**CHAPTER 1: Introduction**

**Practice Exercises**

**1.1** What are the three main purposes of an operating system?

**Answer:**

The three main puropses are:

• To provide an environment for a computer user to execute programs on computer hardware in a convenient and efficient manner.

• To allocate the separate resources of the computer as needed to solve the problem given. The allocation process should be as fair and efficient as possible.

• As a control program it serves two major functions:

(1) supervision of the execution of user programs to prevent errors and improper use of the computer, and

(2) management of the operation and control of I/O devices.

Resource Allocation

Supervision

Managing I/O Devices

**1.2** We have stressed the need for an operating system to make efficient use of the computing hardware. When is it appropriate for the operating system to forsake this principle and to “waste” resources? Why is such a system not really wasteful?

**Answer:**

Single-user systems should maximize use of the system for the user. A GUI might “waste” CPU cycles, but it optimizes the user’s interaction with the system.

**1.3** What is the main difficulty that a programmer must overcome in writing an operating system for a real-time environment?

**Answer:**

The main difficulty is keeping the operating system within the fixed time constraints of a real-time system. If the system does not complete a task in a certain time frame, it may cause a breakdown of the entire system it is running. Therefore when writing an operating system for a real-time system, the writer must be sure that his scheduling schemes don’t allow response time to exceed the time constraint.

**1.4** Keeping in mind the various definitions of ***operating system,*** consider whether the operating system should include applications such as web browsers and mail programs. Argue both that it should and that it should not, and support your answers.

**Answer:**

An argument in favor of including popular applications with the operating system is that if the application is embedded within the operating system, it is likely to be better able to take advantage of features in the kernel and therefore have performance advantages over an application that runs outside of the kernel. Arguments against embedding applications within the operating system typically dominate however: (1) the applications are applications - and not part of an operating system, (2) any performance benefits of running within the kernel are offset by security vulnerabilities, (3) it leads to a bloated operating system.

**1.5** How does the distinction between kernel mode and user mode function as a rudimentary form of protection (security)?

**Answer:**

The distinction between kernel mode and user mode provides a rudimentary form of protection in the following manner. Certain instructions could be executed only when the CPU is in kernel mode. Similarly, hardware devices could be accessed only when the program is executing in kernel mode. Control over when interrupts could be enabled or disabled is also possible only when the CPU is in kernel mode. Consequently, the CPU has very limited capability when executing in user mode, thereby enforcing protection of critical resources.

**1.6** Which of the following instructions should be privileged?

a. Set value of timer.

b. Read the clock.

c. Clear memory.

d. Issue a trap instruction.

e. Turn off interrupts.

f. Modify entries in device-status table.

g. Switch from user to kernel mode.

h. Access I/O device.

**Answer:**

The following operations need to be privileged: Set value of timer, clear memory, turn off interrupts, modify entries in device-status table, access I/O device. The rest can be performed in user mode.

a. Set value of timer (Kernel Mode)

b. Read the clock. (User Mode)

c. Clear memory. (Kernel Mode)

d. Issue a trap instruction. (User Mode)

e. Turn off interrupts. (Kernel Mode)

f. Modify entries in device-status table (Kernel Mode)

g. Switch from User to Kernel Mode (User Mode)

h. Access I/O device (Kernel Mode)

**1.7** Some early computers protected the operating system by placing it in a memory partition that could not be modified by either the user job or the operating system itself. Describe two difficulties that you think could arise with such a scheme.

**Answer:**

The data required by the operating system (passwords, access controls, accounting information, and so on) would have to be stored in or passed through unprotected memory and thus be accessible to unauthorized users.

**1.8** Some CPUs provide for more than two modes of operation. What are two possible uses of these multiple modes?

**Answer:**

Although most systems only distinguish between user and kernel modes, some CPUs have supported multiple modes. Multiple modes could be used to provide a finer-grained security policy. For example, rather than distinguishing between just user and kernel mode, you could distinguish between different types of user mode. Perhaps users belonging to the same group could execute each other’s code. The machine would go into a specified mode when one of these users was running code. When the machine was in this mode, a member of the group could run code belonging to anyone else in the group.

Another possibility would be to provide different distinctions within kernel code. For example, a specific mode could allow USB device drivers to run. This would mean that USB devices could be serviced without having to switch to kernel mode, thereby essentially allowing USB device drivers to run in a quasi-user/kernel mode

**1.9** Timers could be used to compute the current time. Provide a short description of how this could be accomplished.

**Answer:**

A program could use the following approach to compute the current time using timer interrupts. The program could set a timer for some time in the future and go to sleep. When it is awakened by the interrupt, it could update its local state, which it is using to keep track of the number of interrupts it has received thus far. It could then repeat this process of continually setting timer interrupts and updating its local state when the interrupts are actually raised.

**1.10** 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 component 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 systems where more than one process may be accessing a datum. A component 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.

