the example given in Fig. 1-17, the library procedure is called read and the system call itself is called read. Is it essential that both of these have the same name? If not, which one is more important?

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 Modern operating systems decouple a process address space from the machine's physical memory. List two advantages of this design.

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 To a programmer, a system call looks like any other call to a library procedure. Is it important that a programmer know which library procedures result in system calls? Under what circumstances and why?

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 Figure 1-23 shows that a number of UNIX system calls have no Win32 API equivalents. For each of the calls listed as having no Win32 equivalent, what are the consequences for a programmer of converting a UNIX program to run under Windows?

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 A portable operating system is one that can be ported from one system architecture to another without any modification. Explain why it is infeasible to build an operating system that is completely portable. Describe two high-level layers that you will have in designing an operating system that is highly portable.

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.

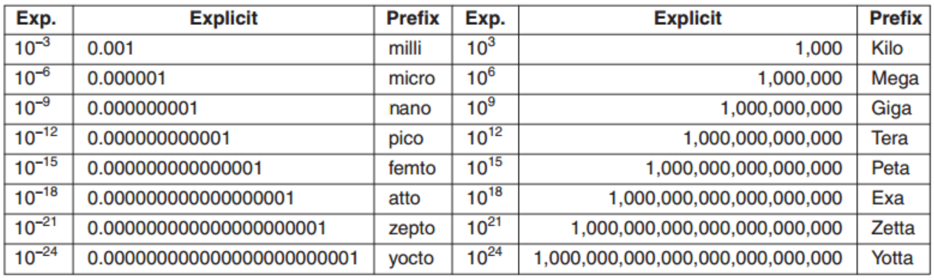
31 Explain how separation of policy and mechanism aids in building microkernel-based operating systems.

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 Virtual machines have become very popular for a variety of reasons. Nevertheless, they have some downsides. Name one.

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

33 Here are some questions for practicing unit conversions:
(a) How long is a nanoyear in seconds?
(b) Micrometers are often called microns. How long is a megamicron?
(c) How many bytes are there in a 1-PB memory?
(d) The mass of the earth is 6000 yottagrams. What is that in kilograms?



Exp.	Explicit	Prefix	Exp.	Explicit	Prefix
10^{-3}	0.001	milli	10^{3}	1,000	Kilo
10^{-6}	0.000001	micro	10^{6}	1,000,000	Mega
10^{-9}	0.000000001	nano	10^{9}	1,000,000,000	Giga
10^{-12}	0.000000000001	pico	10^{12}	1,000,000,000,000	Tera
10^{-15}	0.000000000000001	femto	10^{15}	1,000,000,000,000,000	Peta
10^{-18}	0.000000000000000001	atto	10^{18}	1,000,000,000,000,000,000	Exa
10^{-21}	0.000000000000000000001	zepto	10^{21}	1,000,000,000,000,000,000,000	Zetta
10^{-24}	0.000000000000000000000001	yocto	10^{24}	1,000,000,000,000,000,000,000,000	Yotta

The conversions are straightforward:
(a) A nanoyear is 10−9 × 365 × 24 × 3600 = 31. 536 msec.
(b) 1 meter
(c) There are 250 bytes, which is 1,099,511,627,776 bytes.
(d) It is 6 × 1024 kg or 6 × 1027 g.

34 Write a shell that is similar to Fig. 1-19 but contains enough code that it actually works so you can test it. You might also add some features such as redirection of input and output, pipes, and background jobs.

35. If you have a personal UNIX-like system (Linux, MINIX 3, FreeBSD, etc.) available that you can safely crash and reboot, write a shell script that attempts to create an unlimited number of child processes and observe what happens. Before running the experiment, type sync to the shell to flush the file system buffers to disk to avoid ruining the file system. You can also do the experiment safely in a virtual machine.
Note: Do not try this on a shared system without first getting permission from the system administrator. The consequences will be instantly obvious so you are likely to be caught and sanctions may follow.

36. Examine and try to interpret the contents of a UNIX-like or Windows directory with a tool like the UNIX od program. (Hint: How you do this will depend upon what the OS allows. One trick that may work is to create a directory on a USB stick with one operating system and then read the raw device data using a different operating system that allows such access.)

Chapter 2

1 In Fig. 2-2, three process states are shown. In theory, with three states, there could be six transitions, two out of each state. However, only four transitions are shown. Are there any circumstances in which either or both of the missing transitions might occur?

The transition from blocked to running is conceivable. Suppose that a process is blocked on I/O and the I/O finishes. If the CPU is otherwise idle, the process could go directly from blocked to running. The other missing transition, from ready to blocked, is impossible. A ready process cannot do I/O or anything else that might block it. Only a running process can block.

2. Suppose that you were to design an advanced computer architecture that did process switching in hardware, instead of having interrupts. What information would the CPU need? Describe how the hardware process switching might work.

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.

3. On all current computers, at least part of the interrupt handlers are written in assembly language. Why?

Generally, high-level languages do not allow 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.

4. When an interrupt or a system call transfers control to the operating system, a kernel stack area separate from the stack of the interrupted process is generally used. Why?

There are several reasons for using a separate stack for the kernel. Two of them are as follows. First, you do not want the operating system to crash because a poorly written user program does not allow for enough stack space. Second, if the kernel leaves stack data in a user program's memory space upon return from a system call, a malicious user might be able to use this data to find out information about other processes.

5. A computer system has enough room to hold five programs in its main memory. These programs are idle waiting for I/O half the time. What fraction of the CPU time is wasted?

1 – p^n (n – number of processes, p – waiting time) The chance that all five processes are idle is 1/32, so the CPU idle time is 1/32

6. A computer has 4 GB of RAM of which the operating system occupies 512 MB. The processes are all 256 MB (for simplicity) and have the same characteristics. If the goal is 99% CPU utilization, what is the maximum I/O wait that can be tolerated?

There is enough room for 14 processes in memory. If a process has an I/O of p, then the probability that they are all waiting for I/O is p^{14}. By equating this to 0.01, we get the equation $p^{14} = 0.\ 01$. Solving this, we get p = 0. 72, so we can tolerate processes with up to 72% I/O wait.

7. Multiple jobs can run in parallel and finish faster than if they had run sequentially. Suppose that two jobs, each needing 20 minutes of CPU time, start simultaneously. How long will the last one take to complete if they run sequentially? How long if they run in parallel? Assume 50% I/O wait.

If each job has 50% I/O wait, then it will take 40 minutes to complete in the absence of competition. If run sequentially, the second one will finish 80 minutes after the first one starts. With two jobs, the approximate CPU utilization is $1 - 0.5^2$. Thus, each one gets 0.375 CPU minute per minute of real time. To accumulate 20 minutes of CPU time, a job must run for 20/0.375 minutes, or about 53.33 minutes. Thus running sequentially the jobs finish after 80 minutes, but running in parallel they finish after 53.33 minutes.

8. Consider a multiprogrammed system with degree of 6 (i.e., six programs in memory at the same time). Assume that each process spends 40% of its time waiting for I/O. What will be the CPU utilization?

The probability that all processes are waiting for I/O is $0.\ 4^6$ which is 0.004096. Therefore, CPU utilization = 1 − 0. 004096 = 0: 995904.

9. Assume that you are trying to download a large 2-GB file from the Internet. The file is available from a set of mirror servers, each of which can deliver a subset of the file's bytes; assume that a given request specifies the starting and ending bytes of the file. Explain how you might use threads to improve the download time.

The client process can create separate threads; each thread can fetch a different part of the file from one of the mirror servers. This can help reduce downtime. Of course, there is a single network link being shared by all threads. This link can become a bottleneck as the number of threads becomes very large.

10. In the text it was stated that the model of Fig. 2-11(a) was not suited to a file server using a cache in memory. Why not? Could each process have its own cache?

It would be difficult, if not impossible, to keep the file system consistent. 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.

11. If a multithreaded process forks, a problem occurs if the child gets copies of all the parent's threads. Suppose that one of the original threads was waiting for keyboard input. Now two threads are waiting for keyboard input, one in each process. Does this problem ever occur in single-threaded processes?

No. If a single-threaded process is blocked on the keyboard, it cannot fork.

12. In Fig. 2-8, a multithreaded Web server is shown. If the only way to read from a file is the normal blocking read system call, do you think user-level threads or kernel-level threads are being used for the Web server? Why?

A worker thread will block when it has to read a Web page from the disk. If user-level threads are being used, this action will block the entire process, destroying the value of multithreading. Thus it is essential that kernel threads are used to permit some threads to block without affecting the others.

13. In the text, we described a multithreaded Web server, showing why it is better than a single-threaded server and a finite-state machine server. Are there any circumstances in which a single-threaded server might be better? Give an example.

Yes. If the server is entirely CPU bound, there is no need to have multiple threads. It just adds unnecessary complexity. As an example, consider a telephone directory assistance number (like 555-1212) for an area with 1 million people. If each (name, telephone number) record is, say, 64 characters, the entire database takes 64 megabytes and can easily be kept in the server's memory to provide fast lookup

14. In Fig. 2-12 the register set is listed as a per-thread rather than a per-process item. Why? After all, the machine has only one set of registers.

When a thread is stopped, it has values in the registers. They must be saved, just as when the process is stopped the registers must be saved. Multiprogramming threads is no different than multiprogramming processes, so each thread needs its own register save area.

15. Why would a thread ever voluntarily give up the CPU by calling thread yield? After all, since there is no periodic clock interrupt, it may never get the CPU back.

Threads in a process cooperate. They are not hostile to one another. If yielding is needed for the good of the application, then a thread will yield. After all, it is usually the same programmer who writes the code for all of them

16. Can a thread ever be preempted by a clock interrupt? If so, under what circumstances? If not, why not?

User-level threads cannot be preempted by the clock unless the whole process' quantum has been used up (although transparent clock interrupts can happen). Kernel-level threads can be preempted individually. In the latter case, if a thread runs too long, the clock will interrupt the current process and thus the current thread. The kernel is free to pick a different thread from the same process to run next if it so desires.

17. In this problem you are to compare reading a file using a single-threaded file server and a multithreaded server. It takes 12 msec to get a request for work, dispatch it, and do the rest of the necessary processing, assuming that the data needed are in the block cache. If a disk operation is needed, as is the case one-third of the time, an additional 75 msec is required, during which time the thread sleeps. How many requests/sec can the server handle if it is single threaded? If it is multithreaded?

In the single-threaded case, the cache hits take 12 msec and cache misses take 87 msec. The weighted average is 2/3 × 12 + 1/3 × 87. Thus, the mean request takes 37 msec and the server can do about 27 per second. For a multithreaded server, all the waiting for the disk is overlapped, so every request takes 12 msec, and the server can handle 83 1/3 requests per second.

18. What is the biggest advantage of implementing threads in user space? What is the biggest disadvantage?

The biggest advantage is the efficiency. No traps to the kernel are needed to switch threads. The biggest disadvantage is that if one thread blocks, the entire process blocks.

19. In Fig. 2-15 the thread creations and messages printed by the threads are interleaved at random. Is there a way to force the order to be strictly thread 1 created, thread 1 prints message, thread 1 exits, thread 2 created, thread 2 prints message, thread 2 exists, and so on? If so, how? If not, why not?

Yes, it can be done. After each call to pthread create, the main program could do a pthread join to wait until the thread just created has exited before creating the next thread.

20. In the discussion on global variables in threads, we used a procedure create global to allocate storage for a pointer to the variable, rather than the variable itself. Is this essential, or could the procedures work with the values themselves just as well?

The pointers are really necessary because the size of the global variable is unknown. It could be anything from a character to an array of floating-point numbers. If the value were stored, one would have to give the size to create global, which is all right, but what type should the second parameter of set global be, and what type should the value of read global be?

21. Consider a system in which threads are implemented entirely in user space, with the run-time system getting a clock interrupt once a second. Suppose that a clock interrupt occurs while some thread is executing in the run-time system. What problem might occur? Can you suggest a way to solve it?

It could happen that the runtime system is precisely at the point of blocking or unblocking a thread, and is busy manipulating the scheduling queues. This would be a very inopportune moment for the clock interrupt handler to begin inspecting those queues to see if it was time to do thread switching, since they might be in an inconsistent state. One solution is to set a flag when the runtime system is entered. The clock handler would see this and set its own flag, then return. When the runtime system finished, it would check the clock flag, see that a clock interrupt occurred, and now run the clock handler.

22. Suppose that an operating system does not have anything like the select system call to see in advance if it is safe to read from a file, pipe, or device, but it does allow alarm clocks to be set that interrupt blocked system calls. Is it possible to implement a threads package in user space under these conditions? Discuss.

Yes it is possible, but inefficient. A thread wanting to do a system call first sets an alarm timer, then does the call. If the call blocks, the timer returns control to the threads package. Of course, most of the time the call will not block, and the timer has to be cleared. Thus each system call that might block has to be executed as three system calls. If timers go off prematurely, all kinds of problems develop. This is not an attractive way to build a threads package.

23. Does the busy waiting solution using the turn variable (Fig. 2-23) work when the two processes are running on a shared-memory multiprocessor, that is, two CPUs sharing a common memory?

Yes, it still works, but it still is busy waiting, of course.

24. Does Peterson's solution to the mutual-exclusion problem shown in Fig. 2-24 work when process scheduling is preemptive? How about when it is nonpreemptive?

It certainly works with preemptive scheduling. In fact, it was designed for that case. When scheduling is nonpreemptive, it might fail. Consider the case in which turn is initially 0 but process 1 runs first. It will just loop forever and never release the CPU.

25. Can the priority inversion problem discussed in Sec. 2.3.4 happen with user-level threads? Why or why not?

The priority inversion problem occurs when a low-priority process is in its critical region and suddenly a high-priority process becomes ready and is scheduled. If it uses busy waiting, it will run forever. With user-level threads, it cannot happen that a low-priority thread is suddenly preempted to allow a high-priority thread run. There is no preemption. With kernel-level threads this problem can arise.

26. In Sec. 2.3.4, a situation with a high-priority process, H, and a low-priority process, L, was described, which led to H looping forever. Does the same problem occur if round-robin scheduling is used instead of priority scheduling? Discuss.

With round-robin scheduling it works. Sooner or later L will run, and eventually it will leave its critical region. The point is, with priority scheduling, L never gets to run at all; with round robin, it gets a normal time slice periodically, so it has the chance to leave its critical region.

27. In a system with threads, is there one stack per thread or one stack per process when user-level threads are used? What about when kernel-level threads are used? Explain.

Each thread calls procedures on its own, so it must have its own stack for the local variables, return addresses, and so on. This is equally true for user-level threads as for kernel-level threads.

28. When a computer is being developed, it is usually first simulated by a program that runs one instruction at a time. Even multiprocessors are simulated strictly sequentially like this. Is it possible for a race condition to occur when there are no simultaneous events like this?

Yes. The simulated computer could be multiprogrammed. For example, while process A is running, it reads out some shared variable. Then a simulated clock tick happens and process B runs. It also reads out the same variable. Then it adds 1 to the variable. When process A runs, if it also adds 1 to the variable, we have a race condition.

29. The producer-consumer problem can be extended to a system with multiple producers and consumers that write (or read) to (from) one shared buffer. Assume that each producer and consumer runs in its own thread. Will the solution presented in Fig. 2-28, using semaphores, work for this system?

Yes, it will work as is. At a given time instant, only one producer (consumer) can add (remove) an item to (from) the buffer.

30. Consider the following solution to the mutual-exclusion problem involving two processes P0 and P1. Assume that the variable turn is initialized to 0. Process P0's code is presented below.

```
/* Other code */
while (turn != 0) { } /* Do nothing and wait. */
Cr itical Section /* . . . */
tur n = 0;
/* Other code */
```

For process P1, replace 0 by 1 in above code. Determine if the solution meets all the required conditions for a correct mutual-exclusion solution.

The solution satisfies mutual exclusion since it is not possible for both processes to be in their critical section. That is, when turn is 0, P0 can execute its critical section, but not P1. Likewise, when turn is 1. However, this assumes P0 must run first. If P1 produces something and it puts it in a buffer, then while P0 can get into its critical section, it will find the buffer empty and block. Also, this solution requires strict alternation of the two processes, which is undesirable.

31. How could an operating system that can disable interrupts implement semaphores?

To do a semaphore operation, the operating system first disables interrupts. Then it reads the value of the semaphore. If it is doing a down and the semaphore is equal to zero, it puts the calling process on a list of blocked processes associated with the semaphore. If it is doing an up, it must check to see if any processes are blocked on the semaphore. If one or more processes are blocked, one of them is removed from the list of blocked processes and made runnable. When all these operations have been completed, interrupts can be enabled again.

32. Show how counting semaphores (i.e., semaphores that can hold an arbitrary value) can be implemented using only binary semaphores and ordinary machine instructions.

Associated with each counting semaphore are two binary semaphores, M, used for mutual exclusion, and B, used for blocking. Also associated with each counting semaphore is a counter that holds the number of ups minus the number of downs, and a list of processes blocked on that semaphore. To implement down, a process first gains exclusive access to the semaphores, counter, and list by doing a down on M. It then decrements the counter. If it is zero or more, it just does an up on M and exits. If M is negative, the process is put on the list of blocked processes. Then an up is done on M and a down is done on B to block the process. To implement up, first M is downed to get mutual exclusion, and then the counter is incremented. If it is more than zero, no one was blocked, so all that needs to be done is to up M. If, however, the counter is now neg ative or zero, some process must be removed from the list. Finally, an up is done on B and M in that order.

33. If a system has only two processes, does it make sense to use a barrier to synchronize them? Why or why not?

If the program operates in phases and neither process may enter the next phase until both are finished with the current phase, it makes perfect sense to use a barrier.

34. Can two threads in the same process synchronize using a kernel semaphore if the threads are implemented by the kernel? What if they are implemented in user space? Assume that no threads in any other processes have access to the semaphore. Discuss your answers.

With kernel threads, a thread can block on a semaphore and the kernel can run some other thread in the same process. Consequently, there is no problem using semaphores. With user-level threads, when one thread blocks on a semaphore, the kernel thinks the entire process is blocked and does not run it ever again. Consequently, the process fails.

35. Synchronization within monitors uses condition variables and two special operations, wait and signal. A more general form of synchronization would be to have a single primitive, waituntil, that had an arbitrary Boolean predicate as parameter. Thus, one could say, for example, waituntil $x < 0$ or $y + z < n$. The signal primitive would no longer be needed. This scheme is clearly more general than that of Hoare or Brinch Hansen, but it is not used. Why not? (Hint: Think about the implementation.)

It is very expensive to implement. Each time any variable that appears in a predicate on which some process is waiting changes, the run-time system must re-evaluate the predicate to see if the process can be unblocked. With the Hoare and Brinch Hansen monitors, processes can only be awakened on a signal primitive.

36. A fast-food restaurant has four kinds of employees:
(1) order takers, who take customers' orders;
(2) cooks, who prepare the food;
(3) packaging specialists, who stuff the food into bags; and
(4) cashiers, who give the bags to customers and take their money.
Each employee can be regarded as a communicating sequential process. What form of interprocess communication do they use? Relate this model to processes in UNIX.

The employees communicate by passing messages: orders, food, and bags in this case. In UNIX terms, the four processes are connected by pipes.

37. Suppose that we have a message-passing system using mailboxes. When sending to a full mailbox or trying to receive from an empty one, a process does not block. Instead, it gets an error code back. The process responds to the error code by just trying again, over and over, until it succeeds. Does this scheme lead to race conditions?

It does not lead to race conditions (nothing is ever lost), but it is effectively busy waiting.

38. The CDC 6600 computers could handle up to 10 I/O processes simultaneously using an interesting form of round-robin scheduling called processor sharing. A process switch occurred after each instruction, so instruction 1 came from process 1, instruction 2 came from process 2, etc. The process switching was done by special hardware, and the overhead was zero. If a process needed T sec to complete in the absence of competition, how much time would it need if processor sharing was used with n processes?

It will take nT sec.

39. Consider the following piece of C code:

```
void main( ) {
fork( );
fork( );
exit( );
}
```

How many child processes are created upon execution of this program?

Three processes are created. After the initial process forks, there are two processes running, a parent and a child. Each of them then forks, creating two additional processes. Then all the processes exit.

40. Round-robin schedulers normally maintain a list of all runnable processes, with each process occurring exactly once in the list. What would happen if a process occurred twice in the list? Can you think of any reason for allowing this?

If a process occurs multiple times in the list, it will get multiple quanta per cycle. This approach could be used to give more important processes a larger share of the CPU. But when the process blocks, all entries had better be removed from the list of runnable processes.

41. Can a measure of whether a process is likely to be CPU bound or I/O bound be determined by analyzing source code? How can this be determined at run time?

In simple cases it may be possible to see if I/O will be limiting by looking at source code. For instance a program that reads all its input files into buffers at the start will probably not be I/O bound, but a problem that reads and writes incrementally to a number of different files (such as a compiler) is likely to be I/O bound. If the operating system provides a facility such as the UNIX ps command that can tell you the amount of CPU time used by a program, you can compare this with the total time to complete execution of the program. This is, of course, most meaningful on a system where you are the only user.

42. Explain how time quantum value and context switching time affect each other, in a round-robin scheduling algorithm.

If the context switching time is large, then the time quantum value has to be proportionally large. Otherwise, the overhead of context switching can be quite high. Choosing large time quantum values can lead to an inefficient system if the typical CPU burst times are less than the time quantum. If context switching is very small or negligible, then the time quantum value can be chosen with more freedom.

43. Measurements of a certain system have shown that the average process runs for a time T before blocking on I/O. A process switch requires a time S, which is effectively wasted (overhead). For round-robin scheduling with quantum Q, give a formula for the CPU efficiency for each of the following:
(a) $Q = \infty$
(b) $Q > T$
(c) $S < Q < T$
(d) $Q = S$
(e) Q nearly 0

The CPU efficiency is the useful CPU time divided by the total CPU time. When $Q \geq T$, the basic cycle is for the process to run for T and undergo a process switch for S. Thus, (a) and (b) have an efficiency of $T/(S + T)$. When the quantum is shorter than T, each run of T will require T/Q process switches, wasting a time ST/Q. The efficiency here is then
$T/(T + ST/Q)$
which reduces to $Q/(Q + S)$, which is the answer to (c). For (d), we just substitute Q for S and find that the efficiency is 50%. Finally, for (e), as $Q \to 0$ the efficiency goes to 0.

44. Five jobs are waiting to be run. Their expected run times are 9, 6, 3, 5, and X. In what order should they be run to minimize average response time? (Your answer will depend on X.)

Shortest job first is the way to minimize average response time.
$0 < X \leq 3$: X, 3, 5, 6, 9.
$3 < X \leq 5$: 3, X, 5, 6, 9.
$5 < X \leq 6$: 3, 5, X, 6, 9.
$6 < X \leq 9$: 3, 5, 6, X, 9.
$X > 9$: 3, 5, 6, 9, X.

45. Five batch jobs. A through E, arrive at a computer center at almost the same time. They have estimated running times of 10, 6, 2, 4, and 8 minutes. Their (externally determined) priorities are 3, 5, 2, 1, and 4, respectively, with 5 being the highest priority. For each of the following scheduling algorithms, determine the mean process turnaround time. Ignore process switching overhead.
(a) Round robin.
(b) Priority scheduling.
(c) First-come, first-served (run in order 10, 6, 2, 4, 8).
(d) Shortest job first.
For (a), assume that the system is multiprogrammed, and that each job gets its fair share of the CPU. For (b) through (d), assume that only one job at a time runs, until it finishes. All jobs are completely CPU bound.

For round robin, during the first 10 minutes each job gets 1/5 of the CPU. At the end of 10 minutes, C finishes. During the next 8 minutes, each job gets 1/4 of the CPU, after which time D finishes. Then each of the three remaining jobs gets 1/3 of the CPU for 6 minutes, until B finishes, and so on. The finishing times for the five jobs are 10, 18, 24, 28, and 30, for an average of 22 minutes. For priority scheduling, B is run first. After 6 minutes it is finished. The other jobs finish at 14, 24, 26, and 30, for an average of 18.8 minutes. If the jobs run in the order A through

E, they finish at 10, 16, 18, 22, and 30, for an average of 19.2 minutes. Finally, shortest job first yields finishing times of 2, 6, 12, 20, and 30, for an average of 14 minutes.

46. A process running on CTSS needs 30 quanta to complete. How many times must it be swapped in, including the very first time (before it has run at all)?

The first time it gets 1 quantum. On succeeding runs it gets 2, 4, 8, and 15, so it must be swapped in 5 times.

47. Consider a real-time system with two voice calls of periodicity 5 msec each with CPU time per call of 1 msec, and one video stream of periodicity 33 ms with CPU time per call of 11 msec. Is this system schedulable?

Each voice call needs 200 samples of 1 msec or 200 msec. Together they use 400 msec of CPU time. The video needs 11 msec 33 1/3 times a second for a total of about 367 msec. The sum is 767 msec per second of real time so the system is schedulable.

48. For the above problem, can another video stream be added and have the system still be schedulable?

Another video stream consumes 367 msec of time per second for a total of 1134 msec per second of real time so the system is not schedulable.

49. The aging algorithm with a = 1/2 is being used to predict run times. The previous four runs, from oldest to most recent, are 40, 20, 40, and 15 msec. What is the prediction of the next time?

The sequence of predictions is 40, 30, 35, and now 25.

50. A soft real-time system has four periodic events with periods of 50, 100, 200, and 250 msec each. Suppose that the four events require 35, 20, 10, and x msec of CPU time, respectively. What is the largest value of x for which the system is schedulable?

