# solutions to the exercises

## Chapter 1

1.1 In a multiprogramming and time-sharing environment, several users share the system simultaneously. This situation can result in various security problems.

a. What are two such problems?

b. Can we ensure the same degree of security in a time-shared machine as in a dedicated machine? Explain your answer.

Answer:

a. Stealing or copying one’ s programs or data; using system resources (CPU, memory, disk space, peripherals) without proper accounting.

b. Probably not, since any protection scheme devised by humans can inevitably be broken by a human, and the more complex the scheme, the more difficult it is to feel confident of its correct implementation.

1.2 The issue of resource utilization shows up in different forms in different types of operating systems. List what resources must be managed

carefully in the following settings: a. Mainframe or minicomputer systems b. Workstations connected to servers c. Handheld computers

Answer:

a. Mainframes:memory and CPU resources, storage, network bandwidth.

b. Workstations: memory and CPU resouces

c. Handheld computers: power consumption, memory resources.

1.3 Under what circumstances would a user be better off using a timesharing system rather than a PC or single-user workstation?

Answer: When there are few other users, the task is large, and the hardware is fast, time-sharingmakes sense. The full power of the system can be brought to bear on the user’ s problem. The problemcan be solved faster than on a personal computer. Another case occurs when lots of other users need resources at the same time.

A personal computer is best when the job is small enough to

be executed reasonably on it and when performance is sufficient to execute the program to the user’s satisfaction.

1.4 Which of the functionalities listed below need to be supported by the operating system for the following two settings: (a) handheld devices and (b) real-time systems. a. Batch programming b. Virtual memory c. Time sharing

Answer: For real-time systems, the operating system needs to support virtual memory and time sharing in a fair manner. For handheld systems,the operating system needs to provide virtual memory, but does not need to provide time-sharing. Batch programming is not necessary in both settings.

1.5 Describe the differences between symmetric and asymmetric multiprocessing.What are three advantages and one disadvantage of multiprocessor systems?

Answer: Symmetric multiprocessing treats all processors as equals, and I/O can be processed on any CPU. Asymmetric multiprocessing has one master CPU and the remainder CPUs

are slaves. The master distributes tasks among the slaves, and I/O is usually done by the master only.

Multiprocessors can save money by not duplicating power supplies,housings, and peripherals. They can execute programs more quickly and can have increased reliability. They are also more complex in both hardware and software than uniprocessor systems.

## 1.6 How do clustered systems differ from multiprocessor systems? What is required for two machines belonging to a cluster to cooperate to provide a highly available service?

Answer: Clustered systems are typically constructed by combining multiple computers into a single system to perform a computational task distributed across the cluster. Multiprocessor systems on the other hand could be a single physical entity comprising of multiple CPUs. A clustered system is less tightly coupled than a multiprocessor system.Clustered systems communicate using messages, while processors in a multiprocessor system could communicate using shared memory.

In order for twomachines to provide a highly available service,

the state on the two machines should be replicated and should be consistently updated. When one of the machines fail, the other could then take-over the functionality of the failed machine.

## 1.7 Distinguish between the client-server and peer-to-peer models of distributed systems.

Answer: The client-server model firmly distinguishes the roles of the client and server. Under this model, the client requests services that are provided by the server. The peer-to-peer model doesn’ t have such strict roles. In fact, all nodes in the system are considered peers and thus may act as either clients or servers or both. A node may request a service from another peer, or the node may in fact provide such a service to other peers in the system.

For example, let’ s consider a system of nodes that share cooking recipes.Under the client-server model, all recipes are stored with the server. If a client wishes to access a recipe, it must request the recipe from the specified server. Using the peer-to-peer model, a peer node could ask other peer nodes for the specified recipe. The node (or perhaps nodes) with the

requested recipe could provide it to the requesting node. Notice how each peer may act as both a client (i.e. it may request recipes) and as a server (it may provide recipes.)

1.8 Consider a computing cluster consisting of twonodes running adatabase.Describe two ways in which the cluster software can manage access to the data on the disk. Discuss the benefits and disadvantages of each.

Answer: Consider the following two alternatives: asymmetric clustering and parallel clustering. With asymmetric clustering, one host runs the database application with the other host simply monitoring it. If the server fails, the monitoring host becomes the active server. This is appropriate for providing redundancy. However, it does not utilize the potential processing power of both hosts. With parallel clustering, the database application can run in parallel on both hosts. The difficulty implementing parallel clusters is providing some form of distributed locking mechanism for files on the shared disk.

1.9 How are network computers different from traditional

personal computers? Describe some usage scenarios in which it is advantageous to use network computers.

Answer: A network computer relies on a centralized computer for most of its services. It can therefore have a minimal operating system to manage its resources. A personal computer on the other hand has to be capable of providing all of the required functionality in a standalonemanner without relying on a centralized manner. Scenarios where administrative costs are high and where sharing leads to more efficient use of resources are precisely those settings where network computers are preferred.

1.10 What is the purpose of interrupts? What are the differences between a trap and an interrupt? Can traps be generated intentionally by a user program? If so, for what purpose?

Answer: An interrupt is a hardware-generated change-of-flow within the system. An interrupt handler is summoned to deal with the cause of the interrupt; control is then returned to the interrupted context and instruction. A trap is a

software-generated interrupt. An interrupt can be used to signal the completion of an I/O to obviate the need for device polling. A trap can be used to call operating system routines or to catch arithmetic errors.

1.11 Direct memory access is used for high-speed I/O devices in order to avoid increasing the CPU's execution load.

a. How does the CPU interface with the device to coordinate the transfer?

b. How does the CPU know when the memory operations are complete?

c. The CPU is allowed to execute other programs while the DMA controller is transferring data. Does this process interfere with the execution of the user programs? If so, describe what forms of interference are caused.

Answer: The CPU can initiate a DMA operation by writing values into special registers that can be independently accessed by the device.The device initiates the corresponding operation once it receives a command from the CPU. When the device is finished with its operation, it interrupts the CPU to indicate the completion of the operation.

Both the device and the CPU can be accessing memory simultaneously.The memory controller provides access to the memory bus in a fair manner to these two entities. A CPU might therefore be unable to issue memory operations at peak speeds since it has to compete with the device in order to obtain access to the memory bus.

1.12 Some computer systems do not provide a privileged mode of operation in hardware. Is it possible to construct a secure operating system for these computer systems? Give arguments both that it is and that it is not possible.

Answer: An operating system for a machine of this type would need to remain in control (or monitor mode) at all times. This could be accomplished by two methods:

a. Software interpretation of all user programs (like some BASIC,Java, and LISP systems, for example). The software interpreter would provide, in software, what the hardware does not provide.

b. Require meant that all programs be written in high-level languages so that all object code is compiler-produced. The compiler would generate (either in-line or by function calls)

the protection checks that the hardware is missing.

1.13 Give two reasons why caches are useful.What problems do they solve? What problems do they cause? If a cache can be made as large as the device for which it is caching (for instance, a cache as large as a disk), why not make it that large and eliminate the device?

Answer: Caches are useful when two or more components need to exchange data, and the components perform transfers at differing speeds.Caches solve the transfer problem by providing a buffer of intermediate speed between the components. If the fast device finds the data it needs in the cache, it need not wait for the slower device. The data in the cache must be kept consistent with the data in the components. If a omponent has a data value change, and the datum is also in the cache, the cache must also be updated. This is especially a problem on multiprocessor systemswhere more than one process may be accessing a datum.Acomponent may be eliminated by an equal-sized cache, but only if: (a) the cache and the component have equivalent state-saving capacity (that is,if the component

retains its data when electricity is removed, the cache must retain data as well), and (b) the cache is affordable, because faster storage tends to be more expensive.

1.14 Discuss, with examples, how the problem of maintaining coherence of cached data manifests itself in the following processing environments:

a. Single-processor systems

b. Multiprocessor systems

c. Distributed systems

Answer: In single-processor systems, the memory needs to be updated when a processor issues updates to cached values. These updates can be performed immediately or in a lazy manner. In amultiprocessor system,different processors might be caching the same memory location in its local caches. When updates are made, the other cached locations need to be invalidated or updated. In distributed systems, consistency of cached memory values is not an issue. However, consistency problems might arise when a client caches file data.

1.15 Describe a mechanism for enforcing memory protection

in order to prevent a program from modifying the memory associated with other programs.

Answer: The processor could keep track of what locations are associated with each process and limit access to locations that are outside of a program’s extent. Information regarding the extent of a program’s memory could be maintained by using base and limits registers and by performing a check for every memory access.

1.16 What network configuration would best suit the following environments?

a. A dormitory floor

b. A university campus

c. A state

d. A nation

Answer:

a. A dormitory floor - A LAN.

b. A university campus - A LAN, possible a WAN for very large campuses.

c. A state - AWAN.

d. A nation - A WAN.

1.17 Define the essential properties of the following types of operating systems:

a. Batch

b. Interactive

c. Time sharing

d. Real time

e. Network

f. Parallel

g. Distributed

h. Clustered

i. Handheld

Answer:

a. Batch. Jobs with similar needs are batched together and run through the computer as a group by an operator or automatic job sequencer. Performance is increased by attempting to keep CPU and I/O devices busy at all times through buffering, off-line operation, spooling, andmultiprogramming. Batch is good for executing large jobs that need little interaction; it can be submitted and picked up later.

b. Interactive. This system is composed of many short transactions where the results of the next transaction may be unpredictable. Response time needs to be short (seconds) since the user submits and waits for the result.

c. Time sharing. This systems uses CPU scheduling and multiprogramming to provide economical interactive use of a system. The CPU switches rapidly from one user to another. Instead of having a job defined by spooled card images, each program reads its next control card from the terminal, and output is normally printed immediately to the screen.

d. Real time. Often used in a dedicated application, this system reads information from sensors and must respond within a fixed amount of time to ensure correct performance.

e. Network. Provides operating system features across a network such as file sharing.

f. SMP. Used in systems where there are multiple CPU’s each running the same copy of the operating system.Communication takes place across the system bus.

g. Distributed.This system distributes computation among several physical processors. The processors do not share memory or a clock. Instead, each processor has its own local memory. They communicate with each other through various

communication lines, such as a high-speed bus or local area network.

h. Clustered. A clustered system combines multiple computers into a single system to perform computational task distributed across the cluster.

i. Handheld. A small computer system that performs simple tasks such as calendars, email, and web browsing. Handheld systems differ from traditional desktop systemswith smallermemory and display screens and slower processors.

## 1.18 What are the tradeoffs inherent in handheld computers?

Answer: Handheld computers are much smaller than traditional desktop PC’ s. This results in smaller memory, smaller screens, and slower processing capabilities than a standard desktop PC. Because of these limitations,most handhelds currently can perform only basic tasks such as calendars, email, and simple word processing. However, due to their small size, they are quite portable and, when they are equipped with wireless access, can provide remote access to electronic mail and the world wide web.

# Chapter 2

## 2.1 The services and functions provided by an operating system can be divided into two main categories. Briefly describe the two categories and discuss how they differ.

Answer: One class of services provided by an operating system is to enforce protection between different processes running concurrently in the system. Processes are allowed to access only thosememory locations that are associated with their address spaces. Also, processes are not allowed to corrupt files associated with other users. A process is also not allowed to access devices directly without operating system ntervention.

The second class of services provided by an operating system is to provide new functionality that is not supported directly by the underlying hardware. Virtual memory and file systems are two such examples of new services provided by an operating system.

## 2.2 List five services provided by an operating system that are

designed to make it more convenient for users to use the computer system. In what cases it would be impossible for user-level programs to provide these services? Explain.

Answer:

• Program execution. The operating system loads the contents (or sections) of a file into memory and begins its execution. A user-level program could not be trusted to properly allocate CPU time.

• I/O operations. Disks, tapes, serial lines, and other devices must be communicated with at a very low level. The user need only specify the device and the operation to perform on it, while the system converts that request into device- or controller-specific commands. User-level programs cannot be trusted to only access devices they should have access to and to only access them when they are otherwise unused.

• File-system manipulation. There are many details in file creation, deletion, allocation, and naming that users should not have to perform. Blocks of disk space are used by files and must be tracked. Deleting a file requires removing the name file information and freeing the allocated blocks. Protections must also be checked to assure proper file access. User

programs could neither ensure adherence to protection methods nor be trusted to allocate only free blocks and deallocate blocks on file deletion.

• Communications. Message passing between systems requires messages be turned into packets of information, sent to the network controller,transmitted across a communications medium, and reassembled by the destination system. Packet ordering and data correction must take place. Again, user programs might not coordinate access to the network device, or they might receive packets destined for other processes.

• Error detection. Error detection occurs at both the hardware and software levels. At the hardware level, all data transfers must be inspected to ensure that data have not been corrupted in transit. All data on media must be checked to be sure they have not changed since they were written to the media. At the software level, media must be checked for data consistency; for instance, do the number of allocated and

unallocated blocks of storage match the total number on the device. There, errors are frequently process-independent (for instance, the corruption of data on a disk), so there must be a

global program (the operating system) that handles all types of errors. Also, by having errors processed by the operating system, processes need not contain code to catch and correct all the errors possible on a system.

2.3 Describe three general methods for passing parameters to the operating system.

Answer:

a. Pass parameters in registers

b. Registers pass starting addresses of blocks of parameters

c. Parameters can be placed, or pushed, onto the stack by the program, and popped off the stack by the operating system.

2.4 Describe how you could obtain a statistical profile of the amount of time spent by a program executing different sections of its code.Discuss the importance of obtaining such a statistical profile.

Answer: One could issue periodic timer interrupts and monitor what instructions or what sections of code are currently executing when the interrupts are delivered. A statistical

profile of which pieces of code were active should be consistent with the time spent by the program in different sections of its code. Once such a statistical profile has been obtained, the programmer could optimize those sections of code that are consuming more of the CPU resources.

## 2.5 What are the five major activities of an operating system in regard to file management?

Answer:

- The creation and deletion of files
- The creation and deletion of directories
- The support of primitives for manipulating files and directories
- The mapping of files onto secondary storage
- The backup of files on stable (nonvolatile) storage media

## 2.6 What are the advantages and disadvantages of using the same systemcall interface for manipulating both files and devices?

Answer: Each device can be accessed as though it was a file in

the file system. Since most of the kernel deals with devices through this file interface,it is relatively easy to add a new device driver by implementing the hardware-specific code to support this abstract file interface. Therefore,this benefits the development of both user program code, which can bewritten to access devices and files in the samemanner, and device driver code, which can be written to support a well-defined API. The disadvantage with using the same interface is that it might be difficult to capture the functionality of certain devices within the context of the file access API, thereby either resulting in a loss of functionality or a loss of performance. Some of this could be overcome by the use of ioctl operation that provides a general purpose interface for processes to invoke operations on devices.

2.7 What is the purpose of the command interpreter? Why is it usually separate from the kernel? Would it be possible for the user to develop a new command interpreter using the system-call interface provided by the operating system?

Answer: It reads commands from the user or from a file of commands and executes them, usually by turning them into

one or more system calls. It is usually not part of the kernel since the command interpreter is subject to changes. An user should be able to develop a new command interpreter using the system-call interface provided by the operating system. The command interpreter allows an user to create and manage processes and also determine ways by which they communicate (such as through pipes and files). As all of this functionality could be accessed by an user-level program using the system calls, it should be possible for the user to develop a new command-line interpreter.

## 2.8 What are the two models of interprocess communication? What are the strengths and weaknesses of the two approaches?

Answer: The two models of interprocess communication are message-passing model and the shared-memory model.

## 2.9 Why is the separation of mechanism and policy desirable?

Answer: Mechanism and policymust be separate to ensure that systems are easy to modify. No two system installations

are the same, so each installation may want to tune the operating system to suit its needs.With mechanism and policy separate, the policy may be changed at will while the mechanism stays unchanged. This arrangement provides a more flexible system.

2.10 Why does Java provide the ability to call from a Java program native methods that are written in, say, C or C++? Provide an example of a situation in which a native method is useful.

Answer: Java programs are intended to be platform I/O independent. Therefore, the language does not provide access to most specific system resources such as reading fromI/O devices or ports. To perform a system I/O specific operation, you must write it in a language that supports such features (such as C or C++.) Keep in mind that a Java program that calls a native method written in another language will no longer be architecture-neutral.

2.11 It is sometimes difficult to achieve a layered approach if two components of the operating system are dependent on

each other. Identify a scenario in which it is unclear how to layer two system components that require tight coupling of their functionalities.

Answer: The virtual memory subsystem and the storage subsystem are typically tightly-coupled and requires careful design in a layered system due to the following interactions. Many systems allow files to be mapped into the virtual memory space of an executing process. On the other hand, the virtualmemory subsystem typically uses the storage system to provide the backing store for pages that do not currently reside in memory. Also, updates to the filesystem are sometimes buffered in physical memory before it is flushed to disk, thereby requiring careful coordination of the usage of memory between the virtual memory subsystem and the filesystem.

2.12 What is the main advantage of the microkernel approach to system design? How do user programs and system services interact in amicrokernel architecture? What are the disadvantages of using the microkernel approach?

Answer: Benefits typically include the following (a) adding a new service does not require modifying the kernel, (b) it is more secure as more operations are done in user mode than in kernel mode, and (c) a simpler kernel design and functionality typically results in a more reliable operating system. User programs and system services interact in a microkernel architecture by using interprocess communication mechanisms such asmessaging. These messages are conveyed by the operating system. The primary disadvantage of the microkernel architecture are the overheads associated with interprocess communication and the frequent use of the operating system’ s messaging functions in order to enable the user process and the system service to interact with each other.

2.13 In what ways is the modular kernel approach similar to the layered approach? In what ways does it differ from the layered approach?

Answer: The modular kernel approach requires subsystems to interact with each other through carefully constructed interfaces that are typically narrow (in terms of the

functionality that is exposed to external modules). The layered kernel approach is similar in that respect. However, the layered kernel imposes a strict ordering of subsystems such that subsystems at the lower layers are not allowed to invoke operations corresponding to the upper-layer subsystems. There are no such restrictions in the modular-kernel approach, wherein modules are free to invoke each other without any constraints.

2.14 What is the main advantage for an operating-system designer of using a virtual-machine architecture? What is the main advantage for a user?

Answer: The system is easy to debug, and security problems are easy to solve. Virtual machines also provide a good platform for operating system research sincemany different operating systemsmay run on one physical system.

2.15 Why is a just-in-time compiler useful for executing Java programs?

Answer: Java is an interpreted language. This means that the

JVM interprets the bytecode instructions one at a time. Typically,most interpreted environments are slower than running native binaries, for the interpretation process requires converting each instruction into native machine code. A just-in-time (JIT) compiler compiles the bytecode for a method into native machine code the first time the method is encountered. This means that the Java program is essentially running as a native application(of course, the conversion process of the JIT takes time as well
but not as much as bytecode interpretation.) Furthermore, the JIT caches compiled code so that it may be reused the next time the method is encountered.A Java program that is run by a JIT rather than a traditional interpreter typically runs much faster.

2.16 What is the relationship between a guest operating system and a host operating system in a system like VMware? What factors need to be considered in choosing the host operating system?

Answer: A guest operating system provides its services by mapping them onto the functionality provided by the host

operating system. A key issue that needs to be considered in choosing the host operating system is whether it is sufficiently general in terms of its system-call interface in order to be able to support the functionality associated with the guest operating system.

2.17 The experimental Synthesis operating system has an assembler incorporatedwithin the kernel. To optimize system-call performance, the kernel assembles routineswithin kernel space to minimize the path that the system call must take through the kernel. This approach is the antithesis of the layered approach, in which the path through the kernel is extended to make building the operating system easier. Discuss the pros and cons of the Synthesis approach to kernel design and to system-performance optimization.

Answer: Synthesis is impressive due to the performance it achieves through on-the-fly compilation. Unfortunately, it is difficult to debug problems within the kernel due to the fluidity of the code. Also, such compilation is system specific, making Synthesis difficult to port (a new compiler must be written for each architecture).

2.18 In Section 2.3,we described a program that copies the contents of one file to a destination file. This program works by first prompting the user for the name of the source and destination files. Write this program using either the Windows32 or POSIX API. Be sure to include all necessary error checking, including ensuring that the source file exists. Once you have correctly designed and tested the program, if you used a system that supports it, run the program using a utility that traces system calls. Linux systems provide the ptrace utility, and Solaris systems use the truss or dtrace command. On Mac OS X, the ktrace facility provides similar functionality.

Answer:

**//Solution 1:** The Copy program implemented with file system calls of Linux.

```
//This program was written by Wendell on Mar. 5,2008.
// Usage: copy src dst
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
```

```
#include <unistd.h>

#define BUFSIZE 8192

int main( int argc, char ** argv)
{
   if (argc!=3)
   {
      printf("\n usage: copy src dst\n");
      return -1;
   }
   int src, dst;
   char buf[BUFSIZE];
   int n;
   src=open(argv[1], O_RDONLY);
   dst=open(argv[2], O_RDWR | O_CREAT | O_TRUNC, S_IRUSR
| S_IWUSR | S_IXUSR);
   while ( ( n = read(src, buf, BUFSIZE) ) > 0)
   {
       if (write(dst, buf, n) != n)
                printf("write error!");
   }
```

```
    if (n<0)
        printf("read error !");
    close(src);
    close(dst);
    exit(0);
}
```

**//solution 2:** using Windows CopyFile

```
#include <windows.h>
#include <stdio.h>
#define BUF_SIZE 256

int main (int argc, LPTSTR argv [])
{
    HANDLE hIn, hOut;
    DWORD nIn, nOut;
    CHAR Buffer [BUF_SIZE];
    if (argc != 3) {
        printf ("Usage: cp file1 file2\n");
        return 1;
    }
```

```
    hIn = CreateFile (argv [1], GENERIC_READ,
FILE_SHARE_READ, NULL,
            OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
    if (hIn == INVALID_HANDLE_VALUE) {
        printf ("Cannot open input file. Error: %x\n",
GetLastError ());
        return 2;
    }

    hOut = CreateFile (argv [2], GENERIC_WRITE, 0, NULL,
            CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
    if (hOut == INVALID_HANDLE_VALUE) {
        printf ("Cannot open output file. Error: %x\n",
GetLastError ());
        return 3;
    }
    while (ReadFile (hIn, Buffer, BUF_SIZE, &nIn, NULL) &&
nIn > 0) {
        WriteFile (hOut, Buffer, nIn, &nOut, NULL);
        if (nIn != nOut) {
            printf ("Fatal write error: %x\n", GetLastError ());
            return 4;
```

```
        }
    }
    CloseHandle (hIn);
    CloseHandle (hOut);
    return 0;
}
```

**//solution 3:** using Windows API

```
#include <windows.h>
#include <stdio.h>
#define BUF_SIZE 256

int main (int argc, LPTSTR argv [])
{
    if (argc != 3) {
        printf ("Usage: cp file1 file2\n");
        return 1;
    }
    if (!CopyFile (argv [1], argv [2], FALSE)) {
        printf ("CopyFile Error: %x\n", GetLastError ());
        return 2;
```

```
    }
    return 0;
}

/* Sorry, The copy program hasn’t finished now using POSIX
API, Solaris API or Mac Os X API. */
```

# Chapter 3

## 3.1 Describe the differences among short-term, medium-term, and long-term scheduling.

Answer:

- **Short-term** (CPU scheduler) —— selects from jobs in memory those jobs that are ready to execute and allocates the CPU to them.
- **Medium-term** —— used especially with time-sharing systems as an intermediate scheduling level. A swapping scheme is implemented to remove partially run programs from memory and reinstate them later to continue where they left off.
- **Long-term** (job scheduler) —— determines which jobs are brought into memory for processing.