**1.11** 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 (it may request recipes) and as a server (it may provide recipes).

| Client-Server Model | Peer-To-Peer Model |
| --- | --- |
| In this model, systems are categorized as client systems. | In this model, all nodes are considered peers. |
| Client systems request for services from server systems. Server systems satisfy request of the client systems. | Each peer can act as client or server depending on the operation it is performing. |
| Resources are controlled and available at the server system. | Resources are distributed among the peers and are easy to be shared among the peers. |
| Client-Server model offers greater security than peer-to-peer model as authentication of users is done at the central server | Peer-To-Peer model offers less security than client-server model as there is no central server that authenticate users. |
| The setup of the client-server model is expensive | The setup of peer-to-peer model is less expensive |
| Client server model is difficult to configure | Peer-To-Peer model is easy to configure. |
| For large office setups and for organizations where there is sensitive date, client server model is suggestable. | At home and for small office setups, peer-to-peer model is suggestable. |

**Chapter 1 Exercises**

**1.12** 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?

*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 consisting of multiple CPUs. A clustered system is less tightly couple than a multiprocessor system. Clustered systems use messages to communicate, while processors in a multiprocessor system could communicate using shared memory. In order for two machines to provide a highly available service, the state on the two machines should be replicated and consistently updated. When one of the machines fails, the other could then takeover functionality.*

**1.13** Consider a computing cluster consisting of two nodes running a database. Describe two ways in which the cluster software can manage access to the data on the disk. Discuss the benefits and disadvantages of each.

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.

**1.14** What is the purpose of interrupts? How does an interrupt differ from a trap? Can traps be generated intentionally by a user program? If so, for what purpose?

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 I/O to obviate the need for device polling. A trap can be used to call operating system routines or to catch arithmetic errors.

**1.15** Explain how the Linux kernel variables HZ and jiffies can be used to determine the number of seconds the system has been running since it was booted.

**1.16** 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?

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 one it receives a command from the CPU.

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

When the device is finished with its operation, it interrupts the CPU to indicate completion.

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.

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.

**1.17** 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.

The operating system would need to remain in monitor mode at all times. There would need to be software interpretation of all use programs. The software interpreter would provide in software what the hardware does not provide.

**1.18** Many SMP systems have different levels of caches; one level is local to each processing core, and another level is shared among all processing cores. Why are caching systems designed this way?

The different levels are based on access speed as well as size. In general, the closer the cache is to the CPU, the faster the access. However, faster caches are typically more costly. Therefore, smaller and faster caches are placed local to each CPU, and shared caches that are larger, yest slower, are shared among several different processors.

**1.19** Rank the following storage systems from slowest to fastest:

a. Hard-disk drives

b. Registers

c. Optical disk

d. Main memory

e. Nonvolatile memory

f. Magnetic tapes

g. Cache

The following**storage systems** from slowest to fastest is

1. magnetic tapes

2. optical disk

3. hard-disk drives

4. nonvolatile screen

5. main memory

6. cache

7. registers

**1.20** Consider an SMP system similar to the one shown in Figure 1.8. Illustrate with an example how data residing in memory could in fact have a different value in each of the local caches.

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

a. 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.

b. Multiprocessor systems

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.

c. Distributed systems

Consistency of cached memory values is not an issue. However, consistency problems may arise when a client caches file data.

**1.22** Describe a mechanism for enforcing memory protection in order to prevent a program from modifying the memory associated with other programs.

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

**1.23** Which network configuration—LAN or WAN—would best suit the following environments?

a. A campus student union: LAN

b. Several campus locations across a statewide university system: WAN

c. A neighborhood: LAN/ WAN

**1.24** Describe some of the challenges of designing operating systems for mobile devices compared with designing operating systems for traditional PCs.

**1.25** What are some advantages of peer-to-peer systems over client–server systems?

Services are distributed across a collection of peers, rather than a single centralized server. P2P provides fault tolerance and redundancy. Peers constantly migrate, so they provide a level of security over a server that always exists at a known location on the internet. P2P systems can potentially provide higher bandwidth because you can collectively use the bandwidth of all the peers rather than the bandwidth of a single server.

**1.26** Describe some distributed applications that would be appropriate for a peer-to-peer system.

**1.27** Identify several advantages and several disadvantages of open-source operating systems. Identify the types of people who would find each aspect to be an advantage or a disadvantage.

**CHAPTER 2: Operating - System Structures**

**Practice Exercises**

**2.1** What is the purpose of system calls?

**Answer:**

System calls allow user-level processes to request services of the operating system.

**2.2** What is the purpose of the command interpreter? Why is it usually separate from the kernel?

**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.

**2.3** What system calls have to be executed by a command interpreter or shell in order to start a new process on a UNIX system?

**Answer:**

In Unix systems, a *fork* system call followed by an *exec* system call need to be performed to start a new process. The *fork* call clones the currently executing process, while the *exec* call overlays a new process based on a different executable over the calling process.

**2.4** What is the purpose of system programs?

**Answer:**

System programs can be thought of as bundles of useful system calls. They provide basic functionality to users so that users do not need to write their own programs to solve common problems.

**2.5** What is the main advantage of the layered approach to system design? What are the disadvantages of the layered approach?

**Answer:**

As in all cases of modular design, designing an operating system in a modular way has several advantages. The system is easier to debug and modify because changes affect only limited sections of the system rather than touching all sections of the operating system. Information is kept only where it is needed and is accessible only within a defined and restricted area, so any bugs affecting that data must be limited to a specific module or layer.

**2.6** List five services provided by an operating system, and explain how each creates convenience for users. In which cases would it be impossible for user-level programs to provide these services? Explain your answer.

**Answer:**

The five services are:

a. **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.

b. **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 access only devices they should have access to and to access them only when they are otherwise unused.

c. **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.

d. **Communications**. Message passing between systems requires messages to 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.

e. **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, whether the number of allocated and unallocated blocks of storage match the total number on the device. There, errors are frequently processindependent (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.

**2.7** Why do some systems store the operating system in firmware, while others store it on disk?

**Answer:**

For certain devices, such as handheld PDAs and cellular telephones, a disk with a file system may be not be available for the device. In this situation, the operating system must be stored in firmware.

**2.8** How could a system be designed to allow a choice of operating systems from which to boot? What would the bootstrap program need to do?