The fraction of the CPU used is 35/50 + 20/100 + 10/200 + x/250. To be schedulable, this must be less than 1. Thus x must be less than 12.5 msec.

51. In the dining philosophers problem, let the following protocol be used: An even-numbered philosopher always picks up his left fork before picking up his right fork; an odd-numbered philosopher always picks up his right fork before picking up his left fork. Will this protocol guarantee deadlock-free operation?

Yes. There will be always at least one fork free and at least one philosopher that can obtain both forks simultaneously. Hence, there will be no deadlock. You can try this for N = 2, N = 3 and N = 4 and then generalize.

52. A real-time system needs to handle two voice calls that each run every 6 msec and consume 1 msec of CPU time per burst, plus one video at 25 frames/sec, with each frame requiring 20 msec of CPU time. Is this system schedulable?

Each voice call runs 166.67 times/second and uses up 1 msec per burst, so each voice call needs 166.67 msec per second or 333.33 msec for the two of them. The video runs 25 times a second and uses up 20 msec each time, for a total of 500 msec per second. Together they consume 833.33 msec per second, so there is time left over and the system is schedulable.

53. Consider a system in which it is desired to separate policy and mechanism for the scheduling of kernel threads. Propose a means of achieving this goal.

The kernel could schedule processes by any means it wishes, but within each process it runs threads strictly in priority order. By letting the user process set the priority of its own threads, the user controls the policy but the kernel handles the mechanism.

54. In the solution to the dining philosophers problem (Fig. 2-47), why is the state variable set to HUNGRY in the procedure take forks?

If a philosopher blocks, neighbors can later see that she is hungry by checking his state, in test, so he can be awakened when the forks are available.

55. Consider the procedure put forks in Fig. 2-47. Suppose that the variable state[i] was set to THINKING after the two calls to test, rather than before. How would this change affect the solution?

The change would mean that after a philosopher stopped eating, neither of his neighbors could be chosen next. In fact, they would never be chosen. Suppose that philosopher 2 finished eating. He would run test for philosophers 1 and 3, and neither would be started, even though both were hungry and both forks were available. Similarly, if philosopher 4 finished eating, philosopher 3 would not be started. Nothing would start him.

56. The readers and writers problem can be formulated in several ways with regard to which category of processes can be started when. Carefully describe three different variations of the problem, each one favoring (or not favoring) some category of processes. For each variation, specify what happens when a reader or a writer becomes ready to access the database, and what happens when a process is finished.

Variation 1: readers have priority. No writer may start when a reader is active. When a new reader appears, it may start immediately unless a writer is currently active. When a writer finishes, if readers are waiting, they are all started, regardless of the presence of waiting writers. Variation 2: Writers have priority. No reader may start when a writer is waiting. When the last active process finishes, a writer is started, if there is one; otherwise, all the readers (if any) are started. Variation 3: symmetric version. When a reader is active, new readers may start immediately. When a writer finishes, a new writer has priority, if one is waiting. In other words, once we have started reading, we keep reading until there are no readers left. Similarly, once we have started writing, all pending writers are allowed to run

57. Write a shell script that produces a file of sequential numbers by reading the last number in the file, adding 1 to it, and then appending it to the file. Run one instance of the script in the background and one in the foreground, each accessing the same file. How long does it take before a race condition manifests itself? What is the critical region? Modify the script to prevent the race. (Hint: use ln file file.lock to lock the data file.)

A possible shell script might be

```
if [ ! –f numbers ]; then echo 0 > numbers; fi
count = 0
while (test $count != 200 )
```

```
do
count=′expr $count + 1′
n=′tail –1 numbers′
expr $n + 1 >>numbers
done
```

Run the script twice simultaneously, by starting it once in the background (using &) and again in the foreground. Then examine the file numbers. It will probably start out looking like an orderly list of numbers, but at some point it will lose its orderliness, due to the race condition created by running two copies of the script. The race can be avoided by having each copy of the script test for and set a lock on the file before entering the critical area, and unlocking it upon leaving the critical area. This can be done like this:

```
if ln numbers numbers.lock
then
n=′tail –1 numbers′
expr $n + 1 >>numbers
rm numbers.lock
fi
```

This version will just skip a turn when the file is inaccessible. Variant solutions could put the process to sleep, do busy waiting, or count only loops in which the operation is successful.

58. Assume that you have an operating system that provides semaphores. Implement a message system. Write the procedures for sending and receiving messages.

59. Solve the dining philosophers problem using monitors instead of semaphores.

60. Suppose that a university wants to show off how politically correct it is by applying the U.S. Supreme Court's "Separate but equal is inherently unequal" doctrine to gender as well as race, ending its long-standing practice of gender-segregated bathrooms on campus. However, as a concession to tradition, it decrees that when a woman is in a bathroom, other women may enter, but no men, and vice versa. A sign with a sliding marker on the door of each bathroom indicates which of three possible states it is currently in:

- **Empty**
- **Women present**
- **Men present**

In some programming language you like, write the following procedures:
woman wants to enter, man wants to enter, woman leaves, man leaves. You may use whatever counters and synchronization techniques you like.

61. Rewrite the program of Fig. 2-23 to handle more than two processes.

62. Write a producer-consumer problem that uses threads and shares a common buffer. However, do not use semaphores or any other synchronization primitives to guard the shared data structures. Just let each thread access them when it wants to. Use sleep and wakeup to handle the full and empty conditions. See how long it takes for a fatal race condition to occur. For example, you might have the producer print a number once in a

while. Do not print more than one number every minute because the I/O could affect the race conditions.

63. A process can be put into a round-robin queue more than once to give it a higher priority.
Running multiple instances of a program each working on a different part of a data pool can have the same effect. First write a program that tests a list of numbers for primality. Then devise a method to allow multiple instances of the program to run at once in such a way that no two instances of the program will work on the same number. Can you in fact get through the list faster by running multiple copies of the program? Note that your results will depend upon what else your computer is doing; on a personal computer running only instances of this program you would not expect an improvement, but on a system with other processes, you should be able to grab a bigger share of the CPU this way.

64. The objective of this exercise is to implement a multithreaded solution to find if a given number is a perfect number. N is a perfect number if the sum of all its factors, excluding itself, is N; examples are 6 and 28. The input is an integer, N. The output is true if the number is a perfect number and false otherwise. The main program will read the numbers N and P from the command line. The main process will spawn a set of P threads. The numbers from 1 to N will be partitioned among these threads so that two threads do not work on the name number. For each number in this set, the thread will determine if the number is a factor of N. If it is, it adds the number to a shared buffer that stores factors of N. The parent process waits till all the threads complete. Use the appropriate synchronization primitive here. The parent will then determine if the input number is perfect, that is, if N is a sum of all its factors and then report accordingly. (Note: You can make the computation faster by restricting the numbers searched from 1 to the square root of N.)

65. Implement a program to count the frequency of words in a text file. The text file is partitioned into N segments. Each segment is processed by a separate thread that outputs the intermediate frequency count for its segment. The main process waits until all the threads complete; then it computes the consolidated word-frequency data based on the individual threads' output.

Chapter 3

1. The IBM 360 had a scheme of locking 2-KB blocks by assigning each one a 4-bit key and having the CPU compare the key on every memory reference to the 4-bit key in the PSW. Name two drawbacks of this scheme not mentioned in the text.

First, special hardware is needed to do the comparisons, and it must be fast, since it is used on every memory reference. Second, with 4-bit keys, only 16 programs can be in memory at once (one of which is the operating system).

2. In Fig. 3-3 the base and limit registers contain the same value, 16,384. Is this just an accident, or are they always the same? It is just an accident, why are they the same in this example?

It is an accident. The base register is 16,384 because the program happened to be loaded at address 16,384. It could have been loaded anywhere. The limit register is 16,384 because the program contains 16,384 bytes. It could have been any length. That the load address happens to exactly match the program length is pure coincidence.

3. A swapping system eliminates holes by compaction. Assuming a random distribution of many holes and many data segments and a time to read or write a 32-bit memory word of 4 nsec, about how long does it take to compact 4 GB? For simplicity, assume that word 0 is part of a hole and that the highest word in memory contains valid data.

Almost the entire memory has to be copied, which requires each word to be read and then rewritten at a different location. Reading 4 bytes takes 4 nsec, so reading 1 byte takes 1 nsec and writing it takes another 2 nsec, for a total of 2 nsec per byte compacted. This is a rate of 500,000,000 bytes/sec. To copy 4 GB (2232 bytes, which is about 4. 295 × 109 bytes), the computer needs 232/500, 000, 000 sec, which is about 859 msec. This number is slightly pessimistic because if the initial hole at the bottom of memory is k bytes, those k bytes do not need to be copied. However, if there are many holes and many data segments, the holes will be small, so k will be small and the error in the calculation will also be small.

4. Consider a swapping system in which memory consists of the following hole sizes in memory order: 10 MB, 4 MB, 20 MB, 18 MB, 7 MB, 9 MB, 12 MB, and 15 MB. Which hole is taken for successive segment requests of
(a) 12 MB
(b) 10 MB
(c) 9 MB
for first fit? Now repeat the question for best fit, worst fit, and next fit.

First fit takes 20 MB, 10 MB, 18 MB. Best fit takes 12 MB, 10 MB, and 9 MB. Worst fit takes 20 MB, 18 MB, and 15 MB. Next fit takes 20 MB, 18 MB, and 9 MB

5. What is the difference between a physical address and a virtual address?

Real memory uses physical addresses. These are the numbers that the memory chips react to on the bus. Virtual addresses are the logical addresses that refer to a process' address space. Thus a machine with a 32-bit word can generate virtual addresses up to 4 GB regardless of whether the machine has more or less memory than 4 GB.

6. For each of the following decimal virtual addresses, compute the virtual page number and offset for a 4-KB page and for an 8 KB page: 20000, 32768, 60000.

For a 4-KB page size the (page, offset) pairs are (4, 3616), (8, 0), and (14, 2656). For an 8-KB page size they are (2, 3616), (4, 0), and (7, 2656).

7. Using the page table of Fig. 3-9, give the physical address corresponding to each of the following virtual addresses:
(a) 20
(b) 4100
(c) 8300

(a) 8212. (b) 4100. (c) 24684.

8. The Intel 8086 processor did not have an MMU or support virtual memory. Nevertheless, some companies sold systems that contained an unmodified 8086 CPU and did paging.

Make an educated guess as to how they did it. (Hint: Think about the logical location of the MMU.)

They built an MMU and inserted it between the 8086 and the bus. Thus all 8086 physical addresses went into the MMU as virtual addresses. The MMU then mapped them onto physical addresses, which went to the bus.

9. What kind of hardware support is needed for a paged virtual memory to work?

There needs to be an MMU that can remap virtual pages to physical pages. Also, when a page not currently mapped is referenced, there needs to be a trap to the operating system so it can fetch the page.

10. Copy on write is an interesting idea used on server systems. Does it make any sense on a smartphone?

If the smartphone supports multiprogramming, which the iPhone, Android, and Windows phones all do, then multiple processes are supported. If a process forks and pages are shared between parent and child, copy on write definitely makes sense. A smartphone is smaller than a server, but logically it is not so different.

11. Consider the following C program:

```
int X[N];
int step = M; /* M is some predefined constant */
for (int i = 0; i < N; i += step) X[i] = X[i] + 1;
```

(a) If this program is run on a machine with a 4-KB page size and 64-entry TLB, what values of M and N will cause a TLB miss for every execution of the inner loop?
(b) Would your answer in part (a) be different if the loop were repeated many times? Explain.

For these sizes (a) M has to be at least 4096 to ensure a TLB miss for every access to an element of X. Since N affects only how many times X is accessed, any value of N will do. (b) M should still be at least 4,096 to ensure a TLB miss for every access to an element of X. But now N should be greater than 64K to thrash the TLB, that is, X should exceed 256 KB.

12. The amount of disk space that must be available for page storage is related to the maximum number of processes, n, the number of bytes in the virtual address space, v, and the number of bytes of RAM, r. Give an expression for the worst-case disk-space requirements. How realistic is this amount?

The total virtual address space for all the processes combined is nv, so this much storage is needed for pages. However, an amount r can be in RAM, so the amount of disk storage required is only nv − r. This amount is far more than is ever needed in practice because rarely will there be n processes actually running and even more rarely will all of them need the maximum allowed virtual memory.

13. If an instruction takes 1 nsec and a page fault takes an additional n nsec, give a formula for the effective instruction time if page faults occur every k instructions.

A page fault every k instructions adds an extra overhead of n/k μsec to the average, so the average instruction takes 1 + n/k nsec.

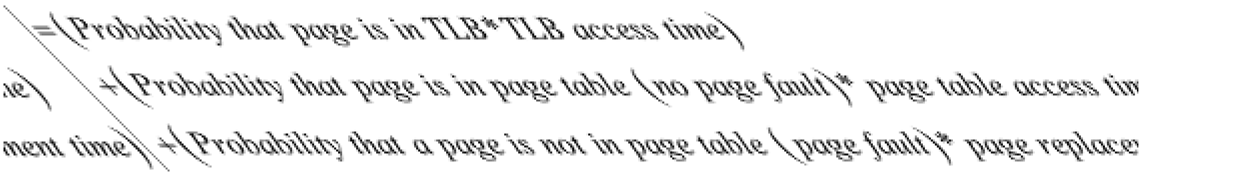
14. A machine has a 32-bit address space and an 8-KB page. The page table is entirely in hardware, with one 32-bit word per entry. When a process starts, the page table is copied to the hardware from memory, at one word every 100 nsec. If each process runs for 100 msec (including the time to load the page table), what fraction of the CPU time is devoted to loading the page tables?

The page table contains $2^{32}/2^{13}$ entries, which is 524,288. Loading the page table takes 52 msec. If a process gets 100 msec, this consists of 52 msec for loading the page table and 48 msec for running. Thus 52% of the time is spent loading page tables.

15. Suppose that a machine has 48-bit virtual addresses and 32-bit physical addresses.
(a) If pages are 4 KB, how many entries are in the page table if it has only a single level? Explain.
(b) Suppose this same system has a TLB (Translation Lookaside Buffer) with 32 entries. Furthermore, suppose that a program contains instructions that fit into one page and it sequentially reads long integer elements from an array that spans thousands of pages. How effective will the TLB be for this case?

Under these circumstances:
(a) We need one entry for each page, or 224 = 16 × 1024 × 1024 entries, since there are 36 = 48 − 12 bits in the page number field.
(b) Instruction addresses will hit 100% in the TLB. The data pages will have a 100 hit rate until the program has moved onto the next data page. Since a 4-KB page contains 1,024 long integers, there will be one TLB miss and one extra memory access for every 1,024 data references.

16. You are given the following data about a virtual memory system:
(a)The TLB can hold 1024 entries and can be accessed in 1 clock cycle (1 nsec).
(b) A page table entry can be found in 100 clock cycles or 100 nsec.
(c) The average page replacement time is 6 msec.
If page references are handled by the TLB 99% of the time, and only 0.01% lead to a page fault, what is the effective address-translation time?



=(Probability that page is in TLB*TLB access time)

ne) +(Probability that page is in page table (no page fault)* page table access tim

ment time)) +(Probability that a page is not in page table (page fault)* page replace

The chance of a hit is 0.99 for the TLB, 0.0099 for the page table, and 0.0001 for a page fault (i.e., only 1 in 10,000 references will cause a page fault). The effective address translation time in nsec is then: 0. 99 × 1 + 0. 0099 × 100 + 0. 0001 × 6 × 106 ≈ 602 clock cycles. Note that the effective address translation time is quite high because it is dominated by the page replacement time even when page faults only occur once in 10,000 references.

17. Suppose that a machine has 38-bit virtual addresses and 32-bit physical addresses.
(a) What is the main advantage of a multilevel page table over a single-level one?
(b) With a two-level page table, 16-KB pages, and 4-byte entries, how many bits should be allocated for the top-level page table field and how many for the nextlevel page table field? Explain.

Consider,
(a) A multilevel page table reduces the number of actual pages of the page table that need to be in memory because of its hierarchic structure. In fact, in a program with lots of instruction and data

locality, we only need the top-level page table (one page), one instruction page, and one data page.
(b) Allocate 12 bits for each of the three page fields. The offset field requires 14 bits to address 16 KB. That leaves 24 bits for the page fields. Since each entry is 4 bytes, one page can hold 212 page table entries and therefore requires 12 bits to index one page. So allocating 12 bits for each of the page fields will address all 238 bytes.

18. Section 3.3.4 states that the Pentium Pro extended each entry in the page table hierarchy to 64 bits but still could only address only 4 GB of memory. Explain how this statement can be true when page table entries have 64 bits.

The virtual address was changed from (PT1, PT2, Offset) to (PT1, PT2, PT3, Offset). But the virtual address still used only 32 bits. The bit configuration of a virtual address changed from (10, 10, 12) to (2, 9, 9, 12)

19. A computer with a 32-bit address uses a two-level page table. Virtual addresses are split into a 9-bit top-level page table field, an 11-bit second-level page table field, and an offset. How large are the pages and how many are there in the address space?

Twenty bits are used for the virtual page numbers, leaving 12 over for the offset. This yields a 4-KB page. Twenty bits for the virtual page implies 2^{20} pages.

20. A computer has 32-bit virtual addresses and 4-KB pages. The program and data together fit in the lowest page (0–4095) The stack fits in the highest page. How many entries are needed in the page table if traditional (one-level) paging is used? How many page table entries are needed for two-level paging, with 10 bits in each part?

For a one-level page table, there are $2^{32}/2^{12}$ or 1M pages needed. Thus the page table must have 1M entries. For two-level paging, the main page table has 1K entries, each of which points to a second page table. Only two of these are used. Thus, in total only three page table entries are needed, one in the top-level table and one in each of the lower-level tables.

21. Below is an execution trace of a program fragment for a computer with 512-byte pages. The program is located at address 1020, and its stack pointer is at 8192 (the stack grows toward 0). Give the page reference string generated by this program. Each instruction occupies 4 bytes (1 word) including immediate constants. Both instruction and data references count in the reference string.
Load word 6144 into register 0
Push register 0 onto the stack
Call a procedure at 5120, stacking the return address
Subtract the immediate constant 16 from the stack pointer
Compare the actual parameter to the immediate constant 4
Jump if equal to 5152

The code and reference string are as follows
LOAD 6144,R0 1(I), 12(D)
PUSH R0 2(I), 15(D)
CALL 5120 2(I), 15(D)
JEQ 5152 10(I)
The code (I) indicates an instruction reference, whereas (D) indicates a data reference.

22. A computer whose processes have 1024 pages in their address spaces keeps its page tables in memory. The overhead required for reading a word from the page table is 5 nsec. To reduce this overhead, the computer has a TLB, which holds 32 (virtual page, physical page frame) pairs, and can do a lookup in 1 nsec. What hit rate is needed to reduce the mean overhead to 2 nsec?

The effective instruction time is 1h + 5(1 − h), where h is the hit rate. If we equate this formula with 2 and solve for h, we find that h must be at least 0.75.

23. How can the associative memory device needed for a TLB be implemented in hardware, and what are the implications of such a design for expandability?

An associative memory essentially compares a key to the contents of multiple registers simultaneously. For each register there must be a set of comparators that compare each bit in the register contents to the key being searched for. The number of gates (or transistors) needed to implement such a device is a linear function of the number of registers, so expanding the design gets expensive linearly

24. A machine has 48-bit virtual addresses and 32-bit physical addresses. Pages are 8 KB. How many entries are needed for a single-level linear page table?

With 8-KB pages and a 48-bit virtual address space, the number of virtual pages is $2^{48}/2^{13}$, which is 2^{35} (about 34 billion).

25. A computer with an 8-KB page, a 256-KB main memory, and a 64-GB virtual address space uses an inverted page table to implement its virtual memory. How big should the hash table be to ensure a mean hash chain length of less than 1? Assume that the hashtable size is a power of two.

The main memory has $2^{28}/2^{13}$ = 32,768 pages. A 32K hash table will have a mean chain length of 1. To get under 1, we have to go to the next size, 65,536 entries. Spreading 32,768 entries over 65,536 table slots will give a mean chain length of 0.5, which ensures fast lookup.

26. A student in a compiler design course proposes to the professor a project of writing a compiler that will produce a list of page references that can be used to implement the optimal page replacement algorithm. Is this possible? Why or why not? Is there anything that could be done to improve paging efficiency at run time?

This is probably not possible except for the unusual and not very useful case of a program whose course of execution is completely predictable at compilation time. If a compiler collects information about the locations in the code of calls to procedures, this information might be used at link time to rearrange the object code so that procedures were located close to the code that calls them. This would make it more likely that a procedure would be on the same page as the calling code. Of course this would not help much for procedures called from many places in the program.

27. Suppose that the virtual page reference stream contains repetitions of long sequences of page references followed occasionally by a random page reference. For example, the sequence: 0, 1, ... , 511, 431, 0, 1, ... , 511, 332, 0, 1, ... consists of repetitions of the sequence 0, 1, ... , 511 followed by a random reference to pages 431 and 332.
(a) Why will the standard replacement algorithms (LRU, FIFO, clock) not be effective in handling this workload for a page allocation that is less than the sequence length?

(b) If this program were allocated 500 page frames, describe a page replacement approach that would perform much better than the LRU, FIFO, or clock algorithms.

Under these circumstances
(a) Every reference will page fault unless the number of page frames is 512, the length of the entire sequence.
(b) If there are 500 frames, map pages 0–498 to fixed frames and vary only one frame.

28. If FIFO page replacement is used with four page frames and eight pages, how many page faults will occur with the reference string 0172327103 if the four frames are initially empty? Now repeat this problem for LRU.

The page frames for FIFO are as follows:
x0172333300
xx017222233
xxx01777722
xxxx0111177

The page frames for LRU are as follows:
x0172327103
xx017232710
xxx01773271
xxxx0111327

FIFO yields six page faults; LRU yields seven.

29. Consider the page sequence of Fig. 3-15(b). Suppose that the R bits for the pages B through A are 11011011, respectively. Which page will second chance remove?

The first page with a 0 bit will be chosen, in this case D.

30. A small computer on a smart card has four page frames. At the first clock tick, the R bits are 0111 (page 0 is 0, the rest are 1). At subsequent clock ticks, the values are 1011, 1010, 1101, 0010, 1010, 1100, and 0001. If the aging algorithm is used with an 8-bit counter, give the values of the four counters after the last tick.

The counters are
Page 0: 01101110
Page 1: 01001001
Page 2: 00110111
Page 3: 10001011

31. Give a simple example of a page reference sequence where the first page selected for replacement will be different for the clock and LRU page replacement algorithms. Assume that a process is allocated 3=three frames, and the reference string contains page numbers from the set 0, 1, 2, 3.

The sequence: 0, 1, 2, 1, 2, 0, 3. In LRU, page 1 will be replaced by page 3. In clock, page 1 will be replaced, since all pages will be marked and the cursor is at page 0.

32. In the WSClock algorithm of Fig. 3-20(c), the hand points to a page with R = 0. If τ = 400, will this page be removed? What about if τ = 1000?

The age of the page is 2204 − 1213 = 991. If τ = 400, it is definitely out of the working set and was not recently referenced so it will be evicted. The τ = 1000 situation is different. Now the page falls within the working set (barely), so it is not removed.

33. Suppose that the WSClock page replacement algorithm uses a τ of two ticks, and the system state is the following:

Page	**Time stamp**	**V**	**R**	**M**
0	**6**	**1**	**0**	**1**
1	**9**	**1**	**1**	**0**
2	**9**	**1**	**1**	**1**
3	**7**	**1**	**0**	**0**
4	**4**	**0**	**0**	**0**

where the three flag bits V, R, and M stand for Valid, Referenced, and Modified, respectively.

(a) If a clock interrupt occurs at tick 10, show the contents of the new table entries. Explain. (You can omit entries that are unchanged.)

(b) Suppose that instead of a clock interrupt, a page fault occurs at tick 10 due to a read request to page 4. Show the contents of the new table entries. Explain. (You can omit entries that are unchanged.)

Consider,

(a) For every R bit that is set, set the time-stamp value to 10 and clear all R bits. You could also change the (0,1) R-M entries to (0,0*). So the entries for pages 1 and 2 will change to:

Page	Time stamp	V	R	M
0	6	1	0	0*
1	10	1	0	0
2	10	1	0	1

(b) Evict page 3 (R = 0 and M = 0) and load page 4:

Page	Time stamp	V	R	M	Notes
0	6	1	0	1	
1	9	1	1	0	
2	9	1	1	1	
3	7	0	0	0	Changed from 7 (1,0,0)
4	10	1	1	0	Changed from 4 (0,0,0)

34. A student has claimed that “in the abstract, the basic page replacement algorithms (FIFO, LRU, optimal) are identical except for the attribute used for selecting the page to be replaced.”

(a) What is that attribute for the FIFO algorithm? LRU algorithm? Optimal algorithm?

(b) Give the generic algorithm for these page replacement algorithms.

Consider,

(a) The attributes are: (FIFO) load time; (LRU) latest reference time; and (Optimal) nearest reference time in the future.

(b) There is the labeling algorithm and the replacement algorithm. The labeling algorithm labels each page with the attribute given in part a. The replacement algorithm evicts the page with the smallest label.

35. How long does it take to load a 64-KB program from a disk whose average seek time is 5 msec, whose rotation time is 5 msec, and whose tracks hold 1 MB
(a) for a 2-KB page size?
(b) for a 4-KB page size?
The pages are spread randomly around the disk and the number of cylinders is so large that the chance of two pages being on the same cylinder is negligible.