The primary difference is in the frequency of their execution. The short-term must select a new process quite often. Long-term is used much less often since it handles placing jobs in the system and may wait a while for a job to finish

before it admits another one.

3.2 Describe the actions taken by a kernel to context-switch between pro-cesses.

Answer: In general, the operating system must save the state of the currently running process and restore the state of the process scheduled to be run next. Saving the state of a process typically includes the values of all the CPU registers in addition to memory allocation. Context switches must also perform many architecture-specific operations, including flushing data and instruction caches.

3.3 Consider the RPC mechanism. Describe the undesirable circumstances that could arise from not enforcing either the "at most once" or "exactly once" semantics. Describe possible uses for a mechanism that had neither of these guarantees.

Answer: If an RPC mechanism could not support either the “at most once” or “at least once” semantics, then the RPC server cannot guarantee that a remote procedure will not be invoked multiple occurrences. Consider if a remote procedure

were withdrawing money from a bank account on a system that did not support these semantics. It is possible that a single invocation of the remote procedure might lead to multiple withdrawals on the server.

For a system to support either of these semantics generally requires the server maintain some form of client state such as the timestamp described in the text.

If a system were unable to support either of these sematics, then such a system could only safely provide remote procedures that do not alter data or provide time-sensitive results. Using our bank accunt as an example, we certainly require "at most once" or "at least once" semantics for performing a withdrawal (or deposit!) However, an inquiry into an account balance or other accunt information such as name, address, etc. does not require these semantics.

3.4 Using the program shown in Figure 3.24, explain what will be output at Line A.

Answer:

PARENT: value = 5

3.5 What are the benefits and detriments of each of the following? Consider both the systems and the programmers'levels.

a. Symmetric and asymmetric communication b. Automatic and explicit buffering

c. Send by copy and send by reference

d. Fixed-sized and variable-sized messages

Answer:

a. **Symmetric and asymmetric communication**—A benefit of symmetric communication is that it allows a rendezvous between the

sender and receiver. A disadvantage of a blocking send is that a rendezvous may not be required and the message could be delivered asynchronously; received at a point of no interest to the sender. As a result, message-passing systems often provide both forms of synchronization.

b. **Automatic and explicit buffering**—Automatic buffering provides a queue with indefinite length; thus ensuring the sender will never have to block while waiting to copy a message. There are no specifications how automatic buffering will be provided; one scheme may reserve sufficiently large

memory where much of the memory is wasted. Explicit buffering specifies how large the buffer is. In this situation, the sender may be blocked while waiting for available space in the queue. However, it is less likely memory will be wasted with explicit buffering.

c. **Send by copy and send by reference**—Send by copy does not allow the receiver to alter the state of the parameter; send by reference does allow it. A benefit of send by reference is that it allows the programmer to write a distributed version of a centralized application. Java's RMI provides both, however passing a parameter by reference requires declaring the parameter as a remote object as well.

d. **Fixed-sized and variable-sized messages** — The implications of this are mostly related to buffering issues; with fixed-size messages, a buffer with a specific size can hold a known number of messages. The number of variable-sized messages that can be held by such a buffer is unknown. Consider how Windows 2000 handles this situation: with fixed-sized messages (anything < 256 bytes), the messages are copied from the address space of the sender to the address space of the receiving process. Larger messages (i.e. variable-sized messages) use shared memory to pass the

message.

3.6 The Fibonacci sequence is the series of numbers 0, 1, 1, 2, 3, 5, 8, .... Formally, it can be expressed as:

$fib_0 = 0$

$fib_1 = 1$

$fib_n = fib_{n-1} + fib_{n-2}$

Write a C program using the fork() system call that that generates the Fibonacci sequence in the child process. The number of the sequence will be provided in the command line. For example, if 5 is provided, the first five numbers in the Fibonacci sequence will be output by the child process. Because the parent and child processes have their own copies of the data, it will be necessary for the child to output the sequence.

Have the parent invoke the wait() call to wait for the child process to complete before exiting the program. Perform necessary error checking to ensure that a non-negative number is passed on the command line.

Answer: // This program solves exercise 3.6

```
int main(int argc, char *argv[])
```

```
{
      if (argc != 2)
            exit(0);

pid_t pid;
int i, a, b, fib;

int n = atoi(argv[1]);

    /* fork another process */
    pid = fork();

    if (pid < 0) { /* error occurred */
        fprintf(stderr, "Fork Failed\n");
        exit(-1);
    }
    else if (pid == 0) { /* child process */
                        if (n == 1)
                              printf("0\n");
                        else if (n == 2)
                              printf("0, 1\n");
                        else if (n > 2) {
```

```
                    a = 0;
                    b = 1;
                    printf("0, 1,");

                    for (i = 3; i < n; i++) {
                        fib = a + b;
                        printf("%d,",fib);
                        a = b;
                        b = fib;
                    }

                    printf("%d\n",a+b);
                }
            }
        else { /* parent process */
            /* parent will wait for the child to complete */
            wait(NULL);

            exit(0);
        }
}
```

3.7 Repeat the preceding exercise, this time using the CreateProcess() in the Win32 API. In this instance, you will need to specify a separate program to be invoked from CreateProcess(). It is this separate program that will run as a child process outputting the Fibonacci sequence. Perform necessary error checking to ensure that a non-negative number is passed on the command line.

Answer: //These programs solve exercise 3.7

//program 1: cp

```
#include <windows.h>
#include <stdio.h>

#define MAX_STRING 256

int main( int argc, char *argv[] )
{
    if (argc < 2)
        exit(0);

     STARTUPINFO si;
     PROCESS_INFORMATION pi;
```

```
    char command[MAX_STRING];

     ZeroMemory( &si, sizeof(si) );
     si.cb = sizeof(si);
     ZeroMemory( &pi, sizeof(pi) );

    sprintf(command,"fib.exe %d",atoi(argv[1]));

     // Start the child process.
     if( !CreateProcess( NULL,     // No module name (use
command line).
            command, // Command line.
            NULL,                 // Process handle not inheritable.
            NULL,                 // Thread handle not inheritable.
            FALSE,                // Set handle inheritance to FALSE.
            0,                    // No creation flags.
            NULL,                 // Use parent's environment
block.
            NULL,                 // Use parent's starting directory.
            &si,                  // Pointer to STARTUPINFO
structure.
            &pi    )              // Pointer to
```

```
PROCESS_INFORMATION structure.
    )
    {
        printf( "CreateProcess failed (%d).\n", GetLastError() );
        return -1;
    }

    // Wait until child process exits.
    WaitForSingleObject( pi.hProcess, INFINITE );

    // Close process and thread handles.
    CloseHandle( pi.hProcess );
    CloseHandle( pi.hThread );
}

//program 2: fib
#include <windows.h>
#include <stdio.h>

int main( int argc, char *argv[] )
{
   int max = atoi(argv[1]);
```

```
if (max <= 0)
    exit(0);
else {
    if (max == 1)
        printf("0\n");
    else if (max == 2)
        printf("0, 1\n");
    else {
        int a = 0;
        int b = 1;
        int fib;

        printf("0, 1, ");
        for (int i = 3; i < max; i++) {
            fib = a + b;
            printf("%d,",fib);
            a = b;
            b = fib;
        }

        printf("%d\n",a+b);
```

```
        }
    }
}
```

3.8 Modify the Date server shown in Figure 3.19 so that it delivers random one-line fortunes rather than the current date. Allow the fortunes to contain multiple lines. The date client shown in Figure 3.20 can be used to read the multi-line fortunes returned by the fortune server.

Answer: //Client program requesting fortune from server.

```
import java.net.*;
import java.io.*;

public class FortuneClient
{
    public static void main(String[] args) throws IOException {
        InputStream in = null;
        BufferedReader bin = null;
        Socket sock = null;

        try {
```

```
            sock = new Socket("127.0.0.1",6013);
            in = sock.getInputStream();
            bin = new BufferedReader(new
InputStreamReader(in));

            String line;
            while( (line = bin.readLine()) != null)
                System.out.println(line);
        }
        catch (IOException ioe) {
                System.err.println(ioe);
        }
                    finally {
                        sock.close();
                    }
    }
}

//Fortune server listening to port 6013.
import java.net.*;
import java.io.*;
```

```
public class FortuneServer
{
    private static final String[] fortunes = { "Buy Low and Sell
High",
                        "Eat Your Vegetables",
                        "Good Walls Make Good Neighbors",
                        "Never Underestimate Your
Competition",
                        "A Clean Camp is a Happy Camp",
                        "Be Sure to Test Every Line of Code You
Write"
                    };

    public static void main(String[] args) throws IOException {
        Socket client = null;
        ServerSocket sock = null;

        try {
            sock = new ServerSocket(6013);
            // now listen for connections
            while (true) {
                client = sock.accept();
```

```
                System.out.println("server = " + sock);
                System.out.println("client = " + client);

                // we have a connection
                PrintWriter pout = new
PrintWriter(client.getOutputStream(), true);
                // write the Date to the socket

    pout.println(fortunes[(int)(java.lang.Math.random() *
fortunes.length)] );

                pout.close();
                client.close();
            }
        }
        catch (IOException ioe) {
                System.err.println(ioe);
        }
        finally {
            if (sock != null)
                sock.close();
            if (client != null)
```

```
                client.close();
            }
        }
}
```

3.9 An echo server is a server that echoes back whatever it receives from a
client. For example, if a client sends the server the string Hello there! the
server will respond with the exact data it received from the client — that
is, Hello there!
Write an echo server using the Java networking API described in Section 3.6.1. This server will wait for a client connection using the accept() method. When a client connection is received, the server will
loop, performing the following steps:

- Read data from the socket into a buffer.
- Write the contents of the buffer back to the client.

The server will break out of the loop only when it has determined that the client has closed the connection.
The date server shown in Figure 3.19 uses the

java.io.BufferedReader class. BufferedReader extends the java.io.Reader class, which is used for reading character streams. However, the echo server cannot guarantee that it will read characters from clients; it may receive binary data as well. The class java.io.InputStream deals with data at the byte level rather than the character level. Thus, this echo server must use an object that extends java.io.InputStream. The read() method in the java.io.InputStream class returns -1 when the client has closed its end of the socket connection.

Answer:

```
// An echo server listening on port 6007.
// This server reads from the client and echoes back the result.
// When the client enters the character '.' – the server closes the connection.
// This conforms to RFC 862 for echo servers.

import java.net.*;
import java.io.*;

public class    EchoServer
{
```

```
public static final int DEFAULT_PORT = 6007;
        public static final int BUFFER_SIZE = 256;



public static void main(String[] args) throws IOException {
    ServerSocket sock = null;
    byte[] buffer = new byte[BUFFER_SIZE];
    InputStream fromClient = null;
    OutputStream toClient = null;

    try {
        // establish the socket
        sock = new ServerSocket(DEFAULT_PORT);

        while (true) {
            /**
             * now listen for connections
             */

            Socket client = sock.accept();

            /**
```

```
                * get the input and output streams associated
with the socket.
                */
                fromClient = new
BufferedInputStream(client.getInputStream());
                toClient = new
BufferedOutputStream(client.getOutputStream());
                int numBytes;

                        /** continually loop until the
client closes the connection */
               while ( (numBytes = fromClient.read(buffer)) != -1)
{

    toClient.write(buffer,0,numBytes);
                    toClient.flush();
                }

                fromClient.close();
                toClient.close();
                client.close();
            }
```

```
        }
        catch (IOException ioe) { }
        finally {
            if (sock != null)
                sock.close();
        }
    }
}
```

3.10 In Exercise 3.6, the child process must output the Fibonacci sequence, since the parent and child have their own copies of the data. Another approach to designing this program is to establish a shared-memory segment between the parent and child processes. This technique allows the child to write the contents of the Fibonacci sequence to the shared-memory segment and has the parent output the sequence when the child completes. Because the memory is shared, any changes the child makes to the shared memory will be reflected in the parent process as well.

This program will be structured using POSIX shared memory as described in Section 3.5.1. The program first requires

creating the data structure for the shared-memory segment. This is most easily accomplished using a struct. This data structure will contain two items: (1) a fixed-sized array of size MAX SEQUENCE that will hold the Fibonacci values; and (2) the size of the sequence the child process is to generate-sequence_size where sequence_size ≤ MAX SEQUENCE. These items can be represented in a struct as follows:

```
#define MAX SEQUENCE 10
typedef struct {
   long fib sequence[MAX SEQUENCE];
   int sequence size;
} shared data;
```

The parent process will progress through the following steps:

a. Accept the parameter passed on the command line and perform error checking to ensure that the parameter is ≤ MAX SEQUENCE.

b. Create a shared-memory segment of size shared data.

c. Attach the shared-memory segment to its address space.

d. Set the value of sequence size to the parameter on the command line.

e. Fork the child process and invoke the wait() system call to

wait for the child to finish.

f. Output the value of the Fibonacci sequence in the shared-memory segment.

g. Detach and remove the shared-memory segment.

Because the child process is a copy of the parent, the shared-memory region will be attached to the child's address space as well. The child process will then write the Fibonacci sequence to shared memory and finally will detach the segment. One issue of concern with cooperating processes involves synchronization issues. In this exercise, the parent and child processes must be synchronized so that the parent does not output the Fibonacci sequence until the child finishes generating the sequence. These two processes will be synchronized using the wait() system call; the parent process will invoke wait(), which will cause it to be suspended until the child process exits.

Answer: /* Example shared memory program. */

```
#include <stdio.h>
#include <unistd.h>
#include <sys/shm.h>
#include <sys/stat.h>
```

```
#include <sys/types.h>

#define PERMS (S_IRUSR | S_IWUSR)
#define MAX_SEQUENCE    10

typedef struct {
    long fib_sequence[MAX_SEQUENCE];
    int sequence_size;
} shared_data;

int main(int argc, char *argv[])
{
    int i, seq_size;

    /* the process identifier */
    pid_t pid;

    /* the id of the shared memory segment */
        int segment_id;

        /* a pointer to the shared memory segment */
        shared_data* shared_memory;
```

```
    /* do some error checking to ensure the parameter was
passed */
    if (argc != 2) {
        fprintf(stderr,"Usage: ./shm-fib <sequence size>\n");
        return -1;
    }

    seq_size = atoi(argv[1]);
    if (seq_size > MAX_SEQUENCE) {
        fprintf(stderr,"sequence size must be
< %d\n",MAX_SEQUENCE);
        return -1;
    }

    /* allocate a shared memory segment */
   if ( (segment_id = shmget(IPC_PRIVATE, sizeof(shared_data),
PERMS)) == -1) {
                fprintf(stderr,"Unable to create shared
memory segment\n");
                return 1;
    }
```

```
    printf("Created shared memory
segment %d\n",segment_id);

            /* now attach the shared memory segment at the
specified address */
            if ( (shared_memory = (shared_data *)
shmat(segment_id, 0, 0)) == (shared_data *)-1) {
                        fprintf(stderr,"Unable to attach to
segment %d\n",segment_id);
                        return 0;
            }

    /* set the size of the sequence */
    shared_memory->sequence_size = seq_size;

    /* now fork a child process */
    if ( (pid = fork()) == (pid_t)-1) {
                        return 1;
    }

    /**
```

```
     * now create a child process and have the child process set
     * the the    shared memory segment to a certain value.
     * The parent process will inquire on this shared value when
     * it returns from wait(). Thus, the call to wait() provides the
synchronization.
     */

    if (pid == 0) { /** child code */
            printf("CHILD: shared memory attached at
address %p\n", shared_memory);

        /* now have the child generate the Fibonacci
sequence ....   */
        shared_memory->fib_sequence[0] = 0;
        shared_memory->fib_sequence[1] = 1;

        for (i = 2; i < shared_memory->sequence_size; i++)
            shared_memory->fib_sequence[i] =
shared_memory->fib_sequence[i-1] +
                shared_memory->fib_sequence[i-2];

            /* now detach the shared memory segment */
```

```
            shmdt((void *)shared_memory);
    }
    else {  /* parent code */
        wait(NULL);

        for (i = 0; i < shared_memory->sequence_size; i++)
            printf("PARENT: %d:%ld\n",i,
shared_memory->fib_sequence[i]);

            /* now detach and remove the shared memory
segment */
            shmdt((void *)shared_memory);
        shmctl(segment_id, IPC_RMID, NULL);
    }
     return 0;
}
```

3.11 Most UNIX and Linux systems provide the ipcs command. This command lists the status of various POSIX interprocess communicationmechanisms,including shared-memory segments. Much of the information for the command comes from the data structure struct shmid_ds,which is available in

the /usr/include/sys/shm.h file. Some of the fields of this structure include:

- int shm segsz—size of the shared-memory segment
- short shm nattch—number of attaches to the shared-memory segment
- struct ipc perm shm perm—permission structure of the shared-memory segment The struct ipc perm data structure (which is available in the file /usr/include/sys/ipc.h) contains the fields:
- unsigned short uid—identifier of the user of the shared-memory segment
- unsigned short mode—permission modes
- key t key (on Linux systems, key)—user-specified key identifier The permission modes are set according to how the shared-memory segment is established with the shmget() system call. Permissions are identified according to the following:

| Mode | Meaning |
|---|---|
| 0400 | Read permission of owner. |
| 0200 | Write permission of owner. |

| | |
|---|---|
| 0040 | Read permission of group. |
| 0020 | Write permission of group. |
| 0004 | Read permission of world. |
| 0002 | Write permission of world. |

Permissions can be accessed by using the bitwise AND operator &. For example, if the statement mode & 0400 evaluates to true, the permission mode allows read permission by the owner of the shared-memory segment.

Shared-memory segments can be identified according to a user-specified key or according to the integer value returned fromthe shmget() system call, which represents the integer identifier of the shared-memory segment created. The shm ds structure for a given integer segment identifier can be obtained with the following shmctl() system call:

```
/* identifier of the shared memory segment*/
int segment id;
shm ds shmbuffer;
shmctl(segment id, IPC STAT, &shmbuffer);
```

If successful, shmctl() returns 0; otherwise, it returns -1. Write a C program that is passed an identifier for a shared-memory segment. This program will invoke the shmctl() function to obtain its shm ds structure. It will then output the following values of the given shared-memory segment:

- Segment ID
- Key
- Mode
- Owner UID
- Size
- Number of attaches

Answer:

```
// This program illustrates the functionality of the ipcs
command on POSIX systems.
// This program is written for Linux 2.4 systems.
// Getting it to work on Solaris, OS X, etc. will require
modifying the source code where commented.
// Usage: gcc -o sm sm.c
//              ./sm

#include <stdio.h>
```

```
#include <sys/shm.h>
#include <sys/stat.h>

int main(int argc, char *argv[])
{
    /* the segment number */
    int segment_id;

    /* permissions of the segment */
    unsigned short mode;

        /** the shared memory segment */
    struct shmid_ds shmbuffer;

    /** do some error checking */
    if (argc < 2) {
        fprintf(stderr,"Usage: sm <segment id>\n");
        return -1;
    }

        /**
         * this needs to be set to the
```

```
            * shared memory segment number
            * being attached to.
            */
           segment_id = atoi(argv[1]);

    /* get the shm_ds information */
    if (shmctl(segment_id, IPC_STAT, &shmbuffer) == -1) {
        fprintf(stderr,"Unable to access
segment %d\n",segment_id);
        return -1;
    }

    /** now report the fields in shm_ds */
    printf("ID \t\t KEY \t MODE \t\t OWNER \t SIZE \t
ATTTACHES \n");
    printf("-- \t\t --- \t ---- \t\t ----- \t ---- \t --------- \n");

    /** Linux has __key rather than key field */
    printf("%d \t %d
\t",segment_id,shmbuffer.shm_perm.__key);

    /** Mac OS X Darwin uses the key field */
```

```
    //printf("%d \t %d
\t",segment_id,shmbuffer.shm_perm.key);

    /** report on the permission */
    mode = shmbuffer.shm_perm.mode;

    /** report on the permission */
    mode = shmbuffer.shm_perm.mode;

    /** OWNER */
    if (mode & 0400)
        printf("r");
    else
        printf("-");
    if (mode & 0200)
        printf("w");
    else
        printf("-");
    if (mode & 0100)
        printf("a");
    else
        printf("-");
```

```
/** GROUP */
if (mode & 0040)
        printf("r");
    else
        printf("-");
    if (mode & 0020)
        printf("w");
    else
        printf("-");
    if (mode & 0010)
        printf("a");
    else
        printf("-");

/** WORLD */
if (mode & 0004)
        printf("r");
    else
        printf("-");
    if (mode & 0002)
        printf("w");
```

```
            else
                        printf("-");
            if (mode & 0001)
                        printf("a");
            else
                        printf("-");

    /** Darwin (Mac OS X) has user_from_uid() function */
    //printf("\t%s\t",user_from_uid(shmbuffer.shm_perm.uid,0)
);
    printf("\t%d\t",shmbuffer.shm_perm.uid);
    printf("%d\t",shmbuffer.shm_segsz);
    printf("%d\t",shmbuffer.shm_nattch);
    //printf("time of last attach %d\n",shmbuffer.shm_atime);

    printf("\n");

            return 0;
}
```

# Chapter 4

## **4.1** Provide two programming examples in which multithreading does *not* provide better performance than a single-threaded solution

**Answer:**

(1) Any kind of sequential program is not a good candidate to be threaded. An example of this is a program that calculates

an individual tax return.

(2) Another example is a "shell" program such as the C-shell or Korn shell. Such a program must closely monitor its own working space such as open files, environment variables, and current working directory.

**4.2** Describe the actions taken by a thread library to context switch between user-level threads.

**Answer:** Context switching between user threads is quite similar to switching between kernel threads, although it is dependent on the threads library and how it maps user threads to kernel threads. In general, context switching between user threads involves taking a user thread of its LWP and replacing it with another thread. This act typically involves saving and restoring the state of the registers.

**4.3** Under what circumstances does a multithreaded solution using multiple kernel threads provide better performance than a single-threaded solution on a single-processor system?

**Answer:** When a kernel thread suffers a page fault, another

kernel thread can be switched in to use the interleaving time in a useful manner.