**Answer:**

Consider a system that would like to run both Windows XP and three different distributions of Linux (e.g., RedHat, Debian, and Mandrake). Each operating system will be stored on disk. During system boot-up, a special program (which we will call the **boot manager**) will determine which operating system to boot into. This means that rather initially booting to an operating system, the boot manager will first run during system startup. It is this boot manager that is responsible for determining which system to boot into. Typically boot managers must be stored at certain locations of the hard disk to be recognized during system startup. Boot managers often provide the user with a selection of systems to boot into; boot managers are also typically designed to boot into a default operating system if no choice is selected by the user.

**Chapter 2 Exercises**

**2.9** 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.

ANS: 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 those memory 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 intervention. 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.

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

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

**2.11** Describe how you could obtain a statistical profile of the amount of time a program spends executing different sections of its code. Discuss the importance of obtaining such a statistical profile.

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.

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

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 be written to access devices and files in the same manner, 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 resulting in either a loss of functionality or a loss of performance. Some of this could be overcome by the use of the ioctl operation that provides a general-purpose interface for processes to invoke operations on devices.

**2.13** Would it be possible for the user to develop a new command interpreter using the system-call interface provided by the operating system?

An user should be able to develop a newcommand 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.

**2.14** Describe why Android uses ahead-of-time (AOT) rather than just-in-time (JIT) compilation.

**2.15** What are the two models of interprocess communication? What are the strengths and weaknesses of the two approaches?

• Message passing

- strengths: program structures better separated, dangerous operations firewalled

- weaknesses: message passes more slowly, for symmetrical copy operations are to be made

• Shared memory

- strengths: fast and direct

- weaknesses: unexpected behavior when unauthentic programs wrongly accesses the shared memory  
segments

**2.16** Contrast and compare an application programming interface (API) and an application binary interface (ABI).

**2.17** Why is the separation of mechanism and policy desirable?

Mechanismand 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.

**2.18** 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.

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 subsystemtypically uses the storage system to provide the backing store for pages that do not currently reside in memory. Also, updates to the file system 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 file system.

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

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.

**2.20** What are the advantages of using loadable kernel modules?

* **Dynamic Functionality Extension**. Kernel modules let administrators and developers add or modify features without recompiling or rebooting the kernel, adapting to changing requirements seamlessly.
* **Device Driver Support**. LKMs are vital for supporting various hardware devices. They enable dynamic loading of device drivers, allowing the kernel to recognize and interact with different components like network interfaces, storage devices, and input/output devices.
* **Memory and Resource Optimization**. Loadable kernel modules optimize system resources by loading modules on demand. This reduces memory usage and system footprint, enhancing performance by unloading unused modules.
* **Easy Debugging and Testing**. Kernel modules facilitate debugging and testing of specific features or drivers. They enable developers to debug code in a controlled environment without affecting system stability.
* **Third-Party Software Integration**. Loadable kernel modules enable integration of third-party software, extending the kernel’s capabilities without modifying the core codebase. This supports proprietary functionality and specialized hardware, empowering vendors and developers.

**2.21** How are iOS and Android similar? How are they different?

Similarities Between iOS and Android

Some of the similarities between iOS and Android are as follows −

* The basic functions in iOS and Android are alike. Both the iOS and Android phones have calling, messaging, web browsing, video chat, maps, voice commands etc.
* The user interfaces of iOS and Android have a lot of similarities. Both of these support swiping, tapping, pinch and zoom etc on their phone screens.
* There is a status bar on both the iOS and Android devices and it offers similar information such as battery life, time, app notifications, wifi etc.
* 4G cellular network can be enjoyed on both the iOS and Android devices. This is very important as cellular network is crucial for internet surfing.
* Privacy settings are paramount in both iOS and Android. Users are presented with app permissions as this lessens the risk of data leakage.

Differences Between iOS and Android

Some of the differences between iOS and Android are as follows −

* iOS is a closed system whereas Android is more open. Users have barely any system permissions in iOS but in Android, users can customize their phones easily.
* Android software is available for many manufacturers such as Samsung, LG etc. and this may lead to some quality problems in the cheaper phones. However, iOS is strictly controlled by Apple and there is no quality problem as there are few models.
* The Android applications are obtained from Google Play while iOS applications are available in the Apple app store.
* Integration with other devices is better in Apple iOS as compared to Google Android.
* There are different voice assistants for iOS and Android namely Siri and Google Assistant. Google assistant is much more powerful than Siri.
* The running speed of iOS devices remains consistent with time. In contrast to this, the performance of Android devices may decline over time.

**2.22** Explain why Java programs running on Android systems do not use the standard Java API and virtual machine.

**2.23** The experimental Synthesis operating system has an assembler incorporated in the kernel. To optimize system-call performance, the kernel assembles routines within 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 system-performance optimization.

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).

**CHAPTER 3: Processes**

**Practice Exercises**

**3.1** Using the program shown in Figure 3.30, explain what the output will be at LINE A.

#include <sys/types.h>

#include <stdio.h>

#include <unistd.h>

int value = 5;

int main() {

pid t pid;

pid = fork();

if (pid == 0) { /\* child process \*/

value += 15;

return 0;

}

else if (pid > 0) { /\* parent process \*/

wait(NULL);

printf("PARENT: value = %d",value); /\* LINE A \*/

return 0;

}

}

**Figure 3.30** What output will be at Line A?

**Answer:**

The result is still 5 as the child updates its copy of value. When control returns to the parent, its value remains at 5.