The seek plus rotational latency is 10 msec. For 2-KB pages, the transfer time is about 0.009766 msec, for a total of about 10.009766 msec. Loading 32 of these pages will take about 320.21 msec. For 4-KB pages, the transfer time is doubled to about 0.01953 msec, so the total time per page is 10.01953 msec. Loading 16 of these pages takes about 160.3125 msec. With such fast disks, all that matters is reducing the number of transfers (or putting the pages consecutively on the disk).

36. A computer has four page frames. The time of loading, time of last access, and the R and M bits for each page are as shown below (the times are in clock ticks):

Page	**Loaded**	**Last ref.**	**R**	**M**
0	**126**	**280**	**1**	**0**
1	**230**	**265**	**0**	**1**
2	**140**	**270**	**0**	**0**
3	**110**	**285**	**1**	**1**

(a) Which page will NRU replace?
(b) Which page will FIFO replace?
(c) Which page will LRU replace?
(d) Which page will second chance replace?

NRU removes page 2. FIFO removes page 3. LRU removes page 1. Second chance removes page 2.

37. Suppose that two processes A and B share a page that is not in memory. If process A faults on the shared page, the page table entry for process A must be updated once the page is read into memory.
(a) Under what conditions should the page table update for process B be delayed even though the handling of process A's page fault will bring the shared page into memory? Explain.
(b) What is the potential cost of delaying the page table update?

Sharing pages brings up all kinds of complications and options:
(a) The page table update should be delayed for process B if it will never access the shared page or if it accesses it when the page has been swapped out again. Unfortunately, in the general case, we do not know what process B will do in the future.
(b) The cost is that this lazy page fault handling can incur more page faults. The overhead of each page fault plays an important role in determining if this strategy is more efficient. (Aside: This cost is similar to that faced by the copy-on-write strategy for supporting some UNIX fork system call implementations.)

38. Consider the following two-dimensional array:
int X[64][64];
Suppose that a system has four page frames and each frame is 128 words (an integer occupies one word). Programs that manipulate the X array fit into exactly one page and always occupy page 0. The data are swapped in and out of the other three frames. The X array is stored in row-major order (i.e., X[0][1] follows X[0][0] in memory). Which of the

two code fragments shown below will generate the lowest number of page faults? Explain and compute the total number of page faults.
Fragment A
for (int j = 0; j < 64; j++)
for (int i = 0; i < 64; i++) X[i][j] = 0;
Fragment B
for (int i = 0; i < 64; i++)
for (int j = 0; j < 64; j++) X[i][j] = 0;

Fragment B since the code has more spatial locality than Fragment A. The inner loop causes only one page fault for every other iteration of the outer loop. (There will be only 32 page faults.) [Aside (Fragment A): Since a frame is 128 words, one row of the X array occupies half of a page (i.e., 64 words). The entire array fits into 64 × 32/128 = 16 frames. The inner loop of the code steps through consecutive rows of X for a given column. Thus, ev ery other reference to X[i][j] will cause a page fault. The total number of page faults will be 64 × 64/2 = 2, 048].

39. You have been hired by a cloud computing company that deploys thousands of servers at each of its data centers. They have recently heard that it would be worthwhile to handle a page fault at server A by reading the page from the RAM memory of some other server rather than its local disk drive.
(a) How could that be done?
(b) Under what conditions would the approach be worthwhile? Be feasible?

It can certainly be done.
(a) The approach has similarities to using flash memory as a paging device in smartphones except now the virtual swap area is a RAM located on a remote server. All of the software infrastructure for the virtual swap area would have to be developed.
(b) The approach might be worthwhile by noting that the access time of disk drives is in the millisecond range while the access time of RAM via a network connection is in the microsecond range if the software overhead is not too high. But the approach might make sense only if there is lots of idle RAM in the server farm. And then, there is also the issue of reliability. Since RAM is volatile, the virtual swap area would be lost if the remote server went down.

40. One of the first timesharing machines, the DEC PDP-1, had a (core) memory of 4K 18-bit words. It held one process at a time in its memory. When the scheduler decided to run another process, the process in memory was written to a paging drum, with 4K 18-bit words around the circumference of the drum. The drum could start writing (or reading) at any word, rather than only at word 0. Why do you suppose this drum was chosen?

The PDP-1 paging drum had the advantage of no rotational latency. This saved half a rotation each time memory was written to the drum.

41. A computer provides each process with 65,536 bytes of address space divided into pages of 4096 bytes each. A particular program has a text size of 32,768 bytes, a data size of 16,386 bytes, and a stack size of 15,870 bytes. Will this program fit in the machine's address space? Suppose that instead of 4096 bytes, the page size were 512 bytes, would it then fit? Each page must contain either text, data, or stack, not a mixture of two or three of them.

The text is eight pages, the data are five pages, and the stack is four pages. The program does not fit because it needs 17 4096-byte pages. With a 512-byte page, the situation is different. Here the text is 64 pages, the data are 33 pages, and the stack is 31 pages, for a total of 128 512-byte pages, which fits. With the small page size it is OK, but not with the large one.

42. It has been observed that the number of instructions executed between page faults is directly proportional to the number of page frames allocated to a program. If the available memory is doubled, the mean interval between page faults is also doubled. Suppose that a normal instruction takes 1 microsec, but if a page fault occurs, it takes 2001 µsec (i.e., 2 msec) to handle the fault. If a program takes 60 sec to run, during which time it gets 15,000 page faults, how long would it take to run if twice as much memory were available?

The program is getting 15,000 page faults, each of which uses 2 msec of extra processing time. Together, the page fault overhead is 30 sec. This means that of the 60 sec used, half was spent on page fault overhead, and half on running the program. If we run the program with twice as much memory, we get half as many memory page faults, and only 15 sec of page fault overhead, so the total run time will be 45 sec.

43. A group of operating system designers for the Frugal Computer Company are thinking about ways to reduce the amount of backing store needed in their new operating system. The head guru has just suggested not bothering to save the program text in the swap area at all, but just page it in directly from the binary file whenever it is needed.
Under what conditions, if any, does this idea work for the program text? Under what conditions, if any, does it work for the data?

It works for the program if the program cannot be modified. It works for the data if the data cannot be modified. However, it is common that the program cannot be modified and extremely rare that the data cannot be modified. If the data area on the binary file were overwritten with updated pages, the next time the program was started, it would not have the original data.

44. A machine-language instruction to load a 32-bit word into a register contains the 32-bit address of the word to be loaded. What is the maximum number of page faults this instruction can cause?

The instruction could lie astride a page boundary, causing two page faults just to fetch the instruction. The word fetched could also span a page boundary, generating two more faults, for a total of four. If words must be aligned in memory, the data word can cause only one fault, but an instruction to load a 32-bit word at address 4094 on a machine with a 4-KB page is legal on some machines (including the x86).

45. Explain the difference between internal fragmentation and external fragmentation. Which one occurs in paging systems? Which one occurs in systems using pure segmentation?

Internal fragmentation occurs when the last allocation unit is not full. External fragmentation occurs when space is wasted between two allocation units. In a paging system, the wasted space in the last page is lost to internal fragmentation. In a pure segmentation system, some space is invariably lost between the segments. This is due to external fragmentation.

46. When segmentation and paging are both being used, as in MULTICS, first the segment descriptor must be looked up, then the page descriptor. Does the TLB also work this way, with two levels of lookup?

No. The search key uses both the segment number and the virtual page number, so the exact page can be found in a single match.

47. We consider a program which has the two segments shown below consisting of instructions in segment 0, and read/write data in segment 1. Segment 0 has read/execute protection, and segment 1 has just read/write protection. The memory system is a demand-paged virtual memory system with virtual addresses that have a 4-bit page number, and a 10-bit offset. The page tables and protection are as follows (all numbers in the table are in decimal):
Segment 0 Segment 1
Read/Execute Read/Write
Virtual Page # Page frame # Virtual Page # Page frame #
0 2 0 On Disk
1 On Disk 1 14
2 11 2 9
3 536
4 On Disk 4 On Disk
5 On Disk 5 13
6 468
7 3 7 12
For each of the following cases, either give the real (actual) memory address which results from dynamic address translation or identify the type of fault which occurs (either page or protection fault).
(a) Fetch from segment 1, page 1, offset 3
(b) Store into segment 0, page 0, offset 16
(c) Fetch from segment 1, page 4, offset 28
(d) Jump to location in segment 1, page 3, offset 32

Here are the results:

	Address	Fault?
(a)	(14, 3)	No (or 0xD3 or 1110 0011)
(b)	NA	Protection fault: Write to read/execute segment
(c)	NA	Page fault
(d)	NA	Protection fault: Jump to read/write segment

48. Can you think of any situations where supporting virtual memory would be a bad idea, and what would be gained by not having to support virtual memory? Explain.

General virtual memory support is not needed when the memory requirements of all applications are well known and controlled. Some examples are smart cards, special-purpose processors (e.g., network processors), and embedded processors. In these situations, we should always consider the possibility of using more real memory. If the operating system did not have to support virtual memory, the code would be much simpler and smaller. On the other hand, some ideas from virtual memory may still be profitably exploited, although with different design requirements. For example, program/thread isolation might be paging to flash memory.

49. Virtual memory provides a mechanism for isolating one process from another. What memory management difficulties would be involved in allowing two operating systems to run concurrently? How might these difficulties be addressed?

This question addresses one aspect of virtual machine support. Recent attempts include Denali, Xen, and VMware. The fundamental hurdle is how to achieve near-native performance, that is, as if the executing operating system had memory to itself. The problem is how to quickly switch to another operating system and therefore how to deal with the TLB. Typically, you want to give some number of TLB entries to each kernel and ensure that each kernel operates within its proper virtual memory context. But sometimes the hardware (e.g., some Intel architectures) wants to handle TLB misses without knowledge of what you are trying to do. So, you need to either handle the TLB miss in software or provide hardware support for tagging TLB entries with a context ID.

50. Plot a histogram and calculate the mean and median of the sizes of executable binary files on a computer to which you have access. On a Windows system, look at all .exe and .dll files; on a UNIX system look at all executable files in /bin, /usr/bin, and /local/bin that are not scripts (or use the file utility to find all executables). Determine the optimal page size for this computer just considering the code (not data). Consider internal fragmentation and page table size, making some reasonable assumption about the size of a page table entry. Assume that all programs are equally likely to be run and thus should be weighted equally.

51. Write a program that simulates a paging system using the aging algorithm. The number of page frames is a parameter. The sequence of page references should be read from a file. For a given input file, plot the number of page faults per 1000 memory references as a function of the number of page frames available.

52. Write a program that simulates a toy paging system that uses the WSClock algorithm. The system is a toy in that we will assume there are no write references (not very

CHAP. 3 PROBLEMS 261

realistic), and process termination and creation are ignored (eternal life). The inputs will be:

- **The reclamation age threshhold**
- **The clock interrupt interval expressed as number of memory references**
- **A file containing the sequence of page references**

(a) Describe the basic data structures and algorithms in your implementation.

(b) Show that your simulation behaves as expected for a simple (but nontrivial) input example.

(c) Plot the number of page faults and working set size per 1000 memory references.

(d) Explain what is needed to extend the program to handle a page reference stream that also includes writes.

53. Write a program that demonstrates the effect of TLB misses on the effective memory access time by measuring the per-access time it takes to stride through a large array.

(a) Explain the main concepts behind the program, and describe what you expect the output to show for some practical virtual memory architecture.

(b) Run the program on some computer and explain how well the data fit your expectations.

(c) Repeat part (b) but for an older computer with a different architecture and explain any major differences in the output.

54. Write a program that will demonstrate the difference between using a local page replacement

policy and a global one for the simple case of two processes. You will need a routine that can generate a page reference string based on a statistical model. This model has N states numbered from 0 to N − 1 representing each of the possible page references and a probability pi associated with each state i representing the chance that the next reference is to the same page. Otherwise, the next page reference will be one of the other pages with equal probability.

(a) Demonstrate that the page reference string-generation routine behaves properly for some small N.

(b) Compute the page fault rate for a small example in which there is one process and a fixed number of page frames. Explain why the behavior is correct.
(c) Repeat part (b) with two processes with independent page reference sequences and twice as many page frames as in part (b).
(d) Repeat part (c) but using a global policy instead of a local one. Also, contrast the per-process page fault rate with that of the local policy approach.
55. Write a program that can be used to compare the effectiveness of adding a tag field to TLB entries when control is toggled between two programs. The tag field is used to effectively
label each entry with the process id. Note that a nontagged TLB can be simulated
by requiring that all TLB entries have the same tag at any one time. The inputs
will be:
• The number of TLB entries available
• The clock interrupt interval expressed as number of memory references
• A file containing a sequence of (process, page references) entries
• The cost to update one TLB entry
262 MEMORY MANAGEMENT CHAP. 3
(a) Describe the basic data structures and algorithms in your implementation.
b) Show that your simulation behaves as expected for a simple (but nontrivial) input example.
(c) Plot the number of TLB updates per 1000 references.

Chapter 4

1. Give five different path names for the file /etc/passwd. (Hint: Think about the directory entries "." and "..".)

/etc/passwd
/./etc/passwd
/././etc/passwd
/./././etc/passwd
/etc/../etc/passwd
/etc/../etc/../etc/passwd
/etc/../etc/../etc/../etc/passwd
/etc/../etc/../etc/../etc/../etc/passwd

2. In Windows, when a user double clicks on a file listed by Windows Explorer, a program is run and given that file as a parameter. List two different ways the operating system could know which program to run.

The Windows way is to use the file extension. Each extension corresponds to a file type and to some program that handles that type. Another way is to remember which program created the file and run that program. The Macintosh works this way.

3. In early UNIX systems, executable files (a.out files) began with a very specific magic number, not one chosen at random. These files began with a header, followed by the text and data segments. Why do you think a very specific number was chosen for executable files, whereas other file types had a more-or-less random magic number as the first word?

These systems loaded the program directly in memory and began executing at word 0, which was the magic number. To avoid trying to execute the header as code, the magic number was a BRANCH instruction with a target address just above the header. In this way it was possible to read the binary file directly into the new process' address space and run it at 0, without even knowing how big the header was.

4. Is the open system call in UNIX absolutely essential? What would the consequences be of not having it?

To start with, if there were no open, on every read it would be necessary to specify the name of the file to be opened. The system would then have to fetch the i-node for it, although that could be cached. One issue that quickly arises is when to flush the i-node back to disk. It could time out, however. It would be a bit clumsy, but it might work.

5. Systems that support sequential files always have an operation to rewind files. Do systems that support random-access files need this, too?

No. If you want to read the file again, just randomly access byte 0.

6. Some operating systems provide a system call rename to give a file a new name. Is there any difference at all between using this call to rename a file and just copying the file to a new file with the new name, followed by deleting the old one?

Yes. The rename call does not change the creation time or the time of last modification, but creating a new file causes it to get the current time as both the creation time and the time of last modification. Also, if the disk is nearly full, the copy might fail.

7. In some systems it is possible to map part of a file into memory. What restrictions must such systems impose? How is this partial mapping implemented?

The mapped portion of the file must start at a page boundary and be an integral number of pages in length. Each mapped page uses the file itself as backing store. Unmapped memory uses a scratch file or partition as backing store.

8. A simple operating system supports only a single directory but allows it to have arbitrarily many files with arbitrarily long file names. Can something approximating a hierarchical file system be simulated? How?

Use file names such as /usr/ast/file. While it looks like a hierarchical path name, it is really just a single name containing embedded slashes.

9. In UNIX and Windows, random access is done by having a special system call that moves the "current position" pointer associated with a file to a given byte in the file. Propose an alternative way to do random access without having this system call.

One way is to add an extra parameter to the read system call that tells what address to read from. In effect, every read then has a potential for doing a seek within the file. The disadvantages of this scheme are (1) an extra parameter in every read call, and (2) requiring the user to keep track of where the file pointer is.

10. Consider the directory tree of Fig. 4-8. If /usr/jim is the working directory, what is the absolute path name for the file whose relative path name is ../ast/x?

The dotdot component moves the search to /usr, so ../ast puts it in /usr/ast. Thus ../ast/x is the same as /usr/ast/x.

11. Contiguous allocation of files leads to disk fragmentation, as mentioned in the text, because some space in the last disk block will be wasted in files whose length is not an integral number of blocks. Is this internal fragmentation or external fragmentation? Make an analogy with something discussed in the previous chapter.

Since the wasted storage is between the allocation units (files), not inside them, this is external fragmentation. It is precisely analogous to the external fragmentation of main memory that occurs with a swapping system or a system using pure segmentation.

12. Describe the effects of a corrupted data block for a given file for: (a) contiguous, (b) linked, and (c) indexed (or table based).

If a data block gets corrupted in a contiguous allocation system, then only this block is affected; the remainder of the file’s blocks can be read. In the case of linked allocation, the corrupted block cannot be read; also, location data about all blocks starting from this corrupted block is lost. In case of indexed allocation, only the corrupted data block is affected.

13. One way to use contiguous allocation of the disk and not suffer from holes is to compact the disk every time a file is removed. Since all files are contiguous, copying a file requires a seek and rotational delay to read the file, followed by the transfer at full speed. Writing the file back requires the same work. Assuming a seek time of 5 msec, a rotational delay of 4 msec, a transfer rate of 80 MB/sec, and an average file size of 8 KB, how long does it take to read a file into main memory and then write it back to the disk at a new location? Using these numbers, how long would it take to compact half of a 16-GB disk?

It takes 9 msec to start the transfer. To read 2^{13} bytes at a transfer rate of 80 MB/sec requires 0.0977 msec, for a total of 9.0977 msec. Writing it back takes another 9.0977 msec. Thus, copying a file takes 18.1954 msec. To compact half of a 16-GB disk would involve copying 8 GB of storage, which is 2^{20} files. At 18.1954 msec per file, this takes 19,079.25 sec, which is 5.3 hours. Clearly, compacting the disk after every file removal is not a great idea.

14. In light of the answer to the previous question, does compacting the disk ever make any sense?

If done right, yes. While compacting, each file should be organized so that all of its blocks are consecutive, for fast access. Windows has a program that defragments and reorganizes the disk. Users are encouraged to run it periodically to improve system performance. But given how long it takes, running once a month might be a good frequency.

15. Some digital consumer devices need to store data, for example as files. Name a modern device that requires file storage and for which contiguous allocation would be a fine idea.

A digital still camera records some number of photographs in sequence on a nonvolatile storage medium (e.g., flash memory). When the camera is reset, the medium is emptied. Thereafter, pictures are recorded one at a time in sequence until the medium is full, at which time they are uploaded to a hard disk. For this application, a contiguous file system inside the camera (e.g., on the picture storage medium) is ideal.

16. Consider the i-node shown in Fig. 4-13. If it contains 10 direct addresses and these were 8 bytes each and all disk blocks were 1024 KB, what would the largest possible file be?

The indirect block can hold 128 disk addresses. Together with the 10 direct disk addresses, the maximum file has 138 blocks. Since each block is 1 KB, the largest file is 138 KB.

17. For a given class, the student records are stored in a file. The records are randomly accessed and updated. Assume that each student's record is of fixed size. Which of the three allocation schemes (contiguous, linked and table/indexed) will be most appropriate?

For random access, table/indexed and contiguous will be both appropriate, while linked allocation is not as it typically requires multiple disk reads for a given record.

18. Consider a file whose size varies between 4 KB and 4 MB during its lifetime. Which of the three allocation schemes (contiguous, linked and table/indexed) will be most appropriate?

Since the file size changes a lot, contiguous allocation will be inefficient requiring reallocation of disk space as the file grows in size and compaction of free blocks as the file shrinks in size. Both linked and table/indexed allocation will be efficient; between the two, table/indexed allocation will be more efficient for random-access scenarios.

19. It has been suggested that efficiency could be improved and disk space saved by storing the data of a short file within the i-node. For the i-node of Fig. 4-13, how many bytes of data could be stored inside the i-node?

There must be a way to signal that the address-block pointers hold data, rather than pointers. If there is a bit left over somewhere among the attributes, it can be used. This leaves all nine pointers for data. If the pointers are k bytes each, the stored file could be up to 9k bytes long. If no bit is left over among the attributes, the first disk address can hold an invalid address to mark the following bytes as data rather than pointers. In that case, the maximum file is 8k bytes.

20. Two computer science students, Carolyn and Elinor, are having a discussion about inodes.
Carolyn maintains that memories have gotten so large and so cheap that when a file is opened, it is simpler and faster just to fetch a new copy of the i-node into the inode table, rather than search the entire table to see if it is already there. Elinor disagrees.
Who is right?

Elinor is right. Having two copies of the i-node in the table at the same time is a disaster, unless both are read only. The worst case is when both are being updated simultaneously. When the i-nodes are written back to the disk, whichever one gets written last will erase the changes made by the other one, and disk blocks will be lost.

21. Name one advantage of hard links over symbolic links and one advantage of symbolic links over hard links.

Hard links do not require any extra disk space, just a counter in the i-node to keep track of how many there are. Symbolic links need space to store the name of the file pointed to. Symbolic links can point to files on other machines, even over the Internet. Hard links are restricted to pointing to files within their own partition.

22. Explain how hard links and soft links differ with respective to i-node allocations.

A single i-node is pointed to by all directory entries of hard links for a given file. In the case of soft-links, a new i-node is created for the soft link and this i-node essentially points to the original file being linked

23. Consider a 4-TB disk that uses 4-KB blocks and the free-list method. How many block addresses can be stored in one block?

The number of blocks on the disk = 4 TB / 4 KB = 2^{30}. Thus, each block address can be 32 bits (4 bytes), the nearest power of 2. Thus, each block can store 4 KB / 4 = 1024 addresses.

24. Free disk space can be kept track of using a free list or a bitmap. Disk addresses require D bits. For a disk with B blocks, F of which are free, state the condition under which the free list uses less space than the bitmap. For D having the value 16 bits, express your answer as a percentage of the disk space that must be free.

The bitmap requires B bits. The free list requires DF bits. The free list requires fewer bits if DF < B. Alternatively, the free list is shorter if F/B < 1/D, where F/B is the fraction of blocks free. For 16-bit disk addresses D = 16 => F/B < 1/16 => F/B < 6.25%, the free list is shorter if 6% or less of the disk is free.

25. The beginning of a free-space bitmap looks like this after the disk partition is first formatted:
1000 0000 0000 0000 (the first block is used by the root directory). The system always searches for free blocks starting at the lowest-numbered block, so after writing file A, which uses six blocks, the bitmap looks like this: 1111 1110 0000 0000.
Show the bitmap after each of the following additional actions:
(a) File B is written, using five blocks.
(b) File A is deleted.
(c) File C is written, using eight blocks.
(d) File B is deleted.

The beginning of the bitmap looks like:
(a) After writing file B: 1111 1111 1111 0000
(b) After deleting file A: 1000 0001 1111 0000
(c) After writing file C: 1111 1111 1111 1100
(d) After deleting file B: 1111 1110 0000 1100

26. What would happen if the bitmap or free list containing the information about free disk blocks was completely lost due to a crash? Is there any way to recover from this disaster, or is it bye-bye disk? Discuss your answers for UNIX and the FAT -16 file system separately.

It is not a serious problem at all. Repair is straightforward; it just takes time. The recovery algorithm is to make a list of all the blocks in all the files and take the complement as the new free list. In UNIX this can be done by scanning all the i-nodes. In the FAT file system, the problem cannot occur because there is no free list. But even if there were, all that would have to be done to recover it is to scan the FAT looking for free entries.

27. Oliver Owl's night job at the university computing center is to change the tapes used for overnight data backups. While waiting for each tape to complete, he works on writing his thesis that proves Shakespeare's plays were written by extraterrestrial visitors. His text

processor runs on the system being backed up since that is the only one they have. Is there a problem with this arrangement?

Ollie's thesis may not be backed up as reliably as he might wish. A backup program may pass over a file that is currently open for writing, as the state of the data in such a file may be indeterminate.

28. We discussed making incremental dumps in some detail in the text. In Windows it is easy to tell when to dump a file because every file has an archive bit. This bit is missing in UNIX. How do UNIX backup programs know which files to dump?

They must keep track of the time of the last dump in a file on disk. At every dump, an entry is appended to this file. At dump time, the file is read and the time of the last entry noted. Any file changed since that time is dumped.

29. Suppose that file 21 in Fig. 4-25 was not modified since the last dump. In what way would the four bitmaps of Fig. 4-26 be different?

In (a) and (b), 21 would not be marked. In (c), there would be no change. In (d), 21 would not be marked.

30. It has been suggested that the first part of each UNIX file be kept in the same disk block as its i-node. What good would this do?

Many UNIX files are short. If the entire file fits in the same block as the inode, only one disk access would be needed to read the file, instead of two, as is presently the case. Even for longer files there would be a gain, since one fewer disk accesses would be needed.

31. Consider Fig. 4-27. Is it possible that for some particular block number the counters in both lists have the value 2? How should this problem be corrected?

It should not happen, but due to a bug somewhere it could happen. It means that some block occurs in two files and also twice in the free list. The first step in repairing the error is to remove both copies from the free list. Next a free block has to be acquired and the contents of the sick block copied there. Finally, the occurrence of the block in one of the files should be changed to refer to the newly acquired copy of the block. At this point the system is once again consistent.

32. The performance of a file system depends upon the cache hit rate (fraction of blocks found in the cache). If it takes 1 msec to satisfy a request from the cache, but 40 msec to satisfy a request if a disk read is needed, give a formula for the mean time required to satisfy a request if the hit rate is h. Plot this function for values of h varying from 0 to 1.0.