A single-threaded process, on the other hand, will not be capable of performing useful work when a page fault takes place. Therefore, in scenarios where a program might suffer from frequent page faults or has to wait for other system events, a multi-threaded solution would perform better even on a single-processor system.

**4.4** Which of the following components of program state are shared across threads in a multithreaded process?

a. Register values

b. Heap memory

c. Global variables

d. Stack memory

**Answer:** The threads of a multithreaded process share heap memory and global variables. Each thread has its separate set of register values and a separate stack.

**4.5** Can a multithreaded solution using multiple user-level threads achieve betterperformance on amultiprocessor

system than on a single-processor system?

**Answer:** A multithreaded system comprising of multiple user-level threads cannot make use of the different processors in a multiprocessor system simultaneously. The operating system sees only a single process and will not schedule the different threads of the process on separate processors. Consequently, there is no performance benefit associated with executing multiple user-level threads on a multiprocessor system.

**4.6** As described in Section 4.5.2, Linux does not distinguish between processes and threads. Instead, Linux treats both in the same way, allowing a task to be more akin to a process or a thread depending on the set of flags passed to the clone() system call. However, many operating systems—such as Windows XP and Solaris—treat processes and threads differently. Typically, such systems use a notationwherein the data structure for a process contains pointers to the separate threads belonging to the process. Contrast these two approaches for modeling processes and threads within the kernel.

**Answer:** On one hand, in systems where processes and threads are considered as similar entities, some of the operating system code could be simplified. A scheduler, for instance, can consider the different processes and threads in equal footing without requiring special code to examine the threads associated with a process during every scheduling step. On the other hand, this uniformity could make it harder to impose process-wide resource constraints in a direct manner. Instead, some extra complexity is required to identifywhich threads correspond towhich process and perform the relevant accounting tasks.

**4.7** The program shown in Figure 4.11 uses the Pthreads API. What would be output from the program at LINE C and LINE P?

```
include <stdio.h>
#include <pthread.h>

int value=0;
void *runner(void *param); /* the thread */
```

```
int main()
{
    int pid;
    pthread_t tid;
    pthread_attr_t attr;
    pid=fork();
    if(pid==0)
    {
        pthread_attr_init(&attr);
        pthread_create(&tid, &attr, runner, NULL);
        pthread_join(tid, NULL);
        printf("CHILD: value=%d\n", value);    /* LINE C */
    }else if(pid>0){
        wait(NULL);
        printf("PARENT: value=%d\n",value);    /* LINE P */
    }
}

void *runner(void *param)
{
    value=5;
    pthread_exit(0);
```

}

**Answer:**

Output at LINE C is **CHILD: value=5**.

Output at LINE P is **PARENT: value=0**.

**4.8** Consider amultiprocessor system and a multithreaded program written using the many-to-many threading model. Let the number of user-level threads in the program be more than the number of processors in the system.Discuss the performance implications of the following scenarios.

a. The number of kernel threads allocated to the program is less than the number of processors.

b. The number of kernel threads allocated to the program is equal to the number of processors.

c. The number of kernel threads allocated to the program is greater than the number of processors but less than the number of userlevel threads.

**Answer:** When the number of kernel threads is less than the number of processors, then some of the processors would remain idle since the scheduler maps only kernel threads to processors and not user-level threads to processors. When the

number of kernel threads is exactly equal to the number of processors, then it is possible that all of the processors might be utilized simultaneously. However, when a kernelthread blocks inside the kernel (due to a page fault or while invoking system calls), the corresponding processor would remain idle. When there are more kernel threads than processors, a blocked kernel thread could be swapped out in favor of another kernel thread that is ready to execute, thereby increasing the utilization of the multiprocessor system.

**4.9** Write a multithreaded Java, Pthreads, or Win32 program that outputs prime numbers. This program should work as follows: The user will run the program and will enter a number on the command line. The program will then create a separate thread that outputs all the prime numbers less than or equal to the number entered by the user.

**Answer:**

**(1)** // a multithreaded Java program

```
class PrimesThread implements Runnable
{
    private int num;
    private int[] primeNums;
```

```
    public PrimesThread(int num) {
        if (num < 2)
            throw new IllegalArgumentException();
        this.num = num;
    }

    public void run() {
        int i, j;
        primeNums = new int[num + 1];
        primeNums[1] = 0;
        for (i = 2; i <= num; i++)
            primeNums[i] = 1;
        for (i = 2; i <= num/2; i++)
            for (j = 2; j <= num/i; j++)
                primeNums[i*j] = 0;
        for (i = 1; i <= num; i++)
            if (primeNums[i] > 0)
                System.out.println(i);
    }
}
```

```
public class Primes
{
    public static void main(String args[]) {
        if (args.length == 0) {
            System.out.println("Usage: Primes <num>");
            System.exit(0);
        }
        else
            new Thread(new
PrimesThread(Integer.parseInt(args[0]))).start();
    }
}
```

(2) Win32 program that outputs prime numbers

```
#include <stdio.h>
#include <windows.h>


/** we will only allow up to 256 prime numbers */
#define MAX_SIZE 256

int primes[MAX_SIZE];
```

```
/* the thread runs in this separate function */
DWORD WINAPI Summation(PVOID Param)
{
      DWORD upper = *(DWORD *)Param;
      int i, j;
    primes[1] = 0;
    for (i = 2; i <= upper; i++)
        primes[i] = 1;
    for (i = 2; i <= upper/2; i++)
        for (j = 2; j <= upper/i; j++)
            primes[i*j] = 0;
    return 0;
}

int main(int argc, char *argv[])
{
    DWORD ThreadId;
    HANDLE ThreadHandle;
    int Param;
    // do some basic error checking
```

```
    if (argc != 2) {
        fprintf(stderr,"An integer parameter is required\n");
        return -1;
    }
    Param = atoi(argv[1]);
    if (Param < 2) {
        fprintf(stderr, "an integer >= 2 is required \n");
        return -1;
    }
    // create the thread
    ThreadHandle = CreateThread(NULL, 0, Summation,
&Param, 0, &ThreadId);

    if (ThreadHandle != NULL) {
        WaitForSingleObject(ThreadHandle, INFINITE);
        CloseHandle(ThreadHandle);
                    /** now output the prime numbers */
                    for (int i = 1; i <= Param; i++)
                    if (primes[i] > 0)
                        printf("%d\n", i);
    }
}
```

(3) Posix program that outputs prime numbers

```
#include <pthread.h>
#include <stdio.h>

/** we will only allow up to 256 prime numbers */
#define MAX_SIZE 256
int primes[MAX_SIZE];
void *runner(void *param); /* the thread */
int main(int argc, char *argv[])
{
      int i;
      pthread_t tid; /* the thread identifier */
      pthread_attr_t attr; /* set of attributes for the thread */

      if (argc != 2) {
         fprintf(stderr,"usage: a.out <integer value>\n");
         return -1;
      }

      if (atoi(argv[1]) < 2) {
         fprintf(stderr,"Argument %d must be >= 2
```

```
\n",atoi(argv[1]));
        return -1;
    }

    /* get the default attributes */
    pthread_attr_init(&attr);
    /* create the thread */
    pthread_create(&tid,&attr,runner,argv[1]);
    /* now wait for the thread to exit */
    pthread_join(tid,NULL);
    /** now output the prime numbers */
    for (i = 1; i <= atoi(argv[1]); i++)
      if (primes[i] > 0)
          printf("%d\n", i);
}

/* Generate primes using the Sieve of Eratosthenes. */
void *runner(void *param)
{
    int i, j;
    int upper = atoi(param);
   primes[1] = 0;
```

```
    for (i = 2; i <= upper; i++)
        primes[i] = 1;
    for (i = 2; i <= upper/2; i++)
        for (j = 2; j <= upper/i; j++)
            primes[i*j] = 0;
    pthread_exit(0);
}
```

**4.10** Modify the socket-based date server (Figure 3.19) in Chapter 3 so that the server services each client request in a separate thread.

**Answer:** /*Server.java

This is the date server where each client is serviced in a separate thread.

The server listens on port 6013. */

```
import java.net.*;
import java.io.*;

public class Connection implements Runnable
{
    private Socket  outputLine;
```

```
    public Connection(Socket s) {
        outputLine = s;
    }
    public void run() {
    // getOutputStream returns an OutputStream object
    // allowing ordinary file IO over the socket.
    try {
            // create a new PrintWriter with automatic flushing
            PrintWriter pout=new
PrintWriter(outputLine.getOutputStream(),true);
            // now send the current date to the client
            pout.println(new java.util.Date() );
            // now close the socket
            outputLine.close();
    }
    catch (java.io.IOException e) {
            System.out.println(e);
    }
   }
}
```

```
/*Connection.java
   This is the separate thread that services each request.
   This reads a random fortune from the list of fortunes.   */

import java.net.*;
import java.io.*;

public class Connection implements Runnable
{
   private Socket    outputLine;

   public Connection(Socket s) {
    outputLine = s;
   }

   public void run() {
    // getOutputStream returns an OutputStream object
    // allowing ordinary file IO over the socket.
    try {
       // create a new PrintWriter with automatic flushing
       PrintWriter pout = new
PrintWriter(outputLine.getOutputStream(), true);
```

```
        // now send the current date to the client
        pout.println(new java.util.Date() );
        // now close the socket
        outputLine.close();
      }
      catch (java.io.IOException e) {
        System.out.println(e);
      }
    }
}
```

**4.11** The Fibonacci sequence is the series of numbers 0, 1, 1, 2, 3, 5, 8, .... Formally,it can be expressed as:

$fib_0 = 0$

$fib_1 = 1$

$fib_n = fib_{n-1} + fib_{n-2}$

Write a multithreaded program that generates the Fibonacci series using either the Java, Pthreads, or Win32 thread library. This program should work as follows: The user will enter on the command line the number of Fibonacci numbers that the program is to generate. The program will then create a

separate thread that will generate the Fibonacci numbers, placing the sequence in data that is shared by the threads (an array is probably the most convenient data structure).When the thread finishes execution, the parent thread will output the sequence generated by the child thread. Because the parent thread cannot begin outputting the Fibonacci sequence until the child thread finishes, this will require having the parent thread wait for the child thread to finish using the techniques described in Section 4.3.

**Answer:** (1) Java program

```
// Generates the Fibonacci sequence in a separate thread.
class FibThread implements Runnable
{
    private int[] fibNums;
    public FibThread(int[] fibNums) {
        this.fibNums = fibNums;
    }

    public void run() {
        int length = fibNums.length;
        if (length == 0)
```

```
            return;
        else if (length == 1)
            fibNums[0] = 0;
        else if (length == 2) {
            fibNums[0] = 0;
            fibNums[1] = 1;
        }
        else { // length > 2
            fibNums[0] = 0;
            fibNums[1] = 1;
            for (int i = 2; i < length; i++)
                fibNums[i] = fibNums[i-1] + fibNums[i-2];
        }
    }
}

public class Fib
{
    public static void main(String args[]) {
        if (args.length == 0) {
            System.out.println("Usage: Fib <num>");
            System.exit(0);
```

```
        }
        else if (Integer.parseInt(args[0]) < 0) {
            System.out.println("Sequence size must be >= 0\n");
            System.exit(0);
        }
        else {
            int[] sequence = new int[Integer.parseInt(args[0])];
            Thread worker = new Thread(new
FibThread(sequence));
            worker.start();
            try {
                worker.join();
            } catch (InterruptedException ie) { }
            for (int i = 0; i < sequence.length; i++)
                System.out.println(sequence[i]);
        }
    }
}
```

(2)Windows program

```
#include <stdio.h>
#include <windows.h>
```

```
/** we will only allow up to 256 Fibonacci numbers */
#define MAX_SIZE 256

int fibs[MAX_SIZE];

/* the thread runs in this separate function */
DWORD WINAPI Summation(PVOID Param)
{
      DWORD upper = *(DWORD *)Param;
    int i;
    if (upper== 0)
        return 0;
    else if (upper == 1)
        fibs[0] = 0;
    else if (upper== 2) {
        fibs[0] = 0;
        fibs[1] = 1;
    }
    else { // sequence > 2
        fibs[0] = 0;
        fibs[1] = 1;
```

```
        for (i = 2; i < upper; i++)
            fibs[i] = fibs[i-1] + fibs[i-2];
    }
    return 0;
}

int main(int argc, char *argv[])
{
    DWORD ThreadId;
    HANDLE ThreadHandle;
    int Param;
    // do some basic error checking
    if (argc != 2) {
        fprintf(stderr,"An integer parameter is required\n");
        return -1;
    }
    Param = atoi(argv[1]);
    if ( (Param < 0) || (Param > 256) ) {
        fprintf(stderr, "an integer >= 0 and < 256 is required
\n");
        return -1;
    }
```

```
    // create the thread
    ThreadHandle = CreateThread(NULL, 0, Summation,
&Param, 0, &ThreadId);

    if (ThreadHandle != NULL) {
        WaitForSingleObject(ThreadHandle, INFINITE);
        CloseHandle(ThreadHandle);
          /** now output the Fibonacci numbers */
          for (int i = 0; i < Param; i++)
                printf("%d\n", fibs[i]);
    }
}
```

(3)Posix program

```
#include <pthread.h>
#include <stdio.h>

/** we will only allow up to 256 fibonacci numbers */
#define MAX_SIZE 256

int fibs[MAX_SIZE];
```

```
void *runner(void *param); /* the thread */

int main(int argc, char *argv[])
{
int i;
pthread_t tid; /* the thread identifier */
pthread_attr_t attr; /* set of attributes for the thread */

if (argc != 2) {
    fprintf(stderr,"usage: a.out <integer value>\n");
    return -1;
}

if (atoi(argv[1]) < 0) {
    fprintf(stderr,"Argument %d must be >= 0
\n",atoi(argv[1]));
    return -1;
}

/* get the default attributes */
pthread_attr_init(&attr);
```

```
/* create the thread */
pthread_create(&tid,&attr,runner,argv[1]);

/* now wait for the thread to exit */
pthread_join(tid,NULL);

/** now output the Fibonacci numbers */
for (i = 0; i < atoi(argv[1]); i++)
    printf("%d\n", fibs[i]);
}

/**
 * Generate primes using the Sieve of Eratosthenes.
 */
void *runner(void *param)
{
int i;
int upper = atoi(param);

    if (upper== 0)
        pthread_exit(0);
    else if (upper == 1)
```

```
        fibs[0] = 0;
    else if (upper== 2) {
        fibs[0] = 0;
        fibs[1] = 1;
    }
    else { // sequence > 2
        fibs[0] = 0;
        fibs[1] = 1;

        for (i = 2; i < upper; i++)
            fibs[i] = fibs[i-1] + fibs[i-2];
    }

    pthread_exit(0);
}
```

**4.12** Exercise 3.9 in Chapter 3 specifies designing an echo server using the Java threading API. However, this server is single-threaded meaning the server cannot respond to concurrent echo clients until the current client exits. Modify the solution to Exercise 3.9 such that the echo server services each client in a spearate request.

**Answer: (1) Handler.java**

//Handler class containing the logic for echoing results back to the client.

```
import java.io.*;
import java.net.*;

public class Handler
{
    public static final int BUFFER_SIZE = 256;

    // this method is invoked by a separate thread
    public void process(Socket client) throws
java.io.IOException {
        byte[] buffer = new byte[BUFFER_SIZE];
        InputStream  fromClient = null;
        OutputStream toClient = null;

        try {
            //get the input and output streams associated with
the socket.
```

```
            fromClient = new
BufferedInputStream(client.getInputStream());
            toClient = new
BufferedOutputStream(client.getOutputStream());
            int numBytes;

        //continually loop until the client closes the connection
            while ( (numBytes = fromClient.read(buffer)) != -1) {
           toClient.write(buffer,0,numBytes);
                toClient.flush();
            }

    }
       catch (IOException ioe) {
             System.err.println(ioe);
       }
    finally {
       // close streams and socket
       if (fromClient != null) fromClient.close();
       if (toClient != null) toClient.close();
       if (client != null) client.close();
    }
```

```
    }
}
```

**(2) Connection.java**

//This is the separate thread that services each incoming echo client request.

```
import java.net.*;
import java.io.*;

public class Connection implements Runnable
{
    public Connection(Socket client) {
        this.client = client;
    }

    public void run() {
        try {
            handler.process(client);
        }
        catch (java.io.IOException ioe) {
            System.err.println(ioe);
```

```
        }
    }

    private Socket   client;
    private static Handler handler = new Handler();
}
```

**(3) EchoServer.java**

//An echo server listening on port 6007.

//This server reads from the client and echoes back the result.

//When the client enters the character '.' - the server closes the connection.

```
import java.net.*;
import java.io.*;

 public class    EchoServer {
    public static final int DEFAULT_PORT = 6007;
    public static void main(String[] args) throws IOException {
        ServerSocket sock = null;
        try {
            // establish the socket
```

```
            sock = new ServerSocket(DEFAULT_PORT);
            while (true) {
                //now listen for connections and service the
connection in a separate thread.
                Thread worker = new Thread(new
Connection(sock.accept()));
                worker.start();
            }
        }
        catch (IOException ioe)
        {
        }
        finally {
            if (sock != null) sock.close();
        }
    }
}
```

# Chapter 5

## 5.1 Why is it important for the scheduler to distinguish I/O-bound programs from CPU-bound programs?

Answer: I/O-bound programs have the property of performing only a small amount of computation before performing IO.

Such programs typically do not use up their entire CPU quantum. CPU-bound programs,on the other hand, use their entire quantum without performing any blocking IO operations. Consequently, one could make better use of the computer ' s resouces by giving higher priority to I/O-bound programs and allow them to execute ahead of the CPU-bound programs.

5.2 Discuss how the following pairs of scheduling criteria conflict in certain settings.

a. CPU utilization and response time

b. Average turnaround time and maximum waiting time c. I/O device utilization and CPU utilization

Answer:

- CPU utilization and response time: CPU utilization is increased if the overheads associated with context switching is minimized. The context switching overheads could be lowered by performing context switches infrequently. This could however result in increasing the response time for processes.
- Average turnaround time and maximum waiting time: Average turnaround time is minimized by executing the

shortest tasks first.

Such a scheduling policy could however starve long-running tasks and thereby increase their waiting time.

■ I/O device utilization and CPU utilization: CPU utilization is maximized by running long-running CPU-bound tasks without performing context switches. I/O device utilization is maximized by scheduling I/O-bound jobs as soon as they become ready to run, thereby incurring the overheads of context switches.

5.3 Consider the exponential average formula used to predict the length of the next CPU burst. What are the implications of assigning the following values to the parameters used by the algorithm?

a. = 0 and 0 = 100milliseconds

b. = 0.99 and 0 = 10milliseconds

Answer: When = 0 and 0 = 100milliseconds, the formula always makes a prediction of 100 milliseconds for the next CPU burst. When =0.99 and 0 = 10milliseconds, the most recent behavior of the process is given much higher weight than the past history associated with the process.

Consequently, the scheduling algorithm is almost memory-less, and simply predicts the length of the previous burst for the next quantum of CPU execution.

5.4 Consider the following set of processes, with the length of the CPU-burst time given in milliseconds:

| Process | Burst Time | Priority |
|---|---|---|
| P1 | 10 | 3 |
| P2 | 1 | 1 |
| P3 | 2 | 3 |
| P4 | 1 | 4 |
| P5 | 5 | 2 |

The processes are assumed to have arrived in the order P1 , P2 , P3 , P4 , P5 , all at time 0.

a. Draw four Gantt charts illustrating the execution of these processes using FCFS, SJF, a nonpreemptive priority (a smaller priority number implies a higher priority), and RR (quantum = 1) scheduling.

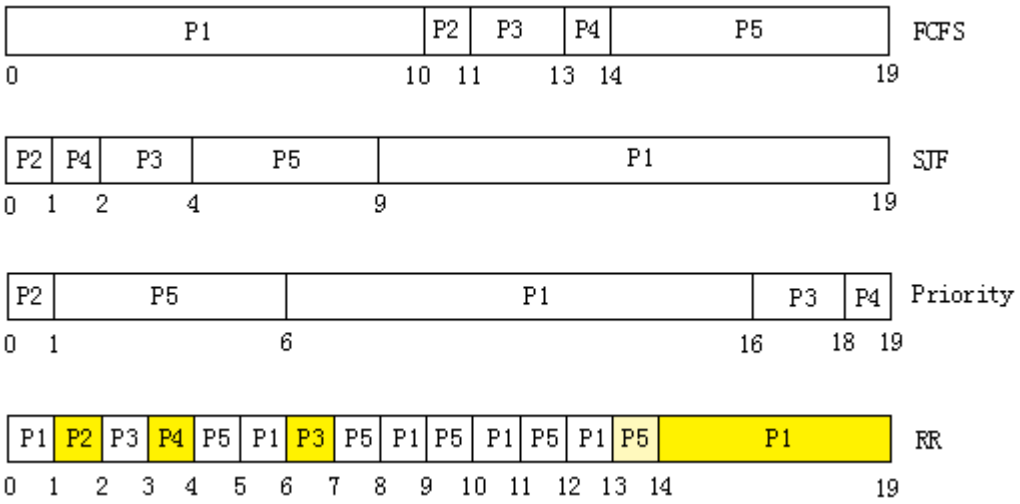
b. What is the turnaround time of each process for each of the scheduling algorithms in part a?

c. What is the waiting time of each process for each of the scheduling algorithms in part a?

d. Which of the schedules in part a results in the minimal average waiting time (over all processes)?

Answer:

a. The four Gantt charts are




b. Turnaround time

| Process | FCFS | RR | SJF | Priority |
|---|---|---|---|---|
| P1 | 10 | 19 | 19 | 16 |
| P2 | 11 | 2 | 1 | 1 |
| P3 | 13 | 7 | 4 | 18 |
| P4 | 14 | 4 | 2 | 19 |
| P5 | 19 | 14 | 9 | 6 |

c. Waiting time (turnaround time minus burst time)

| Process | FCFS | RR | SJF | Priority |
|---|---|---|---|---|
| P1 | 0 | 9 | 9 | 6 |
| P2 | 10 | 1 | 0 | 0 |
| P3 | 11 | 5 | 2 | 16 |
| P4 | 13 | 3 | 1 | 18 |
| P5 | 14 | 9 | 4 | 1 |

d. Shortest Job First

5.5 Which of the following scheduling algorithms could result in starvation?

a. First-come, first-served b. Shortest job first

c. Round robin

d. Priority

Answer: Shortest job first and priority-based scheduling algorithms could result in starvation.

5.6 Consider a variant of the RR scheduling algorithm where the entries in the ready queue are pointers to the PCBs.

a. What would be the effect of putting two pointers to the

same process in the ready queue?

b. What would be the major advantages and disadvantages of this scheme?

c. How would you modify the basic RR algorithm to achieve the same effect without the duplicate pointers?