**3.2** Including the initial parent process, how many processes are created by the program shown in Figure 3.31?

#include <stdio.h>

#include <unistd.h>

int main() {

/\* fork a child process \*/

fork();

/\* fork another child process \*/

fork();

/\* and fork another \*/

fork();

return 0;

}

**Figure 3.31** How many processes are created?

**Answer:**

There are 8 processes created.

**3.3** Original versions of Apple’s mobile iOS operating system provided no means of concurrent processing. Discuss three major complications that concurrent processing adds to an operating system.

**Answer:** FILL

**3.4** Some computer systems provide multiple register sets. Describe what happens when a context switch occurs if the new context is already loaded into one of the register sets. What happens if the new context is in memory rather than in a register set and all the register sets are in use?

**Answer:**

The CPU current-register-set pointer is changed to point to the set containing the new context, which takes very little time. If the context is in memory, one of the contexts in a register set must be chosen and be moved to memory, and the new context must be loaded from memory into the set. This process takes a little more time than on systems with one set of registers, depending on how a replacement victim is selected.

**3.5** When a process creates a new process using the fork() operation, which of the following states is shared between the parent process and the child process?

a. Stack

b. Heap

c. Shared memory segments

**Answer:**

Only the shared memory segments are shared between the parent process and the newly forked child process. Copies of the stack and the heap are made for the newly created process.

**3.6** Consider the “exactly once” semantic with respect to the RPC mechanism. Does the algorithm for implementing this semantic execute correctly even if the ACK message sent back to the client is lost due to a network problem? Describe the sequence of messages, and discuss whether “exactly once” is still preserved.

**Answer:**

The “exactly once” semantics ensure that a remore procedure will be executed exactly once and only once. The general algorithm for ensuring this combines an acknowledgment (ACK) scheme combined with timestamps (or some other incremental counter that allows the server to distinguish between duplicate messages).

The general strategy is for the client to send the RPC to the server along with a timestamp. The client will also start a timeout clock. The client will then wait for one of two occurrences: (1) it will receive an ACK from the server indicating that the remote procedure was performed, or (2) it will time out. If the client times out, it assumes the server was unable to perform the remote procedure so the client invokes the RPC a second time, sending a later timestamp. The client may not receive the ACK for one of two reasons: (1) the original RPC was never received by the server, or (2) the RPC was correctly received—and performed—by the server but the ACK was lost. In situation (1), the use of ACKs allows the server ultimately to receive and perform the RPC. In situation (2), the server will receive a duplicate RPC and it will use the timestamp to identify it as a duplicate so as not to perform the RPC a second time. It is important to note that the server must send a second ACK back to the client to inform the client the RPC has been performed.

**3.7** Assume that a distributed system is susceptible to server failure. What mechanisms would be required to guarantee the “exactly once” semantic for execution of RPCs?

**Answer:**

The server should keep track in stable storage (such as a disk log) information regarding what RPC operations were received, whether they were successfully performed, and the results associated with the operations. When a server crash takes place and a RPC message is received, the server can check whether the RPC had been previously performed and therefore guarantee “exactly once” semanctics for the execution of RPCs.

**Chapter 3 Exercises**

**3.8** Describe the actions taken by a kernel to context-switch between processes.

* The OS must save the PC and user stack pointer of the currently executing process, in response to a clock interrupt and transfers control to the kernel clock interrupt handler
* Saving the rest of the registers, as well as other machine state, such as the state of the floating point registers, in the process PCB is done by the clock interrupt handler.
* The scheduler to determine the next process to execute is invoked the OS.
* Then the state of the next process from its PCB is retrieved by OS and restores the registers. The restore operation takes the processor back to the state in which the previous process was previously interrupted, executing in user code with user-mode privileges.

**3.9** Construct a process tree similar to Figure 3.7. To obtain process information for the UNIX or Linux system, use the command ps -ael.

A diagram of a computer network

Description automatically generated

Use the command man ps to get more information about the ps command. The task manager on Windows systems does not provide the

parent process ID, but the ***process monitor*** tool, available from technet.microsoft.com, provides a process-tree tool.

**3.10** Explain the role of the init (or systemd) process on UNIX and Linux systems in regard to process termination.

**3.11** Including the initial parent process, how many processes are created by the program shown in Figure 3.32?

**3.12** Explain the circumstances under which the line of code marked printf("LINE J") in Figure 3.33 will be reached.

**3.13** Using the program in Figure 3.34, identify the values of pid at lines A, B, C, and D. (Assume that the actual pids of the parent and child are 2600 and 2603, respectively.)

**3.14** Give an example of a situation in which ordinary pipes are more suitable than named pipes and an example of a situation in which named pipes are more suitable than ordinary pipes.

**3.15** Consider the RPC mechanism. Describe the undesirable consequences that could arise from not enforcing either the “at most once” or “exactly once” semantic. Describe possible uses for a mechanism that has neither

of these guarantees.

**3.16** Using the program shown in Figure 3.35, explain what the output will be at lines X and Y.

**3.17** What are the benefits and the disadvantages of each of the following? Consider both the system level and the programmer level.

a. Synchronous and asynchronous communication

b. Automatic and explicit buffering

c. Send by copy and send by reference

d. Fixed-sized and variable-sized messages

**CHAPTER 4: Threads & Concurrency**

**Practice Exercises**

**4.1** Provide three programming examples in which multithreading provides better performance than a single-threaded solution.

**Answer:**

a. A Web server that services each request in a separate thread.

b. A parallelized application such as matrix multiplication where different parts of the matrix may be worked on in parallel.

c. An interactive GUI program such as a debugger where a thread is used to monitor user input, another thread represents the running application, and a third thread monitors performance.