$1h + 40(1-h)$. The plot is just a straight line.

33. For an external USB hard drive attached to a computer, which is more suitable: a write-through cache or a block cache?

In this case, it is better to use a write-through cache since it writes data to the hard drive while also updating the cache. This will ensure that the updated file is always on the external hard drive even if the user accidentally removes the hard drive before disk sync is completed.

34. Consider an application where students' records are stored in a file. The application takes a student ID as input and subsequently reads, updates, and writes the corresponding student record; this is repeated till the application quits. Would the "block readahead" technique be useful here?

The block read-ahead technique reads blocks sequentially, ahead of their use, in order to improve performance. In this application, the records will likely not be accessed sequentially since the user can input any student ID at a given instant. Thus, the read-ahead technique will not be very useful in this scenario.

35. Consider a disk that has 10 data blocks starting from block 14 through 23. Let there be 2 files on the disk: f1 and f2. The directory structure lists that the first data blocks of f1 and f2 are respectively 22 and 16. Given the FAT table entries as below, what are the data blocks allotted to f1 and f2?
(14,18); (15,17); (16,23); (17,21); (18,20); (19,15); (20, −1); (21, −1); (22,19); (23,14).
In the above notation, (x, y) indicates that the value stored in table entry x points to data block y.

The blocks allotted to f1 are: 22, 19, 15, 17, 21.
The blocks allotted to f2 are: 16, 23, 14, 18, 20.

36. Consider the idea behind Fig. 4-21, but now for a disk with a mean seek time of 6 msec, a rotational rate of 15,000 rpm, and 1,048,576 bytes per track. What are the data rates for block sizes of 1 KB, 2 KB, and 4 KB, respectively?

At 15,000 rpm, the disk takes 4 msec to go around once (1/15000 = 60 / 15000 = 0.004 sec). The average access time (in msec) to read k bytes is then 6 + 2 + (k/1 048 576) × 4. For blocks of 1 KB, 2 KB, and 4 KB, the access times are about 6.0039 msec, 6.0078 msec, and 6.0156 msec, respectively (hardly any different). These give rates of about 170.556 KB/sec, 340.890 KB/sec, and 680.896 KB/sec, respectively.

37. A certain file system uses 4-KB disk blocks. The median file size is 1 KB. If all files were exactly 1 KB, what fraction of the disk space would be wasted? Do you think the wastage for a real file system will be higher than this number or lower than it? Explain your answer.

If all files were 1 KB, then each 4-KB block would contain one file and 3 KB of wasted space. Trying to put two files in a block is not allowed because the unit used to keep track of data is the block, not the semiblock. This leads to 75% wasted space. In practice, every file system has large files as well as many small ones, and these files use the disk much more efficiently. For example, a 32,769-byte file would use 9 disk blocks for storage, given a space efficiency of 32,769/36,864, which is about 89%.

38. Given a disk-block size of 4 KB and block-pointer address value of 4 bytes, what is the largest file size (in bytes) that can be accessed using 10 direct addresses and one indirect block?

The indirect block can hold 1024 addresses. Added to the 10 direct addresses, there are 1034 addresses in all. Since each one points to a 4-KB disk block, the largest file is 4,235,264 bytes

39. Files in MS-DOS have to compete for space in the FAT-16 table in memory. If one file uses k entries, that is k entries that are not available to any other file, what constraint does this place on the total length of all files combined?

It constrains the sum of all the file lengths to being no larger than the disk. This is not a very serious constraint. If the files were collectively larger than the disk, there would be no place to store all of them on the disk.

40. A UNIX file system has 4-KB blocks and 4-byte disk addresses. What is the maximum file size if i-nodes contain 10 direct entries, and one single, double, and triple indirect entry each?

The i-node holds 10 pointers. The single indirect block holds 1024 pointers. The double indirect block is good for 1024^2 pointers. The triple indirect block is good for 1024^3 pointers. Adding these up, we get a maximum file size of 1,074,791,434 blocks, which is about 16.06 GB.

41. How many disk operations are needed to fetch the i-node for a file with the path name /usr/ast/courses/os/handout.t? Assume that the i-node for the root directory is in memory, but nothing else along the path is in memory. Also assume that all directories fit in one disk block.

The following disk reads are needed:
directory for /
i-node for /usr
directory for /usr
i-node for /usr/ast
directory for /usr/ast
i-node for /usr/ast/courses
directory for /usr/ast/courses
i-node for /usr/ast/courses/os
directory for /usr/ast/courses/os
i-node for /usr/ast/courses/os/handout.t

42. In many UNIX systems, the i-nodes are kept at the start of the disk. An alternative design is to allocate an i-node when a file is created and put the i-node at the start of the first block of the file. Discuss the pros and cons of this alternative.

Some pros are as follows. First, no disk space is wasted on unused i-nodes. Second, it is not possible to run out of i-nodes. Third, less disk movement is needed since the i-node and the initial data can be read in one operation. Some cons are as follows. First, directory entries will now need a 32-bit disk address instead of a 16-bit i-node number. Second, an entire disk will be used even for files which contain no data (empty files, device files). Third, file-system integrity checks will be slower because of the need to read an entire block for each i-node and because i-nodes will be scattered all over the disk. Fourth, files whose size has been carefully designed to fit the block size will no longer fit the block size due to the i-node, messing up performance.

43. Write a program that reverses the bytes of a file, so that the last byte is now first and the first byte is now last. It must work with an arbitrarily long file, but try to make it reasonably efficient.

44. Write a program that starts at a given directory and descends the file tree from that point recording the sizes of all the files it finds. When it is all done, it should print a histogram of the file sizes using a bin width specified as a parameter (e.g., with 1024, file sizes of 0 to 1023 go in one bin, 1024 to 2047 go in the next bin, etc.).

45. Write a program that scans all directories in a UNIX file system and finds and locates all i-nodes with a hard link count of two or more. For each such file, it lists together all

file names that point to the file.
46. Write a new version of the UNIX ls program. This version takes as an argument one or more directory names and for each directory lists all the files in that directory, one line per file. Each field should be formatted in a reasonable way given its type. List only the first disk address, if any.
47. Implement a program to measure the impact of application-level buffer sizes on read time. This involves writing to and reading from a large file (say, 2 GB). Vary the application buffer size (say, from 64 bytes to 4 KB). Use timing measurement routines (such as gettimeofday and getitimer on UNIX) to measure the time taken for different buffer sizes. Analyze the results and report your findings: does buffer size make a difference to the overall write time and per-write time?
48. Implement a simulated file system that will be fully contained in a single regular file stored on the disk. This disk file will contain directories, i-nodes, free-block information, file data blocks, etc. Choose appropriate algorithms for maintaining free-block information and for allocating data blocks (contiguous, indexed, linked). Your program will accept system commands from the user to create/delete directories, create/delete/open files, read/write from/to a selected file, and to list directory contents.

Chapter 5

1. Advances in chip technology have made it possible to put an entire controller, including all the bus access logic, on an inexpensive chip. How does that affect the model of Fig. 1-6?

In the figure, we see controllers and devices as separate units. The reason is to allow a controller to handle multiple devices, and thus eliminate the need for having a controller per device. If controllers become almost free, then it will be simpler just to build the controller into the device itself. This design will also allow multiple transfers in parallel and thus give better performance.

2. Given the speeds listed in Fig. 5-1, is it possible to scan documents from a scanner and transmit them over an 802.11g network at full speed? Defend your answer.

Easy. The scanner puts out 400 KB/sec maximum. The wireless network runs at 6.75 MB/sec, so there is no problem at all.

3. Figure 5-3(b) shows one way of having memory-mapped I/O even in the presence of separate buses for memory and I/O devices, namely, to first try the memory bus and if that fails try the I/O bus. A clever computer science student has thought of an improvement on this idea: try both in parallel, to speed up the process of accessing I/O devices. What do you think of this idea?

It is not a good idea. The memory bus is surely faster than the I/O bus, otherwise why bother with it? Consider what happens with a normal memory request. The memory bus finishes first, but the I/O bus is still busy. If the CPU waits until the I/O bus finishes, it has reduced memory performance to that of the I/O bus. If it just tries the memory bus for the second reference, it will fail if this one is an I/O device reference. If there were some way to instantaneously abort the previous I/O bus reference to try the second one, the improvement might work, but there is never such an option. All in all, it is a bad idea.

4. Explain the tradeoffs between precise and imprecise interrupts on a superscalar machine.

An advantage of precise interrupts is simplicity of code in the operating system since the machine state is well defined. On the other hand, in imprecise interrupts, OS writers have to figure out what instructions have been partially executed and up to what point. However, precise interrupts increase complexity of chip design and chip area, which may result in slower CPU.

5. A DMA controller has five channels. The controller is capable of requesting a 32-bit word every 40 nsec. A response takes equally long. How fast does the bus have to be to avoid being a bottleneck?

Each bus transaction has a request and a response, each taking 50 nsec, or 100 nsec per bus transaction. This gives 10 million bus transactions/sec. If each one is good for 4 bytes, the bus has to handle 40 MB/sec. The fact that these transactions may be sprayed over five I/O devices in round-robin fashion is irrelevant. A bus transaction takes 100 nsec, regardless of whether consecutive requests are to the same device or different devices, so the number of channels the DMA controller has does not matter. The bus does not know or care

6. Suppose that a system uses DMA for data transfer from disk controller to main memory. Further assume that it takes t1 nsec on average to acquire the bus and t2 nsec to transfer one word over the bus (t1 >> t2). After the CPU has programmed the DMA controller, how long will it take to transfer 1000 words from the disk controller to main memory, if
(a) word-at-a-time mode is used,
(b) burst mode is used? Assume that commanding the disk controller requires acquiring the bus to send one word and acknowledging a transfer also requires acquiring the bus to send one word.

. (a) Word-at-a-time mode: 1000 × [(t1 + t2) + (t1 + t2) + (t1 + t2)] Where the first term is for acquiring the bus and sending the command to the disk controller, the second term is for transferring the word, and the third term is for the acknowledgement. All in all, a total of 3000 × (t1 + t2) nsec.

(b) Burst mode: (t1 + t2) + t1 + 1000 times t2 + (t1 + t2) where the first term is for acquiring the bus and sending the command to the disk controller, the second term is for the disk controller to acquire the bus, the third term is for the burst transfer, and the fourth term is for acquiring the bus and doing the acknowledgement. All in all, a total of 3t1 + 1002t2.

7. One mode that some DMA controllers use is to have the device controller send the word to the DMA controller, which then issues a second bus request to write to memory. How can this mode be used to perform memory to memory copy? Discuss any advantage or disadvantage of using this method instead of using the CPU to perform memory to memory copy.

Memory to memory copy can be performed by first issuing a read command that will transfer the word from memory to DMA controller and then issuing a write to memory to transfer the word from the DMA controller to a different address in memory. This method has the advantage that the CPU can do other useful work in parallel. The disadvantage is that this memory to memory copy is likely to be slow since DMA controller is much slower than CPU and the data transfer takes place over system bus as opposed to the dedicated CPU-memory bus.

8. Suppose that a computer can read or write a memory word in 5 nsec. Also suppose that when an interrupt occurs, all 32 CPU registers, plus the program counter and PSW are

pushed onto the stack. What is the maximum number of interrupts per second this machine can process?

An interrupt requires pushing 34 words onto the stack. Returning from the interrupt requires fetching 34 words from the stack. This overhead alone is 340 nsec. Thus the maximum number of interrupts per second is no more than about 2.94 million, assuming no work for each interrupt.

9. CPU architects know that operating system writers hate imprecise interrupts. One way to please the OS folks is for the CPU to stop issuing new instructions when an interrupt is signaled, but allow all the instructions currently being executed to finish, then force the interrupt. Does this approach have any disadvantages? Explain your answer.

The execution rate of a modern CPU is determined by the number of instructions that finish per second and has little to do with how long an instruction takes. If a CPU can finish 1 billion instructions/sec it is a 1000 MIPS machine, even if an instruction takes 30 nsec. Thus there is generally little attempt to make instructions finish quickly. Holding the interrupt until the last instruction currently executing finishes may increase the latency of interrupts appreciably. Furthermore, some administration is required to get this right.

10. In Fig. 5-9(b), the interrupt is not acknowledged until after the next character has been output to the printer. Could it have equally well been acknowledged right at the start of the interrupt service procedure? If so, give one reason for doing it at the end, as in the text. If not, why not?

It could have been done at the start. A reason for doing it at the end is that the code of the interrupt service procedure is very short. By first outputting another character and then acknowledging the interrupt, if another interrupt happens immediately, the printer will be working during the interrupt, making it print slightly faster. A disadvantage of this approach is slightly longer dead time when other interrupts may be disabled.

11. A computer has a three-stage pipeline as shown in Fig. 1-7(a). On each clock cycle, one new instruction is fetched from memory at the address pointed to by the PC and put into the pipeline and the PC advanced. Each instruction occupies exactly one memory word. The instructions already in the pipeline are each advanced one stage. When an interrupt occurs, the current PC is pushed onto the stack, and the PC is set to the address of the interrupt handler. Then the pipeline is shifted right one stage and the first instruction of the interrupt handler is fetched into the pipeline. Does this machine have precise interrupts? Defend your answer.

Yes. The stacked PC points to the first instruction not fetched. All instructions before that have been executed and the instruction pointed to and its successors have not been executed. This is the condition for precise interrupts. Precise interrupts are not hard to achieve on machine with a single pipeline. The trouble comes in when instructions are executed out of order, which is not the case here.

12. A typical printed page of text contains 50 lines of 80 characters each. Imagine that a certain printer can print 6 pages per minute and that the time to write a character to the printer's output register is so short it can be ignored. Does it make sense to run this printer using interrupt-driven I/O if each character printed requires an interrupt that takes 50 µsec all-in to service?

The printer prints 50 × 80 × 6 = 24, 000 characters/min, which is 400 characters/sec. Each character uses 50 µsec of CPU time for the interrupt, so collectively in each second the interrupt overhead is 20 msec. Using interrupt-driven I/O, the remaining 980 msec of time is available for other work. In other words, the interrupt overhead costs only 2% of the CPU, which will hardly affect the running program at all.

13. Explain how an OS can facilitate installation of a new device without any need for recompiling the OS.

UNIX does it as follows. There is a table indexed by device number, with each table entry being a C struct containing pointers to the functions for opening, closing, reading, and writing and a few other things from the device. To install a new device, a new entry has to be made in this table and the pointers filled in, often to the newly loaded device driver.

14. In which of the four I/O software layers is each of the following done.
(a) Computing the track, sector, and head for a disk read.
(b) Writing commands to the device registers.
(c) Checking to see if the user is permitted to use the device.
(d) Converting binary integers to ASCII for printing.

(a) Device driver.
(b) Device driver.
(c) Device-independent software.
(d) User-level software.

15. A local area network is used as follows. The user issues a system call to write data packets to the network. The operating system then copies the data to a kernel buffer. Then it copies the data to the network controller board. When all the bytes are safely inside the controller, they are sent over the network at a rate of 10 megabits/sec. The receiving network controller stores each bit a microsecond after it is sent. When the last bit arrives, the destination CPU is interrupted, and the kernel copies the newly arrived packet to a kernel buffer to inspect it. Once it has figured out which user the packet is for, the kernel copies the data to the user space. If we assume that each interrupt and its associated processing takes 1 msec, that packets are 1024 bytes (ignore the headers), and that copying a byte takes 1 µsec, what is the maximum rate at which one process can pump data to another? Assume that the sender is blocked until the work is finished at the receiving side and an acknowledgement comes back. For simplicity, assume that the time to get the acknowledgement back is so small it can be ignored.

A packet must be copied four times during this process, which takes 4.1 msec. There are also two interrupts, which account for 2 msec. Finally, the transmission time is 0.83 msec, for a total of 6.93 msec per 1024 bytes. The maximum data rate is thus 147,763 bytes/sec, or about 12% of the nominal 10 megabit/sec network capacity. (If we include protocol overhead, the figures get even worse.)

16. Why are output files for the printer normally spooled on disk before being printed?

If the printer were assigned as soon as the output appeared, a process could tie up the printer by printing a few characters and then going to sleep for a week.

17. How much cylinder skew is needed for a 7200-RPM disk with a track-to-track seek time of 1 msec? The disk has 200 sectors of 512 bytes each on each track.

The disk rotates at 120 RPS, so 1 rotation takes 1000/120 msec. With 200 sectors per rotation, the sector time is 1/200 of this number or 5/120 = 1/24 msec. During the 1-msec seek, 24 sectors pass under the head. Thus the cylinder skew should be 24.

18. A disk rotates at 7200 RPM. It has 500 sectors of 512 bytes around the outer cylinder. How long does it take to read a sector?

At 7200 RPM, there are 120 rotations per second, so 1 rotation takes about 8.33 msec. Dividing this by 500 we get a sector time of about 16.67 µsec.

19. Calculate the maximum data rate in bytes/sec for the disk described in the previous problem.

There are 120 rotations in a second. During one of them, 500 × 512 bytes pass under the head. So the disk can read 256,000 bytes per rotation or 30,720,000 bytes/sec.

20. RAID level 3 is able to correct single-bit errors using only one parity drive. What is the point of RAID level 2? After all, it also can only correct one error and takes more drives to do so.

RAID level 2 can not only recover from crashed drives, but also from undetected transient errors. If one drive delivers a single bad bit, RAID level 2 will correct this, but RAID level 3 will not.

21. A RAID can fail if two or more of its drives crash within a short time interval. Suppose that the probability of one drive crashing in a given hour is p. What is the probability of a k-drive RAID failing in a given hour?

The probability of 0 failures, P0, is $(1 - p)^k$. The probability of 1 failure, P1, is $kp(1 - p)^{k-1}$. The probability of a RAID failure is then 1 − P0 − P1. This is $1 - (1 - p)^k - kp(1 - p)^{k-1}$.

22. Compare RAID level 0 through 5 with respect to read performance, write performance, space overhead, and reliability.

Read performance: RAID levels 0, 2, 3, 4, and 5 allow for parallel reads to service one read request. However, RAID level 1 further allows two read requests to simultaneously proceed. Write performance: All RAID levels provide similar write performance. Space overhead: There is no space overhead in level 0 and 100% overhead in level 1. With 32-bit data word and six parity drives, the space overhead is about 18.75% in level 2. For a 32-bit data word, the space overhead in level 3 is about 3.13%. Finally, assuming 33 drives in levels 4 and 31 5, the space overhead is 3.13% in them. Reliability: There is no reliability support in level 0. All other RAID levels can survive one disk crash. In addition, in levels 3, 4 and 5, a single random bit error in a word can be detected, while in level 2, a single random bit error in a word can be detected and corrected.

23. How many pebibytes are there in a zebibyte?

A zebibyte is 2^{70} bytes; a pebibyte is 2^{50}. 2^{20} pebibytes fit in a zebibyte.

24. Why are optical storage devices inherently capable of higher data density than magnetic storage devices? Note: This problem requires some knowledge of high-school physics and how magnetic fields are generated.

A magnetic field is generated between two poles. Not only is it difficult to make the source of a magnetic field small, but also the field spreads rapidly, which leads to mechanical problems trying to keep the surface of a magnetic medium close to a magnetic source or sensor. A semiconductor laser generates light in a very small place, and the light can be optically manipulated to illuminate a very small spot at a relatively great distance from the source.

25. What are the advantages and disadvantages of optical disks versus magnetic disks?

The main advantage of optical disks is that they have much higher recording densities than magnetic disks. The main advantage of magnetic disks is that they are an order of magnitude faster than the optical disks.

26. If a disk controller writes the bytes it receives from the disk to memory as fast as it receives them, with no internal buffering, is interleaving conceivably useful? Discuss your answer.

Possibly. If most files are stored in logically consecutive sectors, it might be worthwhile interleaving the sectors to give programs time to process the data just received, so that when the next request is issued, the disk would be in the right place. Whether this is worth the trouble depends strongly on the kind of programs run and how uniform their behavior is.

27. If a disk has double interleaving, does it also need cylinder skew in order to avoid missing data when making a track-to-track seek? Discuss your answer.

Maybe yes and maybe no. Double interleaving is effectively a cylinder skew of two sectors. If the head can make a track-to-track seek in fewer than two sector times, than no additional cylinder skew is needed. If it cannot, then additional cylinder skew is needed to avoid missing a sector after a seek.

28. Consider a magnetic disk consisting of 16 heads and 400 cylinders. This disk has four 100-cylinder zones with the cylinders in different zones containing 160, 200, 240. And 280 sectors, respectively. Assume that each sector contains 512 bytes, average seek time between adjacent cylinders is 1 msec, and the disk rotates at 7200 RPM. Calculate the
(a) disk capacity,
(b) optimal track skew, and
(c) maximum data transfer rate.

Consider,
(a) The capacity of a zone is tracks × cylinders × sectors/cylinder × bytes/sect.
Capacity of zone 1: 16 × 100 × 160 × 512 = 131072000 bytes
Capacity of zone 2: 16 × 100 × 200 × 512 = 163840000 bytes
Capacity of zone 3: 16 × 100 × 240 × 512 = 196608000 bytes
Capacity of zone 4: 16 × 100 × 280 × 512 = 229376000 bytes
Sum = 131072000 + 163840000 + 196608000 + 229376000 = 720896000
(b) A rotation rate of 7200 means there are 120 rotations/sec. In the 1 msec track-to-track seek time, 0.120 of the sectors are covered. In zone 1, the disk head will pass over 0.120 × 160 sectors in 1 msec, so, optimal track skew for zone 1 is 19.2 sectors. In zone 2, the disk head will pass over 0.120 × 200 sectors in 1 msec, so, optimal track skew for zone 2 is 24 sectors. In zone 3, the disk head will pass over 0.120 × 240 sectors in 1 msec, so, optimal track skew for zone 3 is 28.8 sectors. In zone 4, the disk head will pass over 0.120 × 280 sectors in 1 msec, so, optimal track skew for zone 3 is 33.6 sectors.

(c) The maximum data transfer rate will be when the cylinders in the outermost zone (zone 4) are being read/written. In that zone, in one second, 280 sectors are read 120 times. Thus the data rate is 280 × 120 × 512 = 17,203,200 bytes/sec

29. A disk manufacturer has two 5.25-inch disks that each have 10,000 cylinders. The newer one has double the linear recording density of the older one. Which disk properties are better on the newer drive and which are the same? Are any worse on the newer one?

The drive capacity and transfer rates are doubled. The seek time and average rotational delay are the same. No properties are worse.

30. A computer manufacturer decides to redesign the partition table of a Pentium hard disk to provide more than four partitions. What are some consequences of this change?

One fairly obvious consequence is that no existing operating system will work because they all look there to see where the disk partitions are. Changing the format of the partition table will cause all the operating systems to fail. The only way to change the partition table is to simultaneously change all the operating systems to use the new format.

31. Disk requests come in to the disk driver for cylinders 10, 22, 20, 2, 40, 6, and 38, in that order. A seek takes 6 msec per cylinder. How much seek time is needed for
(a) First-come, first served.
(b) Closest cylinder next.
(c) Elevator algorithm (initially moving upward).
In all cases, the arm is initially at cylinder 20.

(a) 10 + 12 + 2 + 18 + 38 + 34 + 32 = 146 cylinders = 876 msec.
(b) 0 + 2 + 12 + 4 + 4 + 36 +2 = 60 cylinders = 360 msec.
(c) 0 + 2 + 16 + 2 + 30 + 4 + 4 = 58 cylinders = 348 msec.

32. A slight modification of the elevator algorithm for scheduling disk requests is to always scan in the same direction. In what respect is this modified algorithm better than the elevator algorithm?

In the worst case, a read/write request is not serviced for almost two full disk scans in the elevator algorithm, while it is at most one full disk scan in the modified algorithm.

33. A personal computer salesman visiting a university in South-West Amsterdam remarked during his sales pitch that his company had devoted substantial effort to making their version of UNIX very fast. As an example, he noted that their disk driver used the elevator algorithm and also queued multiple requests within a cylinder in sector order. A student, Harry Hacker, was impressed and bought one. He took it home and wrote a program to randomly read 10,000 blocks spread across the disk. To his amazement, the performance that he measured was identical to what would be expected from first-come, first-served. Was the salesman lying?

Not necessarily. A UNIX program that reads 10,000 blocks issues the requests one at a time, blocking after each one is issued until after it is completed. Thus the disk driver sees only one request at a time; it has no opportunity to do anything but process them in the order of arrival. Harry should have started up many processes at the same time to see if the elevator algorithm worked.

34. In the discussion of stable storage using nonvolatile RAM, the following point was glossed over. What happens if the stable write completes but a crash occurs before the operating system can write an invalid block number in the nonvolatile RAM? Does this race condition ruin the abstraction of stable storage? Explain your answer.

There is a race but it does not matter. Since the stable write itself has already completed, the fact that the nonvolatile RAM has not been updated just means that the recovery program will know which block was being written. It will read both copies. Finding them identical, it will change neither, which is the correct action. The effect of the crash just before the nonvolatile RAM was updated just means the recovery program will have to make two disk reads more than it should.

35. In the discussion on stable storage, it was shown that the disk can be recovered to a consistent state (a write either completes or does not take place at all) if a CPU crash occurs during a write. Does this property hold if the CPU crashes again during a recovery procedure. Explain your answer.