Answer: a. In effect, that process will have increased its priority since by getting time more often it is receiving preferential treatment.

b. The advantage is that more important jobs could be given more time, in other words, higher priority in treatment. The consequence, of course, is that shorter jobs will suffer.

c. Allot a longer amount of time to processes deserving higher priority. In other words, have two or more quantums possible in the Round-Robin scheme.

5.7 Consider a system running ten I/O-bound tasks and one CPU-bound task. Assume that the I/O-bound tasks issue an I/O operation once for every millisecond of CPU computing and that each I/O operation takes 10 milliseconds to complete. Also assume that the context switching overhead is 0.1 millisecond and that all processes are long-running tasks.

What is the CPU utilization for a round-robin scheduler when:

a. The time quantum is 1 millisecond

b. The time quantum is 10 milliseconds

Answer:

- The time quantum is 1 millisecond: Irrespective of which process is scheduled, the scheduler incurs a 0.1 millisecond context-switching cost for every context-switch. This results in a CPU utilization of 1/1.1 * 100 = 91%.
- The time quantum is 10 milliseconds: The I/O-bound tasks incur a context switch after using up only 1 millisecond of the time quantum. The time required to cycle through all the processes is therefore 10*1.1 + 10.1 (as each I/O-bound task executes for 1 millisecond and then incur the context switch task, whereas the CPU-bound task executes for 10 milliseconds before incurring a context switch). The CPU utilization is therefore 20/21.1 * 100 = 94%.

5.8 Consider a system implementing multilevel queue scheduling. What strategy can a computer user employ to maximize the amount of CPU time allocated to the user's process?

Answer: The program could maximize the CPU time allocated to it by not fully utilizing its time quantums. It could use a large fraction of its assigned quantum, but relinquish the CPU before the end of the quantum, thereby increasing the priority associated with the process.

5.9 Consider a preemptive priority scheduling algorithm based on dynamically changing priorities. Larger priority numbers imply higher priority. When a process is waiting for the CPU (in the ready queue, but not running), its priority changes at a rate; when it is running, its priority changes at a rate . All processes are given a priority of 0 when they enter the ready queue. The parameters and can be set to give many different scheduling algorithms.

a. What is the algorithm that results from $\beta > \alpha > 0$?

b. What is the algorithm that results from $\alpha < \beta < 0$?

Answer: a. FCFS b. LIFO

5.10 Explain the differences in the degree to which the following scheduling algorithms discriminate in favor of short

processes:

a. FCFS

b. RR

c. Multilevel feedback queues

Answer:

a. FCFS — discriminates against short jobs since any short jobs arriving after long jobs will have a longer waiting time.

b. RR — treats all jobs equally (giving them equal bursts of CPU time) so short jobs will be able to leave the system faster since they will finish first.

c. Multilevel feedback queues — work similar to the RR algorithm —they discriminate favorably toward short jobs.

5.11 Using the Windows XP scheduling algorithm, what is the numeric priority of a thread for the following scenarios?

a. A thread in the REALTIME PRIORITY CLASS with a relative priority of HIGHEST.

b. A thread in the NORMAL PRIORITY CLASS with a relative priority of NORMAL.

c. A thread in the HIGH PRIORITY CLASS with a relative priority of ABOVE NORMAL.

Answer: a. 26 b. 8 c. 14

5.12 Consider the scheduling algorithm in the Solaris operating system for time sharing threads:

a. What is the time quantum (in milliseconds) for a thread with priority 10? With priority 55?

b. Assume a thread with priority 35 has used its entire time quantum without blocking. What new priority will the scheduler assign this thread?

c. Assume a thread with priority 35 blocks for I/O before its time quantum has expired. What new priority will the scheduler assign this thread?

Answer: a. 160 and 40 b. 35 c. 54

5.13 The traditional UNIX scheduler enforces an inverse relationship between priority numbers and priorities: The higher the number, the lower the priority. The scheduler recalculates process priorities once per second using the following function:

Priority = (Recent CPU usage / 2) + Base

where base = 60 and recent CPU usage refers to a value indicating how often a process has used the CPU since priorities were last recalculated. Assume that recent CPU usage for process P1 is 40, process P2 is 18, and process P3 is 10. What will be the new priorities for these three processes when priorities are recalculated? Based on this information, does the traditional UNIX scheduler raise or lower the relative priority of a CPU-bound process?

Answer: The priorities assigned to the processes are 80, 69, and 65 respectively. The scheduler lowers the relative priority of CPU-bound processes.

# Chapter 6

6.1 The first known correct software solution to the critical-section problem

for two processes was developed by Dekker. The two processes, P0 and P1 , share the following variables:

```
boolean flag[2]; /* initially false */
int turn;
```

The structure of process Pi (i == 0 or 1) is shown in Figure 6.25; the other process is Pj (j == 1 or 0). Prove that the algorithm satisfies all three requirements for the critical-section problem.

Answer: This algorithm satisfies the three conditions of mutual exclusion.

(1) Mutual exclusion is ensured through the use of the flag and turn variables. If both processes set their flag to true, only one will succeed. Namely, the process whose turn it is. The waiting process can only enter its critical section when the other process updates the value of turn.

(2) Progress is provided, again through the flag and turn

variables. This algorithm does not provide strict alternation. Rather, if a process wishes to access their critical section, it can set their flag variable to true and enter their critical section. It only sets turn to the value of the other process upon exiting its critical section. If this process wishes to enter its critical section again - before the other process - it repeats the process of entering its critical section and setting turn to the other process upon exiting.

(3) Bounded waiting is preserved through the use of the TTturn variable. Assume two processes wish to enter their respective critical sections. They both set their value of flag to true, however only the thread whose turn it is can proceed, the other thread waits.

If bounded waiting were not preserved, it would therefore be possible that the waiting process would have to wait indefinitely while the first process repeatedly entered - and exited - its critical section. However, Dekker’ s algorithm has a process set the value of turn to the other process, thereby ensuring that the other process will enter its critical section next.

## 6.2 The first known correct software solution to the

critical-section problem for n processes with a lower bound on waiting of n? 1 turns was presented by Eisenberg and McGuire. The processes share the following variables:

```
enum    pstate {idle, want_in, in_cs};
pstate    flag[n];
int    turn;
```

All the elements of flag are initially idle; the initial value of turn is immaterial (between 0 and n-1). The structure of process Pi is shown in Figure 6.26. Prove that the algorithm satisfies all three requirements for the critical-section problem.

Answer: This algorithm satisfies the three conditions. Before we show that the three conditions are satisfied, we give a brief explanation of what the algorithm does to ensure mutual exclusion. When a process I requires access to critical section, it first sets its flag variable to want in to indicate its desire. It then performs the following steps: (1) It ensures that all processes whose index lies between turn and i are idle. (2)If so, it updates its flag to in_cs and checks whether there is already some other process that has updated its flag to in_cs. (3) If not and if it is this process’ s turn to enter the critical section or if the process indicated by the turn variable is idle, it enters the

critical section. Given the above description, we can reason about how the algorithm satisfies the requirements in the following manner:

- Mutual exclusion is ensured: Notice that a process enters the critical section only if the following requirements is satisfied: no other process has its flag variable set to in_cs. Since the process sets its own flag variable set to in_cs before checking the status of other processes, we are guaranteed that no two processes will enter the critical section simultaneously.
- Progress requirement is satisfied: Consider the situation where multiple processes simultaneously set their flag variables to in_cs and then check whether there is any other process has the flag variable set to in_cs. When this happens, all processes realize that there are competing processes, enter the next iteration of the outer while (1) loop and reset their flag variables to want in. Now the

only process that will set its turn variable to in_cs is the process whose index is closest to turn. It is however possible that new processes whose index values are even closer to turn might decide to enter the critical section at this point and therefore might be able to simultaneously set its flag to in_cs. These processes would then realize there are competing processes

and might restart the process of entering the critical section. However, at each iteration, the index values of processes that set their flag variables to in_cs become closer to turn and eventually we reach the following condition:

only one process (say k) sets its flag to in_cs and no other process whose index lies between turn and k has set its flag to in_cs. This process then gets to enter the critical section.

- Bounded-waiting requirement is met: The bounded waiting requirement is satisfied by the fact that when a process k desires to enter the critical section, its flag is no longer set to idle. Therefore, any process whose index does not lie between turn and k cannot enter the critical section. In the meantime, all processes whose index falls between turn and k and desire to enter the critical section would indeed enter the critical section (due to the fact that the system always makes progress) and the turn value monotonically becomes closer to k. Eventually, either turn becomes k or there are no processes whose index values lie between turn and k, and therefore process k gets to enter the critical section.

6.3 What is the meaning of the term busy waiting? What other kinds of waiting are there in an operating system? Can busy

waiting be avoided altogether? Explain your answer.

Answer: Busy waiting means that a process is waiting for a condition to be satisfied in a tight loop without relinquish the processor. Alternatively,a process could wait by relinquishing the processor, and block on a condition and wait to be awakened at some appropriate time in the future. Busy waiting can be avoided but incurs the overhead associated with putting a process to sleep and having to wake it up when the appropriate program state is reached.

6.4 Explain why spinlocks are not appropriate for single-processor systems yet are often used in multiprocessor systems.

**Answer:** Spinlocks are not appropriate for single-processor systems because the condition that would break a process out of the spinlock could be obtained only by executing a different process. If the process is not relinquishing the processor, other processes do not get the opportunity to set the program condition required for the first process to make progress. In a multiprocessor system, other processes execute on other

processors and thereby modify the program state in order to release the first process from the spinlock.

6.5 Explain why implementing synchronization primitives by disabling interrupts is not appropriate in a single-processor system if the synchronization primitives are to be used in user-level programs.

Answer: If a user-level program is given the ability to disable interrupts, then it can disable the timer interrupt and prevent context switching from taking place, thereby allowing it to use the processor without letting other processes to execute.

6.6 Explain why interrupts are not appropriate for implementing synchronization primitives in multiprocessor systems.

Answer: Interrupts are not sufficient in multiprocessor systems since disabling interrupts only prevents other processes from executing on the processor in which interrupts were disabled; there are no limitations on what processes could be executing on other processors and therefore the process disabling

interrupts cannot guarantee mutually exclusive access to program state.

6.7 Describe how the Swap() instruction can be used to provide mutual exclusion that satisfies the bounded-waiting requirement.

Answer:

```
do {
    waiting[i] = TRUE;
    key = TRUE;
    while (waiting[i] && key) key = Swap(&lock, &key);
    waiting[i] = FALSE;
    /* critical section */
    j = (i+1) % n;
    while ((j != i) && !waiting[j]) j = (j+1) % n;
    if (j == i) lock = FALSE;
    else waiting[j] = FALSE;
    /* remainder section */
} while (TRUE);
```

6.8 Servers can be designed to limit the number of open

connections. For example, a server may wish to have only N socket connections at any point in time. As soon as N connections are made, the server will not accept another incoming connection until an existing connection is released. Explain how semaphores can be used by a server to limit the number of concurrent connections.

Answer: A semaphore is initialized to the number of allowable open socket connections. When a connection is accepted, the acquire() method is called, when a connection is released, the release() method is called. If the system reaches the number of allowable socket connections, subsequent calls to acquire() will block until an existing connection is terminated and the release method is invoked.

6.9 Show that, if the wait() and signal() semaphore operations are not executed atomically, then mutual exclusion may be violated.

Answer: A wait operation atomically decrements the value associated with a semaphore. If two wait operations are executed on a semaphore when its value is 1, if the two

operations are not performed atomically,then it is possible that both operations might proceed to decrement the semaphore value thereby violating mutual exclusion.

6.10 Show how to implement the wait() and signal() semaphore operations in multiprocessor environments using the TestAndSet() instruction. The solution should exhibit minimal busy waiting.

Answer: Here is the pseudocode for implementing the operations:

```
int guard = 0;
int semaphore value = 0;
wait()
{
    while (TestAndSet(&guard) == 1);
    if (semaphore value == 0) {
        atomically add process to a queue of processes waiting
        for the semaphore and set guard to 0;
    } else {
        semaphore value--;
        guard = 0;
```

```
    }
}

signal()
{
    while (TestAndSet(&guard) == 1);
    if (semaphore value == 0 && there is a process on the wait
queue)
        wake up the first process in the queue of waiting
processes;
    else
        semaphore value++;
    guard = 0;
}
```

6.11 The Sleeping-Barber Problem. A barbershop consists of a waiting room with n chairs and a barber room with one barber chair. If there are no customers to be served, the barber goes to sleep. If a customer enters the barbershop and all chairs are occupied, then the customer leaves the shop. If the barber is busy but chairs are available, then the customer sits in one of the free chairs. If the barber is asleep, the customer wakes up

the barber. Write a program to coordinate the barber and the customers.

Answer:

(1) BarberShop.java

```
//This class defines the Barber Shop Scenario.
//This scenario allows N customers to enter it.
//It also contains a number of chairs that allow the customers to wait in.

public class BarberShop
{
    private int chairNum;
    private int barber;
    private int chairState[];
    static final int FULL = -1;
    static final int EMPTY = 0;
    static final int OCUPIED = 1;
    static final int SLEEPING = 2;
    static final int DONE = 3;
    static final int WAITED = 4;
```

```
/*
==========================================
======================
```

Construct a new Barber Shop scenario for customers to get haircuts at. Set the barber shop to have N number of chairs to wait in. Then make all the chairs to be initially empty. Also set the barber to be asleep so that when his first customer comes in he can wake him up.

@param pChairNum The number of waiting chairs in this barber shop

```
==========================================
============================*/
    public BarberShop(int pChairNum)
    {
        chairNum = pChairNum;
        chairState = new int[chairNum];
        barber = SLEEPING;

        //initialize every chair in the waiting room to be
empty
        for(int i = 0; i < chairNum; i++)
```

```
            chairState[i] = EMPTY;
    }

/*
==========================================
======================
This method is called when a customer sees that the Barber is
busy. This means that the customer must wait for the barber.
Therefore, find a chair in the waiting room.  When the
customer finds a chair they set its state to OCUPIED (so there's
only one customer per chair) and return true. Otherwise, there
are no chairs available so return false.
@param pCustomer The customer wanting to find a chair to
wait in.
@return boolean True if chair is found; False otherwise.
==========================================
====================== */
    private boolean findChair(int pCustomer)
    {
        //try to find a chair to wait in
        int test = getFirstEmptyChair();
```

```
        //if barber shop is full return false
        if(test == FULL)
            return false;
        //otherwise sit down in this chair
        else
            chairState[test] = OCUPIED;

        return true;
    }

/*
==========
==========
```

This method is called by the Customer thread to get a haircut. IF the barber's asleep then the customer wakes him up and the customer gets their haircut. IF the barber's already busy then the customer tries to find a chair to wait in. However, if there are no chairs and the barbers busy, then the shop is full, so customer will then leave. If the barber's state is not sleeping or ocupied, then he can take the customer immediately.

This solution doesn't prevent starvation for the waiting customers. If A customer finnaly gets notified to leave the

waiting state they will be the next haircut. If another customer enters the shop at the same time, then that customer will always have to wair for existing customers to be serviced.

@param customer The customer wanting to get a haircut

@return int The state of the Barber Shop and/or barber

========== */

```
    public synchronized int getHairCut(int customer)
    {
        //if the barber's sleeping, then wake him up and tell him to get to work
        if(barber == SLEEPING)
        {
            barber = OCUPIED;
            return SLEEPING;
        }
        //the barber's busy try to wait for him in the waiting room
        else if(barber == OCUPIED)
        {
```

```
boolean test = findChair(customer);

//if barber shop is full return full and get out.
if(test == false)
    return FULL;
else
{
    //wait as long as the barber is busy
    while(barber == OCUPIED)
    {
        try{ this.wait(); }
        catch(InterruptedException e)
        {}
    }

//waiting customer will get to be the next haircut scheduled. Therefore they
//stand up and give up their chair.  This is why a chair gets reset to empty.
    for(int i = 0; i < chairNum; i++)
    {
        if(chairState[i] == OCUPIED)
```

```
                    {
                        chairState[i] = EMPTY;
                        break;
                    }
                }

//set barber to ocupied since this waiting customer will get next haircut.
                barber = OCUPIED;
                return WAITED;
            }
        }
//barber's done.  This customer can get their haircut immediately
        else
        {
            barber = OCUPIED;
            return DONE;
        }
    }

/*
```

```
==========================================
======================
```

This method is called when the customer has recieved their haircut and they leave the shop. They first check to see if anyone else is waiting in the shop. If there isn't anyone, then they were the last customer.This means the barber must go back to sleep. Otherwise poeple are waiting, so just set barber's state to done. Then notify anyone waiting.

@param customer The customer finished with haircut and leaving shop

@return void

```
==========================================
====================== */
    public synchronized void leaveBarberShop(int customer)
    {
        boolean test = isAnyoneWaiting();
        if(test == true)
            barber = DONE;
        else
            barber = SLEEPING;
        //notify only one waiting customer (if any exist)
        //this helps on performance of the program.
```

```
            notify();
        }

/*
==========================================
======================
Find the first empty chair in the waiting room and return it.  If
no chairs are found then all chairs are ocupied by a waiting
customer.
@return int The number of the empty chair; FULL otherwise
==========================================
====================== */
        private int getFirstEmptyChair()
        {
            //if an empty chair is found return it
            for(int i = 0; i < chairNum; i++)
            {
                if(chairState[i] == EMPTY)
                    return i;
            }
            //all chairs are occupied so return FULL
            return FULL;
```

```
    }

/*
==========================================
======================
Test to see if the waiting room has any customers in it.
@return boolean True if there are customers waiting; False if
empty
==========================================
====================== */
    private boolean isAnyoneWaiting()
    {
        //see if anyone is in a chair waiting
        for(int i = 0; i < chairNum; i++)
        {
            if(chairState[i] == OCUPIED)
                return true;
        }
        //couldn't find anyone waiting
        return false;
    }
}
```

(2) Customer.java

```
//This class creates a customer who is wanting a haircut.
//Once the haircut is received the customer then has finished their task
//and they have to be re-instantiated if they want another haircut.
import java.util.*;

public class Customer implements Runnable
{
    private BarberShop shop;
    private int customer;
    private int HAIRCUT_TIME = 5;

   // Construct a new customer with their unique identity
   //and give them a barber shop to enter for their haircut.
    //@param pCustomer This customer's identifier.
    //@param pShop The barber shop they will get a haircut in.
    public Customer(int pCustomer, BarberShop pShop)
    {
```

```
        shop = pShop;
        customer = pCustomer;
    }

    /**
     * The run method for the customer thread tries to get the customer a
     * haircut.
    */
    public void run()
    {
        int sleeptime = (int) (HAIRCUT_TIME * Math.random() );
        System.out.println("ENTERING SHOP: Customer [" + customer + "] entering barber shop for haircut.");
        int  test = BarberShop.OCUPIED;
        //test for this customer's haircut posibility
        test = shop.getHairCut(customer);
        //if the barber was busy, then this customer has waited in the waiting room.
        //This waiting customer will now get the next haircut. Entering threads will
```

```
        //have to wait for the existing customers to be serviced.  As far as which
        //waiting thread will get serviced next is up to the JVM.
        if(test == BarberShop.WAITED)
            System.out.println("Barber's busy: Customer [" + customer + "] has waited and now wants haircut.");
        //otherwise, no one in barber shop, wake up barber and get haircut
        else if (test == BarberShop.SLEEPING)
            System.out.println("Barber's asleep: Customer [" + customer + "] is waking him up and getting haircut.");
        //this barber shop is full.  Therefore leave and never return.
        else if (test == BarberShop.FULL)
        {
            System.out.println("Barber Shop full: Customer [" + customer + "] is leaving shop.");
            return;
        }
        //Barber's ready to take this customer right away for a haircut.
```

```
            else
                System.out.println("HAIRCUT: Customer [" +
customer + "] is getting haircut.");
            //customer is now getting their haircut for an amount
of time
            SleepUtilities.nap();
            //haircut finished.  Leave the shop and notify anyone
waiting.
            System.out.println("LEAVING SHOP: Customer [" +
customer + "] haircut finished: leaving shop.");
            shop.leaveBarberShop(customer);
        }
}
```

(3) SleepUtilities.java

```
//Utilities for causing a thread to sleep.
//Note, we should be handling interrupted exceptions
//but choose not to do so for code clarity.
public class SleepUtilities
{
    // Nap between zero and NAP_TIME seconds.
    public static void nap() {
```

```
        nap(NAP_TIME);
    }

    //Nap between zero and duration seconds.
    public static void nap(int duration) {
            int sleeptime = (int) (NAP_TIME * Math.random() );
            try { Thread.sleep(sleeptime*1000); }
            catch (InterruptedException e) {}
    }
    private static final int NAP_TIME = 5;
}
```

(4) CreateBarberShopTest.java

//Test Stub for the BarberShop\Customer problem.

//This class adds customers to a constructed barber shop indefinately.

```
public class CreateBarberShopTest implements Runnable
{
     static final private int WAIT_TIME = 3;
     static public void main(String [] args)
     {
```

```
        new Thread(new CreateBarberShopTest()).start();
    }

    public void run()
    {
        //create the barber shop
        BarberShop newShop = new BarberShop(15);
        int customerID = 1;

        //add the specifid number of threads to shop
        while(customerID <= 10000)
        {
            //add customers to the barber shop
            new Thread(new Customer(customerID, newShop)).start();
            customerID++;
            //wait this amount of time before adding another customer to shop
            SleepUtilities.nap();
        }
    }
}
```

## 6.12 Demonstrate that monitors and semaphores are equivalent insofar as they can be used to implement the same types of synchronization problems.

Answer: A semaphore can be implemented using the following monitor code:

```
monitor semaphore {
    int value = 0;
    condition c;
    semaphore increment() {
        value++; c.signal();
    }
    semaphore decrement() {
        while (value == 0) c.wait();
        value--;
    }
}
```

A monitor could be implemented using a semaphore in the following manner. Each condition variable is represented by a queue of threads waiting for the condition. Each thread has a semaphore associated with its queue entry. When a thread

performs a wait operation, it creates a new semaphore (initialized to zero), appends the semaphore to the queue associated with the condition variable, and performs a blocking semaphore decrement operation on the newly created semaphore. When a thread performs a signal on a condition variable, the first process in the queue is awakened by performing an increment on the corresponding semaphore.

6.13 Write a bounded-buffer monitor in which the buffers (portions) are em- bedded within the monitor itself.

Answer:

```
monitor bounded_buffer {
    int items[MAX_ITEMS]; int numItems = 0; condition full, empty;
    void produce(int v) {
        while (numItems == MAX_ITEMS) full.wait();
        items[numItems++] = v; empty.signal();
    }
    int consume() {
        int retVal;
```

```
        while (numItems == 0) empty.wait();
        retVal = items[--numItems]; full.signal(); return retVal;
    }
}
```

6.14 The strict mutual exclusion within a monitor makes the bounded-buffer monitor of Exercise 6.13 mainly suitable for small portions.

a. Explain why this is true.

b. Design a new scheme that is suitable for larger portions.