**4.2** Using Amdahl’s Law, calculate the speedup gain of an application that has a 60 percent parallel component for (a) two processing cores and (b) four processing cores.

**Answer:**

a. With two processing cores we get a speedup of 1.42 times.

b. With four processing cores, we get a speedup of 1.82 times.

**4.3** Does the multithreaded web server described in Section 4.1 exhibit task or data parallelism?

**Answer:**

Data parallelism. Each thread is performing the same task, but on different data.

**4.4** What are two differences between user-level threads and kernel-level threads? Under what circumstances is one type better than the other?

**Answer:**

a. User-level threads are unknown by the kernel, whereas the kernel is aware of kernel threads.

b. On systems using either many-to-one or many-to-many model mapping, user threads are scheduled by the thread library, and the kernel schedules kernel threads.

c. Kernel threads need not be associated with a process, whereas every user thread belongs to a process. Kernel threads are generally more expensive to maintain than user threads, as they must be represented with a kernel data structure.

**4.5** Describe the actions taken by a kernel to context-switch between kernellevel threads.

**Answer:**

Context switching between kernel threads typically requires saving the value of the CPU registers from the thread being switched out and restoring the CPU registers of the new thread being scheduled.

**4.6** What resources are used when a thread is created? How do they differ from those used when a process is created?

**Answer:**

Because a thread is smaller than a process, thread creation typically uses fewer resources than process creation. Creating a process requires allocating a process control block (PCB), a rather large data structure. The PCB includes a memory map, list of open files, and environment variables. Allocating and managing the memory map is typically the most time-consuming activity. Creating either a user or kernel thread involves allocating a small data structure to hold a register set, stack, and priority.

**4.7** Assume that an operating system maps user-level threads to the kernel using the many-to-many model and that the mapping is done through LWPs. Furthermore, the system allows developers to create real-time threads for use in real-time systems. Is it necessary to bind a real-time thread to an LWP? Explain.

**Answer:**

Yes. Timing is crucial to real-time applications. If a thread is marked as real-time but is not bound to an LWP, the thread may have to wait to be attached to an LWP before running. Consider if a real-time thread is running (is attached to an LWP) and then proceeds to block (i.e. must perform I/O, has been preempted by a higher-priority real-time thread, is waiting for a mutual exclusion lock, etc.) While the real-time thread is blocked, the LWP it was attached to has been assigned to another thread. When the real-time thread has been scheduled to run again, it must first wait to be attached to an LWP. By binding an LWP to a real-time thread you are ensuring the thread will be able to run with minimal delay once it is scheduled.

**Chapter 4 Exercises**

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

ANS:

(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.

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

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 multithreaded solution would perform better even on a single-processor system.

**4.10** 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

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.

**4.11** Can a multithreaded solution using multiple user-level threads achieve better performance on a multiprocessor system than on a single-processor system? Explain.

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.

**4.12** In Chapter 3, we discussed Google’s Chrome browser and its practice of opening each new tab in a separate process. Would the same benefits have been achieved if, instead, Chrome had been designed to open each

new tab in a separate thread? Explain.

* Processes and threads both can typically be defined as sequences of execution, for any program. The main difference lies beneath the memory space they share on the RAM or main memory of the system while getting executed.
  + Each process acquires a unique address space on the memory; whereas threads of a program usually share the same memory space for their execution.
  + Processes act independently while threads always remain dependent on their consequent thread for their execution. Threads remain a chain of instructions where a small breakdown may terminate the entire execution.
  + Google Chrome browser maintains to open each new website as an independent process rather than opening them as threads to ensure that no website breakdown affects others' service.
  + One cannot maintain the efficiency of the browser while opening each new website as a thread, because threads are allocated with shared memory spaces. Hence they may affect each other if one of them crashes unexpectedly.

**4.13** Is it possible to have concurrency but not parallelism? Explain.

**Answer:**

* Both are a form of an operating system. Sometimes, to complete a task, both methods needed to finish the task and sometimes one. The priority to select, which form is better, depends upon the requirement of system and according to that operating system coding created.

Yes, it is possible to have concurrency but not parallelism.

Concurrency:

Concurrency means where two different tasks or threads start working together in an overlapped time period, however, it does not mean they run at same instant.

Example:

Multi-task switch on a single-cored processor

In a Concurrency, minimum two threads are to be executed for processing.

A more generalized form of parallelism includes time-slicing which is a form of virtual parallelism.

Explanation:

Consider a scenario where Process 'A' and 'B' have four different task P1, P2, P3, and P4. So in order to go for execution, first Process 'A' of P1 executes, then Process 'B' of P1 executes, Secondly, Process 'A' of P2 executes, then Process 'B' of P2 executes and goes on until all the threads are finished for their execution.

Parallelism:

Parallelism is where two or more different tasks start their execution at the same time. It means that the two tasks or threads start working simultaneously.

Example:

Multi-cored Processor

Explanation:

Consider a Scenario, where Process 'A' and 'B' and each have four different tasks P1, P2, P3, and P4, so both process go for simultaneous execution and each works independently.

Therefore, concurrency can be occurring number of times which are same as parallelism if the process switching is quick and rapid. So, yes, it is possible to have concurrency but not parallelism.