Yes the disk remains consistent even if the CPU crashes during a recovery procedure. Consider Fig. 5-31. There is no recovery involved in (a) or (e). Suppose that the CPU crashes during recovery in (b). If CPU crashes before the block from drive 2 has been completely copied to drive 1, the situation remains same as earlier. The subsequent recovery procedure will detect an ECC error in drive 1 and again copy the block from drive 2 to drive 1. If CPU crashes after the block from drive 2 has been copied to drive 1, the situation is same as that in case (e). Suppose that the CPU crashes during recovery in (c). If CPU crashes before the block from drive 1 has been completely copied to drive 2, the situation is same as that in case (d). The subsequent recovery procedure will detect an ECC error in drive 2 and copy the block from drive 1 to drive 2. If CPU crashes after the block from drive 1 has been copied to drive 2, the situation is same as that in case (e). Finally, suppose the CPU crashes during recovery in (d). If CPU crashes before the block from drive 1 has been completely copied to drive 2, the situation remains same as earlier. The subsequent recovery procedure will detect an ECC error in drive 2 and again copy the block from drive 1 to drive 2. If CPU crashes after the block from drive 1 has been copied to drive 2, the situation is same as that in case (e).

36. In the discussion on stable storage, a key assumption is that a CPU crash that corrupts a sector leads to an incorrect ECC. What problems might arise in the five crash-recovery scenarios shown in Figure 5-27 if this assumption does not hold?

Problems arise in scenarios shown in Figure 5-27 (b) and 5-27 (d), because they may look like scenario 5-27 (c), if the ECC of the corrupted block is correct. In this case, it is not possible to detect which disk contains the valid (old or new) lock, and a recovery is not possible

37. The clock interrupt handler on a certain computer requires 2 msec (including process switching overhead) per clock tick. The clock runs at 60 Hz. What fraction of the CPU is devoted to the clock?

Two msec 60 times a second is 120 msec/sec, or 12% of the CPU

38. A computer uses a programmable clock in square-wave mode. If a 500 MHz crystal is used, what should be the value of the holding register to achieve a clock resolution of
(a) a millisecond (a clock tick once every millisecond)?
(b) 100 microseconds?

With these parameters,

(a) Using a 500 MHz crystal, the counter can be decremented every 2 nsec. So, for a tick every millisecond, the register should be 1000000/2 = 500,000.
(b) To get a clock tick every 100 μsec, holding register value should be 50,000.

39. A system simulates multiple clocks by chaining all pending clock requests together as shown in Fig. 5-30. Suppose the current time is 5000 and there are pending clock requests for time 5008, 5012, 5015, 5029, and 5037. Show the values of Clock header, Current time, and Next signal at times 5000, 5005, and 5013. Suppose a new (pending) signal arrives at time 5017 for 5033. Show the values of Clock header, Current time and Next signal at time 5023.

At time 5000:
Current time = 5000; Next Signal = 8; Header → 8 → 4 → 3 → 14 → 8.
At time 5005:
Current time = 5005; Next Signal = 3; Header → 3 → 4 → 3 → 14 → 8.
At time 5013:
Current time = 5013; Next Signal = 2; Header 2 → 14 → 8.
At time 5023:
Current time = 5023; Next Signal = 6; Header → 6 → 4 → 5.

40. Many versions of UNIX use an unsigned 32-bit integer to keep track of the time as the number of seconds since the origin of time. When will these systems wrap around (year and month)? Do you expect this to actually happen?

The number of seconds in a mean year is 365.25 × 24 × 3600. This number is 31,557,600. The counter wraps around after 2^{32} seconds from 1 January 1970. The value of $2^{32}/31{,}557{,}600$ is 136.1 years, so wrapping will happen at 2106.1, which is early February 2106. Of course, by then, all computers will be at least 64 bits, so it will not happen at all.

41. A bitmap terminal contains 1600 by 1200 pixels. To scroll a window, the CPU (or controller) must move all the lines of text upward by copying their bits from one part of the video RAM to another. If a particular window is 80 lines high by 80 characters wide (6400 characters, total), and a character's box is 8 pixels wide by 16 pixels high, how long does it take to scroll the whole window at a copying rate of 50 nsec per byte? If all lines are 80 characters long, what is the equivalent baud rate of the terminal? Putting a character on the screen takes 5 μsec. How many lines per second can be displayed?

Scrolling the window requires copying 79 lines of 80 characters or 6320 characters. Copying 1 character (16 bytes) takes 800 nsec, so the whole window takes 5.056 msec. Writing 80 characters to the screen takes 400 nsec, so scrolling and displaying a new line take 5.456 msec. This gives about 183.2 lines/sec.

42. After receiving a DEL (SIGINT) character, the display driver discards all output currently queued for that display. Why?

Suppose that the user inadvertently asked the editor to print thousands of lines. Then he hits DEL to stop it. If the driver did not discard output, output might continue for several seconds after the DEL, which would make the user hit DEL again and again and get frustrated when nothing happened.

43. A user at a terminal issues a command to an editor to delete the word on line 5 occupying character positions 7 through and including 12. Assuming the cursor is not on

line 5 when the command is given, what ANSI escape sequence should the editor emit to delete the word?

It should move the cursor to line 5 position 7 and then delete 6 characters. The sequence is ESC [5 ; 7 H ESC [6 P

44. The designers of a computer system expected that the mouse could be moved at a maximum rate of 20 cm/sec. If a mickey is 0.1 mm and each mouse message is 3 bytes, what is the maximum data rate of the mouse assuming that each mickey is reported separately?

The maximum rate the mouse can move is 200 mm/sec, which is 2000 mickeys/sec. If each report is 3 byte, the output rate is 6000 bytes/sec.

45. The primary additive colors are red, green, and blue, which means that any color can be constructed from a linear superposition of these colors. Is it possible that someone could have a color photograph that cannot be represented using full 24-bit color?

With a 24-bit color system, only 224 colors can be represented. This is not all of them. For example, suppose that a photographer takes pictures of 300 cans of pure blue paint, each with a slightly different amount of pigment. The first might be represented by the (R, G, B) value (0, 0, 1). The next one might be represented by (0, 0, 2), and so forth. Since the B coordinate is only 8 bits, there is no way to represent 300 different values of pure blue. Some of the photographs will have to be rendered as the wrong color. Another example is the color (120.24, 150.47, 135.89). It cannot be represented, only approximated by (120, 150, 136).

46. One way to place a character on a bitmapped screen is to use BitBlt from a font table. Assume that a particular font uses characters that are 16 × 24 pixels in true RGB color.
(a) How much font table space does each character take?
(b) If copying a byte takes 100 nsec, including overhead, what is the output rate to the screen in characters/sec?

(a) Each pixel takes 3 bytes in RGB, so the table space is 16 × 24 × 3 bytes, which is 1152 bytes.
(b) At 100 nsec per byte, each character takes 115.2 μsec. This gives an output rate of about 8681 chars/sec.

47. Assuming that it takes 2 nsec to copy a byte, how much time does it take to completely rewrite the screen of an 80 character × 25 line text mode memory-mapped screen? What about a 1024 × 768 pixel graphics screen with 24-bit color?

Rewriting the text screen requires copying 2000 bytes, which can be done in 4 μseconds. Rewriting the graphics screen requires copying 1024 × 768 × 3 = 2,359,296 bytes, or about 4.72 msec.

48. In Fig. 5-36 there is a class to RegisterClass. In the corresponding X Window code, in Fig. 5-34, there is no such call or anything like it. Why not?

In Windows, the OS calls the handler procedures itself. In X Windows, nothing like this happens. X just gets a message and processes it internally.

49. In the text we gave an example of how to draw a rectangle on the screen using the Windows GDI:
Rectangle(hdc, xleft, ytop, xright, ybottom);

Is there any real need for the first parameter (hdc), and if so, what? After all, the coordinates of the rectangle are explicitly specified as parameters.

The first parameter is essential. First of all, the coordinates are relative to some window, so hdc is needed to specify the window and thus the origin. Second, the rectangle will be clipped if it falls outside the window, so the window coordinates are needed. Third, the color and other properties of the rectangle are taken from the context specified by hdc. It is quite essential.

50. A thin-client terminal is used to display a Web page containing an animated cartoon of size 400 pixels × 160 pixels running at 10 frames/sec. What fraction of a 100-Mbps Fast Ethernet is consumed by displaying the cartoon?

The display size is 400 × 160 × 3 bytes, which is 192,000 bytes. At 10 fps this is 1,920,000 bytes/sec or 15,360,000 bits/sec. This consumes 15% of the Fast Ethernet.

51. It has been observed that a thin-client system works well with a 1-Mbps network in a test. Are any problems likely in a multiuser situation? (Hint: Consider a large number of users watching a scheduled TV show and the same number of users browsing the World Wide Web.)

The bandwidth on a network segment is shared, so 100 users requesting different data simultaneously on a 1-Mbps network will each see a 10-Kbps effective speed. With a shared network, a TV program can be multicast, so the video packets are only broadcast once, no matter how many users there are and it should work well. With 100 users browsing the Web, each user will get 1/100 of the bandwidth, so performance may degrade very quickly.

52. Describe two advantages and two disadvantages of thin client computing?

Advantages of thin clients include low cost and no need for complex management for the clients. Disadvantages include (potentially) lower performance due to network latency and (potential) loss of privacy because the client’s data/information is shared with the server.

53. If a CPU’s maximum voltage, V, is cut to V/n, its power consumption drops to $1/n^2$ of its original value and its clock speed drops to 1/n of its original value. Suppose that a user is typing at 1 char/sec, but the CPU time required to process each character is 100 msec. What is the optimal value of n and what is the corresponding energy saving in percent compared to not cutting the voltage? Assume that an idle CPU consumes no energy at all.

If n = 10, the CPU can still get its work done on time, but the energy used drops appreciably. If the energy consumed in 1 sec at full speed is E, then running at full speed for 100 msec then going idle for 900 msec uses E/10. Running at 1/10 speed for a whole second uses E/100, a saving of 9E/100. The percent savings by cutting the voltage is 90%.

54. A notebook computer is set up to take maximum advantage of power saving features including shutting down the display and the hard disk after periods of inactivity. A user sometimes runs UNIX programs in text mode, and at other times uses the X Window System. She is surprised to find that battery life is significantly better when she uses text-only programs. Why?

The windowing system uses much more memory for its display and uses virtual memory more than the text mode. This makes it less likely that the hard disk will be inactive for a period long enough to cause it to be automatically powered down.

55. Write a program that simulates stable storage. Use two large fixed-length files on your disk to simulate the two disks.

56. Write a program to implement the three disk-arm scheduling algorithms. Write a driver
program that generates a sequence of cylinder numbers (0–999) at random, runs the
three algorithms for this sequence and prints out the total distance (number of cylinders)
the arm needs to traverse in the three algorithms.
57. Write a program to implement multiple timers using a single clock. Input for this program
consists of a sequence of four types of commands (S <int>, T, E <int>, P): S
<int> sets the current time to <int>; T is a clock tick; and E <int> schedules a signal to
occur at time <int>; P prints out the values of Current time, Next signal, and Clock
header. Your program should also print out a statement whenever it is time to raise a
signal.

1. Give an example of a deadlock taken from politics.

In the U.S., consider a presidential election in which three or more candidates are trying for the nomination of some party. After all the primary elections are finished, when the delegates arrive at the party convention, it could happen that no candidate has a majority and that no delegate is willing to change his or her vote. This is a deadlock. Each candidate has some resources (votes) but needs more to get the job done. In countries with multiple political parties in the parliament, it could happen that each party supports a different version of the annual budget and that it is impossible to assemble a majority to pass the budget. This is also a deadlock.

2. Students working at individual PCs in a computer laboratory send their files to beprinted by a server that spools the files on its hard disk. Under what conditions may a deadlock occur if the disk space for the print spool is limited? How may the deadlock be avoided?

Disk space on the spooling partition is a finite resource. Every block that comes in de facto claims a resource and every new one arriving wants more resources. If the spooling space is, say, 10 MB and the first half of ten 2-MB jobs arrive, the disk will be full and no more blocks can be stored so we have a deadlock. The deadlock can be avoided by allowing a job to start printing before it is fully spooled and reserving the space thus released for the rest of that job. In this way, one job will actually print to completion, then the next one can do the same thing. If jobs cannot start printing until they are fully spooled, deadlock is possible.

3. In the preceding question, which resources are preemptable and which are nonpreemptable?

The printer is nonpreemptable; the system cannot start printing another job until the previous one is complete. The spool disk is preemptable; you can delete an incomplete file that is growing too large and have the user send it later, assuming the protocol allows that.

4. In Fig. 6-1 the resources are returned in the reverse order of their acquisition. Would giving them back in the other order be just as good?

Yes. It does not make any difference whatsoever.

5. The four conditions (mutual exclusion, hold and wait, no preemption and circular wait) are necessary for a resource deadlock to occur. Give an example to show that these conditions are not sufficient for a resource deadlock to occur. When are these conditions sufficient for a resource deadlock to occur?

Suppose that there are three processes, A, B and C, and two resource types, R and S. Further assume that there are one instance of R and two instance of S. Consider the following execution scenario: A requests R and gets it; B requests S and gets; C requests S and gets it (there are two instances of S); B requests R and is blocked; A requests S and is blocked. At this stage all four conditions hold. However, there is no deadlock. When C finishes, one instance of S is released that is allocated to A. Now A can complete its execution and release R that can be allocated to B, which can then complete its execution. These four conditions are enough if there is one resource of each type.

6. City streets are vulnerable to a circular blocking condition called gridlock, in which intersections are blocked by cars that then block cars behind them that then block the cars that are trying to enter the previous intersection, etc. All intersections around a city block are filled with vehicles that block the oncoming traffic in a circular manner. Gridlock is a resource deadlock and a problem in competition synchronization. New York City's prevention algorithm, called "don't block the box," prohibits cars from entering an intersection unless the space following the intersection is also available. Which prevention algorithm is this? Can you provide any other prevention algorithms for gridlock?

“Don't block the box” is a pre-allocation strategy, negating the hold and wait deadlock precondition, since we assume that cars can enter the street space following the intersection, thus freeing the intersection. Another strategy might allow cars to temporarily pull into garages and release enough space to clear the gridlock. Some cities have a traffic control policy to shape traffic; as city streets become more congested, traffic supervisors adjust the settings for red lights in order to throttle traffic entering heavily congested areas. Lighter traffic ensures less competition over resources and thus lowers the probability of gridlock occurring.

7. Suppose four cars each approach an intersection from four different directions simultaneously. Each corner of the intersection has a stop sign. Assume that traffic regulations require that when two cars approach adjacent stop signs at the same time, the car on the left must yield to the car on the right. Thus, as four cars each drive up to their individual stop signs, each waits (indefinitely) for the car on the left to proceed. Is this anomaly a communication deadlock? Is it a resource deadlock?

The above anomaly is not a communication deadlock since these cars are independent of each other and would drive through the intersection with a minimal delay if no competition occurred. It is not a resource deadlock, since no car is holding a resource that is requested by another car. Nor would the mechanisms of resource pre-allocation or of resource preemption assist in controlling this anomaly. This anomaly is one of competition synchronization, however, in which cars are waiting for resources in a circular chain and traffic throttling may be an effective strategy for control. To distinguish from resource deadlock, this anomaly might be termed a “scheduling deadlock.” A similar deadlock could occur following a law that required two trains merging onto a shared railroad track to wait for the other to proceed. Note that a policeman signaling one of the competing cars or trains to proceed (and not the others) can break this dead state without rollback or any other overhead.

8. Is it possible that a resource deadlock involves multiple units of one type and a single unit of another? If so, give an example.

It is possible that one process holds some or all of the units of one resource type and requests another resource type, while another process holds the second resource while requesting the available units of the first resource type. If no other process can release units of the first resource type and the resource cannot be preempted or used concurrently, the system is deadlocked. For example, two processes are both allocated memory cells in a real memory system. (We assume that swapping of pages or processes is not supported, while dynamic requests for memory are supported.) The first process locks another resource - perhaps a data cell. The second process requests the locked data and is blocked. The first process needs more memory in order to execute the code to release the data. Assuming that no other processes in the system can complete and release memory cells, a deadlock exists in the system.

9. Fig. 6-3 shows the concept of a resource graph. Do illegal graphs exist, that is, graphs that structurally violate the model we have used of resource usage? If so, give an example of one.

Yes, illegal graphs exist. We stated that a resource may only be held by a single process. An arc from a resource square to a process circle indicates that the process owns the resource. Thus, a square with arcs going from it to two or more processes means that all those processes hold the resource, which violates the rules. Consequently, any graph in which multiple arcs leave a square and end in different circles violates the rules unless there are multiple copies of the resources. Arcs from squares to squares or from circles to circles also violate the rules.

10. Consider Fig. 6-4. Suppose that in step (o) C requested S instead of requesting R. Would this lead to deadlock? Suppose that it requested both S and R.

Neither change leads to deadlock. There is no circular wait in either case.

11. Suppose that there is a resource deadlock in a system. Give an example to show that the set of processes deadlocked can include processes that are not in the circular chain in the corresponding resource allocation graph.

Consider three processes, A, B and C and two resources R and S. Suppose A is waiting for I that is held by B, B is waiting for S held by A, and C is waiting for R held by A. All three processes, A, B and C are deadlocked. However, only A and B belong to the circular chain

12. In order to control traffic, a network router, A periodically sends a message to its neighbor, B, telling it to increase or decrease the number of packets that it can handle. At some point in time, Router A is flooded with traffic and sends B a message telling it to cease sending traffic. It does this by specifying that the number of bytes B may send (A's window size) is 0. As traffic surges decrease, A sends a new message, telling B to restart transmission. It does this by increasing the window size from 0 to a positive number. That message is lost. As described, neither side will ever transmit. What type of deadlock is this?

This is clearly a communication deadlock, and can be controlled by having A time out and retransmit its enabling message (the one that increases the window size) after some period of time (a heuristic). It is possible, however, that B has received both the original and the duplicate message. No harm will occur if the update on the window size is given as an absolute value and not as a differential. Sequence numbers on such messages are also effective to detect duplicates.

13. The discussion of the ostrich algorithm mentions the possibility of process-table slots or other system tables filling up. Can you suggest a way to enable a system administrator to recover from such a situation?

A portion of all such resources could be reserved for use only by processes owned by the administrator, so he or she could always run a shell and programs needed to evaluate a deadlock and make decisions about which processes to kill to make the system usable again.

14. Consider the following state of a system with four processes, P1, P2, P3, and P4, and five types of resources, RS1, RS2, RS3, RS4, and RS5:

$$C = \begin{bmatrix} 0 & 1 & 1 & 1 & 2 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 2 & 1 & 0 & 0 & 0 \end{bmatrix} \qquad R = \begin{bmatrix} 1 & 1 & 0 & 2 & 1 \\ 0 & 1 & 0 & 2 & 1 \\ 0 & 2 & 0 & 3 & 1 \\ 0 & 2 & 1 & 1 & 0 \end{bmatrix} \qquad \begin{array}{l} E = (24144) \\ A = (01021) \end{array}$$

Using the deadlock detection algorithm described in Section 6.4.2, show that there is a deadlock in the system. Identify the processes that are deadlocked.

First, the set of unmarked processes, P = (P1 P2 P3 P4)
R1 is not less than or equal to A
R2 is less than A; Mark P2; A = (0 2 0 3 1); P = (P1 P3 P4)
R1 is not less than or equal to A
R3 is equal to A; Mark P3; A = (0 2 0 3 2); P = (P1 P4)
R1 is not less than or equal to A
R4 is not less than or equal to A
So, processes P1 and P4 remain unmarked. They are deadlocked.

15. Explain how the system can recover from the deadlock in previous problem using
(a) recovery through preemption.
(b) recovery through rollback.
(c) recovery through killing processes.

Recovery through preemption: After processes P2 and P3 complete, process P1 can be forced to preempt 1 unit of RS3. This will makeA=(0 2 1 3 2), and allow process P4 to complete. Once P4 completes and release its resources P1 may complete. Recovery through rollback: Rollback P1 to the state checkpointed before it acquired RS3. Recovery through killing processes: Kill P1.

16. Suppose that in Fig. 6-6 Cij + Rij > Ej for some i. What implications does this have for the system?

The process is asking for more resources than the system has. There is no conceivable way it can get these resources, so it can never finish, even if no other processes want any resources at all.

17. All the trajectories in Fig. 6-8 are horizontal or vertical. Can you envision any circumstances in which diagonal trajectories are also possible?

If the system had two or more CPUs, two or more processes could run in parallel, leading to diagonal trajectories.

18. Can the resource trajectory scheme of Fig. 6-8 also be used to illustrate the problem of deadlocks with three processes and three resources? If so, how can this be done? If not, why not?

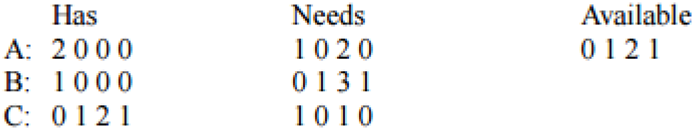
Yes. Do the whole thing in three dimensions. The z-axis measures the number of instructions executed by the third process.

19. In theory, resource trajectory graphs could be used to avoid deadlocks. By clever scheduling, the operating system could avoid unsafe regions. Is there a practical way of actually doing this?

The method can only be used to guide the scheduling if the exact instant at which a resource is going to be claimed is known in advance. In practice, this is rarely the case.

20. Can a system be in a state that is neither deadlocked nor safe? If so, give an example. If not, prove that all states are either deadlocked or safe.

There are states that are neither safe nor deadlocked, but which lead to deadlocked states. As an example, suppose we have four resources: tapes, plotters, scanners, and CD-ROMs, as in the text, and three processes competing for them. We could have the following situation:



	Has	Needs	Available
A:	2 0 0 0	1 0 2 0	0 1 2 1
B:	1 0 0 0	0 1 3 1	
C:	0 1 2 1	1 0 1 0	

This state is not deadlocked because many actions can still occur, for example, A can still get two printers. However, if each process asks for its remaining requirements, we have a deadlock.

21. Take a careful look at Fig. 6-11(b). If D asks for one more unit, does this lead to a safe state or an unsafe one? What if the request came from C instead of D?

A request from D is unsafe, but one from C is safe

22. A system has two processes and three identical resources. Each process needs a maximum of two resources. Is deadlock possible? Explain your answer.

The system is deadlock free. Suppose that each process has one resource. There is one resource free. Either process can ask for it and get it, in which case it can finish and release both resources. Consequently, deadlock is impossible.

23. Consider the previous problem again, but now with p processes each needing a maximum of m resources and a total of r resources available. What condition must hold to make the system deadlock free?

If a process has m resources it can finish and cannot be involved in a deadlock. Therefore, the worst case is where every process has $m - 1$ resources and needs another one. If there is one resource left over, one process can finish and release all its resources, letting the rest finish too. Therefore the condition for avoiding deadlock is $r \geq p(m - 1) + 1$

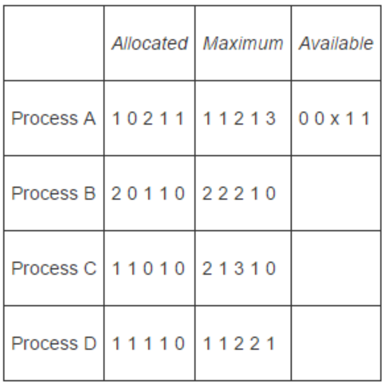
24. Suppose that process A in Fig. 6-12 requests the last tape drive. Does this action lead to a deadlock?

No. D can still finish. When it finishes, it returns enough resources to allow E (or A) to finish, and so on.

25. The banker's algorithm is being run in a system with m resource classes and n processes. In the limit of large m and n, the number of operations that must be performed to check a state for safety is proportional to $m^a n^b$. What are the values of a and b?

Comparing a row in the matrix to the vector of available resources takes m operations. This step must be repeated on the order of n times to find a process that can finish and be marked as done. Thus, marking a process as done takes on the order of mn steps. Repeating the algorithm for all n processes means that the number of steps is then mn^2 . Thus, a = 1 and b = 2.

26. A system has four processes and five allocatable resources. The current allocation and maximum needs are as follows:



	Allocated	*Maximum*	*Available*
Process A	1 0 2 1 1	1 1 2 1 3	0 0 x 1 1
Process B	2 0 1 1 0	2 2 2 1 0	
Process C	1 1 0 1 0	2 1 3 1 0	
Process D	1 1 1 1 0	1 1 2 2 1	

What is the smallest value of x for which this is a safe state?

The needs matrix is as follows:
01002
02100
10300
00111
If x is 0, we have a deadlock immediately. If x is 1, process D can run to completion. When it is finished, the available vector is 1 1 2 2 1. Unfortunately we are now deadlocked. If x is 2, after D runs, the available vector is 11321 and C can run. After it finishes and returns its resources the available vector is 2 2 3 3 1, which will allow B to run and complete, and then A to run and complete. Therefore, the smallest value of x that avoids a deadlock is 2.

27. One way to eliminate circular wait is to have rule saying that a process is entitled only to a single resource at any moment. Give an example to show that this restriction is unacceptable in many cases.

Consider a process that needs to copy a huge file from a tape to a printer. Because the amount of memory is limited and the entire file cannot fit in this memory, the process will have to loop through the following statements until the entire file has been printed:
Acquire tape drive
Copy the next portion of the file in memory (limited memory size)
Release tape drive
Acquire printer
Print file from memory
Release printer
This will lengthen the execution time of the process. Furthermore, since the printer is released after every print step, there is no guarantee that all portions of the file will get printed on continuous pages.