Answer: The solution to the bounded buffer problem given above copies the produced value into the monitor’ s local buffer and copies it back from the monitor’ s local buffer to the consumer. These copyoperations could be expensive if one were using large extents of memory for each buffer region. The increased cost of copy operation means that the monitor is held for a longer period of time while a process is in the produce or consume operation. This decreases the overall throughput of the system. This problem could be alleviated by storing pointers to buffer regions within the monitor instead of storing the buffer regions themselves. Consequently, one

could modify the code given above to simply copy the pointer to the buffer region into and out of the monitor’ s state. This operation should be relatively inexpensive and therefore the period of time that the monitor is being held will be much shorter, thereby increasing the throughput of the monitor.

6.15 Discuss the tradeoff between fairness and throughput of operations in the readers-writers problem. Propose a method for solving the readers- writers problem without causing starvation.

Answer: Throughput in the readers-writers problem is increased by favoring multiple readers as opposed to allowing a single writer to exclusively access the shared values. On the other hand, favoring readers could result in starvation for writers. The starvation in the readers-writers problem could be avoided by keeping timestamps associated with waiting processes. When a writer is finished with its task, it would wake up the process that has been waiting for the longest duration.When a reader arrives and notices that another reader is accessing the database, then it would enter the critical section only if there are no waiting writers. These

restrictions would guarantee fairness.

## 6.16 How does the signal() operation associated with monitors differ from the corresponding operation defined for semaphores?

Answer: The signal() operations associated with monitors is not persistent in the following sense: if a signal is performed and if there are no waiting threads, then the signal is simply ignored and the system does not remember the fact that the signal took place. If a subsequent wait operation is performed, then the corresponding thread simply blocks. In semaphores, on the other hand, every signal results in a corresponding increment of the semaphore value even if there are no waiting threads. A future wait operation would immediately succeed because of the earlier increment.

## 6.17 Suppose the signal() statement can appear only as the last statement in a monitor procedure. Suggest how the implementation described in Section 6.7 can be simplified.

Answer: If the signal operation were the last statement, then

the lock could be transferred from the signalling process to the process that is the recipient of the signal. Otherwise, the signalling process would have to explicitly release the lock and the recipient of the signal would have to compete with all other processes to obtain the lock to make progress.

6.18 Consider a system consisting of processes P1 , P2 , ..., Pn, each of which has a unique priority number. Write a monitor that allocates three identical line printers to these processes, using the priority numbers for deciding the order of allocation.

Answer: Here is the pseudocode:

```
monitor printers {
    int num avail = 3;
    int waiting processes[MAX PROCS];
    int num waiting;
    condition c;
    void request printer(int proc number) {
        if (num avail > 0) {
        num avail--;
        return;
    }
```

```
    waiting processes[num waiting] = proc number;
    num waiting++; sort(waiting processes);
    while (num avail == 0 && waiting processes[0] != proc
number) c.wait();
    waiting processes[0] =waiting processes[num waiting-1];
        num waiting--;
    sort(waiting processes);
    num avail--;
}
void release printer() { num avail++; c.broadcast();
}
}
```

6.19 A file is to be shared among different processes, each of which has a unique number. The file can be accessed simultaneously by several processes, subject to the following constraint: The sum of all unique numbers associated with all the processes currently accessing the file must be less than n. Write a monitor to coordinate access to the file.

Answer: The pseudocode is as follows:

```
monitor file access {
```

```
    int curr sum = 0;
    int n;
    condition c;
    void access file(int my num) {
        while (curr sum+ my num >= n) c.wait();
        curr sum += my num;
    }
    void finish access(int my num) {
        curr sum -= my num;
        c.broadcast();
    }
}
```

6.20 When a signal is performed on a condition inside a monitor, the signaling process can either continue its execution or transfer control to the process that is signaled. How would the solution to the preceding exercise differ with the two different ways in which signaling can be performed?

Answer: The solution to the previous exercise is correct under both situations. However, it could suffer from the problem that a process might be awakened only to find that it is still not

possible for it to make forward progress either because there was not sufficient slack to begin with when a process was awakened or if an intervening process gets control, obtains the monitor and starts accessing the file. Also, note that the broadcast operation wakes up all of the waiting processes. If the signal also transfers control and the monitor from the current thread to the target, then one could check whether the target would indeed be able to make forward progress and perform the signal only if it it were possible. Then the "while" loop for the waiting thread could be replaced by "if" condition since it is guaranteed that the condition will be satisfied when the process is woken up.

6.21 Suppose we replace the wait() and signal() operations of monitors with a single construct await(B), where B is a general Boolean expression that causes the process executing it to wait until B becomes true.

a. Write a monitor using this scheme to implement the readers–writers problem.

b. Explain why,in general, this construct cannot be implemented efficiently.

c. What restrictions need to be put on the await statement so

that it can be implemented efficiently? (Hint: Restrict the generality of B; see Kessels [1977].)

Answer:

- The readers-writers problem could be modified with the following more generate await statements: A reader can perform "await(active writers == 0 && waiting writers == 0)" to check that there are no active writers and there are no waiting writers before it enters the critical section. The writer can perform a "await(active writers == 0 && active readers == 0)" check to ensure mutually exclusive access.
- The system would have to check which one of the waiting threads have to be awakened by checking which one of their waiting conditions are satisfied after a signal. This requires considerable complexity as well as might require some interaction with the compiler to evaluate the conditions at different points in time. One could restrict the boolean condition to be a disjunction of conjunctions with each component being a simple check (equality or inequality with respect to a static value) on a program variable. In that case, the boolean condition could be communicated to the runtime system, which could perform the check every time it needs to

determine which thread to be awakened.

6.22 Write a monitor that implements an alarm clock that enables a calling program to delay itself for a specified number of time units (ticks). You may assume the existence of a real hardware clock that invokes a procedure tick in your monitor at regular intervals.

Answer: Here is a pseudocode for implementing this:

```
monitor alarm {
    condition c;
    void delay(int ticks) {
        int begin time = read clock();
        while (read clock() < begin time + ticks) c.wait();
    }
    void tick() {
        c.broadcast();
    }
}
```

6.23 Why do Solaris, Linux, and Windows 2000 use spinlocks as a synchronization mechanism only on multiprocessor

systems and not on single-processor systems?

Answer: Solaris, Linux,and Windows 2000 use spinlocks as a synchronization mechanism only on multiprocessor systems.Spinlocks are not appropriate for single-processor systems because the condition that would break a process out of the spinlock could be obtained only by executing a different process.If the process is not relinquishing the processor, other processes do not get the opportunity to set the program condition required for the first process to make progress. In a multiprocessor system, other processes execute on other processors and thereby modify the program state in order to release the first process from the spinlock.

6.24 In log-based systems that provide support for transactions, updates to data items cannot be performed before the corresponding entries are logged. Why is this restriction necessary?

Answer: If the transaction needs to be aborted, then the values of the updated data values need to be rolled back to the old values. This requires the old values of the data entries to be

logged before the updates are performed.

## 6.25 Show that the two-phase locking protocol ensures conflict serializability.

Answer: A schedule refers to the execution sequence of the operations for one or more transactions. A serial schedule is the situation where each transaction of a schedule is performed atomically. If a schedule consists of two different transactions where consecutive operations from the different transactions access the same data and at least one of the operations is a write, then we have what is known as a conflict. If a schedule can be transformed into a serial schedule by a series of swaps on nonconflicting operations, we say that such a schedule is conflict serializable.

The two-phase locking protocol ensures conflict serializabilty because exclusive locks (which are used for write operations) must be acquired serially, without releasing any locks during the acquire (growing) phase. Other transactions that wish to acquire the same locks must wait for the first transaction to begin releasing locks. By requiring that all locks must first be acquired before releasing any locks, we are ensuring that

potential conflicts are avoided.

6.26 What are the implications of assigning a new timestamp to a transaction that is rolled back? How does the system process transactions that were issued after the rolled-back transaction but that have timestamps smaller than the new timestamp of the rolled-back transaction?

Answer: If the transactions that were issued after the rolled-back transaction had accessed variables that were updated by the rolled-back transaction, then these transactions would have to rolled-back as well. If they have not performed such operations (that is, there is no overlap with the rolled-back transaction in terms of the variables accessed), then these operations are free to commit when appropriate.

6.27 Assume that a finite number of resources of a single resource type must be managed. Processes may ask for a number of these resources and — once finished — will return them. As an example, many commercial software packages provide a given number of licenses, indicating the number of applications that may run concurrently. When the application

is started, the license count is decremented. When the application is terminated, the license count is incremented. If all licenses are in use, requests to start the application are denied. Such requests will only be granted when an existing license holder terminates the application and a license is returned.

The following program segment is used to manage a finite number of instances of an available resource. The maximum number of resources and the number of available resources are declared as follows:

```
#define MAX RESOURCES 5
int available resources = MAX RESOURCES;
```

When a process wishes to obtain a number of resources, it invokes the

decrease count() function:

```
/* decrease available resources by count resources */
/* return 0 if sufficient resources available, */
/* otherwise return -1 */
int decrease count(int count) {
    if (available resources <count) return -1;
    else {
        available resources -= count;
```

```
        return  0;
    }
}
```

When a process wants to return a number of resources, it calls the decrease count() function:

```
/*  increase  available resources  by  count  */
int  increase count(int  count)  {
    available resources  +=  count;
    return  0;
}
```

The preceding program segment produces a race condition. Do the following:

a. Identify the data involved in the race condition.

b. Identify the location (or locations) in the code where the race condition occurs.

c. Using a semaphore, fix the race condition.

Answer:

- Identify the data involved in the race condition: The variable available resources.
- Identify the location (or locations) in the code where the race condition occurs: The code that decrements available

resources and the code that increments available resources are the statements that could be involved in race conditions.

■ Using a semaphore, fix the race condition: Use a semaphore to represent the available resources variable and replace increment and decrement operations by semaphore increment and semaphore decrement operations.

6.28 The decrease count() function in the previous exercise currently returns 0 if sufficient resources are available and -1 otherwise. This leads to awkward programming for a process that wishes obtain a number of resources:

```
while (decrease count(count) == -1);
```

Rewrite the resource-manager code segment using a monitor and condition variables so that the decrease count() function suspends the process until sufficient resources are available. This will allow a process to invoke decrease count() by simply calling decrease_count(count);

The process will only return from this function call when sufficient resources are available.

Answer:

monitor resources

```
{   int available_resources;
    condition resources_avail;
    int decrease_count(int count)
    {           while (available_resources < count)
resources_avail.wait();
        available_resources -= count;
    }
    int increase_count(int count)
    {   available_resources += count;
        resources_avail.signal();
    }
 }
```

# Chapter 7

7.1 Consider the traffic deadlock depicted in Figure 7.1.

a. Show that the four necessary conditions for deadlock indeed hold in this example.

b. State a simple rule for avoiding deadlocks in this system.

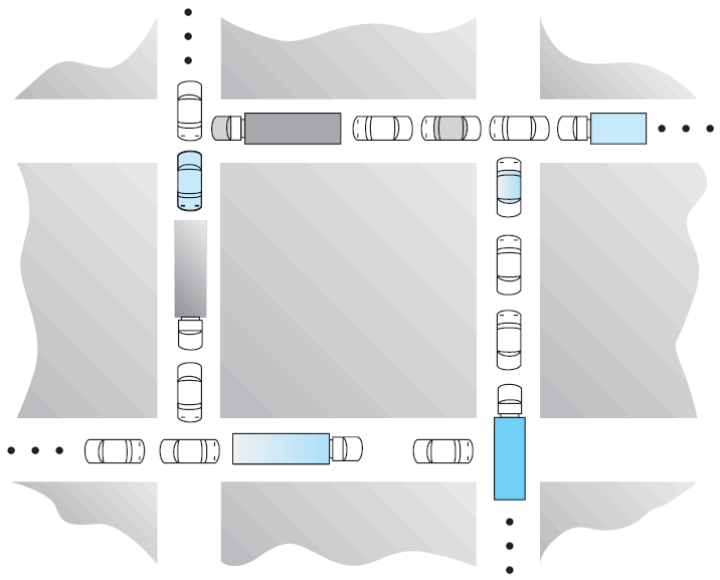


Figure 7.1 Traffic deadlock for Exercise 7.1.

Answer:

a. The four necessary conditions for a deadlock are (1) mutual exclusion;(2) hold-and-wait; (3) no preemption; and (4) circular wait. The mutual exclusion condition holds as only one car can occupy a space in the roadway. Hold-and-wait occurs where a car holds onto their place in the roadway while they wait to advance in the roadway. A car cannot be removed (i.e. preempted) from its position in the roadway. Lastly, there is indeed a circular wait as each car is waiting for a subsequent car to advance. The circular wait condition is also easily observed from the graphic.

b. A simple rule that would avoid this traffic deadlock is that a car may not advance into an intersection if it is clear they will not be able to immediately clear the intersection.

7.2 Consider the deadlock situation that could occur in the dining-philosophers problem when the philosophers obtain the chopsticks one at a time. Discuss how the four necessary conditions for deadlock indeed hold in this setting. Discuss how deadlocks could be avoided by eliminating any one of the four conditions.

Answer: Deadlock is possible because the four necessary conditions hold in the following manner: 1) mutual exclusion is required for chopsticks, 2) the philosophers tend to hold onto the chopstick in hand while they wait for the other chopstick, 3) there is no preemption of chopsticks in the sense that a chopstick allocated to a philosopher cannot be forcibly taken away, and 4) there is a possibility of circular wait. Deadlocks could be avoided by overcoming the conditions in the following manner: 1) allow simultaneous sharing of chopsticks, 2) have the philosophers relinquish the first chopstick if they are unable to obtain the other chopstick, 3) allow for chopsticks to be forcibly taken away if a philosopher has had a chopstick for a long period of time, and 4) enforce a numbering of the chopsticks and always obtain the lower numbered chopstick before obtaining the higher numbered one.

7.3 A possible solution for preventing deadlocks is to have a single, higher-order resource that must be requested before any other resource. For example, if multiple threads attempt to access the synchronization objects A… E , deadlock is possible. (Such synchronization objects may include mutexes,

semaphores, condition variables, etc.) We can prevent the deadlock by adding a sixth object F . Whenever a thread wants to acquire the synchronization lock for any object A...E , it must first acquire the lock for object F . This solution is known as containment: The locks for objects A ... E are contained within the lock for object F . Compare this scheme with the circular-wait scheme of Section 7.4.4.

Answer: This is probably not a good solution because it yields too large a scope. It is better to define a locking policy with as narrow a scope as possible.

7.4 Compare the circular-wait scheme with the deadlock-avoidance schemes (like the banker’ s algorithm) with respect to the following issues:
a. Runtime overheads b. System throughput

Answer: A deadlock-avoidance scheme tends to increase the runtime overheads due to the cost of keep track of the current resource allocation. However, a deadlock-avoidance scheme allows for more concurrent use of resources than schemes that statically prevent the formation of deadlock. In that sense, a

deadlock-avoidance scheme could increase system throughput.

7.5 In a real computer system, neither the resources available nor the demands of processes for resources are consistent over long periods (months).

Resources break or are replaced, new processes come and go, new resources are bought and added to the system. If deadlock is controlled by the banker’s algorithm, which of the following changes can be made safely (without introducing the possibility of deadlock), and under what circumstances?

a. Increase Available (new resources added).

b. Decrease Available (resource permanently removed from system)

c. Increase Max for one process (the process needs more resources than allowed, it may want more)

d. Decrease Max for one process (the process decides it does not need that many resources)

e. Increase the number of processes.

f. Decrease the number of processes.

Answer:

a. Increase Available (new resources added) - This could safely be changed without any problems.

b. Decrease Available (resource permanently removed from system) - This could have an effect on the system and introduce the possibility of deadlock as the safety of the system assumed there were a certain number of available resources.

c. Increase Max for one process (the process needs more resources than allowed, it may want more) - This could have an effect on the system and introduce the possibility of deadlock.

d. Decrease Max for one process (the process decides it does not need that many resources) - This could safely be changed without any problems.

e. Increase the number of processes - This could be allowed assuming that resources were allocated to the new process(es) such that the system does not enter an unsafe state.

f. Decrease the number of processes - This could safely be changed without any problems.

## 7.6 Consider a system consisting of four resources of the same

type that are shared by three processes, each of which needs at most two resources.Show that the system is deadlock-free.

Answer: Suppose the system is deadlocked. This implies that each process is holding one resource and is waiting for one more. Since there are three processes and four resources, one process must be able to obtain two resources. This process requires no more resources and, therefore it will return its resources when done.

7.7 Consider a system consisting of m resources of the same type, being shared by n processes. Resources can be requested and released by processes only one at a time. Show that the system is deadlock free if the following two conditions hold:

a. The maximum need of each process is between 1 and m resources b. The sum of all maximum needs is less than m + n

Answer: Using the terminology of Section 7.6.2, we have:

a. $\sum_{i=1}^{n} Max_i < m + n$

b. $Max_i \geq 1$ for all i

Proof: $Need_i = Max_i - Allocation_i$

If there exists a deadlock state then:

c. $\sum_{i=1}^{n} Allocation_i = m$

Use a. to get: $\sum Need_i + \sum Allocation_i = \sum Max_i < m+n$

Use c. to get: $\sum Need_i + m < m+n$

Rewrite to get: $\sum_{i=1}^{n} Need_i < n$

This implies that there exists a process Pi such that $Need_i=0$. Since $Max_i \geq 1$ it follows that Pi has at least one resource that it can release. Hence the system cannot be in a deadlock state.

7.8 Consider the dining-philosophers problem where the chopsticks are placed at the center of the table and any two of them could be used by a philosopher. Assume that requests for chopsticks are made one at a time. Describe a simple rule for determining whether a particular request could be satisfied without causing deadlock given the current allocation of chopsticks to philosophers.

Answer: The following rule prevents deadlock: when a philosopher makes a request for the first chopstick, do not satisfy the request only if there is no other philosopher with two chopsticks and if there is only one chopstick remaining.

7.9 Consider the same setting as the previous problem. Assume now that each philosopher requires three chopsticks to eat and that resource requests are still issued separately. Describe some simple rules for determining whether a particular request could be satisfied without causing deadlock given the current allocation of chopsticks to philosophers.

Answer: When a philosopher makes a request for a chopstick, allocate the request if: 1) the philosopher has two chopsticks and there is at least one chopstick remaining, 2) the philosopher has one chopstick and there is at least two chopsticks remaining, 3) there is at least one chopstick remaining, and there is at least one philosopher with three chopsticks, 4) the philosopher has no chopsticks, there are two chopsticks remaining, and there is at least one other philosopher with two chopsticks assigned.

7.10 We can obtain the banker's algorithm for a single resource type from the general banker's algorithm simply by reducing the dimensionality of the various arrays by 1. Show through an example that the multiple-resource-type

banker’ s scheme cannot be implemented by individual application of the single-resource-type scheme to each resource type.

Answer: Consider a system with resources A, B, and C and processes P0, P1, P2, P3, and P4 with the following values of Allocation:

| Allocation | | | |
|---|---|---|---|
| | A | B | C |
| P0 | 0 | 1 | 0 |
| P1 | 3 | 0 | 2 |
| P2 | 3 | 0 | 2 |
| P3 | 2 | 1 | 1 |
| P4 | 0 | 0 | 2 |

And the following value of Need:

| Need | | | |
|---|---|---|---|
| | A | B | C |
| P0 | 7 | 4 | 3 |
| P1 | 0 | 2 | 0 |
| P2 | 6 | 0 | 0 |
| P3 | 0 | 1 | 1 |
| P4 | 4 | 3 | 1 |

If the value of Available is (2 3 0), we can see that a request from process P0 for (0 2 0) cannot be satisfied as this lowers Available to (2 1 0) and no process could safely finish.

However, if we were to treat the three resources as three single-resource types of the banker’ s algorithm, we get the following:

For resource A (which we have 2 available),

| | Allocated | Need |
|---|---|---|
| P0 | 0 | 7 |
| P1 | 3 | 0 |
| P2 | 3 | 6 |
| P3 | 2 | 0 |
| P4 | 0 | 4 |

Processes could safely finish in the order of P1, P3, P4, P2, P0.For resource B (which we now have 1 available as 2 were assumed assigned to process P0 ),

| | Allocated | Need |
|---|---|---|
| P0 | 3 | 2 |
| P1 | 0 | 2 |
| P2 | 0 | 0 |
| P3 | 1 | 1 |

| | | |
|---|---|---|
| P4 | 0 | 3 |

Processes could safely finish in the order of P2, P3, P1, P0, P4.And finally, for For resource C (which we have 0 available),

| | Allocated | Need |
|---|---|---|
| P0 | 0 | 3 |
| P1 | 2 | 0 |
| P2 | 2 | 0 |
| P3 | 1 | 1 |
| P4 | 2 | 1 |

Processes could safely finish in the order of P1, P2, P0, P3, P4.As we can see, if we use the banker’ s algorithm for multiple resource types, the request for resources (0 2 0) from process P0 is denied as it leaves the system in an unsafe state. However, if we consider the banker’ s algorithm for the three separate resources where we use a single resource type, the request is granted. Therefore, if we have multiple resource types, we must use the banker ’ s algorithm for multiple resource types.

## 7.11 Consider the following snapshot of a system:

| Allocation | Max | Available |
|---|---|---|
| A B C D | A B C D | A B C D |

| | | | |
|---|---|---|---|
| P0 | 0 0 1 2 | 0 0 1 2 | 1 5 2 0 |
| P1 | 1 0 0 0 | 1 7 5 0 | |
| P2 | 1 3 5 4 | 2 3 5 6 | |
| P3 | 0 6 3 2 | 0 6 5 2 | |
| P4 | 0 0 1 4 | 0 6 5 6 | |

Answer the following questions using the banker's algorithm:

a. What is the content of the matrix Need?

b. Is the system in a safe state?

c. If a request from process P1 arrives for (0,4,2,0), can the request be granted immediately?

Answer:

a. What is the content of the matrix Need? The values of Need for processes P0 through P4 respectively are (0, 0, 0, 0), (0, 7, 5, 0), (1, 0, 0, 2), (0, 0, 2, 0), and (0, 6, 4, 2).

b. Is the system in a safe state? Yes. With Available being equal to (1, 5, 2, 0), either process P0 or P3 could run. Once process P3 runs, it releases its resources which allow all other existing processes to run.

c. If a request from process P1 arrives for (0,4,2,0), can the request be granted immediately? Yes it can. This results in the

value of Available being (1, 1, 0, 0). One ordering of processes that can finish is P0 , P2 , P3 , P1 , and P4 .