**4.14** Using Amdahl’s Law, calculate the speedup gain for the following applications:

• 40 percent parallel with (a) eight processing cores and (b) sixteen processing cores

• 67 percent parallel with (a) two processing cores and (b) four processing cores

• 90 percent parallel with (a) four processing cores and (b) eight processing cores

*The question calculates the****speedup gain****for****varying degrees****of parallelism (40 percent, 67 percent, and 90 percent) and varying numbers of processing cores (two, four, eight, sixteen), using Amdahl's Law. The results show that as the number of processing cores and degree of parallelism increase, speedup gain increases.*

***Explanation:***

*Amdahl's Law is used to find the maximum improvement in performance of a system task when only part of the system can be improved. It is defined as Speedup = 1 / ( (1 -P) + P/N), where P is the portion of the task that can be made parallel and N is the number of processors.*

***a) 40 percent (0.4) parallel:***

* ***With eight processing cores:****Speedup = 1 / ((1 - 0.4) + (0.4 / 8)) = 1.538*
* ***With sixteen processing cores:****Speedup = 1 / ((1 - 0.4) + (0.4 / 16)) = 1.905*

***b) 67 percent (0.67) parallel:***

* ***With two processing cores:****Speedup = 1 / ((1 - 0.67) + (0.67 / 2)) = 1.764*
* ***With four processing cores:****Speedup = 1 / ((1 - 0.67) + (0.67 / 4)) = 2.258*

***c) 90 percent (0.9) parallel:***

* ***With four processing cores:****Speedup = 1 / ((1 - 0.9) + (0.9 / 4)) = 2.957*
* ***With eight processing cores:****Speedup = 1 / ((1 - 0.9) + (0.9 / 8)) = 3.902*

**4.15** Determine if the following problems exhibit task or data parallelism:

• Using a separate thread to generate a thumbnail for each photo in a collection

• Transposing a matrix in parallel

• A networked application where one thread reads from the network and another writes to the network

• The fork-join array summation application described in Section 4.5.2

• The Grand Central Dispatch system

1. Data parallelism: using a **separate** **thread** to generate a thumbnail for each photo in a **collection**.
2. Data parallelism: **transposing** a matrix in **parallel**.
3. Task Parallelism: a networked application where one **thread** reads from the network and another writes to the network.
4. Data parallelism: the fork-join array summation application described in Section 4.5.2.
5. Task parallelism: the Grand Central Dispatch system

**4.16** A system with two dual-core processors has four processors available for scheduling. A CPU-intensive application is running on this system. All input is performed at program start-up, when a single file must be opened. Similarly, all output is performed just before the program terminates, when the program results must be written to a single file. Between start-up and termination, the program is entirely CPU-bound. Your task is to improve the performance of this application by multithreading it. The application runs on a system that uses the one-to-one threading model (each user thread maps to a kernel thread).

• How many threads will you create to perform the input and output? Explain.

• How many threads will you create for the CPU-intensive portion of the application? Explain.

* Threads count depends upon the priority and requirements of the application. So only thread is enough for this kind of application and this thread is going to handle both input and output operation.
  + It is a concurrency approach. Here, it only makes sense to create as many threads as there are blocking system calls, as the threads will be spent blocking.
  + It doesn't provides any benefits to create an additional threads.
  + Thus, only a signal thread creation makes sense for input and a single thread for output.
  + Four threads are created to perform the CPU-intensive portion of the application. It is because, there should be as many threads as there are processing cores.
  + It would be the waste of processing resources to use fewer threads.
  + Also any number greater than four would be unable to run.

**4.17** Consider the following code segment:

pid t pid;

pid = fork();

if (pid == 0) *{* /\* child process \*/

fork();

thread create( . . .);

*}*

fork();

a. How many unique processes are created?

b. How many unique threads are created?

**Answer:**

* The statement pid = fork(); before the if statement creates one process. The parent process say p creates this process. Let it be p1.
* The statement fork(); in the if statement creates one process. The parent process p creates this process. Let it be p2.
* After the if statement, parent process p, process p1 and process p2 will execute fork(); creating three new processes.
  + One process is created by parent process p.
  + One process is created by process p1.
  + One process is created by process p2.

Hence, 5 unique processes (p1, p2, p3, p4, p5) will be created. If the parent process is also considered, then 6 unique processes (p, p1, p2, p3, p4, p1, p5) will be created.

* Thread creation is done in if block. Only child process p1 is executed in the if block. Therefore, process p1 will be created one thread.
* In the if block one process p2 is created using fork(). Therefore, process p2 will also create a thread.

Hence, 2 unique threads will be created.

**4.18** As described in Section 4.7.2, 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, other operating systems, such as Windows, treat processes and threads differently. Typically, such systems use a notation in which 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:**

Linux operating systems consider both threads and processes as tasks; it cannot distinguish between them. In contrast, windows operating system threads and processes differently.

This approach has pros and cons while modeling threads and processes inside the kernel.

**Pros:**

* Linux consider this as similar, so codes belong to operating system can be cut down easily.
* Scheduler present in the Linux operating systems do not need special code to test threads coupled with each processes.
* It considers different threads and processes as a single task during the time of scheduling.

**Cons:**

- This ability makes it harder for the Linux operating system to inflict process-wide resource limitations directly.

- Extra steps are needed to recognize the each processes belong to appropriate threads and complexity in performing relevant tasks.

**4.19** The program shown in Figure 4.23 uses the Pthreads API. What would be the output from the program at LINE C and LINE P? *Output at LINE C is 5. Output at LINE P is 0*

#include *<*pthread.h*>*

#include *<*stdio.h*>*

int value = 0;

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

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

*{*

pid t pid;

pthread t tid;

pthread attr t attr;

pid = fork();

if (pid == 0) *{* /\* child process \*/

pthread attr init(&attr);

pthread create(&tid,&attr,runner,NULL);

pthread join(tid,NULL);

printf("CHILD: value = %d",value); /\* LINE C \*/

*}*

else if (pid > 0) *{* /\* parent process \*/

wait(NULL);

printf("PARENT: value = %d",value); /\* LINE P \*/

*}*

*}*

void \*runner(void \*param) *{*

value = 5;

pthread exit(0);

*}*

**Figure 4.22** C program for Exercise 4.19.

**4.20** Consider a multicore system and a multithreaded program written using the many-to-many threading model. Let the number of user-level threads in the program be greater than the number of processing cores 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 processing cores.