28. Two processes, A and B, each need three records, 1, 2, and 3, in a database. If A asks for them in the order 1, 2, 3, and B asks for them in the same order, deadlock is not possible. However, if B asks for them in the order 3, 2, 1, then deadlock is possible. With three resources, there are 3! or six possible combinations in which each process can request them. What fraction of all the combinations is guaranteed to be deadlock free?

Suppose that process A requests the records in the order a, b, c. If process B also asks for a first, one of them will get it and the other will block. This situation is always deadlock free since the winner can now run to completion without interference. Of the four other combinations, some may lead to deadlock and some are deadlock free. The six cases are as follows:
abc deadlock free
acb deadlock free
bac possible deadlock
bca possible deadlock
cab possible deadlock
cba possible deadlock
Since four of the six may lead to deadlock, there is a 1/3 chance of avoiding a deadlock and a 2/3 chance of getting one.

29. A distributed system using mailboxes has two IPC primitives, send and receive. The latter primitive specifies a process to receive from and blocks if no message from that process is available, even though messages may be waiting from other processes. There are no shared resources, but processes need to communicate frequently about other matters. Is deadlock possible? Discuss.

Yes. Suppose that all the mailboxes are empty. Now A sends to B and waits for a reply, B sends to C and waits for a reply, and C sends to A and waits for a reply. All the conditions for a communications deadlock are now fulfilled.

30. In an electronic funds transfer system, there are hundreds of identical processes that work as follows. Each process reads an input line specifying an amount of money, the account to be credited, and the account to be debited. Then it locks both accounts and transfers the money, releasing the locks when done. With many processes running in parallel, there is a very real danger that a process having locked account x will be unable to lock y because y has been locked by a process now waiting for x. Devise a scheme that avoids deadlocks. Do not release an account record until you have completed the transactions. (In other words, solutions that lock one account and then release it immediately if the other is locked are not allowed.)

To avoid circular wait, number the resources (the accounts) with their account numbers. After reading an input line, a process locks the lower-numbered account first, then when it gets the lock (which may entail waiting), it locks the other one. Since no process ever waits for an account lower than what it already has, there is never a circular wait, hence never a deadlock.

31. One way to prevent deadlocks is to eliminate the hold-and-wait condition. In the text it was proposed that before asking for a new resource, a process must first release whatever resources it already holds (assuming that is possible). However, doing so introduces the danger that it may get the new resource but lose some of the existing ones to competing processes. Propose an improvement to this scheme.

Change the semantics of requesting a new resource as follows. If a process asks for a new resource and it is available, it gets the resource and keeps what it already has. If the new resource is not available, all existing resources are released. With this scenario, deadlock is impossible and there is no danger that the new resource is acquired but existing ones lost. Of course, the process only works if releasing a resource is possible (you can release a scanner between pages or a CD recorder between CDs).

32. A computer science student assigned to work on deadlocks thinks of the following brilliant way to eliminate deadlocks. When a process requests a resource, it specifies a time limit. If the process blocks because the resource is not available, a timer is started. If the time limit is exceeded, the process is released and allowed to run again. If you were the professor, what grade would you give this proposal and why?

I'd give it an F (failing) grade. What does the process do? Since it clearly needs the resource, it just asks again and blocks again. This is no better than staying blocked. In fact, it may be worse since the system may keep track of how long competing processes have been waiting and assign a newly freed resource to the process that has been waiting longest. By periodically timing out and trying again, a process loses its seniority.

33. Main memory units are preempted in swapping and virtual memory systems. The processor is preempted in time-sharing environments. Do you think that these preemption methods were developed to handle resource deadlock or for other purposes? How high is their overhead?

Both virtual memory and time-sharing systems were developed mainly to assist system users. Virtualizing hardware shields users from the details of prestating needs, resource allocation, and overlays, in addition to preventing deadlock. The cost of context switching and interrupt handling, however, is considerable. Specialized registers, caches, and circuitry are required. Probably this cost would not have been incurred for the purpose of deadlock prevention alone.

34. Explain the differences between deadlock, livelock, and starvation.

A deadlock occurs when a set of processes are blocked waiting for an event that only some other process in the set can cause. On the other hand, processes in a livelock are not blocked. Instead, they continue to execute checking for a condition to become true that will never become true. Thus, in addition to the resources they are holding, processes in livelock continue to consume precious CPU time. Finally, starvation of a process occurs because of the presence of other processes as well as a stream of new incoming processes that end up with higher priority that the process being starved. Unlike deadlock or livelock, starvation can terminate on its own, e.g. when existing processes with higher priority terminate and no new processes with higher priority arrive.

35. Assume two processes are issuing a seek command to reposition the mechanism to access the disk and enable a read command. Each process is interrupted before executing its read, and discovers that the other has moved the disk arm. Each then reissues the seek command, but is again interrupted by the other. This sequence continually repeats. Is this a resource deadlock or a livelock? What methods would you recommend to handle the anomaly?

This dead state is an anomaly of competition synchronization and can be controlled by resource pre-allocation. Processes, however, are not blocked from resources. In addition, resources are already requested in a linear order. This anomaly is not a resource deadlock; it is a livelock. Resource preallocation will prevent this anomaly. As a heuristic, processes may time-out and release their resources if they do not complete within some interval of time, then go to sleep for a random period and then try again.

36. Local Area Networks utilize a media access method called CSMA/CD, in which stations sharing a bus can sense the medium and detect transmissions as well as collisions. In the Ethernet protocol, stations requesting the shared channel do not transmit frames if they sense the medium is busy. When such transmission has terminated, waiting stations each transmit their frames. Two frames that are transmitted at the same time will collide. If stations immediately and repeatedly retransmit after collision detection, they will continue to collide indefinitely.
(a) Is this a resource deadlock or a livelock?
(b) Can you suggest a solution to this anomaly?
(c) Can starvation occur with this scenario?

Here are the answers, albeit a bit complicated.
(a) This is a competition synchronization anomaly. It is also a livelock. We might term it a scheduling livelock. It is not a resource livelock or deadlock, since stations are not holding resources that are requested by others and thus a circular chain of stations holding resources while requesting others does not exist. It is not a communication deadlock, since stations are executing independently and would complete transmission were scheduled sequentially.
(b) Ethernet and slotted Aloha require that stations that detect a collision of their transmission must wait a random number of time slots before retransmitting. The interval within which the time slot is chosen is doubled after each successive collision, dynamically adjusting to heavy traffic loads. After sixteen successive retransmissions a frame is dropped.
(c) Because access to the channel is probabilistic, and because newly arriving stations can compete and be allocated the channel before stations that have retransmitted some number of times, starvation is enabled.

37. A program contains an error in the order of cooperation and competition mechanisms, resulting in a consumer process locking a mutex (mutual exclusion semaphore) before it blocks on an empty buffer. The producer process blocks on the mutex before it can place a value in the empty buffer and awaken the consumer. Thus, both processes are blocked forever, the producer waiting for the mutex to be unlocked and the consumer waiting for a signal from the producer. Is this a resource deadlock or a communication deadlock? Suggest methods for its control.

The anomaly is not a resource deadlock. Although processes are sharing a mutex, i.e., a competition mechanism, resource pre-allocation and deadlock avoidance methods are all ineffective for this dead state. Linearly ordering resources is also ineffective. Indeed one could argue that linear orders may be the problem; executing a mutex should be the last step before entering and the first after leaving a critical section. A circular dead state does exist in which both processes wait for an event that can only be caused by the other process. This is a communication deadlock. To make progress, a time-out will work to break this deadlock if it preempts the consumer’s mutex. Writing careful code or using monitors for mutual exclusion are better solutions.

38. Cinderella and the Prince are getting divorced. To divide their property, they have agreed on the following algorithm. Every morning, each one may send a letter to the other’s lawyer requesting one item of property. Since it takes a day for letters to be delivered, they have agreed that if both discover that they have requested the same item on the same day, the next day they will send a letter canceling the request. Among their property is their dog, Woofer, Woofer’s doghouse, their canary, Tweeter, and Tweeter’s

cage. The animals love their houses, so it has been agreed that any division of property separating an animal from its house is invalid, requiring the whole division to start over from scratch. Both Cinderella and the Prince desperately want Woofer. So that they can go on (separate) vacations, each spouse has programmed a personal computer to handle the negotiation. When they come back from vacation, the computers are still negotiating. Why? Is deadlock possible? Is starvation possible? Discuss your answer.

If both programs ask for Woofer first, the computers will starve with the endless sequence: request Woofer, cancel request, request Woofer, cancel request, and so on. If one of them asks for the doghouse and the other asks for the dog, we have a deadlock, which is detected by both parties and then broken, but it is just repeated on the next cycle. Either way, if both computers have been programmed to go after the dog or the doghouse first, either starvation or deadlock ensues. There is not really much difference between the two here. In most deadlock problems, starvation does not seem serious because introducing random delays will usually make it very unlikely. That approach does not work here.

39. A student majoring in anthropology and minoring in computer science has embarked on a research project to see if African baboons can be taught about deadlocks. He locates a deep canyon and fastens a rope across it, so the baboons can cross hand-overhand. Several baboons can cross at the same time, provided that they are all going in the same direction. If eastward-moving and westward-moving baboons ever get onto the rope at the same time, a deadlock will result (the baboons will get stuck in the middle) because it is impossible for one baboon to climb over another one while suspended over the canyon. If a baboon wants to cross the canyon, he must check to see that no other baboon is currently crossing in the opposite direction. Write a program using semaphores that avoids deadlock. Do not worry about a series of eastward-moving baboons holding up the westward-moving baboons indefinitely.

40. Repeat the previous problem, but now avoid starvation. When a baboon that wants to cross to the east arrives at the rope and finds baboons crossing to the west, he waits until the rope is empty, but no more westward-moving baboons are allowed to start until at least one baboon has crossed the other way.

470 DEADLOCKS CHAP. 6

41. Program a simulation of the banker's algorithm. Your program should cycle through each of the bank clients asking for a request and evaluating whether it is safe or unsafe. Output a log of requests and decisions to a file.

42. Write a program to implement the deadlock detection algorithm with multiple resources of each type. Your program should read from a file the following inputs: the number of processes, the number of resource types, the number of resources of each type in existence (vector E), the current allocation matrix C (first row, followed by the second row, and so on), the request matrix R (first row, followed by the second row, and so on). The output of your program should indicate whether there is a deadlock in the system. In case there is, the program should print out the identities of all processes that are deadlocked.

43. Write a program that detects if there is a deadlock in the system by using a resource allocation graph. Your program should read from a file the following inputs: the number of processes and the number of resources. For each process if should read four numbers: the number of resources it is currently holding, the IDs of resources it is holding, the number of resources it is currently requesting, the IDs of resources it is requesting. The output of program should indicate if there is a deadlock in the system. In case there is, the program should print out the identities of all processes that are deadlocked.

44. In certain countries, when two people meet they bow to each other. The protocol is that one of them bows first and stays down until the other one bows. If they bow at the same time, they will both stay bowed forever. Write a program that does not deadlock

1. Give a reason why a data center might be interested in virtualization.

There are numerous reasons, among them consolidating servers to save hardware investment cost, rack space, and electrical power, and make management of thousands of servers easier.
Data center might be interested in virtualization due to following reasons

- Most of the areas spending dollars for heat dissipation and control in the data center. Virtualize your servers and using less physical hardware.
- Use less physical hardware and generate less heat.
- In physical server re-deploy takes more time and depends upon number of factors.
- By using virtualization, the re-deploy can occur within minutes.
- Disaster recovery is quite a bit easier when your data center is virtualized.

2. Give a reason why a company might be interested in running a hypervisor on a machine that has been in use for a while.

If the hardware configuration was upgraded, virtualization could hide this and allow old software to continue working.

3. Give a reason why a software developer might use virtualization on a desktop machine being used for development.

There are various reasons. A key one is to have many platforms such as Windows 7, Windows 8, Linux, FreeBSD, OS X, etc. available on a single desktop machine to test the software being developed. Also, rebooting a virtual machine after a crash induced by a software bug is much faster

4. Give a reason why an individual at home might be interested in virtualization.

After upgrading to a new computer and operating system, the person might want to run some software that he had on the old one. Virtualization makes it possible to run the old system and new one on the same computer, thus preserving the old software.

5. Why do you think virtualization took so long to become popular? After all, the key paper was written in 1974 and IBM mainframes had the necessary hardware and software throughout the 1970s and beyond.

Very few programmers had access to an IBM mainframe. Starting on the 1980s, the Intel x86 series dominated computing and it was not virtualizable. While binary translation could solve that problem, that idea was not thought of until the late 1990s.

6. Name two kinds of instructions that are sensitive in the Popek and Goldberg sense.

Any instruction changing the page tables or memory map is certainly sensitive, as well as anything involving I/O. Any instruction capable of reading the true state of the machine is also sensitive.
Instruction that manipulate interrupt flag (PUSHF, POPF): Usually they are used to changes the bit that disables or enables interrupts. While in user mode, this bit change does not occur.

Instructions that provide read-only access to privileged registers (SGDT, SIDT, and SLDT): Usually they are used to load and store the values of Global, Interrupt and Local Descriptor Tables, In user mode, the values can only be stored and not loaded.

7. Name three machine instructions that are not sensitive in the Popek and Goldberg sense.

There are many, including moves, arithmetic instructions, jump and call instructions, shifts, etc.

8. What is the difference between full virtualization and paravirtualization? Which do you think is harder to do? Explain your answer.

Full virtualization means emulating the hardware exactly so every operating system running on the virtual machine behaves exactly as it would on the bare metal. Paravirtualization consists of changing the operating system so it does not do anything that is hard to virtualize. Full virtualization in the absence of hardware support is complicated on any architecture that is complex, like the x86. It is easier on RISC machines. If virtualization hardware is present, full virtualization is not so difficult. So, which is harder probably depends on whether hardware support is available. If it is, then paravirtualizing an operating system is probably more work. If there is no hardware support, it may be easier to change the operating system to be more friendly. If there are many operating systems that have to be paravirtualized, that could be more work.

9. Does it make sense to paravirtualize an operating system if the source code is available? What if it is not?

Yes, of course. Linux has been paravirtualized precisely because the source code is available. Windows has been paravirtualized by Microsoft (which has the source code), but has not released any paravirtualized versions.

10. Consider a type 1 hypervisor that can support up to n virtual machines at the same time. PCs can have a maximum of four disk primary partitions. Can n be larger than 4? If so, where can the data be stored?

Virtual machines have nothing to do with disk partitions. The hypervisor can take a disk partition and divide it up into subpartitions and give each virtual machine one of them. In principle, there can be hundreds. It can either statically partition the disk into n pieces or do this on demand. In hosted virtual machines, it is common to use files on the host to store disk images of the guest.

11. Briefly explain the concept of process-level virtualization.

An application or process is virtualized during runtime, by using a virtualization layer between the application and the OS. This layer executes the application’s instructions, modifying them as required prior to execution. The application is transparent to the presence of the underlying layer. Windows Emulator (WINE) is an example, where Microsoft Windows binary executables can be executed on another operating system such as Linux. This is done using onthe-fly mapping of Windows API calls to POSIX calls.

12. Why do type 2 hypervisors exist? After all, there is nothing they can do that type 1 hypervisors cannot do and the type 1 hypervisors are generally more efficient as well.

Type 1 hypervisors generally require changing the boot procedure of the computer to load the hypervisor first, then create virtual machines, and then install operating systems in them. At data centers run by expert system administrators, this is not a problem, but for most ordinary users, doing this is far too complicated. Type 2 hypervisors were invented to make installing a hypervisor no more difficult than installing an application program, something that users frequently do. Also, by using a host operating system to service local peripherals, it was not necessary for the hypervisor to have drivers for all of them since it could use the ones inside the host OS.

13. Is virtualization of any use to type 2 hypervisors?

Yes. When a guest OS does I/O, for example, the virtualization hardware catches it and gets control to the type 2 hypervisor, which then figures out what to do. Usually this will involve making a request to the host OS to perform the I/O, but not having to worry about trapping the I/O instruction definitely simplifies matters for the hypervisor

14. Why was binary translation invented? Do you think it has much of a future? Explain your answer.

It was invented in the early days, before virtualization hardware existed. It was necessary to prevent guest operating systems, which were running in user mode, from executing sensitive instructions that were not privileged. Going forward, this is less necessary since modern hardware traps when a user-mode program executes a sensitive instruction. However, in some circumstances, binary translation is faster than trapping. Nevertheless, as the hardware improves, the need for binary translation will decrease.

15. Explain how the x86's four protection rings can be used to support virtualization.

Typically, ring 0 (with the highest set of privileges) is used for running in kernel mode; and ring 3 for user mode. Some hypervisors used ring 0 for running the hypervisor in kernel mode; the guest OS was run in ring 1. When the guest OS invokes privileged instructions, it could trap to the hypervisor that runs these instructions after verifying the access rights, permissions, etc. Other ways are also possible

16. State one reason as to why a hardware-based approach using VT-enabled CPUs can perform poorly when compared to translation-based software approaches.

It has been shown that the VT-enabled CPU approach results in a lot of traps due to the use of trap-and-emulate approach. Since traps are expensive to handle, there are instances where translation based approaches outperform the hardware based approach.

17. Give one case where a translated code can be faster than the original code, in a system using binary translation.

When the guest OS issues a ‘‘Clear Interrupt’’ instruction (such as CLI), doing it in hardware can be very time consuming in some CPUs such as those with deep pipelines. On the other hand, in a virtualized system, the hypervisor need not actually disable interrupts in hardware and simply use a variable to indicate that the specified guest OS's ‘‘Interrupt Flag’’ is set to zero, making it faster to execute the CLI instruction.

18. VMware does binary translation one basic block at a time, then it executes the block and starts translating the next one. Could it translate the entire program in advance and then execute it? If so, what are the advantages and disadvantages of each technique?

It could translate the entire program in advance. The reason for not doing so is that many programs have large pieces of code that are never executed. By translating basic blocks on demand, no unused code is ever translated. A potential disadvantage of on-demand translation is that it might be slightly less efficient to keep starting and stopping the translator, but this effect is probably small. In addition, static analysis and translation of x86 code is complicated due to indirect branches (branches whose targets are computed at run time). This is made worse by the variable-size instructions on the x86. Thus you may not be sure which instructions to translate. Finally, there is the issue of selfmodifying code.

19. What is the difference between a pure hypervisor and a pure microkernel?

A pure hypervisor just emulates the real hardware and nothing else. A pure microkernel is a small operating system that offers basic services to the programs using it. The virtual machines running on a pure hypervisor run traditional operating systems such as Windows and Linux. On top of a microkernel are generally processes that implement operating system services but in a decentralized way.

20. Briefly explain why memory is so difficult to virtualize well in practice? Explain your answer.

If multiple guest OSes all allocate what they think is physical page k to one of their processes, there is a problem. Some way is needed to perform a second mapping between pages because the guests do not really control the physical pages, despite what they may think. This is why nested page tables are needed.

21. Running multiple virtual machines on a PC is known to require large amounts of memory. Why? Can you think of any ways to reduce the memory usage? Explain.

Not only does the machine need memory for the normal (guest) operating system and all its applications, but it also needs memory for the hypervisor functions and data structures needed to execute sensitive instructions on behalf of the guest OS. Type 2 hypervisors have the added cost of the host operating system. Moreover, each virtual machine will have its own operating system, so there will be N operating system copies stored in memory. One way to reduce memory usage would be to identified "shared code" and keep only one copy of this code in memory. For example, a Web hosting company may run multiple VMs, each running an identical version of Linux and an identical copy of the Apache web server code. In this case, the code segment can be shared across VMs, even though the data regions must be private.

22. Explain the concept of shadow page tables, as used in memory virtualization.

Each guest OS will maintain a page table that maps its virtual page numbers to physical frame numbers (on its share of the virtualized memory). In order to prevent different guest operating systems from incorrectly referring to the same physical page number, the hypervisor creates a shadow page table that maps the virtual machine’s virtual page number to the physical frame number provided by the hypervisor.

23. One way to handle guest operating systems that change their page tables using ordinary (nonprivileged) instructions is to mark the page tables as read only and take a trap when they are modified. How else could the shadow page tables be maintained? Discuss the efficiency of your approach vs. the read-only page tables.

Page tables can be modified only by the guest operating system, not the application programs in the guest. When the guest OS is finished modifying the tables, it must switch back to user mode by issuing a sensitive instruction like RETURN FROM TRAP. This instruction will trap and give the hypervisor control. It could then examine the page tables in the guest OS to see if they had been modified. While this could work, all the page tables would have to be checked on every system made by a guest application, that is, every time the guest OS returned to user mode. There could be thousands of these transitions per second, so it is not likely to be as efficient as using read-only pages for the page table.

24. Why are balloon drivers used? Is this cheating?

When a hypervisor runs out of pages, it has no way of figuring out which pages the guest operating systems really value. The solution is to cheat and include balloon drivers in the guests. The hypervisor then signals the balloon drivers to expand their memory usage, forcing the guest operating systems to decide which pages to evict. This is definitely cheating because the hypervisor is not supposed to talk to specific pieces of the guest operating systems. It is not supposed to know what is going on in the virtual machines at all. But this technique solves a problem in a simple way, so everyone pretends there is nothing iffy going on.

25. Descibe a situation in which balloon drivers do not work.

Balloon drivers do not work if the hypervisor does not know anything about the guest operating systems running on its virtual machines. It also does not work if there is no way to include a balloon driver in them, for example, if they do not support loadable drivers and the source code is not available so they cannot be recompiled to include the balloon driver.

26. Explain the concept of deduplication as used in memory virtualization.

Consider a case where multiple virtual machines copies of the same guest OS reside in a system. In this case, it is not necessary to maintain multiple copies of the read-only portion of the OS (such as code segments) in memory. Only one copy needs to be maintained, thereby reducing memory requirements and allowing more virtual machines on a system. This technique is called deduplication. VMware calls this ‘‘transparent page sharing.’’

27. Computers have had DMA for doing I/O for decades. Did this cause any problems before there were I/O MMUs?

Yes. Early DMA hardware used absolute memory addresses. If a guest operating system started a DMA operating to what it thought was physical address k, this would probably not go to the buffer it was supposed to go to and might overwrite something important. Early hypervisors had to rewrite code that used DMA to use addresses that would not cause trouble.

28. Give one advantage of cloud computing over running your programs locally. Give one disadvantage as well.

Using cloud services means you do not have to set up and maintain a computing infrastructure. You may also be able to outsource making backups. Furthermore, if your computing needs change rapidly, you can add or remove machines easily. On the downside, the cloud provider could easily steal your confidential data, and the promised expandability might be illusory if you need extra capacity just at the moment Walmart or some other big customer decides to grab 10,000 machines. Also, the bandwidth between you and the cloud might be an issue. It is likely to be far less than the local bandwidth, so if a lot of data needs to move between you and the cloud that could be an issue. Also, if you are doing real-time work, the bandwidth between you and the cloud could vary wildly from moment to moment, causing trouble.

29. Give an example of IAAS, PAAS, and SAAS.

Obviously there are many, but a provider offering empty virtual x86 machines would be offering IAAS. A provider offering Windows 8 or Linux machines would be offering PAAS. A provider offering a word-processing program, such as Microsoft Word, running in the cloud would be offering software as a service.

30. Why is virtual machine migration important? Under what circumstances might it be useful?

Suppose many virtual machines were started up on a single server. Initially, all of them did about the same amount of work and required the same resources and the situation was fine. Then all of a sudden, one of them began using massive resources (CPU, memory, etc.) disturbing all the other virtual machines. This might be a good time to migrate it to a dedicated server of its own.

31. Migrating virtual machines may be easier than migrating processes, but migration can still be difficult. What problems can arise when migrating a virtual machine?

Physical I/O devices still present problems because they do not migrate with the virtual machine, yet their registers may hold state that is critical to the proper functioning of the system. Think of read or write operations to devices (e.g., the disk) that have been issued but have not yet completed. Network I/O is particularly difficult because other machines will continue to send packets to the hypervisor, unaware that the virtual machine has moved. Even if packets can be redirected to the new hypervisor, the virtual machine will be unresponsive during the migration period, which can be long because the entire virtual machine, including the guest operating system and all processes executing on it, must be moved to the new machine. As a result packets can experience large delays or even packet loss if the device/hypervisor buffers overflow.

32. Why is migration of virtual machines from one machine to another easier than migrating processes from one machine to another?

In order to migrate a specific process, process state information has to be stored and then transferred, including open files, alarms, signal handlers, etc. Errors may creep in during the state capture task leading to potentially incorrect, incomplete or inconsistent state information. In the case of VM migration, the entire memory and disk images are moved to the new system, which is easier.

33. What is the difference between live migration and the other kind (dead migration?)?

Standard (dead) migration consists of stopping the virtual machine and saving its memory image as a file. The file is then transported to the destination, installed in a virtual machine, and restarted. Doing so causes the application to stop for a little while during transport. In many circumstances having the application stop is undesirable. With live migration, the pages of the virtual machine are moved while it is running. After they all arrive at the destination, a check is made to see if any of them have changed since being migrated. If so, they are copied again. This process is repeated until all the pages at the destination are up to date. Working this way (live migration) means applications can be moved with no downtime.

34. What were the three main requirements considered while designing VMware?

The three main requirements were: Compability (ability to run an existing guest OS without any modifications as a virtual machine); Performance (minimal overhead during VM execution; else, users would not choose to run their applications inside a VM) and Isolation (protecting the hardware resources from malicious or otherwise unauthorized access).