7.12 What is the optimistic assumption made in the deadlock-detection algorithm? How could this assumption be violated?

Answer: The optimistic assumption is that there will not be any form of circular-wait in terms of resources allocated and processes making requests for them. This assumption could be violated if a circular-wait does indeed in practice.

7.13 Write a multithreaded Java program that implements the banker ' s algorithm discussed in Section 7.5.3. Create n threads that request and release resources from the banker. A banker will only grant the request if it leaves the system in a safe state. Ensure that access to shared data is thread-safe by employing Java thread synchronization as discussed in Section 7.8.

Answer: (I have not completed it up to now. Maybe someday it will be released.)

7.14 A single-lane bridge connects the two Vermont villages of North Tunbridge and South Tunbridge. Farmers in the two villages use this bridge to deliver their produce to the neighboring town. The bridge can become deadlocked if both a northbound and a southbound farmer get on the bridge at the same time (Vermont farmers are stubborn and are unable to back up.) Using semaphores, design an algorithm that prevents deadlock. Initially, do not be concerned about starvation (the situation
in which northbound farmers prevent southbound farmers from using the bridge, or vice versa).

Answer: semaphore ok to cross = 1;

```
void enter bridge() {
        ok to cross.wait();
}
void exit bridge() {
        ok to cross.signal();
}
```

7.15 Modify your solution to Exercise 7.15 so that it is

starvation-free.

Answer: monitor bridge {

int num waiting north = 0; int num waiting south = 0; int on bridge = 0;

condition ok to cross;

int prev = 0;

void enter bridge north() {

num waiting north++;

while (on bridge ||(prev == 0 && num waiting south > 0)) ok to cross.wait();

num waiting north--;

prev = 0;

}

void exit bridge north() {

on bridge = 0;

ok to cross.broadcast();

}

void enter bridge south() {

num waiting south++;

while (on bridge ||(prev == 1 && num waiting north > 0)) ok to cross.wait();

```
    num waiting south--;
    prev = 1;
}
void exit bridge south() {
    on bridge = 0;
    ok to cross.broadcast();
}
}
```

## Chapter 8

8.1 Explain the difference between internal and external fragmentation.

Answer: Internal Fragmentation is the area in a region or a page that is not used by the job occupying that region or page. This space is unavailable for use by the system until that job is finished and the page or region is released.

8.2 Consider the following process for generating binaries. A compiler is used to generate the object code for individual modules, and a linkage editor is used to combine multiple

object modules into a single program binary. How does the linkage editor change the binding of instructions and data to memory addresses? What information needs to be passed from the compiler to the linkage editor to facilitate the memory binding tasks of the linkage editor?

Answer: The linkage editor has to replace unresolved symbolic addresses with the actual addresses associated with the variables in the final program binary. In order to perform this, the modules should keep track of instructions that refer to unresolved symbols. During linking, each module is assigned a sequence of addresses in the overall program binary and when this has been performed, unresolved references to symbols exported by this binary could be patched in other modules since every other module would contain the list of instructions that need to be patched.

8.3 Given five memory partitions of 100 KB, 500 KB, 200 KB, 300 KB, and 600KB (in order), how would each of the first-fit, best-fit, and worst-fit algorithms place processes of 212 KB, 417 KB, 112 KB, and 426 KB (in order)? Which algorithm makes the most efficient use of memory?

Answer:

(1) First-fit:

a. 212K is put in 500K partition

b. 417K is put in 600K partition

c. 112K is put in 288K partition (new partition 288K = 500K - 212K)

d. 426K must wait

(2) Best-fit:

e. 212K is put in 300K partition

f. 417K is put in 500K partition

g. 112K is put in 200K partition

h. 426K is put in 600K partition

(3) Worst-fit:

i. 212K is put in 600K partition

j. 417K is put in 500K partition

k. 112K is put in 388K partition

l. 426K must wait

In this example, Best-fit turns out to be the best.

8.4 Most systems allow programs to allocate more memory to its address space during execution. Data allocated in the heap

segments of programs is an example of such allocated memory. What is required to support dynamic memory allocation in the following schemes:

a. contiguous-memory allocation b. pure segmentation

c. pure paging

Answer:

- contiguous-memory allocation: might require relocation of the entire program since there is not enough space for the program to grow its allocated memory space.
- pure segmentation: might also require relocation of the segment that needs to be extended since there is not enough space for the segment to grow its allocated memory space.
- pure paging: incremental allocation of new pages is possible in this scheme without requiring relocation of the program's address space.

8.5 Compare the main memory organization schemes of contiguous-memory allocation, pure segmentation, and pure paging with respect to the following issues:

a. external fragmentation b. internal fragmentation

c. ability to share code across processes

Answer: ontiguous memory allocation scheme suffers from external fragmentation as address spaces are allocated contiguously and holes develop as old processes dies and new processes are initiated. It also does not allow processes to share code, since a process' s virtual memory segment is not broken into non-contiguous finegrained segments. Pure segmentation also suffers from external fragmentation as a segment of a process is laid out contiguously in physical memory and fragmentation would occur as segments of dead processes are replaced by segments of new processes. Segmentation, however, enables processes to share code;
for instance, two different processes could share a code segment but have distinct data segments. Pure paging does not suffer from external fragmentation, but instead suffers from internal fragmentation. Processes are allocated in page granularity and if a page is not completely utilized, it results in internal fragmentation and a corresponding wastage of space. Paging also enables processes to share code at the granularity of pages.

8.6 On a system with paging, a process cannot access memory

that it does not own; why? How could the operating system allow access to other memory? Why should it or should it not?

Answer: An address on a paging system is a logical page number and an offset. The physical page is found by searching a table based on the logical page number to produce a physical page number. Because the operating system controls the contents of this table, it can limit a process to accessing only those physical pages allocated to the process. There is no way for a process to refer to a page it does not own because the page will not be in the page table. To allow such access, an operating system

simply needs to allow entries for non-process memory to be added to the process' s page table. This is useful when two or more processes need to exchange data — they just read and write to the same physical addresses (which may be at varying logical addresses). This makes for very efficient interprocess communication.

8.7 Compare paging with segmentation with respect to the amount of memory required by the address translation structures in order to convert

virtual addresses to physical addresses.

Answer: Paging requires more memory overhead to maintain the translation structures. Segmentation requires just two registers per segment: one to maintain the base of the segment and the other to maintain the extent of the segment. Paging on the other hand requires one entry per page, and this entry provides the physical address in which the page is located.

8.8 Program binaries in many systems are typically structured as follows. Code is stored starting with a small fixed virtual address such as 0. The code segment is followed by the data segment that is used for storing the program variables. When the program starts executing, the stack is allocated at the other end of the virtual address space and is allowed to grow towards lower virtual addresses. What is the significance of the above structure on the following schemes:

a. contiguous-memory allocation b. pure segmentation

c. pure paging

Answer: 1) Contiguous-memory allocation requires the

operating system to allocate the entire extent of the virtual address space to the program when it starts executing. This could be much higher than the actual memory requirements of the process. 2) Pure segmentation gives the operating system flexibility to assign a small extent to each segment at program startup time and extend the segment if required. 3) Pure paging does not require the operating system to allocate the maximum extent of

the virtual address space to a process at startup time, but it still requires the operating system to allocate a large page table spanning all of the program’ s virtual address space. When a program needs to extend the stack or the heap, it needs to allocate a new page but the corresponding page table entry is preallocated.

8.9 Consider a paging system with the page table stored in memory.

a. If a memory reference takes 200 nanoseconds, how long does a paged memory reference take?

b. If we add associative registers, and 75 percent of all page-table references are found in the associative registers, what is the effective memory reference time? (Assume that

finding a page-table entry in the associative registers takes zero time, if the entry is there.)

Answer:

a. 400 nanoseconds; 200 nanoseconds to access the page table and 200 nanoseconds to access the word in memory.

b. Effective access time = 0.75 × (200 nanoseconds) + 0.25 × (400 nanoseconds) = 250 nanoseconds.

8.10 Why are segmentation and paging sometimes combined into one scheme?

Answer: Segmentation and paging are often combined in order to improve upon each other. Segmented paging is helpful when the page table becomes very large. A large contiguous section of the page table that is unused can be collapsed into a single segment table entry with a page-table address of zero. Paged segmentation handles the case of having very long segments that require a lot of time for allocation. By paging the segments, we reduce wasted memory due to external fragmentation as well as simplify the allocation.

8.11 Explain why it is easier to share a reentrant module using segmentation
than it is to do so when pure paging is used.

Answer: Since segmentation is based on a logical division of memory rather than a physical one, segments of any size can be shared with only one entry in the segment tables of each user. With paging there must be a common entry in the page tables for each page that is shared.

8.12 Consider the following segment table:

| Segment | Base | Length |
|---|---|---|
| 0 | 219 | 600 |
| 1 | 2300 | 14 |
| 2 | 90 | 100 |
| 3 | 1327 | 580 |
| 4 | 1952 | 96 |

What are the physical addresses for the following logical addresses?

a. 0,430 b. 1,10 c. 2,500 d. 3,400 e. 4,112

Answer:

a. 219 + 430 = 649

b. 2300 + 10 = 2310

c. illegal reference, trap to operating system d. 1327 + 400 = 1727

e. illegal reference, trap to operating system

8.13 What is the purpose of paging the page tables?

Answer: In certain situations the page tables could become large enough that by paging the page tables, one could simplify the memory allocation problem (by ensuring that everything is allocated as fixed-size pages as opposed to variable-sized chunks) and also enable the swapping of portions of page table that are not currently used.

8.14 Consider the hierarchical paging scheme used by the VAX architecture. How many memory operations are performed when an user program executes a memory load operation?

Answer: When a memory load operation is performed, there are three memory operations that might be performed. One is

to translate the position where the page table entry for the page could be found (since page tables themselves are paged). The second access is to access the page table entry itself, while the third access is the actual memory load operation.

8.15 Compare the segmented paging scheme with the hashed page tables scheme for handling large address spaces. Under what circumstances is one scheme preferrable over the other?

Answer: When a program occupies only a small portion of its large virtual address space, a hashed page table might be preferred due to its smaller size. The disadvantage with hashed page tables however is the problem that arises due to conflicts in mapping multiple pages onto the same hashed page table entry. If many pages map to the same entry, then traversing the list corresponding to that hash table entry could incur a significant overhead; such overheads are minimal in the segmented paging scheme where each page table entry maintains information regarding only one page.

8.16 Consider the Intel address translation scheme shown in Figure 8.22.

a. Describe all the steps that the Intel 80386 takes in translating a logical address into a physical address.

b. What are the advantages to the operating system of hardware that provides such complicated memory translation hardware?

c. Are there any disadvantages to this address-translation system? If so, what are they? If not, why is it not used by every manufacturer?

Answer:

a. The selector is an index into the segment descriptor table. The segment descriptor result plus the original offset is used to produce a linear address with a dir, page, and offset. The dir is an index into a page directory. The entry from the page directory selects the page table, and the page field is an index into the page table. The entry from the page table, plus the offset, is the physical address.

b. Such a page translation mechanism offers the flexibility to allow most operating systems to implement their memory scheme in hardware, instead of having to implement some parts in hardware and some in software. Because it can be done in hardware, it is more efficient (and the kernel is

simpler).

c. Address translation can take longer due to the multiple table lookups it can invoke. Caches help, but there will still be cache misses.

# Chapter 9

## 9.1 Give an example that illustrates the problem with

restarting the block move instruction (MVC) on the IBM 360/370 when the source and destination regions are overlapping.

Answer: Assume that the page boundary is at 1024 and the move instruction is moving values from a source region of 800:1200 to a target region of 700:1100. Assume that a page fault occurs while accessing location 1024. By this time the locations of 800:923 have been overwritten with the new values and therefore restarting the block move instruction would result in copying the new values in 800:923 to locations 700:823, which is incorrect.

9.2 Discuss the hardware support required to support demand paging.

Answer: For every memory access operation, the page table needs to be consulted to check whether the corresponding page is resident or not and whether the program has read or write privileges for accessing the page. These checks would have to be performed in hardware. A TLB could serve as a cache and improve the performance of the lookup operation.

9.3 What is the copy-on-write feature and under what circumstances is it beneficial to use this feature? What is the hardware support required to implement this feature?

Answer: When two processes are accessing the same set of program values (for instance, the code segment of the source binary), then it is useful to map the corresponding pages into the virtual address spaces of the two programs in a write-protected manner. When a write does indeed take place, then a copy must be made to allow the two programs to individually access the different copies without interfering with each other. The hardware support required to implement is simply the following: on each memory access, the page table needs to be consulted to check whether the page is write-protected. If it is indeed write-protected, a trap would occur and the operating system could resolve the issue.

9.4 A certain computer provides its users with a virtual-memory space of $2^{32}$ bytes. The computer has $2^{18}$ bytes of physical memory. The virtual memory is implemented by paging, and the page size is 4096 bytes. A user process

generates the virtual address 11123456. Explain how the system establishes the corresponding physical location. Distinguish between software and hardware operations.

Answer: The virtual address in binary form is 0001 0001 0001 0010 0011 0100 0101 0110 Since the page size is $2^{12}$ , the page table size is $2^{20}$ . Therefore the low-order 12 bits "0100 0101 0110" are used as the displacement into the page,while the remaining 20 bits "0001 0001 0001 0010 0011" are used as the displacement in the page table.

9.5 Assume we have a demand-paged memory. The page table is held in registers. It takes 8 milliseconds to service a page fault if an empty page is available or the replaced page is not modified, and 20 milliseconds if the replaced page is modified. Memory access time is 100 nanoseconds. Assume that the page to be replaced is modified 70 percent of the time. What is the maximum acceptable page-fault rate for an effective access time of no more than 200 nanoseconds?

Answer: 0.2μsec = (1 - P) × 0.1μsec + (0.3P)×8 millisec + (0.7P) × 20 millisec

$0.1P = -0.1P + 2400P + 14000P$

$P \approx 0.000006$

9.6 Assume that you are monitoring the rate at which the pointer in the clock algorithm (which indicates the candidate page for replacement) moves. What can you say about the system if you notice the following behavior:

a. pointer is moving fast

b. pointer is moving slow

Answer: If the pointer is moving fast, then the program is accessing a large number of pages simultaneously. It is most likely that during the period between the point at which the bit corresponding to a page is cleared and it is checked again, the page is accessed again and therefore cannot be replaced. This results in more scanning of the pages before a victim page is found. If the pointer is moving slow, then the virtual memory system is finding candidate pages for replacement extremely efficiently, indicating that many of the resident pages are not being accessed.

9.7 Discuss situations under which the least frequently used

page-replacement algorithm generates fewer page faults than the least recently used page- replacement algorithm. Also discuss under what circumstance does the opposite holds.

Answer: Consider the following sequence of memory accesses in a system that can hold four pages in memory. Sequence: 1 1 2 3 4 5 1.

When page 5 is accessed, the least frequently used page-replacement algorithm would replace a page other than 1, and therefore would not incur a page fault when page 1 is accessed again. On the other hand, for the sequence "1 2 3 4 5 2," the least recently used algorithm performs better.

9.8 Discuss situations under which the most frequently used page-replacement algorithm generates fewer page faults than the least recently used page-replacement algorithm. Also discuss under what circumstance does the opposite holds.

Answer: Consider the sequence in a system that holds four pages in memory: 1 2 3 4 4 4 5 1. The most frequently used page replacement algorithm evicts page 4 while fetching page 5, while the LRU algorithm evicts page 1. This is unlikely to

happen much in practice. For the sequence "1 2 3 4 4 4 5 1," the LRU algorithm makes the right decision.

9.9 The VAX/VMS system uses a FIFO replacement algorithm for resident pages and a free-frame pool of recently used pages. Assume that the free-frame pool is managed using the least recently used replacement policy. Answer the following questions:

a. If a page fault occurs and if the page does not exist in the free-frame pool, how is free space generated for the newly requested page?

b. If a page fault occurs and if the page exists in the free-frame pool, how is the resident page set and the free-frame pool managed to make space for the requested page?

c. What does the system degenerate to if the number of resident pages is set to one?

d. What does the system degenerate to if the number of pages in the free-frame pool is zero?

Answer:

- When a page fault occurs and if the page does not exist in the free-frame pool, then one of the pages in the free-frame

pool is evicted to disk, creating space for one of the resident pages to be moved to the free-frame pool. The accessed page is then moved to the resident set.

- When a page fault occurs and if the page exists in the free-frame pool, then it is moved into the set of resident pages, while one of the resident pages is moved to the free-frame pool.
- When the number of resident pages is set to one, then the system degenerates into the page replacement algorithm used in the free-frame pool, which is typically managed in a LRU fashion.
- When the number of pages in the free-frame pool is zero, then the system degenerates into a FIFO page replacement algorithm.

9.10 Consider a demand-paging system with the following time-measured utilizations: CPU utilization 20% Paging disk 97.7% Other I/O devices 5%

Which (if any) of the following will (probably) improve CPU utilization? Explain your answer.

a. Install a faster CPU.

b. Install a bigger paging disk.

c. Increase the degree of multiprogramming.

d. Decrease the degree of multiprogramming.

e. Install more main memory.

f. Install a faster hard disk or multiple controllers with multiple hard disks.

g. Add prepaging to the page fetch algorithms. h. Increase the page size.

Answer: The system obviously is spending most of its time paging, indicating over-allocation of memory. If the level of multiprogramming is reduced resident processes would page fault less frequently and the CPU utilization would improve. Another way to improve performance would be to get more physical memory or a faster paging drum.

a. Get a faster CPU —No.

b. Get a bigger paging drum — No.

c. Increase the degree of multiprogramming — No.

d. Decrease the degree of multiprogramming — Yes.

e. Install more main memory — Likely to improve CPU utilization as more pages can remain resident and not require paging to or from the disks.

f. Install a faster hard disk, or multiple controllers with multiple

hard disks — Also an improvement, for as the disk bottleneck is removed by faster response and more throughput to the disks, the CPU will get more data more quickly.

g. Add prepaging to the page fetch algorithms — Again, the CPU will get more data faster, so it will be more in use. This is only the case if the paging action is amenable to prefetching (i.e., some of the access is sequential).

h. Increase the page size — Increasing the page size will result in fewer page faults if data is being accessed sequentially. If data access is more or less random, more paging action could ensue because fewer pages can be kept in memory and more data is transferred per page fault. So this change is as likely to decrease utilization as it is to increase it.

9.11 Suppose that a machine provides instructions that can access memory locations using the one-level indirect addressing scheme. What is the sequence of page faults incurred when all of the pages of a program are currently non-resident and the first instruction of the program is an indirect memory load operation? What happens when the operating system is using a per-process frame allocation technique and only two pages are allocated to this process?

Answer: The following page faults take place: page fault to access the instruction, a page fault to access the memory location that contains a pointer to the target memory location, and a page fault when the target memory location is accessed. The operating system will generate three page faults with the third page replacing the page containing the instruction. If the instruction needs to be fetched again to repeat the trapped instruction, then the sequence of page faults will continue indefinitely.

If the instruction is cached in a register, then it will be able to execute

completely after the third page fault.

9.12 Suppose that your replacement policy (in a paged system) is to examine each page regularly and to discarding that page if it has not been used since the last examination. What would you gain and what would you lose by using this policy rather than LRU or second-chance replacement?

Answer: Such an algorithm could be implemented with the use of a reference bit. After every examination, the bit is set to

zero; set back to one if the page is referenced. The algorithm would then select an arbitrary page for replacement from the set of unused pages since the last examination.

The advantage of this algorithm is its simplicity - nothing other than a reference bit need be maintained. The disadvantage of this algorithm is that it ignores locality by only using a short time frame for determining

whether to evict a page or not. For example, a page may be part of the working set of a process, but may be evicted because it was not referenced since the last examination (i.e. not all pages in the working set may be referenced between examinations.)

9.13 A page-replacement algorithm should minimize the number of page faults. We can do this minimization by distributing heavily used pages evenly over all of memory, rather than having them compete for a small number of page frames. We can associate with each page frame a counter of the number of pages that are associated with that frame. Then, to replace a page, we search for the page frame with the smallest counter.

《页替换算法应尽量减少页错误。我们可以做到这一点通过尽量减少

大量使用分布均匀页上的所有存储，而不是让他们竞争少数页面帧。我们可以联系每个页内柜台的数量，页相关的框架。然后，以取代一个页，我们寻求的最小的数量的页帧。》

a. Define a page-replacement algorithm using this basic idea. Specifically address the problems of (1) what the initial value of the counters is, (2) when counters are increased, (3) when counters are decreased, and (4) how the page to be replaced is selected.《定义一个页面替换算法利用这一基本思想。具体处理的问题（ 1 ）计数器的初始值是什么，（ 2 ）当计数器增加，（ 3 ）当计数器减少，（ 4 ）如何选中被取代的页。》

b. How many page faults occur for your algorithm for the following reference string, for four page frames?

1, 2, 3, 4, 5, 3, 4, 1, 6, 7, 8, 7, 8, 9, 7, 8, 9, 5, 4, 5, 4, 2.

c. What is the minimum number of page faults for an optimal page-replacement strategy for the reference string in part b with four page frames?

Answer:

a. Define a page-replacement algorithm addressing the problems of:

(1) Initial value of the counters — 0.

(2) Counters are increased — whenever a new page is

associated with that frame.

(3) Counters are decreased — whenever one of the pages associated with that frame is no longer required.

(4) How the page to be replaced is selected — find a frame with the smallest counter. Use FIFO for breaking ties.

b. 14 page faults

c. 11 page faults

9.14 Consider a demand-paging system with a paging disk that has an average access and transfer time of 20 milliseconds. Addresses are translated through a page table in main memory, with an access time of 1 microsecond per memory access. Thus, each memory reference through the page table takes two accesses. To improve this time, we have added an associative memory that reduces access time to one memory reference, if the page-table entry is in the associative memory.

Assume that 80 percent of the accesses are in the associative memory and that, of the remaining, 10 percent (or 2 percent of the total) cause page faults. What is the effective memory access time?

Answer:

effective access time = 0.8×1μsec+ 0.1×2μsec+0.1×5002μsec

= 501.2 sec

= 0.5 millisec

9.15 What is the cause of thrashing? How does the system detect thrashing? Once it detects thrashing, what can the system do to eliminate this problem?

《系统抖动的原因是什么？系统如何检测系统抖动？一旦侦测到系统抖动，怎么做的系统，以消除这个问题？》

Answer: Thrashing is caused by underallocation of the minimum number of pages required by a process, forcing it to continuously page fault. The system can detect thrashing by evaluating the level of CPU utilization as compared to the level of multiprogramming. It can be eliminated by reducing the level of multiprogramming.

9.16 Is it possible for a process to have two working sets? One representing

data and another representing code? Explain.