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.

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

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 kernel thread blocks inside the kernel, due to a page fault or while invoking system calls, the corresponding processor would remain idle.

c. The number of kernel threads allocated to the program is greater than the number of processing cores but less than the number of user-level threads.

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.

**4.21** Pthreads provides an API for managing thread cancellation. The pthread setcancelstate() function is used to set the cancellation state. Its prototype appears as follows:

pthread setcancelstate(int state, int \*oldstate)

The two possible values for the state are PTHREAD CANCEL ENABLE and PTHREAD CANCEL DISABLE.

Using the code segment shown in Figure 4.24, provide examples of two operations that would be suitable to perform between the calls to disable and enable thread cancellation.

int oldstate;

pthread setcancelstate(PTHREAD CANCEL DISABLE, &oldstate);

/\* What operations would be performed here? \*/

pthread setcancelstate(PTHREAD CANCEL ENABLE, &oldstate);

**Figure 4.23** C program for Exercise 4.21.

**CHAPTER 5: CPU Scheduling**

**Practice Exercises**

**5.1** A CPU-scheduling algorithm determines an order for the execution of its scheduled processes. Given *n* processes to be scheduled on one processor, how many different schedules are possible? Give a formula in terms of *n.*

**Answer:** *n!* (*n* factorial = *n* × *n* – 1 × *n* – 2 × ... × 2 × 1).

**5.2** Explain the difference between preemptive and nonpreemptive scheduling.

**Answer:** Preemptive scheduling allows a process to be interrupted in the midst of its execution, taking the CPU away and allocating it to another process. Nonpreemptive scheduling ensures that a process relinquishes control of the CPU only when it finishes with its current CPU burst.

**5.3** Suppose that the following processes arrive for execution at the times indicated. Each process will run for the amount of time listed. In answering the questions, use nonpreemptive scheduling, and base all decisions

on the information you have at the time the decision must be made.

|  |  |  |
| --- | --- | --- |
| Process | Arrival Time | Burst Time |
| *P*1 | 0.0 | 8 |
| *P*2 | 0.4 | 4 |
| *P*3 | 1.0 | 1 |

a. What is the average turnaround time for these processes with the FCFS scheduling algorithm?

b. What is the average turnaround time for these processes with the SJF scheduling algorithm?

c. The SJF algorithm is supposed to improve performance, but notice that we chose to run process *P*1 at time 0 because we did not know that two shorter processes would arrive soon. Compute what the average turnaround time will be if the CPU is left idle for the first 1 unit and then SJF scheduling is used. Remember that processes *P* 1 and *P*2 are waiting during this idle time, so their waiting time may increase. This algorithm could be known as ***future-knowledge scheduling***.

**Answer:**

a. 10.53

b. 9.53

c. 6.86

Remember that turnaround time is finishing time minus arrival time, so you have to subtract the arrival times to compute the turnaround times. FCFS is 11 if you forget to subtract arrival time

a/ Average turnaround for these processes: ( 8 + (12 - 0.4) + (13 - 1)) / 3 = 10.53

b/ Average turnaround for these processes: ( 8 + (9 - 1) + (13 – 0.4)) / 3 = 9.53

c/ CPU is left idle: Average turnaround for these processes: ((2 - 1) + ( 6 – 0.4 ) + ( 14 - 0)) / 3 = 6.87

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

|  |  |  |
| --- | --- | --- |
| Process | Burst Time | Priority |
| *P* 1 | 2 | 2 |
| *P* 2 | 1 | 1 |
| *P* 3 | 8 | 4 |
| *P* 4 | 4 | 2 |
| *P* 5 | 5 | 3 |

The processes are assumed to have arrived in the order *P*1, *P*2, *P*3, *P*4, *P*5, all at time 0.

a. Draw four Gantt charts that illustrate the execution of these processes using the following scheduling algorithms: FCFS, SJF, nonpreemptive priority (a larger priority number implies a higher priority), and RR (quantum = 2).

a. The four Gantt charts:

A rectangular object with numbers

Description automatically generated

A white rectangular object with black letters

Description automatically generated

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

A white sheet with black text

Description automatically generated

c. What is the waiting time of each process for each of these scheduling algorithms?

A white sheet with black text

Description automatically generated

d. Which of the algorithms results in the minimum average waiting time (over all processes)?

🡪 SJF has the shortest wait time.