35. Why was the enormous number of peripheral devices available a problem when VMware Workstation was first introduced?

There was no way that VMware could have drivers for the thousands of different I/O devices in existence. By having VMware Workstation be a type 2 hypervisor, it could solve the problem by indirectly using the drivers already installed in the host OS.

36. VMware ESXi has been made very small. Why? After all, servers at data centers usually have tens of gigabytes of RAM. What difference does a few tens of megabytes more or less make?

VMware ESXi has been made small so it can be put into the firmware of the servers. When the server is turned on, the BIOS can then copy itself to RAM and start creating virtual machines. This greatly simplifies the booting and startup process.

37. Do an Internet search to find two real-life examples of virtual appliances.

Several examples can be found at: virtualboximages.com. These include various distributions of preinstalled Open Source Operating Systems. For example, rather than get an ISO for a new Linux flavor go through the install process and then get the VM running, it is easier to download the preinstalled VDI. There are similar appliances that run on VMWare. Other examples can be found at: http://www.turnkeylinux.org

Chapter 8

1. Can the USENET newsgroup system or the SETI@home project be considered distributed systems? (SETI@home uses several million idle personal computers to analyze radio telescope data to search for extraterrestrial intelligence.) If so, how do they relate to the categories described in Fig. 8-1?

Both USENET and SETI@home could be described as wide area distributed systems. However, USENET is actually more primitive than the scheme of Fig. 8-1(c), since it does not require any network infrastructure other than point-to-point connections between pairs of machines. Also, since it does no processing work beyond that necessary to ensure proper dissemination of news articles, it could be debated whether it is really a distributed system of the sort we are concerned with in this chapter. SETI@home is a more typical example of a wide area distributed system; data are distributed to remote nodes which then return results of calculations to the coordinating node.

2. What happens if three CPUs in a multiprocessor attempt to access exactly the same word of memory at exactly the same instant?

Depending on how CPUs are connected to memory, one of them gets through first, for example, seizes the bus first. It completes its memory operation, then another one happens, etc. It is not predictable which one goes first, but if the system has been designed for sequential consistency, it should not matter.

3. If a CPU issues one memory request every instruction and the computer runs at 200 MIPS, about how many CPUs will it take to saturate a 400-MHz bus? Assume that a memory reference requires one bus cycle. Now repeat this problem for a system in which caching is used and the caches have a 90% hit rate. Finally, what cache hit rate would be needed to allow 32 CPUs to share the bus without overloading it?

A 200-MIPS machine will issue 200 million memory references/sec, consuming 200 million bus cycles or half of the bus' capacity. It takes only two CPUs to consume the entire bus. Caching drops the number of memory requests/sec to 20 million, allowing 20 CPUs to share the bus. To get 32 CPUs on the bus, each one could request no more than 12.5 million requests/sec. If only 12.5 million of the 200 million of the memory references go out on the bus, the cache miss rate must be 12.5/200, or 6.25%. This means the hit rate is 93.75%.

4. Suppose that the wire between switch 2A and switch 3B in the omega network of Fig. 8-5 breaks. Who is cut off from whom?

CPUs 000, 010, 100, and 110 are cut off from memories 010 and 011.

5. How is signal handling done in the model of Fig. 8-7?

Each CPU manages its own signals completely. If a signal is generated from the keyboard and the keyboard is not assigned to any particular CPU (the usual case), somehow the signal has to be given to the correct CPU to handle.

6. When a system call is made in the model of Fig. 8-8, a problem has to be solved immediately after the trap that does not occur in the model of Fig. 8-7. What is the nature of this problem and how might it be solved?

To issue a system call, a process generates a trap. The trap interrupts its own CPU. Somehow, the information that a slave CPU has had a trap has to be conveyed to the master CPU. This does not happen in the first model. If there are interprocessor trap instructions, this can be used to signal the master. If no such instructions exist, the slave can collect the parameters of the system call and put them in a data structure in memory that the master polls continuously when it is idle.

7. Rewrite the enter region code of Fig. 2-22 using the pure read to reduce thrashing induced by the TSL instruction.

Here is a possible solution:

```
enter region:
        TST LOCK                 |       Test the value of lock
        JNE ENTER REGION         |       If it is nonzero, go try again
        TSL REGISTER,LOCK        |       Copy lock to register and set lock to 1
        CMP REGISTER,#0          |       Was lock zero?
        JNE ENTER REGION         |       If it was nonzero, lock was set, so loop
        RET                      |       Return to caller; critical region entered
```

8. Multicore CPUs are beginning to appear in conventional desktop machines and laptop computers. Desktops with tens or hundreds of cores are not far off. One possible way to harness this power is to parallelize standard desktop applications such as the word processor or the web browser. Another possible way to harness the power is to parallelize

the services offered by the operating system -- e.g., TCP processing -- and commonly-used library services -- e.g., secure http library functions). Which approach appears the most promising? Why?

As noted in the text, we have little experience (and tools) for writing highly parallel desktop applications. Although desktop applications are sometimes multithreaded, the threads are often used to simplify I/O programming and thus they are not compute-intensive threads. The one desktop application area that has some chance at large-scale parallelization is video gaming, where many aspects of the game require significant (parallel) computation. A more promising approach is to parallelize operating system and library services. We have already seen examples of this in current hardware and OS designs. For example, network cards now have on-board parallel processors (network processors) that are used to speed up packet processing and offer higher-level network services at line speeds (e.g., encryption, intrusion detection, etc). As another example, consider the powerful processors found on video cards used to offload video rendering from the main CPU and offer higher-level graphics APIs to applications (e.g., Open GL). One can imagine replacing these special-purpose cards with single-chip multicore processors. Moreover, as the number of cores increases, the same basic approach can be used to parallelize other operating system and common library services.

9. Are critical regions on code sections really necessary in an SMP operating system to avoid race conditions or will mutexes on data structures do the job as well?

Probably locks on data structures are sufficient. It is hard to imagine anything a piece of code could do that is critical and does not involve some kernel data structure. All resource acquisition and release uses data structures, for example. While it cannot be proven, probably locks on data structures are enough.

10. When the TSL instruction is used for multiprocessor synchronization, the cache block containing the mutex will get shuttled back and forth between the CPU holding the lock and the CPU requesting it if both of them keep touching the block. To reduce bus traffic, the requesting CPU executes one TSL every 50 bus cycles, but the CPU holding the lock always touches the cache block between TSL instructions. If a cache block consists of 16 32-bit words, each of which requires one bus cycle to transfer, and the bus runs at 400 MHz, what fraction of the bus bandwidth is eaten up by moving the cache block back and forth?

It takes 16 bus cycles to move the block and it goes both ways for each TSL. Thus every 50 bus cycles, 32 of them are wasted on moving the cache block. Consequently, 64% of the bus bandwidth is wasted on cache transfers.

11. In the text, it was suggested that a binary exponential backoff algorithm be used between uses of TSL to poll a lock. It was also suggested to have a maximum delay between polls. Would the algorithm work correctly if there were no maximum delay?

Yes, it would, but the interpoll time might end up being very long, degrading performance. But it would be correct, even without a maximum.

12. Suppose that the TSL instruction was not available for synchronizing a multiprocessor. Instead, another instruction, SWP, was provided that atomically swapped the contents of a register with a word in memory. Could that be used to provide multiprocessor synchronization? If so, how could it be used? If not, why does it not work?

It is just as good as TSL. It is used by preloading a 1 into the register to be used. Then that register and the memory word are atomically swapped. After the instruction, the memory word is locked (i.e., has a value of 1). Its previous value is now contained in the register. If it was previously locked, the word has not been changed and the caller must loop. If it was previously unlocked, it is now locked.

13. In this problem you are to compute how much of a bus load a spin lock puts on the bus. Imagine that each instruction executed by a CPU takes 5 nsec. After an instruction has completed, any bus cycles needed, for example, for TSL are carried out. Each bus cycle takes an additional 10 nsec above and beyond the instruction execution time. If a process is attempting to enter a critical region using a TSL loop, what fraction of the bus bandwidth does it consume? Assume that normal caching is working so that fetching an instruction inside the loop consumes no bus cycles.

The loop consists of a TSL instruction (5 nsec), a bus cycle (10 nsec), and a JMP back to the TSL instruction (5 nsec). Thus in 20 nsec, 1 bus cycle is requested occupying 10 nsec. The loop consumes 50% of the bus.

14. Affinity scheduling reduces cache misses. Does it also reduce TLB misses? What about page faults?

Affinity scheduling has to do with putting the right thread on the right CPU. Doing so might well reduce TLB misses since these are kept inside each CPU. On the other hand, it has no effect on page faults, since if a page is in memory for one CPU, it is in memory for all CPUs.

15. For each of the topologies of Fig. 8-16, what is the diameter of the interconnection network? Count all hops (host-router and router-router) equally for this problem.

(a) 2 (b) 4 (c) 8 (d) 5 (e) 3 (f) 4.

16. Consider the double-torus topology of Fig. 8-16(d) but expanded to size k × k. What is the diameter of the network? (Hint: Consider odd k and even k differently.)

On a grid, the worst case is nodes at opposite corners trying to communicate. However, with a torus, opposite corners are only two hops apart. The worst case is one corner trying to talk to a node in the middle. For odd k, it takes (k − 1)/2 hops to go from a corner to the middle horizontally and another (k − 1)/2 hops to go to the middle vertically, for a total of k − 1. For even k, the middle is a square of four dots in the middle, so the worst case is from a corner to the most distant dot in that four-dot square. It takes k/2 hops to get there horizontally and also k/2 vertically, so the diameter is k.

17. The bisection bandwidth of an interconnection network is often used as a measure of its capacity. It is computed by removing a minimal number of links that splits the network into two equal-size units. The capacity of the removed links is then added up. If there are many ways to make the split, the one with the minimum bandwidth is the bisection bandwidth. For an interconnection network consisting of an 8 × 8 × 8 cube, what is the bisection bandwidth if each link is 1 Gbps?

The network can be sliced in two by a plane through the middle, giving two systems, each with a geometry of 8 × 8 × 4. There are 128 links running between the two halves, for bisection bandwidth of 128 Gbps.

18. Consider a multicomputer in which the network interface is in user mode, so only three copies are needed from source RAM to destination RAM. Assume that moving a 32-bit word to or from the network interface board takes 20 nsec and that the network itself operates at 1 Gbps. What would the delay be for a 64-byte packet being sent from source to destination if we could ignore the copying time? What is it with the copying time? Now consider the case where two extra copies are needed, to the kernel on the sending side and from the kernel on the receiving side. What is the delay in this case?

If we just consider the network time, we get 1 nsec per bit or 512-nsec delay per packet. To copy 64 bytes 4 bytes at a time, 320 nsec are needed on each side, or 640 nsec total. Adding the 512-nsec wire time, we get 1132 nsec total. If two additional copies are needed, we get 1792 nsec.

19. Repeat the previous problem for both the three-copy case and the five-copy case, but this time compute the bandwidth rather than the delay.

If we consider only the wire time, a 1-Gbps network delivers 125 MB/sec. Moving 64 bytes in 1152 nsec is 55.6 MB/sec. Moving 64 bytes in 1792 nsec is 35.7 MB/sec.

20. When transferring data from RAM to a network interface, pinning a page can be used, but suppose that system calls to pin and unpin pages each take 1 µsec. Copying takes 5 bytes/nsec using DMA but 20 nsec per byte using programmed I/O. How big does a packet have to be before pinning the page and using DMA is worth it?

The time to move k bytes by programmed I/O is 20k nsec. The time for DMA is 2000 + 5k nsec. Equating these and solving for k we get the breakeven point at 133 bytes.

21. When a procedure is scooped up from one machine and placed on another to be called by RPC, some problems can occur. In the text, we pointed out four of these: pointers, unknown array sizes, unknown parameter types, and global variables. An issue not discussed is what happens if the (remote) procedure executes a system call. What problems might that cause and what might be done to handle them?

Clearly, the wrong thing happens if a system call is executed remotely. Trying to read a file on the remote machine will not work if the file is not there. Also, setting an alarm on the remote machine will not send a signal back to the calling machine. One way to handle remote system calls is to trap them and send them back to the originating site for execution.

22. In a DSM system, when a page fault occurs, the needed page has to be located. List two possible ways to find the page.

First, on a broadcast network, a broadcast request could be made. Second, a centralized database of who has which page could be maintained. Third, each page could have a home base, indicated by the upper k bits of its virtual address; the home base could keep track of the location of each of its pages.

23. Consider the processor allocation of Fig. 8-24. Suppose that process H is moved from node 2 to node 3. What is the total weight of the external traffic now?

In this split, node 1 has A, E, and G, node 2 has B and F, and node 3 has C, D, H, and I. The cut between nodes 1 and 2 now contains AB and EB for a weight of 5. The cut between nodes 2 and 3 now contains CD, CI, FI, and FH for a weight of 14. The cut between nodes 1 and 3 now contains EH and GH for a weight of 8. The sum is 27.

24. Some multicomputers allow running processes to be migrated from one node to another. Is it sufficient to stop a process, freeze its memory image, and just ship that off to a different node? Name two hard problems that have to be solved to make this work.

The table of open files is kept in the kernel, so if a process has open files, when it is unfrozen and tries to use one of its files, the new kernel does not know about them. A second problem is the signal mask, which is also stored on the original kernel. A third problem is that if an alarm is pending, it will go off on the wrong machine. In general, the kernel is full of bits and pieces of information about the process, and they have to be successfully migrated as well.

25. Why is there a limit to cable length on an Ethernet network?

Ethernet nodes must be able to detect collisions between packets, so the propagation delay between the two most widely separated nodes must be less than the duration of the shortest packet to be sent. Otherwise the sender may fully transmit a packet and not detect a collision even though the packet suffers a collision close to the other end of the cable.

26. In Fig. 8-27, the third and fourth layers are labeled Middleware and Application on all four machines. In what sense are they all the same across platforms, and in what sense are they different?

The middleware runs on different operating systems, so the code is clearly different because the embedded system calls are different. What they have in common is producing a common interface to the application layer above them. If the application layer makes calls only to the middleware layer and no system calls, then all the versions of it can have the same source code. If they also make true system calls, these will differ.

27. Figure 8-30 lists six different types of service. For each of the following applications, which service type is most appropriate?
(a) Video on demand over the Internet.
(b) Downloading a Web page.

The most appropriate services are
(a) Unreliable connection.
(b) Reliable byte stream.

28. DNS names have a hierarchical structure, such as sales.general-widget.com or cs.uni.edu One way to maintain the DNS database would be as one centralized database, but that is not done because it would get too many requests/sec. Propose a way that the DNS database could be maintained in practice.

It is maintained hierarchically. There is a worldwide server for .edu that knows about all the universities and a .com server that knows about all the names ending in .com. Thus to look up cs.uni.edu, a machine would first look up uni at the .edu server, then go there to ask about cs, and so on.

29. In the discussion of how URLs are processed by a browser, it was stated that connections are made to port 80. Why?

A computer may have many processes waiting for incoming connections. These could be the Web server, mail server, news server, and others. Some way is needed to make it possible to direct an incoming connection to some particular process. That is done by having each process listen to a specific port. It has been agreed upon that Web servers will listen to port 80, so incoming connections directed to the Web server are sent to port 80. The number itself was an arbitrary choice, but some number had to be chosen.

30. Migrating virtual machines may be easier than migrating processes, but migration can still be difficult. What problems can arise when migrating a virtual machine?

Physical I/O devices still present problems because they do not migrate with the virtual machine, yet their registers may hold state that is critical to the proper functioning of the system. Think of read or write operations to devices (e.g., the disk) that have been issued but have not yet completed. Network I/O is particularly difficult because other machines will continue to send packets to the hypervisor, unaware that the virtual machine has moved. Even if packets can be redirected to the new hypervisor, the virtual machine will be unresponsive during the migration period, which can be long because the entire virtual machine, including the guest operating system and all processes executing on it, must be moved to the new machine. As a result packets can experience large delays or even packet loss if the device/hypervisor buffers overflow.

31. When a browser fetches a Web page, it first makes a TCP connection to get the text on the page (in the HTML language). Then it closes the connection and examines the page. If there are figures or icons, it then makes a separate TCP connection to fetch each one. Suggest two alternative designs to improve performance here.

One way would be for the Web server to package the entire page, including all the images, in a big zip file and send the whole thing the first time so that only one connection is needed. A second way would be to use a connectionless protocol like UDP. This would eliminate the connection overhead, but would require servers and browsers to do their own error control.

32. When session semantics are used, it is always true that changes to a file are immediately visible to the process making the change and never visible to processes on other machines. However, it is an open question as to whether or not they should be immediately visible to other processes on the same machine. Give an argument each way.

Having the value of a read depend on whether a process happens to be on the same machine as the last writer is not at all transparent. This argues for making changes only visible to the process making the changes. On the other hand, having a single cache manager per machine is easier and cheaper to implement. Such a manager becomes a great deal more complicated if it has to maintain multiple copies of each modified file, with the value returned depending on who is doing the reading.

33. When multiple processes need access to data, in what way is object-based access better than shared memory?

Shared memory works with whole pages. This can lead to false sharing, in which access to unrelated variables that happen to lie on the same page causes thrashing. Putting each variable on a separate page is wasteful. Object-based access eliminates these problems and allows a finer grain of sharing.

34. When a Linda in operation is done to locate a tuple, searching the entire tuple space linearly is very inefficient. Design a way to organize the tuple space that will speed up searches on all in operations.

Hashing on any of the fields of the tuple when it is inserted into the tuple space does not help because the in may have mostly formal parameters. One optimization that always works is noting that all the fields of both out and in are typed. Thus the type signature of all tuples in the tuple space is known, and the tuple type needed on an in is also known. This suggests creating a tuple subspace for each type signature. For example, all the (int, int, int) tuples go into one space, and all the (string, int, float) tuples go into a different space. When an in is executed, only the matching subspace has to be searched.

35. Copying buffers takes time. Write a C program to find out how much time it takes on a system to which you have access. Use the clock or times functions to determine how long it takes to copy a large array. Test with different array sizes to separate copying time from overhead time.

36. Write C functions that could be used as client and server stubs to make an RPC call to the standard printf function, and a main program to test the functions. The client and server should communicate by means of a data structure that could be transmitted over a network. You may impose reasonable limits on the length of the format string and the number, types, and sizes of the variables your client stub will accept.

37. Write a program that implements the sender-initiated and receiver-initiated load balancing algorithms described in Sec. 8.2. The algorithms should take as input a list of newly created jobs specified as (creating processor, start time, required CPU time) where the creating processor is the number of the CPU that created the job, the start time is the time at which the job was created, and the required CPU time is the amount of CPU time the job needs to complete (specified in seconds). Assume a node is overloaded when it has one job and a second job is created. Assume a node is underloaded when it has no jobs. Print the number of probe messages sent by both algorithms under heavy and light workloads. Also print the maximum and minimum number of probes sent by any host and received by any host. To create the workloads, write two workload generators. The first should simulate a heavy workload, generating, on average, N jobs every AJL seconds, where AJL is the average job length and N is the number of processors. Job lengths can be a mix of long and short jobs, but the average job length must be AJL. The jobs should be randomly created (placed) across all processors. The second generator should simulate a light load, randomly generating N/3 jobs every AJL seconds. Play with other parameter settings for the workload generators and see how it affects the number of probe messages.

38. One of the simplest ways to implement a publish/subscribe system is via a centralized broker that receives published articles and distributes them to the appropriate subscribers. Write a multithreaded application that emulates a broker-based pub/sub system. Publisher and subscriber threads may communicate with the broker via (shared) memory. Each message should start with a length field followed by that many characters. Publishers send messages to the broker where the first line of the message contains a hierarchical subject line separated by dots followed by one or more lines that comprise the published article. Subscribers send a message to the broker with a single line containing a hierarchical interest line separated by dots expressing the articles they are interested in. The interest line may contain the wildcard symbol “*”. The broker must respond by sending all (past)

articles that match the subscriber's interest. Articles in the message are separated by the line "BEGIN NEW ARTICLE." The subscriber should print each message it receives along with its subscriber identity (i.e., its interest line). The subscriber should continue to receive any new articles that are posted and match its interests. Publisher and subscriber threads can be created dynamically from the terminal by typing "P" or "S" (for publisher or subscriber) followed by the hierarchical subject/interest line. Publishers will then prompt for the article. Typing a single line containing "." will signal the end of the article. (This project can also be implemented using processes communicating via TCP.)

Chapter 9

1. Confidentiality, integrity, and availability are three components of security. Describe an application that integrity and availability but not confidentiality, an application that requires confidentiality and integrity but not (high) availability, and an application that requires confidentiality, integrity, and availability

Popular news services require integrity and availability but not confidentiality. Backup storage systems require confidentiality and integrity but not necessarily 24/7 availability. Finally, banking services require confidentiality, integrity, and availability.

2. One of the techniques to build a secure operating system is to minimize the size of TCB. Which of the following functions needs to be implemented inside the TCB and which can be implemented outside TCB:
(a) Process context switch;
(b) Read a file from disk;
(c) Add more swapping space;
(d) Listen to music;
(e) Get the GPS coordinates of a smartphone.

(a) and (c) have to be a part of the TCB and (b), (d) and (e) can be implemented outside the TCB.

3. What is a covert channel? What is the basic requirement for a covert channel to exist?

A covert channel is an unauthorized communications channel that can be created in a system by observing and manipulating measurable performance characteristics of the system. The key requirement of a covert channel to exist is to have some shared system resources such as CPU, disk, or network that can be used for sending secret signals.

4. In a full access-control matrix, the rows are for domains and the columns are for objects. What happens if some object is needed in two domains?

It is just entered into the matrix twice. In the example given in the text, printer1 is in two domains simultaneously. There is no problem here.

5. Suppose that a system has 5000 objects and 100 domains at some time. 1% of the objects are accessible (some combination of r, w and x) in all domains, 10% are accessible in two domains, and the remaining 89% are accessible in only one domain. Suppose one unit of space is required to store an access right (some combination of r, w, x), object ID, or a domain ID. How much space is needed to store the full protection matrix, protection matrix as ACL, and protection matrix as capability list?

Full protection matrix: 5000 × 100 = 500,000 units of space. ACL: 50 × 100 (1% objects accessible in all domains; 100 entries/ACL) + 500 × 2 (10% objects accessible in two domains; two entries/ACL) + 4450 × 1 (89% objects accessible in one domain; one entry/ACL) = 10,450 units of space

6. Explain which implementation of the protection matrix is more suitable for the following operations:
(a) Granting read access to a file for all users.
(b) Revoking write access to a file from all users.
(c) Granting write access to a file to John, Lisa, Christie, and Jeff.
(d) Revoking execute access to a file from Jana, Mike, Molly, and Shane.

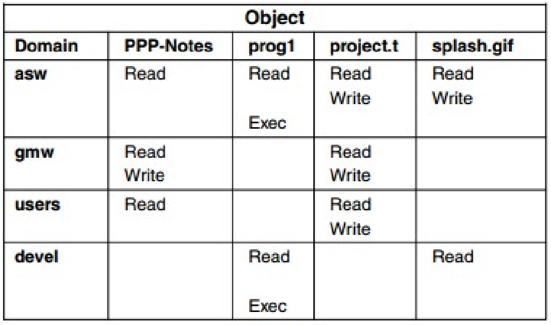
(a) Capability list
(b) Capability list
(c) ACL
(d) ACL

7. Two different protection mechanisms that we have discussed are capabilities and access-control lists. For each of the following protection problems, tell which of these mechanisms can be used.
(a) Ken wants his files readable by everyone except his office mate.
(b) Mitch and Steve want to share some secret files.
(c) Linda wants some of her files to be public.

To make a file readable by everyone except one person, access-control lists are the only possibility. For sharing private files, access-control lists or capabilities can be used. To make files public, access-control lists are easiest, but it may also be possible to put a capability for the file or files in a well-known place in a capability system.

8. Represent the ownerships and permissions shown in this UNIX directory listing as a protection matrix. (Note: asw is a member of two groups: users and devel; gmw is a member only of users.) Treat each of the two users and two groups as a domain, so that the matrix has four rows (one per domain) and four columns (one per file).
– rw– r– – r– – 2 gmw users 908 May 26 16:45 PPP– Notes
– rwx r– x r– x 1 asw dev el 432 May 13 12:35 prog1
– rw– rw– – – – 1 asw users 50094 May 30 17:51 project.t
– rw– r– – – – – 1 asw dev el 13124 May 31 14:30 splash.gif

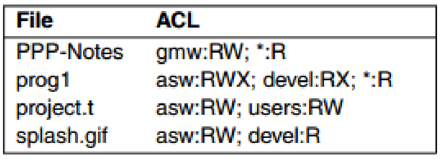
Here is the protection matrix:



	Object			
Domain	**PPP-Notes**	**prog1**	**project.t**	**splash.gif**
asw	Read	Read Exec	Read Write	Read Write
gmw	Read Write		Read Write	
users	Read		Read Write	
devel		Read Exec		Read

9. Express the permissions shown in the directory listing of the previous problem as access-control lists.

. The ACLs are as follows:



File	ACL
PPP-Notes	gmw:RW; *:R
prog1	asw:RWX; devel:RX; *:R
project.t	asw:RW; users:RW
splash.gif	asw:RW; devel:R

Assume that * means all

10. Modify the ACL from the previous problem for one file to grant or deny an access that cannot be expressed using the UNIX rwx system. Explain this modification.

If asw wants to allow gmw but no other members of users to look at splash.gif, he could modify the ACL to asw:RW; devel:R; gmw:R.