Answer: Yes, in fact many processors provide two TLB’ s for

this very reason. As an example, the code being accessed by a process may retain the same working set for a long period of time. However, the data the code accesses may change, thus reflecting a change in the working set for data accesses.

9.17 Consider the parameter used to define the working-set window in the working-set model. What is the effect of setting to a small value on the page fault frequency and the number of active (non-suspended) processes currently executing in the system? What is the effect when is set to a very high value?

Answer: When is set to a small value, then the set of resident pages for a process might be underestimated allowing a process to be scheduled even though all of its required pages are not resident. This could result in a large number of page faults. When is set to a large value, then a process' s resident set is overestimated and this might prevent many processes from being scheduled even though their required pages are resident. However, once a process is scheduled, it is unlikely to generate

page faults since its resident set has been overestimated.

9.18 Assume there is an initial 1024 KB segment where memory is allocated using the Buddy System. Using Figure 9.27 as a guide, draw the tree illustrating how the following memory requests are allocated:

- request 240 bytes
- request 120 bytes
- request 60 bytes
- request 130 bytes

Next, modify the tree for the following releases of memory. Perform coalescing whenever possible:

- release 240 bytes
- release 60 bytes
- release 120 bytes

Answer: The following allocation is made by the Buddy system: The 240 byte request is assigned a 256 byte segment. The 120 byte request is assigned a 128 byte segement, the 60 byte request is assigned a 64 byte segment and the 130 byte request is assigned a 256 byte segment. After the allocation, the following segment sizes are available: 64 bytes, 256bytes, 1K, 2K, 4K, 8K, 16K, 32K, 64K, 128K, 256K, and 512K. After the releases of memory, the only segment in use would be a

256byte segment containing 130 bytes of data. The following segments will be free: 256 bytes, 512 bytes, 1K, 2K, 4K, 8K, 16K, 32K, 64K, 128K, 256K, and 512K.

9.19 The slab allocation algorithm uses a separate cache for each different object type. Assuming there is one cache per object type, explain why this doesn't scale well with multiple CPUs. What could be done to address this scalability issue?

Answer: This had long been a problem with the slab allocator - poor scalability with multiple CPUs. The issue comes from having to lock the global cache when it is being accesses. This has the effect of serializing cache accesses on multiprocessor systems. Solaris has addressed this by introducing a per-CPU cache, rather than a single global cache.

9.20 Consider a system that allocates pages of different sizes to its processes. What are the advantages of such a paging scheme? What modifications to the virtual memory system are provide this functionality?

Answer: The program could have a large code segment or use

large sized arrays as data. These portions of the program could be allocated to larger pages, thereby decreasing the memory overheads associated with a page table. The virtual memory system would then have to maintain multiple free lists of pages for the different sizes and should also need to have more complex code for address translation to take into account different page sizes.

9.21 Write a program that implements the FIFO and LRU page-replacement algorithms presented in this chapter. First, generate a random page-reference string where page numbers range from 0..9. Apply the random page-reference string to each algorithm and record the number of page faults incurred by each algorithm. Implement the replacement algorithms such that the number of page frames can vary from 1..7. Assume that demand paging is used.

Answer: Please refer to the experiment manual for source code solution.

9.22 The Catalan numbers are an integer sequence Cn that appear in tree enumeration problems. The first Catalan numbers for n = 1, 2, 3, ... are 1, 2, 5, 14, 42, 132, .... A formula

generating Cn is

$$C_n = \left(\frac{1}{n+1}\right)\left(\frac{2n}{n}\right) = \frac{(2n)!}{(n+1)n!}$$

Design two programs that communicate with shared memory using the Win32 API as outlined in Section 9.7.2. The producer process will generate the Catalan sequence and write it to a shared memory object. The consumer process will then read and output the sequence from shared memory.

In this instance, the producer process will be passed an integer parameter on the command line specifying the number of Catalan numbers to produce, i.e. providing 5 on the command line means the producer process will generate the first 5 Catalan numbers.

Answer: (I have not completed it up to now. Maybe someday it will be released.)

# Chapter 10

10.1 Consider a file system where a file can be deleted and its disk space reclaimed while links to that file still exist. What problems may occur if a new file is created in the same storage area or with the same absolute path name? How can these problems be avoided?

Answer: Let F1 be the old file and F2 be the new file. A user wishing to access F1 through an existing link will actually access F2. Note that the access protection for file F1 is used rather than the one associated with F2.

This problem can be avoided by insuring that all links to a deleted file are deleted also. This can be accomplished in several ways:

a. maintain a list of all links to a file, removing each of them when the file is deleted

b. retain the links, removing them when an attempt is made to access a deleted file

c. maintain a file reference list (or counter), deleting the file only after all links or references to that file have been deleted.

10.2 The open-file table is used to maintain information about files that are currently open. Should the operating system

maintain a separate table for each user or just maintain one table that contains references to files that are being accessed by all users at the current time? If the same file is being accessed by two different programs or users, should there be separate entries in the open file table?

Answer: By keeping a central open-file table, the operating system can perform the following operation that would be infeasible other- wise. Consider a file that is currently being accessed by one or more processes. If the file is deleted, then it should not be removed from the disk until all processes accessing the file have closed it. This check could be performed only if there is centralized accounting of number of processes accessing the file. On the other hand, if two processes are accessing the file, then separate state needs to be maintained to keep track of the current location of which parts of the file are being accessed by the two processes. This requires the operating system to maintain separate entries for the two processes.

10.3 What are the advantages and disadvantages of a system providing mandatory locks instead of providing advisory locks

whose usage is left to the users' discretion?

Answer: In many cases, separate programs might be willing to tolerate concurrent access to a file without requiring the need to obtain locks and thereby guaranteeing mutual exclusion to the files. Mutual exclusion could be guaranteed by other program structures such as memory locks or other forms of synchronization. In such situations, the mandatory locks would limit the flexibility in how files could be accessed and might also increase the overheads associated with accessing files.

10.4 What are the advantages and disadvantages of recording the name of the creating program with the file's attributes (as is done in the Macintosh Operating System)?

Answer: By recording the name of the creating program, the operating system is able to implement features (such as automatic program invocation when the file is accessed) based on this information. It does add overhead in the operating system and require space in the file descriptor, however.

10.5 Some systems automatically open a file when it is referenced for the first time, and close the file when the job terminates. Discuss the advantages and disadvantages of this scheme as compared to the more traditional one, where the user has to open and close the file explicitly.

Answer: Automatic opening and closing of files relieves the user from the invocation of these functions, and thus makes it more convenient to the user; however, it requires more overhead than the case where explicit opening and closing is required.

10.6 If the operating system were to know that a certain application is going to access the file data in a sequential manner, how could it exploit this information to improve performance?

Answer: When a block is accessed, the file system could prefetch the subsequent blocks in anticipation of future requests to these blocks. This prefetching optimization would reduce the waiting time experienced by the process for future requests.

10.7 Give an example of an application that could benefit from operating system support for random access to indexed files.

Answer: An application that maintains a database of entries could benefit from such support. For instance, if a program is maintaining a student database, then accesses to the database cannot be modeled by any predetermined access pattern. The access to records are random and locating the records would be more efficient if the operating system were to provide some form of tree-based index.

10.8 Discuss the merits and demerits of supporting links to files that cross mount points (that is, the file link refers to a file that is stored in a different volume).

Answer: The advantage is that there is greater transparency in the sense that the user does not need to be aware of mount points and create links in all scenarios. The disadvantage however is that the filesystem containing the link might be mounted while the filesystem containing the target file might not be, and therefore one cannot provide transparent access

to the file in such a scenario; the error condition would expose to the user that a link is a dead link and that the link does indeed cross filesystem boundaries.

10.9 Some systems provide file sharing by maintaining a single copy of a file; other systems maintain several copies, one for each of the users sharing the file. Discuss the relative merits of each approach.

Answer: With a single copy, several concurrent updates to a file may result in user obtaining incorrect information, and the file being left in an incorrect state. With multiple copies, there is storage waste and the various copies may not be consistent with respect to each other.

10.10 Discuss the advantages and disadvantages of associating with remote file systems (stored on file servers) a different set of failure semantics from that associated with local file systems.

Answer: The advantage is that the application can deal with the failure condition in a more intelligent manner if it realizes

that it incurred an error while accessing a file stored in a remote filesystem. For instance, a file open operation could simply fail instead of hanging when accessing a remote file on a failed server and the application could deal with the failure in the best possible manner; if the operation were to simply hang, then the entire application hangs which is not desirable. The disadvantage however is the lack of uniformity in failure semantics and the resulting complexity in application code.

10.11 What are the implications of supporting UNIXconsistency semantics for shared access for those files that are stored on remote file systems.

Answer: UNIXconsistency semantics requires updates to a file to be immediately available to other processes. Supporting such a semantics for shared files on remote file systems could result in the following inefficiencies: all updates by a client have to be immediately reported to the fileserver instead of being batched (or even ignored if the updates are to a temporary file), and updates have to be communicated by the fileserver to clients caching the data immediately again resulting in more communication.

# Chapter 11

11.1 Consider a file system that uses a modifed contiguous-allocation scheme with support for extents.A file is a collection of extents, with each extent corresponding to a contiguous set of blocks. A key issue in such systems is the degree of variability in the size of the extents. What are the advantages and disadvantages of the following schemes:

a. All extents are of the same size, and the size is predetermined.

b. Extents can be of any size and are allocated dynamically.

c. Extents can be of a few fixed sizes, and these sizes are

predetermined.

Answer: If all extents are of the same size, and the size is predetermined, then it simplifies the block allocation scheme. A simple bit map or free list for extents would suffice. If the extents can be of any size and are allocated dynamically, then more complex allocation schemes are required. It might be difficult to find an extent of the appropriate size and there might be external fragmentation. One could use the Buddy system allocator discussed in the previous chapters to design an appropriate allocator. When the extents can be of a few fixed sizes, and these sizes are predetermined, one would have to maintain a separate bitmap or free list for each possible size. This scheme is of intermediate complexity and of intermediate flexibility in comparison to the earlier schemes.

11.2 What are the advantages of the variation of linked allocation that uses a FAT to chain together the blocks of a file?

Answer: The advantage is that while accessing a block that is stored at the middle of a file, its location can be determined by

chasing the pointers stored in the FAT as opposed to accessing all of the individual blocks of the file in a sequential manner to find the pointer to the target block. Typically, most of the FAT can be cached in memory and therefore the pointers can be determined with just memory accesses instead of having to access the disk blocks.

11.3 Consider a system where free space is kept in a free-space list.

a. Suppose that the pointer to the free-space list is lost. Can the system reconstruct the free-space list? Explain your answer.

b. Consider a file system similar to the one used by UNIX with indexed allocation. How many disk I/O operations might be required to read the contents of a small local file at /a/b/c? Assume that none of the disk blocks is currently being cached.

c. Suggest a scheme to ensure that the pointer is never lost as a result of memory failure.

Answer:

a. In order to reconstruct the free list, it would be necessary to perform "garbage collection." This would entail searching

the entire directory structure to determine which pages are already allocated to jobs. Those remaining unallocated pages could be relinked as the free-space list.

b. The free-space list pointer could be stored on the disk, perhaps in several places.

c.

11.4 Some file systems allow disk storage to be allocated at different levels of granularity. For instance, a file system could allocate 4 KB of disk space as a single 4-KB block or as eight 512-byte blocks. How could we take advantage of this flexibility to improve performance? What modifications would have to be made to the free-space management scheme in order to support this feature?

Answer: Such a scheme would decrease internal fragmentation. If a file is 5KB, then it could be allocated a 4KB block and two contiguous 512-byte blocks. In addition to maintaining a bitmap of free blocks, one would also have to maintain extra state regarding which of the subblocks are currently being used inside a block. The allocator would then have to examine this extra state to allocate subblocks and

coallesce the subblocks to obtain the larger block when all of the subblocks become free.

11.5 Discuss how performance optimizations for file systems might result in difficulties in maintaining the consistency of the systems in the event of computer crashes.

Answer: The primary difficulty that might arise is due to delayed updates of data and metadata. Updates could be delayed in the hope that the same data might be updated in the future or that the updated data might be temporary and might be deleted in the near future. However, if the system were to crash without having committed the delayed updates, then the consistency of the file system is destroyed.

11.6 Consider a file system on a disk that has both logical and physical block sizes of 512 bytes. Assume that the information about each file is already in memory. For each of the three allocation strategies (contiguous,linked, and indexed), answer these questions:

a. How is the logical-to-physical address mapping accomplished in this system? (For the indexed allocation,

assume that a file is always less than 512 blocks long.)

b. If we are currently at logical block 10 (the last block accessed was block 10) and want to access logical block 4, how many physical blocks must be read from the disk?

Answer: Let Z be the starting file address (block number).

a. Contiguous. Divide the logical address by 512 with X and Y the resulting quotient and remainder respectively.

(1) Add X to Z to obtain the physical block number. Y is the displacement into that block.

(2) 1

b. Linked. Divide the logical physical address by 511 with X and Y the resulting quotient and remainder respectively.

(1) Chase down the linked list (getting X + 1 blocks). Y + 1 is the displacement into the last physical block.

(2) 4

c. Indexed. Divide the logical address by 512 with X and Y the resulting quotient and remainder respectively.

(1) Get the index block into memory. Physical block address is contained in the index block at location X. Y is the displacement into the desired physical block.

(2) 2

11.7 Fragmentation on a storage device could be eliminated by recompaction of the information. Typical disk devices do not have relocation or base registers (such as are used when memory is to be compacted),so how can we relocate files? Give three reasons why recompacting and relocation of files often are avoided.

Answer: Relocation of files on secondary storage involves considerable overhead — data blocks would have to be read into main memory and written back out to their new locations. Furthermore, relocation registers apply only to sequential files, and many disk files are not sequential. For this same reason, many new files will not require contiguous disk space; even sequential files can be allocated noncontiguous blocks if links between logically sequential blocks are maintained by the disk system.

11.8 In what situations would using memory as a RAM disk be more useful than using it as a disk cache?

Answer: In cases where the user (or system) knows exactly

what data is going to be needed. Caches are algorithm-based, while a RAM disk is user-directed.

11.9 Consider the following augmentation of a remote-file-access protocol. Each client maintains a name cache that caches translations from file names to corresponding file handles. What issues should we take into account in implementing the name cache?

Answer: One issue is maintaining consistency of the name cache. If the cache entry becomes inconsistent, then it should be either updated or its inconsistency should be detected when it is used next. If the inconsistency is detected later, then there should be a fallback mechanism for determining the new translation for the name. Also, another related issue is whether a name lookup is performed one element at a time for each subdirectory in the pathname or whether it is performed in a single shot at the server. If it is perfomed one element at a time, then the client might obtain more information regarding the translations for all of the intermediate directories. On the other hand, it increases the network traffic as a single name lookup causes a sequence of partial name lookups.

## 11.10 Explain why logging metadata updates ensures recovery of a file system after a file system crash.

Answer: For a file system to be recoverable after a crash, it must be consistent or must be able to be made consistent. Therefore, we have to prove that logging metadata updates keeps the file system in a consistent or able-to-be-consistent state. For a file system to become inconsistent, the metadata must be written incompletely or in the wrong order to the file system data structures. With metadata logging, the writes are made to a sequential log. The complete transaction is written there before it is moved to the file system structures. If the system crashes during file system data updates, the updates can be completed based on the information in the log. Thus, logging ensures that file system changes are made completely (either before or after a crash).

The order of the changes are guaranteed to be correct because of the sequential writes to the log. If a change was made incompletely to the log, it is discarded, with no changes made to the file system structures.

Therefore, the structures are either consistent or can be

trivially made consistent via metadata logging replay.

11.11 Consider the following backup scheme:

- Day 1. Copy to a backup medium all files from the disk.
- Day 2. Copy to another medium all files changed since day 1.
- Day 3. Copy to another medium all files changed since day 1.

This contrasts to the schedule given in Section 11.7.2 by having all subsequent backups copy all files modified since the first full backup. What are the benefits of this system over the one in Section 11.7.2? What are the drawbacks? Are restore operations made easier or more difficult? Explain your answer.

Answer: Restores are easier because you can go to the last backup tape, rather than the full tape. No intermediate tapes need be read. More tape is used as more files change.

# Chapter 12

12.1 None of the disk-scheduling disciplines, except FCFS, is truly fair (star vation may occur).

a. Explain why this assertion is true.

b. Describe a way to modify algorithms such as SCAN to ensure fairness.

c. Explain why fairness is an important goal in a time-sharing system.

d. Give three or more examples of circumstances in which it is important that the operating system be unfair in serving I/O requests.

Answer:

a. New requests for the track over which the head currently resides can theoretically arrive as quickly as these requests are being serviced.

b. All requests older than some predetermined age could be "forced" to the top of the queue, and an associated bit for

each could be set to indicate that no new request could be moved ahead of these requests. For SSTF, the rest of the queue would have to be reorganized with respect to the last of these "old" requests.

c. To prevent unusually long response times.

d. Paging and swapping should take priority over user requests. It may be desirable for other kernel-initiated I/O, such as the writing of file system metadata, to take precedence over user I/O. If the kernel supports real-time process priorities, the I/O requests of those processes should be favored.

12.2 Suppose that a disk drive has 5000 cylinders, numbered 0 to 4999. The drive is currently serving a request at cylinder 143, and the previous request was at cylinder 125. The queue of pending requests, in FIFO order, is 86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130 Starting from the current head position, what is the total distance (in cylinders) that the disk arm moves to satisfy all the pending requests, for each of the following disk-scheduling algorithms?

a. FCFS b. SSTF c. SCAN d. LOOK e. C-SCAN

Answer:

a. The FCFS schedule is 143, 86, 1470, 913, 1774, 948,1509,1022,1750, 130. The total seek distance is 7081.

b. The SSTF schedule is 143,130,86,913,948,1022,1470,1509,1750,1774. The total seek distance is 1745.

c. The SCAN schedule is 143, 913,948,1022,1470,1509,1750,1774,4999,130,86. The total seek distance is 9769.

d. The LOOK schedule is 143,913,948,1022,1470,1509,1750,1774,130,86. The total seek distance is 3319.

e. The C-SCAN schedule is 143,913,948,1022,1470,1509,1750,1774,4999, 86, 130. The total seek distance is 9813.

f. (Bonus.) The C-LOOK schedule is 143,913,948,1022,1470,1509,1750, 1774, 86, 130. The total seek distance is 3363.

12.3 From elementary physics, we know that when an object is subjected to a constant acceleration a ,the relationship between distance d and time t is given by

$$d = \frac{1}{2}at^2.$$

Suppose that, during a seek, the disk in Exercise 12.2 accelerates the disk arm at a constant rate for the first half of the seek, then decelerates the disk arm at the same rate for the second half of the seek. Assume that the disk can perform a seek to an adjacent cylinder in 1 millisecond and a full-stroke seek over all 5000 cylinders in 18 milliseconds.

a. The distance of a seek is the number of cylinders that the head moves. Explain why the seek time is proportional to the square root of the seek distance.

b. Write an equation for the seek time as a function of the s√eek distance. This equation should be of the form $t = x + y\sqrt{L}$, where t is the time in milliseconds and L is the seek distance in cylinders.

c. Calculate the total seek time for each of the schedules in Exercise 12.2. Determine which schedule is the fastest (has the smallest total seek time).

d. The percentage speedup is the time saved divided by the original time. What is the percentage speedup of the fastest schedule over FCFS?

Answer:

a. Solving $d = \frac{1}{2}at^2$ for t gives $t = \sqrt{(2d/a)}$.

b. Solve the simultaneous equations $t = x + y\sqrt{L}$ that result from (t = 1, L = 1) and (t = 18, L = 4999) to obtain $t = 0.7561 + 0.2439\sqrt{L}$

c. The total seek times are: FCFS 65.20; SSTF 31.52; SCAN 62.02; LOOK 40.29; C-SCAN 62.10; (and C-LOOK 40.42). Thus, SSTF is fastest here.

d. (65.20-31.52)/65.20 = 0.52 The percentage speedup of SSTF over FCFS is 52%, with respect to the seek time. If we include the overhead of rotational latency and data transfer, the percentage speedup will be less.

12.4 Suppose that the disk in Exercise 12.3 rotates at 7200 RPM.

a. What is the average rotational latency of this disk drive?

b. What seek distance can be covered in the time that you found for part a?

Answer:

a. 7200 rpm gives 120 rotations per second. Thus, a full rotation takes 8.33 ms, and the average rotational latency (a half rotation) takes 4.167 ms.

b. Solving $t = 0.7561 + 0.2439\sqrt{L}$ for t = 4.167 gives L = 195.58,so we can seek over 195 tracks (about 4% of the disk) during an average rotational latency.

12.5 Write a Java program for disk scheduling using the SCAN and C-SCAN disk-scheduling algorithms.

Answer: (I have not completed it up to now. Maybe someday it will be released.)

12.6 Compare the performance of C-SCAN and SCAN scheduling, assuming a uniform distribution of requests. Consider the average response time (the time between the arrival of a request and the completion of that request' s service), the variation in response time, and the effective bandwidth. How does performance depend on the relative sizes of seek time and rotational latency?

Answer:

There is no simple analytical argument to answer the first part of this question. It would make a good small simulation experiment for the students. The answer can be found in

Figure 2 of Worthington et al.[1994]. (Worthington et al. studied the LOOK algorithm, but similar results obtain for SCAN. Figure 2 in Worthington et al. shows that C-LOOK has an average response time just a few percent higher than LOOK but that C-LOOK has a significantly lower variance in response time for medium and heavy workloads. The intuitive reason for the difference in variance is that LOOK (and SCAN) tend to favor requests near the middle cylinders, whereas the C-versions do not have this imbalance. The intuitive reason for the slower response time of C-LOOK is the "circular" seek from one end of the disk to the farthest request

at the other end. This seek satisfies no requests. It only causes a small performance degradation because the square-root dependency of seek time on distance implies that a long seek isn't terribly expensive by comparison with moderate length seeks.For the second part of the question, we observe that these algorithms do not schedule to improve rotational latency; therefore, as seek times decrease relative to rotational latency, the performance differences between the algorithms will decrease.

12.7 Requests are not usually uniformly distributed. For

example, a cylinder containing the file system FAT or inodes can be expected to be accessed more frequently than a cylinder that only contains files. Suppose you know that 50 percent of the requests are for a small, fixed number of cylinders.

a. Would any of the scheduling algorithms discussed in this chapter be particularly good for this case? Explain your answer.

b. Propose a disk-scheduling algorithm that gives even better performance by taking advantage of this "hot spot" on the disk.

c. File systems typically find data blocks via an indirection table, such as a FAT in DOS or inodes in UNIX. Describe one or more ways to take advantage of this indirection to improve the disk performance.

