CHAPTER 1
QUESTIONS
ANSWERS

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.

In Section 1.4, 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.

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.

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.

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).

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 userlevel programs cannot interfere with them.

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

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.

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.

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.

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.

Which of the following instructions should be allowed only in kernel mode? (a) Disable all interrupts. (b) Read the time-of-day clock. (c) Set the time-of-day clock. (d) Change the memory map.

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

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.

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.

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 × 0.99 × 10 nsec (word is in RAM, but not in the cache) + 0.05 × 0.01 × 10,000,000 nsec (word on disk only) = 5001.445 nsec = 5.001445 μsec

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.

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.

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.

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.

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.

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

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.

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.

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

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.

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.

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.

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.

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.

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.

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.

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

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

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?

The conversions are straightforward:

(a) A nanoyear is $10^{-9} \times 365 \times 24 \times 3600 = 31.536$ msec.

(b) 1 meter

(c) There are 2^{50} bytes, which is 1,099,511,627,776 bytes.

(d) It is 6×10^{24} kg or 6×10^{27} g.

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.

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.

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.)