11. Suppose there are four security levels, 1, 2 and 3. Objects A and B are at level 1, C and D are at level 2, and E and F are at level 3. Processes 1 and 2 are at level 1, 3 and 4 are at level 2, and 5 and 6 are at level 3. For each of the following operations, specify whether they are permissible under Bell-LaPadula model, Biba model, or both.
(a) Process 1 writes object D
(b) Process 4 reads object A
(c) Process 3 reads object C
(d) Process 3 writes object C
(e) Process 2 reads object D
(f) Process 5 writes object F
(g) Process 6 reads object E
(h) Process 4 write object E
(i) Process 3 reads object F

Bell-LaPadula model only: (a), (b), (h)
Biba model only: (e), (i)
Both: (c), (d), (f), (g)

12. In the Amoeba scheme for protecting capabilities, a user can ask the server to produce a new capability with fewer rights, which can then be given to a friend. What happens if the friend asks the server to remove even more rights so that the friend can give it to someone else?

The server will verify that the capability is valid and then generate a weaker capability. This is legal. After all, the friend can just give away the capability it already has. Giving it the power to give away something even weaker is not a security threat. If you have the ability to give away, say, read/write power, giving away read-only power is not a problem.

13. In Fig. 9-11, there is no arrow from process B to object 1. Would such an arrow be allowed? If not, what rule would it violate?

No. That would be writing down, which violates the * property.

14. If process-to-process messages were allowed in Fig. 9-11, what rules would apply to them? For process B in particular, to which processes could it send messages and which not?

A process writing to another process is similar to a process writing to a file. Consequently, the * property would have to hold. A process could write up but not write down. Process B could send to C, D, and E, but not to A.

15. Consider the steganographic system of Fig. 9-14. Each pixel can be represented in a color space by a point in the three-dimensional system with axes for the R, G, and B values. Using this space, explain what happens to the color resolution when steganography is employed as it is in this figure.

In the original photo, the R, G, and B axes each allow discrete integral values from 0 to 255, inclusive. This means that there are 2^{24} valid points in color space that a pixel can occupy. When 1 bit is taken away for the covert channel, only the even values are allowed (assuming the secret bit is replaced by a 0 everywhere). Thus as much of the space is covered, but the color resolution is only half as good. In total, only 1/8 of the colors can be represented. The disallowed colors are mapped onto the adjacent color all of whose values are even numbers, for example, the colors (201, 43, 97), (201, 42, 97), (200, 43, 96), and (200, 42, 97) now all map onto the point (200, 42, 96) and can no longer be distinguished.

16. Break the following monoalphabetic cipher. The plaintext, consisting of letters only, is a well-known excerpt from a poem by Lewis Carroll.
kfd ktbd fzm eubd kfd pzyiom mztx ku kzyg ur bzha kfthcm
ur mfudm zhx mftnm zhx mdzythc pzq ur ezsszcdm zhx gthcm
zhx pfa kfd mdz tm sutythc fuk zhx pfdkfdi ntcm fzld pthcm
sok pztk z stk kfd uamkdim eitdx sdruid pd fzld uoi efzk
rui mubd ur om zid uok ur sidzkf zhx zyy ur om zid rzk
hu foiia mztx kfd ezindhkdi kfda kfzhgdx ftb boef rui kfzk

the time has come the walrus said to talk of many things
of ships and shoes and sealing wax of cabbages and kings
of why the sea is boiling hot and whether pigs have wings
but wait a bit the oysters cried before we have our chat
for some of us are out of breath and all of us are fat
no hurry said the carpenter they thanked him much for that

From Through the Looking Glass (Tweedledum and Tweedledee).

17. Consider a secret-key cipher that has a 26 × 26 matrix with the columns headed by ABC ... Z and the rows also named ABC ... Z. Plaintext is encrypted two characters at a time. The first character is the column; the second is the row. The cell formed by the intersection of the row and column contains two ciphertext characters. What constraint must the matrix adhere to and how many keys are there?

The constraint is that no two cells contain the same two letters, otherwise decryption would be ambiguous. Thus each of the 676 matrix elements contains a different one of the 676 digrams. The number of different combinations is thus 676! This is a very big number.

18. Consider the following way to encrypt a file. The encryption algorithm uses two n-byte arrays, A and B. The first n bytes are read from the file into A. Then A[0] is copied to B[i], A[1] is copied to B[j], A[2] is copied to B[k], etc. After all n bytes are copied to the B array, that array is written to the output file and n more bytes are read into A. This procedure continues until the entire file has been encrypted. Note that here encryption is not being done by replacing characters with other ones, but by changing their order. How many keys

have to be tried to exhaustively search the key space? Give an advantage of this scheme over a monoalphabetic substitution cipher.

The number of permutations is n!, so this is the size of the key space. One advantage is that the statistical attack based on properties of natural languages does not work because an E really does represent an E, and so on.

19. Secret-key cryptography is more efficient than public-key cryptography, but requires the sender and receiver to agree on a key in advance. Suppose that the sender and receiver have never met, but there exists a trusted third party that shares a secret key with the sender and also shares a (different) secret key with the receiver. How can the sender and receiver establish a new shared secret key under these circumstances?

The sender picks a random key and sends it to the trusted third party encrypted with the secret key that they share. The trusted third party then decrypts the random key and recrypts it with the secret key it shares with the receiver. This message is then sent to the receiver

20. Give a simple example of a mathematical function that to a first approximation will do as a one-way function.

A function like $y = x^k$ is easy to compute but taking the kth root of y is far more difficult.

21. Suppose that two strangers A and B want to communicate with each other using secretkey cryptography, but do not share a key. Suppose both of them trust a third party C whose public key is well known. How can the two strangers establish a new shared secret key under these circumstances?

A and B pick random keys Ka and Kb and send them to C encrypted with C's public key. C picks a random key K and sends it to A encrypted using Ka and to B encrypted using Kb

22. As Internet cafes become more widespread, people are going to want ways of going to one anywhere in the world and conducting business there. Describe a way to produce signed documents from one using a smart card (assume that all the computers are equipped with smart-card readers). Is your scheme secure?

One way to sign a document would be for the smart card to read in the document, make a hash of it, and then encrypt the hash with the user's private key, stored in the card. The encrypted hash would be output to the Internet cafe computer, but the secret key would never leave the smart card, so the scheme is secure.

23. Natural-language text in ASCII can be compressed by at least 50% using various compression algorithms. Using this knowledge, what is the steganographic carrying capacity for ASCII text (in bytes) of a 1600 × 1200 image stored using the low-order bits of each pixel? How much is the image size increased by the use of this technique (assuming no encryption or no expansion due to encryption)? What is the efficiency of the scheme, that is, its payload/(bytes transmitted)?

The image contains 1,920,000 pixels. Each pixel has 3 bits that can be used, given a raw capacity of 720,000 bytes. If this is effectively doubled due to compressing the text before storing it, the image can hold ASCII text occupying about 1,440,000 bytes before compression. There is no expansion due to the steganography. The image with the hidden data is the same size as the original image. The efficiency is 25%. This can be easily seen from the fact that 1 bit of every 8-

bit color sample contains payload, and the compression squeezes 2 bits of ASCII text per payload bit. Thus per 24-bit pixel, effectively 6 bits of ASCII text are being encoded.

24. Suppose that a tightly knit group of political dissidents living in a repressive country are using steganography to send out messages to the world about conditions in their country. The government is aware of this and is fighting them by sending out bogus images containing false steganographic messages. How can the dissidents try to help people tell the real messages from the false ones?

The dissidents could sign the messages using a private key and then try to widely publicize their public key. This might be possible by having someone smuggle it out of the country and then post it to the Internet from a free country

25. Go to www.cs.vu.nl/ ast and click on covered writing link. Follow the instructions to extract the plays. Answer the following questions:
(a) What are the sizes of the original-zebras and zebras files?
(b) What plays are secretly stored in the zebras file?
(c) How many bytes are secretly stored in the zebras file?

(a) Both files are 2.25 MB.
(b) Hamlet, Julius Caesar, King Lear, Macbeth, and Merchant of Venice.
(c) There are six text files secretly stored, totaling about 722 KB.

26. Not having the computer echo the password is safer than having it echo an asterisk for each character typed, since the latter discloses the password length to anyone nearby who can see the screen. Assuming that passwords consist of upper and lowercase letters and digits only, and that passwords must be a minimum of fiv e characters and a maximum of eight characters, how much safer is not displaying anything?

It depends on how long the password is. The alphabet from which passwords is built has 62 symbols. The total search space is $62^5 + 62^6 + 62^7 + 62^8$, which is about 2×10^{14}. If the password is known to be k characters, the search space is reduced to only 62k . The ratio of these is thus $2 \times 10^{14}/62^k$. For k from 5 to 8, these values are 242,235, 3907, 63, and 1. In other words, learning that the password is only five characters reduces the search space by a factor of 242,235 because all the long passwords do not have to be tried. This is a big win. However, learning that it is eight characters does not help much because it means that all the short (easy) passwords can be skipped.

27. After getting your degree, you apply for a job as director of a large university computer center that has just put its ancient mainframe system out to pasture and switched over to a large LAN server running UNIX. You get the job. Fifteen minutes after you start work, your assistant bursts into your office screaming: “Some students have discovered the algorithm we use for encrypting passwords and posted it on the Internet.” What should you do?

Try to calm the assistant. The password encryption algorithm is public. Passwords are encrypted by the login program as soon as they are typed in, and the encrypted password is compared to the entry in the password file.

28. The Morris-Thompson protection scheme with n-bit random numbers (salt) was designed to make it difficult for an intruder to discover a large number of passwords by encrypting common strings in advance. Does the scheme also offer protection against a

student user who is trying to guess the superuser password on his machine? Assume the password file is available for reading.

No, it does not. The student can easily find out what the random number for his superuser is. This information is in the password file unencrypted. If it is 0003, for example, then he just tries encrypting potential passwords as Susan0003, Boston0003, IBMPC0003, and so on. If another user has password Boston0004, he will not discover it, however.

29. Suppose the password file of a system is available to a cracker. How much extra time does the cracker need to crack all passwords if the system is using the Morris-Thompson protection scheme with n-bit salt versus if the system is not using this scheme?

Suppose there are m users in the systems. The cracker can then collect the m salt values, assumed all different here. The cracker will have to try encrypting each guessed password m times, once with each of the m salts used by the system. Thus the cracker's time to crack all passwords is increased m-fold.

30. Name three characteristics that a good biometric indicator must have in order to be useful as a login authenticator.

There are many criteria. Here are a few of them:
It should be easy and painless to measure (not blood samples).
There should be many values available (not eye color).
The characteristic should not change over time (not hair color).
It should be difficult to forge the characteristic (not weight).

31. Authentication mechanisms are divided into three categories: Something the user knows, something the user has, and something the user is. Imagine an authentication system that uses a combination of these three categories. For example, it first asks the user to enter a login and password, then insert a plastic card (with magnetic strip) and enter a PIN, and finally provide fingerprints. Can you think of two drawbacks of this design?

The combination of different authentication mechanisms will provide stronger authentication. However, there are two drawbacks. First, the cost involved in implementing this system is high. The system incurs the cost of three different authentication mechanisms. Second, this authentication mechanism puts extra burden on the user. The user has to remember his login/password, carry his plastic card and remember its PIN, and has to go through the process of fingerprint matching. The key issue is that all this will increase the time it takes to authenticate a user, resulting in increased user dissatisfaction.

32. A computer science department has a large collection of UNIX machines on its local network. Users on any machine can issue a command of the form rexec machine4 who and have the command executed on machine4, without having the user log in on the remote machine. This feature is implemented by having the user's kernel send the command and his UID to the remote machine. Is this scheme secure if the kernels are all trustworthy? What if some of the machines are students' personal computers, with no protection?

If all the machines can be trusted, it works OK. If some cannot be trusted, the scheme breaks down, because an untrustworthy machine could send a message to a trustworthy machine asking it to carry out some command on behalf of the superuser. The machine receiving the message has no way of telling if the command really did originate with the superuser, or with a student.

33. Lamport's one-time password scheme uses the passwords in reverse order. Would it not be simpler to use f (s) the first time, f (f(s)) the second time, and so on?

It would not work to use them forward. If an intruder captured one, he would know which one to use next time. Using them backward prevents this danger.

34. Is there any feasible way to use the MMU hardware to prevent the kind of overflow attack shown in Fig. 9-21? Explain why or why not.

No, it is not feasible. The problem is that array bounds are not checked. Arrays do not line up with page boundaries, so the MMU is not of any help. Furthermore, making a kernel call to change the MMU on every procedure call would be prohibitively expensive.

35. Describe how stack canaries work and how they can be circumvented by the attackers.

The attacker exploits the race condition by executing an operation like symbolic link after the access rights are checked and before the file is opened. If the file system access is a transaction, the access rights check and file open will be a part of a single transaction and the serializability property will ensure that symbolic link cannot be created in between. The main downside of this approach is that the performance of the file system will suffer since transactions incur extra overhead.

36. The TOCTOU attack exploits race condition between the attacker and the victim. One way to prevent race conditions is make file system accesses transactions. Explain how this approach might work and what problems might arise?

The compiler could insert code on all array references to do bounds checking. This feature would prevent buffer overflow attacks. It is not done because it would slow down all programs significantly. In addition, in C it is not illegal to declare an array of size 1 as a procedure parameter and then reference element 20, but clearly the actual array whose address has been passed had better have at least 20 elements. In addition, C functions like memset and memcpy are used all the time to copy entire structures at once even if they contain separate arrays. In other words, they are buffer overflows by design.

37. Name a C compiler feature that could eliminate a large number of security holes. Why is it not more widely implemented?

If the capabilities are used to make it possible to have small protection domains, no; otherwise yes. If an editor, for example, is started up with only the capabilities for the file to be edited and its scratch file, then no matter what tricks are lurking inside the editor, all it can do is read those two files. On the other hand, if the editor can access all of the user's objects, then Trojan horses can do their dirty work, capabilities or not.

38. Can the Trojan-horse attack work in a system protected by capabilities?

From a security point of view, it would be ideal. Used blocks sometimes are exposed, leaking valuable information. From a performance point of view, zeroing blocks wastes CPU time, thus degrading performance.

9. When a file is removed, its blocks are generally put back on the free list, but they are not erased. Do you think it would be a good idea to have the operating system erase each block

before releasing it? Consider both security and performance factors in your answer, and explain the effect of each.

For any operating system all programs must either start execution at a known address or have a starting address stored in a known position in the program file header. (a) The virus first copies the instructions at the normal start address or the address in the header to a safe place, and then inserts a jump to itself into the code or its own start address into the header. (b) When done with its own work, the virus executes the instructions it borrowed, followed by a jump to the next instruction that would have been executed, or transfers control to the address it found in the original header.

40. How can a parasitic virus (a) ensure that it will be executed before its host program, and (b) pass control back to its host after doing whatever it does?

A master boot record requires only one sector, and if the rest of the first track is free, it provides space where a virus can hide the original boot sector as well as a substantial part of its own code. Modern disk controllers read and buffer entire tracks at a time, so there will be no perceivable delay or sounds of additional seeks as the extra data are read.

41. Some operating systems require that disk partitions must start at the beginning of a track. How does this make life easier for a boot-sector virus?

C programs have extension .c. Instead of using the access system call to test for execute permission, examine the file name to see if it ends in .c. This code will do it:

```
char *file name;
int len;
file name = dp->d name;
len = strlen(file name);
if (strcmp(&file name[len − 2], ".c") == 0) infect(s);
```

42. Change the program of Fig. 9-28 so that it finds all the C programs instead of all the executable files.

They probably cannot tell, but they can guess that XORing one word within the virus with the rest will produce valid machine code. Their computers can just try each virus word in turn and see if any of them produce valid machine code. To slow down this process, Virgil can use a better encryption algorithm, such as using different keys for the odd and even words, and then rotating the first word left by some number of bits determined by a hash function on the keys, rotating the second word that number of bits plus one, and so on.

43. The virus in Fig. 9-33(d) is encrypted. How can the dedicated scientists at the antivirus lab tell which part of the file is the key so that they can decrypt the virus and reverse engineer it? What can Virgil do to make their job a lot harder?

The compressor is needed to compress other executable programs as part of the process of infecting them.

44. The virus of Fig. 9-33(c) has both a compressor and a decompressor. The decompressor is needed to expand and run the compressed executable program. What is the compressor for?

Most viruses do not want to infect a file twice. It might not even work. Therefore it is important to be able to detect the virus in a file to see if it is already infected. All the techniques used to make it hard for antivirus software to detect viruses also make it hard for the virus itself to tell which files have been infected.

45. Name one disadvantage of a polymorphic encrypting virus from the point of view of the virus writer.

First, running the Ifdisk program from the hard disk is a mistake. It may be infected and it may infect the boot sector. It has to be run from the original CDROM or a write-protected floppy disk. Second, the restored files may be infected. Putting them back without cleaning them may just reinstall the virus.

46. Often one sees the following instructions for recovering from a virus attack:
1. Boot the infected system.
2. Back up all files to an external medium.
3. Run fdisk (or a similar program) to format the disk.
4. Reinstall the operating system from the original CD-ROM.
5. Reload the files from the external medium.
Name two serious errors in these instructions.

Yes, but the mechanism is slightly different from Windows. In UNIX a companion virus can be installed in a directory on the search path ahead of the one in which the real program lives. The most common example is to insert a program ls in a user directory, which effectively overrides /bin/ls because it is found first.

47. Are companion viruses (viruses that do not modify any existing files) possible in UNIX? If so, how? If not, why not?

Obviously, executing any program from an unknown source is dangerous. Self-extracting archives can be especially dangerous, because they can release multiple files into multiple directories, and the extraction program itself could be a Trojan horse. If a choice is available it is much better to obtain files in the form of an ordinary archive, which you can then extract with tools you trust.

48. Self-extracting archives, which contain one or more compressed files packaged with an extraction program, are frequently used to deliver programs or program updates. Discuss the security implications of this technique.

Since a rootkit is designed to conceal its existence, it infects operating system, libraries and applications. So, any detection software that relies on any system functionality cannot be trusted. Essentially, a rootkit subverts the software that is intended to find it. As a result, rootkit detectors have to rely on external components such as scanning from an external TCB.

49. Why are rookits extremely difficult or almost impossible to detect as opposed to viruses and worms?

Since a rootkit can subvert the recovery software, for example, by resetting the system restore points, this approach to system recovery does not work.

50. Could a machine infected with a rootkit be restored to good health by simply rolling back the software state to a previously stored system restore point?

It is not possible to write such a program, because if such a program is possible, a cracker can use that program to circumvent virus checking in the virusladen program she writes.

51. Discuss the possibility of writing a program that takes another program as input and determines if that program contains a virus.

The source IP address of all incoming packets can be inspected. The second set of rules will drop all incoming IP packets with source IP addresses belonging to known spammers.

52. Section 9.10.1 describes a set of firewall rules that limit outside access to only three services. Describe another set of rules that you can add to this firewall to further restrict access to these services.

It does not matter. If zero fill is used, then S2 must contain the true prefix as an unsigned integer in the low-order k bits. If sign extension is used, then S2 must also be sign extended. As long as S2 contains the correct results of shifting a true address, it does not matter what is in the unused upper bits of S2.

53. On some machines, the SHR instruction used in Fig. 9-38(b) fills the unused bits with zeros; on others the sign bit is extended to the right. For the correctness of Fig. 9-38(b), does it matter which kind of shift instruction is used? If so, which is better?

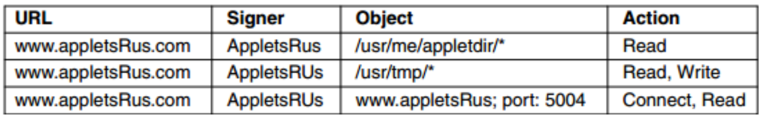
Existing browsers come preloaded with the public keys of sev eral trusted third parties such as the Verisign Corporation. Their business consists of verifying other companies' public keys and making up certificates for them. These certificates are signed by, for example, Verisign's private key. Since Verisign's public key is built into the browser, certificates signed with its private key can be verified.

54. To verify that an applet has been signed by a trusted vendor, the applet vendor may include a certificate signed by a trusted third party that contains its public key. However, to read the certificate, the user needs the trusted third party's public key. This could be provided by a trusted fourth party, but then the user needs that public key. It appears that there is no way to bootstrap the verification system, yet existing browsers use it. How could it work?

First, Java does not provide pointer variables. This limits a process' ability to overwrite an arbitrary memory location. Second, Java does not allow user-controlled storage allocation (malloc/free). This simplifies memory management. Third, Java is a type-safe language, ensuring that a variable is used in exactly way it is supposed to be, based on its type.

55. Describe three features that make Java a better programming language than C to write secure programs.

Here are the rules.



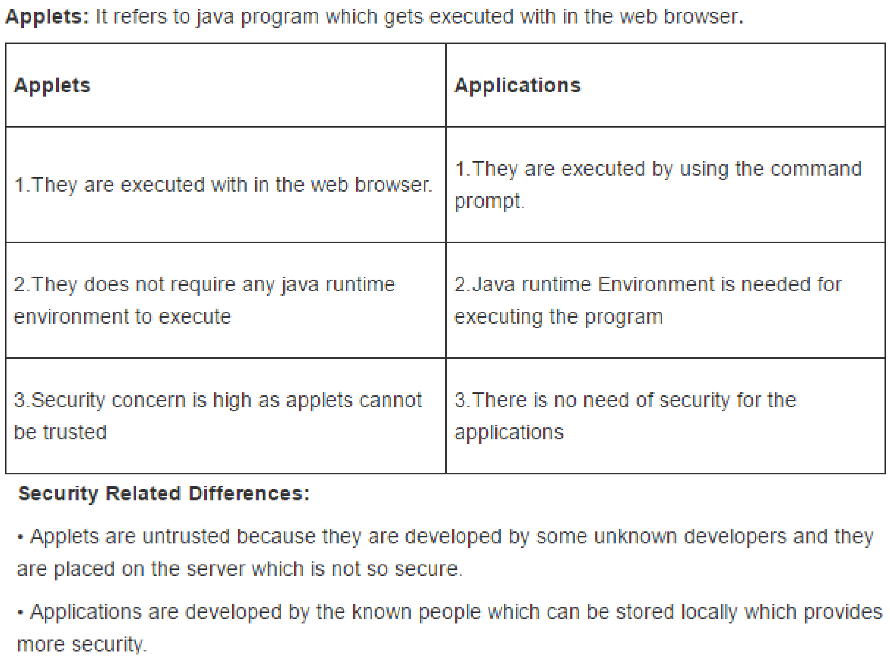
URL	Signer	Object	Action
www.appletsRus.com	AppletsRus	/usr/me/appletdir/*	Read
www.appletsRus.com	AppletsRUs	/usr/tmp/*	Read, Write
www.appletsRus.com	AppletsRUs	www.appletsRus; port: 5004	Connect, Read

56. Assume that your system is using JDK 1.2. Show the rules (similar to those in Figure 9-40) you will use to allow an applet from www.appletsRus.com to run on your machine. This

applet may download additional files from www.appletsRus.com, read/write files in /usr/tmp/, and also read files from /usr/me/appletdir.

An applet is any small program that performs a specific task such as filling up a form. The main difference between an applet and an application is that an applet runs within the scope of a dedicated larger program such rendering a webpage. Applets are typical examples of auxiliary applications that don’t monopolize the user’s attention and are intended to be easily accessible. Since applets are typically downloaded from a third party, they essentially contain foreign code designed to run on a user’s machine. They may contain viruses, worms, or other harmful code.

57. How are applets different from applications? How does this difference relate to security?



Applets: It refers to java program which gets executed with in the web browser.

Applets	Applications
1.They are executed with in the web browser.	1.They are executed by using the command prompt.
2.They does not require any java runtime environment to execute	2.Java runtime Environment is needed for executing the program
3.Security concern is high as applets cannot be trusted	3.There is no need of security for the applications

Security Related Differences:

- Applets are untrusted because they are developed by some unknown developers and they are placed on the server which is not so secure.
- Applications are developed by the known people which can be stored locally which provides more security.

58. Write a pair of programs, in C or as shell scripts, to send and receive a message by a covert channel on a UNIX system. (Hint: A permission bit can be seen even when a file is otherwise inaccessible, and the sleep command or system call is guaranteed to delay for a fixed time, set by its argument.) Measure the data rate on an idle system. Then create an artificially heavy load by starting up numerous different background processes and measure the data rate again.

59. Several UNIX systems use the DES algorithm for encrypting passwords. These systems typically apply DES 25 times in a row to obtain the encrypted password. Download an implementation of DES from the Internet and write a program that encrypts a password and checks if a password is valid for such a system. Generate a list of 10 encrypted passwords using the Morris-Thomson protection scheme. Use 16-bit salt for your program.

60. Suppose a system uses ACLs to maintain its protection matrix. Write a set of management functions to manage the ACLs when (1) a new object is created; (2) an object is deleted; (3) a new domain is created; (4) a domain is deleted; (5) new access rights (a combination of r, w, x) are granted to a domain to access an object; (6) existing access rights of a domain to access an object are revoked; (7) new access rights are granted to all domains to access an object; (8) access rights to access an object are revoked

from all domains.

61. Implement the program code outlined in Sec. 9.7.1 to see what happens when there is buffer overflow. Experiment with different string sizes.

62. Write a program that emulates overwriting viruses outlined in Sec. 9.9.2 under the heading "Executable Program Viruses". Choose an existing executable file that you know can be overwritten without any harm. For the virus binary, choose any harmless executable binary