Answer:

a. SSTF would take greatest advantage of the situation. FCFS could cause unnecessary head movement if references to the "high-demand" cylinders were interspersed with references to cylinders far away.

b. Here are some ideas. Place the hot data near the middle of

the disk. Modify SSTF to prevent starvation. Add the policy that if the disk becomes idle for more than, say, 50 ms, the operating system generates an anticipatory seek to the hot region, since the next request is more likely to be there.

c. Cache the metadata in primary memory, and locate a file's data and metadata in close physical proximity on the disk. (UNIX accomplishes the latter goal by allocating data and metadata in regions called cylinder groups.)

12.8 Could a RAID Level 1 organization achieve better performance for read requests than a RAID Level 0 organization (with nonredundant striping of data)? If so, how?

Answer: Yes, a RAID Level 1 organization could achieve better performance for read requests. When a read operation is performed, a RAID Level 1 system can decide which of the two copies of the block should be accessed to satisfy the request. This choice could be based on the current location of the disk head and could therefore result in performance optimizations by choosing a disk head which is closer to the target data.

12.9 Consider a RAID Level 5 organization comprising five

disks, with the parity for sets of four blocks on four disks stored on the fifth disk. How many blocks are accessed in order to perform the following?

a. A write of one block of data

b. A write of seven continuous blocks of data

Answer:

a. A write of one block of data requires the following: read of the parity block, read of the old data stored in the target block, computation of the new parity based on the differences between the new and old contents of the target block, and the write of the parity block and the target block.

b. Assume that the seven contiguous blocks begin at a four-block boundary. A write of seven contiguous blocks of data could be performed by writing the seven contiguous blocks,writing the parity block of the first four blocks, reading the eight block, computing the parity for the next set of four blocks and writing the corresponding parity block onto disk.

12.10 Compare the throughput achieved by a RAID Level 5 organization with that achieved by a RAID Level 1 organization for the following:

a. Read operations on single blocks

b. Read operations on multiple contiguous blocks

Answer: 1) The amount of throughput depends on the number of disks in the RAID system. A RAID Level 5 comprising of a parity block for every set of four blocks spread over five disks can support four to five operations simultaneously. A RAID Level 1 comprising of two disks can support two simultaneous operations. Of course, there is greater flexibility in RAID Level 1 as to which copy of a block could be accessed and that could provide performance benefits by taking into account position of disk head. 2) RAID Level 5 organization achieves greater bandwidth for accesses to multiple contiguous blocks since the adjacent blocks could be simultaneously accessed. Such bandwidth improvements are not possible in RAID Level 1.

12.11 Compare the performance of write operations achieved by a RAID Level 5 organization with that achieved by a RAID Level 1 organization.

Answer: RAID Level 1 organization can perform writes by simply issuing the writes to mirrored data concurrently. RAID

Level 5, on the other hand, would require the old contents of the parity block to be read before it is updated based on the new contents of the target block. This results in more overhead for the write operations on a RAID Level 5 system.

12.12 Assume that you have a mixed configuration comprising disks organized as RAID Level 1 and as RAID Level 5 disks. Assume that the system has flexibility in deciding which disk organization to use for storing a particular file. Which files should be stored in the RAID Level 1 disks and which in the RAID Level 5 disks in order to optimize performance?

Answer: Frequently updated data need to be stored on RAID Level 1 disks while data which is more frequently read as opposed to being written should be stored in RAID Level 5 disks.

12.13 Is there any way to implement truly stable storage? Explain your answer.

Answer: Truly stable storage would never lose data. The fundamental technique for stable storage is to maintain

multiple copies of the data,so that if one copy is destroyed, some other copy is still available for use. But for any scheme, we can imagine a large enough disaster that all copies are destroyed.

12.14 The reliability of a hard-disk drive is typically described in terms of a quantity called mean time between failures (MTBF). Although this quantity is called a "time," the MTBF actually is measured in drive-hours per failure.

a. If a disk farm contains 1000 drives, each of which has a 750,000 hour MTBF, which of the following best describes how often a drive failure will occur in that disk farm: once per thousand years, once per century, once per decade, once per year, once per month, once per week, once per day, once per hour, once per minute, or once per second?

b. Mortality statistics indicate that, on the average, a U.S. resident has about 1 chance in 1000 of dying between ages 20 and 21 years. Deduce the MTBF hours for 20 year olds. Convert this figure from hours to years. What does this MTBF tell you about the expected lifetime of a 20 year old?

c. The manufacturer claims a 1-million hour MTBF for a certain model of disk drive. What can you say about the number of

years that one of those drives can be expected to last?

Answer:

a. 750,000 drive-hours per failure divided by 1000 drives gives 750 hours per failure—about 31 days or once per month.

b. The human-hours per failure is 8760 (hours in a year) divided by 0.001 failure, giving a value of 8,760,000 "hours" for the MTBF.8760,000 hours equals 1000 years. This tells us nothing about the expected lifetime of a person of age 20.

c. The MTBF tells nothing about the expected lifetime. Hard disk drives are generally designed to have a lifetime of 5 years. If such a drive truly has a million-hour MTBF, it is very unlikely that the drive will fail during its expected lifetime.

12.15 Discuss the relative advantages and disadvantages of sector sparing and sector slipping.

Answer: Sector sparing can cause an extra track switch and rotational latency,causing an unlucky request to require an extra 8 ms of time. Sector slipping has less impact during future reading, but at sector remapping time it can require the reading and writing of an entire track' s worth of data to slip

the sectors past the bad spot.

12.16 Discuss the reasons why the operating system might require accurate information on how blocks are stored on a disk. How could the operating system improve file system performance with this knowledge? Answer: While allocating blocks for a file, the operating system could allocate blocks that are geometrically close by on the disk if it had more information regarding the physical location of the blocks on the disk.

In particular, it could allocate a block of data and then allocate the second block of data in the same cylinder but on a different surface at a rotationally optimal place so that the access to the next block could be made with minimal cost.

12.17 The operating system generally treats removable disks as shared file systems but assigns a tape drive to only one application at a time.

Give three reasons that could explain this difference in treatment of disks and tapes. Describe additional features that would be required of the operating system to support shared file-system access to a tape jukebox. Would the applications

sharing the tape jukebox need any special properties, or could they use the files as though the files were disk-resident? Explain your answer.

Answer:

a. Disks have fast random-access times, so they give good performance for interleaved access streams. By contrast, tapes have high positioning times. Consequently, if two users attempt to share a tape drive for reading, the drive will spend most of its time switching tapes and positioning to the desired data, and relatively little time performing data transfers. This performance problem is similar to the thrashing of a virtual memory system that has insufficient physical memory.

b. Tape cartridges are removable. The owner of the data may wish to store the cartridge off-site (far away from the computer) to keep a copy of the data safe from a fire at the location of the computer.

c. Tape cartridges are often used to send large volumes of data from a producer of data to the consumer. Such a tape cartridge is reserved for that particular data transfer and cannot be used for general-purpose shared storage space.To support shared file-system access to a tape jukebox, the

operating system would need to perform the usual file-system duties, including

- Manage a file-system name space over the collection of tapes
- Perform space allocation
- Schedule the I/O operations

The applications that access a tape-resident file system would need to be tolerant of lengthy delays. For improved performance, it would be desirable for the applications to be able to disclose a large number of I/O operations so that the tape-scheduling algorithms could generate efficient schedules.

12.18 What would be the effect on cost and performance if tape storage were to achieve the same areal density as disk storage? (Areal density is the number of gigabits per square inch.)

Answer: To achieve the same areal density as a magnetic disk, the areal density of a tape would need to improve by two orders of magnitude. This would cause tape storage to be much cheaper than disk storage. The storage capacity of a

tape would increase to more than 1 terabyte, which could enable a single tape to replace a jukebox of tapes in today's technology, further reducing the cost. The areal density has no direct bearing on the data transfer rate, but the higher capacity per tape might reduce the overhead of tape switching.

12.19 You can use simple estimates to compare the cost and performance of a terabyte storage system made entirely from disks with one that incorporates tertiary storage. Suppose that magnetic disks each hold 10 gigabytes, cost $1000, transfer 5 megabytes per second, and have an average access latency of 15 milliseconds. Suppose that a tape library costs $10 per gigabyte, transfers 10 megabytes per second, and has an average access latency of 20 seconds. Compute the total cost, the maximum total data rate, and the average waiting time for a pure disk system. If you make any assumptions about the workload, describe and justify them. Now, suppose that 5 percent of the data are frequently used, so they must reside on disk, but the other 95 percent are archived in the tape library. Further suppose that 95 percent of the requests are handled by the disk system and the other 5 percent are handled by the

library. What are the total cost, the maximum total data rate, and the

average waiting time for this hierarchical storage system?

Answer: First let' s consider the pure disk system. One terabyte is1024 GB. To be correct, we need 103 disks at 10GB each.But since this question is about approximations, we will simplify the arithmetic by rounding off the numbers. The pure disk system will have 100 drives. The cost of the disk drives would be $100,000, plus about 20% for cables, power supplies, and enclosures, i.e., around $120,000. The aggregate data rate would be 100×5 MB/s, or 500 MB/s. The average waiting time depends on the workload. Suppose that the requests are for transfers of size 8 KB, and suppose that the requests are randomly distributed over the disk drives. If the system is lightly loaded, a typical request will arrive at an idle disk, so the response time will be 15 ms access time plus about 2 ms transfer time. If the system is heavily loaded, the delay will increase, roughly in proportion to the queue length. Now let' s consider the hierarchical storage system. The total disk space required is 5% of 1 TB, which is 50 GB. Consequently, we need 5 disks, so the cost of the disk storage is $5,000 (plus

20%, i.e., $6,000). The cost of the 950 GB tape library is $9500. Thus the total storage cost is $15,500.

The maximum total data rate depends on the number of drives in the tape library. We suppose there is only 1 drive. Then the aggregate data rate is 6×10 MB/s, i.e., 60 MB/s. For a lightly loaded system, 95% of the requests will be satisfied by the disks with a delay of about 17 ms.The other 5% of the requests will be satisfied by the tape library, with a delay of slightly more than 20 seconds. Thus the average delay will be (95×0.017+5×20)/100, or about 1 second. Even with an empty request queue at the tape library, the latency of the tape drive is responsible for almost all of the system’ s response latency, because 1/20th of the workload is sent to a device that has a 20 second latency. If the system is more heavily loaded, the average delay will increase in proportion to the length of the queue of requests waiting for service from the tape drive.

The hierarchical system is much cheaper. For the 95% of the requests that are served by the disks, the performance is as good as a pure-disk system. But the maximum data rate of the hierarchical system is much worse than for the pure-disk system, as is the average response time.

12.20 Imagine that a holographic storage drive has been invented. Suppose that a holographic drive costs $10,000 and has an average access time of 40 milliseconds. Suppose that it uses a $100 cartridge the size of a CD. This cartridge holds 40,000 images, and each image is a square black-and-white picture with resolution 6000 × 6000 pixels (each pixel stores 1 bit). Suppose that the drive can read or write 1 picture in 1 millisecond. Answer the following questions.

a. What would be some good uses for this device?

b. How would this device affect the I/O performance of a computing system?

c. Which other kinds of storage devices, if any, would become obsolete as a result of this device being invented?

Answer: First, calculate performance of the device. 6000×6000bits per millisecond = 4394KB per millisecond = 4291MB/sec(!). Clearly this is orders of magnitude greater than current hard disk technology, as the best production hard drives do less than 40MB/sec. The following answers assume that the device cannot store data in smaller chunks than 4MB.

a. This device would find great demand in the storage of images,audio files, and other digital media.

b. Assuming that interconnection speed to this device would equal its throughput ability (that is, the other components of the system could keep it fed), large-scale digital load and store performance would be greaty enchanced. Manipulation time of the digital object would stay the same of course. The result would be greatly enhanced overall performance.

c. Currently, objects of that size are stored on optical media, tape media, and disk media. Presumably, demand for those would decrease as the holographic storage became available. There are likely to be uses for all of those media even in the presence of holographic storage, so it is unlikely that any would become obsolete. Hard disks would still be used for random access to smaller items (such as user files). Tapes would still be used for off-site, archiving, and disaster recovery uses, and optical disks (CDRW for instance) for easy interchange with other computers, and low cost bulk storage.Depending on the size of the holographic device, and its power requirements, it would also find use in replacing solid state memory for digital cameras, MP3 players, and hand-held computers.

12.21 Suppose that a one-sided 5.25-inch optical-disk

cartridge has an areal density of 1 gigabit per square inch. Suppose that a magnetic tape has an areal density of 20 megabits per square inch, and is 1/2 inch wide and 1800 feet long. Calculate an estimate of the storage capacities of these two kinds of storage cartridges. Suppose that an optical tape exists that has the same physical size as the tape, but the same storage density as the optical disk. What volume of data could the optical tape hold? What would be a marketable price for the optical tape if the magnetic tape cost $25?

Answer: The area of a 5.25 inch disk is about 19.625 square inches. If we suppose that the diameter of the spindle hub is 1.5 inches, the hub occupies an area of about 1.77 square inches, leaving 17.86 square inches for data storage. Therefore, we estimate the storage capacity of the optical disk to be 2.2 gigabytes. The surface area of the tape is 10,800 square inches, so its storage capacity is about 26 gigabytes. If the 10,800 square inches of tape had a storage density of 1 gigabit per square inch, the capacity of the tape would be about 1,350 gigabytes, or 1.3 terabytes. If we charge the same price per gigabyte for the optical tape as for magnetic tape, the optical tape cartridge would cost about 50 times more than the

magnetic tape, i.e., $1,250.

12.22 Discuss how an operating system could maintain a free-space list for a tape-resident file system. Assume that the tape technology is append-only, and that it uses the EOT mark and locate, space, and read position commands as described in Section 12.9.2.1.

Answer: Since this tape technology is append-only, all the free space is at the end of the tape. The location of this free space does not need to be stored at all, because the space command can be used to position to the EOT mark. The amount of available free space after the EOT mark can be represented by a single number. It may be desirable to maintain a second number to represent the amount of space occupied by files that have been logically deleted (but their space has not been reclaimed since the tape is append-only) so that we can decide when it would pay to copy the nondeleted files to a new tape in order to reclaim the old tape for reuse. We can store the free and deleted space numbers on disk for easy access. Another copy of these numbers can be stored at the end of the tape as the last data block. We can

overwrite this last data block when we allocate new storage on the tape.

# Chapter 13

13.1 When multiple interrupts from different devices appear at about the same time, a priority scheme could be used to determine the order in which the interrupts would be serviced. Discuss what issues need to be considered in assigning priorities to different interrupts.

Answer: A number of issues need to be considered in order to determine the priority scheme to be used to determine the order in which the interrupts need to be serviced. First, interrupts raised by devices should be given higher priority than traps generated by the user program; a device interrupt can therefore interrupt code used for handling system calls. Second, interrupts that control devices might be given higher priority than interrupts that simply perform tasks such as copying data served up a device to user/kernel buffers, since such tasks can always be delayed. Third, devices that have

realtime constraints on when its data is handled should be given higher priority than other devices. Also, devices that do not have any form of buffering for its data would have to be assigned higher priority since the data could be available only for a short period of time.

## 13.2 What are the advantages and disadvantages of supporting memory- mapped I/O to device control registers?

Answer: The advantage of supporting memory-mapped I/O to device control registers is that it eliminates the need for special I/O instructions from the instruction set and therefore also does not require the enforcement of protection rules that prevent user programs from executing these I/O instructions. The disadvantage is that the resulting flexibility needs to be handled with care; the memory translation units need to ensure that the memory addresses associated with the device control registers are not accessible by user programs in order to ensure protection.

## 13.3 Consider the following I/O scenarios on a single-user PC.

a. A mouse used with a graphical user interface

b. A tape drive on a multitasking operating system (assume no device preallocation is available)

c. A disk drive containing user files

d. A graphics card with direct bus connection, accessible through memory-mapped I/O For each of these I/O scenarios, would you design the operating system to use buffering, spooling, caching, or a combination? Would you use polled I/O, or interrupt-driven I/O? Give reasons for your choices.

Answer:

a. A mouse used with a graphical user interface Buffering may be needed to record mouse movement during

times when higher-priority operations are taking place. Spooling and caching are inappropriate. Interrupt driven I/O is most appropriate.

b. A tape drive on a multitasking operating system (assume no device preallocation is available) Buffering may be needed to manage throughput difference between the tape drive and the source or destination of the I/O, Caching can be used to hold copies of data that resides on the tape, for faster access. Spooling could be used to stage data to the device when multiple users desire to read from or write to it. Interrupt

driven I/O is likely to allow the best performance.

c. A disk drive containing user files Buffering can be used to hold data while in transit from user space to the disk, and visa versa. Caching can be used to hold disk-resident data for improved performance. Spooling is not necessary because disks are shared-access devices. Interrupt-driven I/O is best for devices such as disks that transfer data at slow rates.

d. A graphics card with direct bus connection, accessible through memory-mapped I/O Buffering may be needed to control multiple access and for performance (double-buffering can be used to hold the next screen image while displaying the current one). Caching and spooling are not necessary due to the fast and shared-access natures of the device. Polling and interrupts are only useful for input and for I/O completion detection, neither of which is needed for a memory-mapped device.

13.4 In most multiprogrammed systems, user programs access memory through virtual addresses, while the operating system uses raw physical addresses to access memory. What are the implications of this design on the initiation of I/O operations by the user program and their execution by the

operating system?

Answer: The user program typically specifies a buffer for data to be transmitted to or from a device. This buffer exists in user space and is specified by a virtual address. The kernel needs to issue the I/O operation and needs to copy data between the user buffer and its own kernel buffer before or after the I/O operation. In order to access the user buffer, the kernel needs to translate the virtual address provided by the user program to the corresponding physical address within the context of the user program' s virtual address space. This translation is typically performed in software and therefore incurs overhead. Also, if the user
buffer is not currently present in physical memory, the corresponding page(s) need to obtained from the swap space. This operation might require careful handling and might delay the data copy operation.

13.5 What are the various kinds of performance overheads associated with
servicing an interrupt?

Answer: When an interrupt occurs the currently executing process is interrupts and its state is stored in the appropriate process control block. The interrupt service routine is then dispatch in order to deal with the interrupt. On completion of handling of the interrupt, the state of the process is restored and the process is resumed. Therefore, the performance overheads include the cost of saving and restoring process state and the cost of flushing the instruction pipeline and restoring the instructions into the pipeline when the process is restarted.

13.6 Describe three circumstances under which blocking I/O should be used. Describe three circumstances under which nonblocking I/O should be used. Why not just implement nonblocking I/O and have processes busy-wait until their device is ready?

Answer: Generally, blocking I/O is appropriate when the process will only be waiting for one specific event. Examples include a disk, tape, or keyboard read by an application program. Non-blocking I/O is useful when I/O may come from more than one source and the order of the I/O arrival is not

predetermined. Examples include network daemons listening to more than one network socket, window managers that accept mouse movement as well as keyboard input, and I/O-management programs, such as a copy command that copies data between I/O devices. In the last case, the program could optimize its performance by buffering the input and output and using non-blocking I/O to keep both devices fully occupied.

Non-blocking I/O is more complicated for programmers, because of the asynchonous rendezvous that is needed when an I/O occurs. Also, busy waiting is less efficient than interrupt-driven I/O so the overall system performance would decrease.

13.7 Typically, at the completion of a device I/O, a single interrupt is raised and appropriately handled by the host processor. In certain settings, however, the code that is to be executed at the completion of the I/O can be broken into two separate pieces, one of which executes immediately after the I/O completes and schedules a second interrupt for the remaining piece of code to be executed at a later time. What is the purpose of using this strategy in the design of interrupt

handlers?

Answer: The purpose of this strategy is to ensure that the most critical aspect of the interrupt handling code is performed first and the less critical portions of the code is delayed for the future. For instance, when a device finishes an I/O operation, the device control operations corresponding to declaring the device as no longer being busy are more important in order to issue future operations. However, the task of copying the data provided by the device to the appropriate user or kernel memory regions can be delayed for a future point when the CPU is idle. In such a scenario, a latter lower priority interrupt handler is used to perform the copy operation.

13.8 Some DMA controllers support direct virtual memory access, where the targets of I/O operations are specified as virtual addresses and a translation from virtual to physical address is performed during the DMA. How does this design complicate the design of the DMA controller? What are the advantages of providing such a functionality?

Answer: Direct virtual memory access allows a device to

perform a transfer from two memory-mapped devices without the intervention of the CPU or the use of main memory as a staging ground; the device simply issue memory operations to the memory-mapped addresses of a target device and the ensuing virtual address translation guarantees that the data is transferred to the appropriate device. This functionality, however, comes at the cost of having to support virtual address translation on addresses accessed by a DMA controller and requires the addition of an address translation unit to the DMA controller. The address translation results in both hardware and software costs and might also result in coherence problems between the data structures maintained by the CPU for address translation and corresponding structures used by the DMA controller. These coherence issues would also need to be dealt with and results in further increase in system complexity.

13.9 UNIX coordinates the activities of the kernel I/O components by manipulating shared in-kernel data structures, whereas Windows NT uses object-oriented message passing between kernel I/O components. Discuss three pros and three cons of each approach.

Answer: Three pros of the UNIX method: Very efficient, low overhead and low amount of data movement Fast implementation — no coordination needed with other kernel components Simple, so less chance of data loss Three cons: No data protection, and more possible side-effects from changes so more difficult to debug Difficult to implement new I/O methods: new data structures needed rather than just new objects Complicated kernel I/O subsystem, full of data structures, access routines, and locking mechanisms

13.10 Write (in pseudocode) an implementation of virtual clocks, including the queuing and management of timer requests for the kernel and applications. Assume that the hardware provides three timer channels.

Answer: Each channel would run the following algorithm:

/** data definitions **/

// a list of interrupts sorted by earliest-time-first order List interruptList

// the list that associates a request with an entry in interruptList

```
List requestList
// an interrupt-based timer
Timer timer
while (true) {
   /** Get the next earliest time in the list **/
   timer.setTime = interruptList.next();
   /** An interrupt will occur at time timer.setTime **/
   /** now wait for the timer interrupt i.e. for the timer to
expire **/
   notify( requestList.next() );
}
```

13.11 Discuss the advantages and disadvantages of guaranteeing reliable transfer of data between modules in the STREAMS abstraction.

Answer: Reliable transfer of data requires modules to check whether space is available on the target module and to block the sending module if space is not available. This check requires extra communication between the modules, but the overhead enables the system to provide a stronger abstraction than one which does not guarantee reliable transfer. The

stronger abstraction typically reduces the complexity of the code in the modules. In the STREAMS abstraction, however, there is unreliability introduced by the driver end, which is allowed to drop messages if the corresponding device cannot handle the incoming data.

Consequently, even if there is reliable transfer of data between the modules, messages could be dropped at the device if the corresponding buffer fills up. This requires retransmission of data and special code for handling such retransmissions, thereby somewhat limiting the advantages that are associated with reliable transfer between the modules.