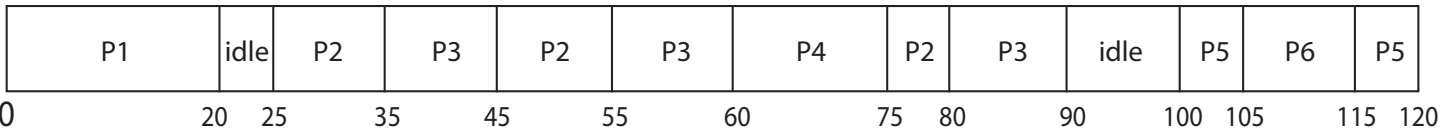
**5.5** The following processes are being scheduled using a preemptive, roundrobin scheduling algorithm.

|  |  |  |  |
| --- | --- | --- | --- |
| Process | Priority | Burst | Arrival |
| *P* 1 | 40 | 20 | 0 |
| *P* 2 | 30 | 25 | 25 |
| *P* 3 | 30 | 25 | 30 |
| *P* 4 | 35 | 15 | 60 |
| *P* 5 | 5 | 10 | 100 |
| *P* 6 | 10 | 10 | 105 |

Each process is assigned a numerical priority, with a higher number indicating a higher relative priority. In addition to the processes listed below, the system also has an **idle task** (which consumes no CPU resources and is identified as *Pidle*). This task has priority 0 and is scheduled whenever the system has no other available processes to run. The length of a time quantum is 10 units. If a process is preempted by a higher-priority process, the preempted process is placed at the end of the queue.

a. Show the scheduling order of the processes using a Gantt chart.

The Gantt chart:



b. What is the turnaround time for each process?

P1: 20-0 - 20, P2: 80-25 = 55, P3: 90 - 30 = 60, P4: 75-60 = 15, P5: 120-100 = 20, P6: 115-105 = 10

c. What is the waiting time for each process?

P1: 0, P2: 40, P3: 35, P4: 0, P5: 10, P6: 0

d. What is the CPU utilization rate? 105/120 = 87.5 percent

**5.6** What advantage is there in having different time-quantum sizes at different levels of a multilevel queueing system?

**Answer:**

Processes that need more frequent servicing—for instance, interactive processes such as editors—can be in a queue with a small time quantum. Processes with no need for frequent servicing can be in a queue with a larger quantum, requiring fewer context switches to complete the processing and thus making more efcient use of the computer.

**5.7** Many CPU-scheduling algorithms are parameterized. For example, the RR algorithm requires a parameter to indicate the time slice. Multilevel feedback queues require parameters to define the number of queues, the scheduling algorithms for each queue, the criteria used to move processes between queues, and so on.

These algorithms are thus really sets of algorithms (for example, the set of RR algorithms for all time slices, and so on). One set of algorithms may include another (for example, the FCFS algorithm is the RR algorithm with an infinite time quantum). What (if any) relation holds between the following pairs of algorithm sets?

a. Priority and SJF

b. Multilevel feedback queues and FCFS

c. Priority and FCFS

d. RR and SJF

**Answer:**

a. The shortest job has the highest priority.

b. The lowest level of MLFQ is FCFS.

c. FCFS gives the highest priority to the job that has been in existence the longest.

d. None.

**5.8** Suppose that a CPU scheduling algorithm favors those processes that have used the least processor time in the recent past. Why will this algorithm favor I/O-bound programs and yet not permanently starve CPU-bound programs?

**Answer:**

It will favor the I/O-bound programs because of the relatively short CPU bursts requested by them; however, the CPU-bound programs will not starve, because the I/O-bound programs will relinquish the CPU relatively often to do their I/O.

**5.9** Distinguish between PCS and SCS scheduling.

**Answer:**

PCS scheduling is local to the process. It is how the thread library schedules threads onto available LWPs. SCS scheduling is used when the operating system schedules kernel threads. On systems using either the many-to-one or the many-to-many model, the two scheduling models are fundamentally different. On systems using the one-to-one model, PCS and SCS are the same.

**5.10** 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 *P*1 is 40, for process *P*2 is 18, and for process *P*3 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 will be 80, 69, and 65, respectively. The scheduler lowers the relative priority of CPU-bound processes.

**Chapter 5 Exercises**

**5.11** Of these two types of programs:

a. I/O-bound

b. CPU-bound

which is more likely to have voluntary context switches, and which is more likely to have nonvoluntary context switches? Explain your answer.

The **correct context switches** for the **two types of programs** are as follows:

* For I/O-bound we require voluntary context switches.
* For CPU-bound we require non-voluntary context switches.

According to the **given question**, we are asked to show the **correct context switches**for the two types of programs,**I/O bound and a CPU bound program.**

As a result of this, we can see that a **voluntary context switch**has to do with when a **process in the processing unit***does not have control*because it **lacks the proper resource.**

On the other hand, a **non voluntary context switch**has to do with there is lack of control as a result of **process allocation** or time slice and this occurs in the **CPU bound program,**

**5.12** 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

A white text on a black background

Description automatically generated

**5.13** One technique for implementing **lottery scheduling** works by assigning processes lottery tickets, which are used for allocating CPU time. Whenever a scheduling decision has to be made, a lottery ticket is chosen at random, and the process holding that ticket gets the CPU. The BTV operating system implements lottery scheduling by holding a lottery 50 times each second, with each lottery winner getting 20 milliseconds of CPU

time (20 milliseconds × 50 = 1 second). Describe how the BTV scheduler can ensure that higher-priority threads receive more attention from the CPU than lower-priority threads.

**Answer:**

By assigning more lottery tickets to higher-priority processes.

**5.14** Most scheduling algorithms maintain a **run queue**, which lists processes eligible to run on a processor. On multicore systems, there are two general options: (1) each processing core has its own run queue, or

(2) a single run queue is shared by all processing cores. What are the advantages and disadvantages of each of these approaches?

**Answer:**

The primary advantage of each processing core having its own run queue is that there is no contention over a single run queue when the scheduler is running concurrently on 2 or more processors. When a scheduling decision must be made for a processing core, the scheduler only need to look no further than its private run queue. A disadvantage of a single run queue is that it must be protected with locks to prevent a race condition and a processing core may be available to run a thread, yet it must first acquire the lock to retrieve the thread from the single queue. However, load balancing would likely not be an issue with a single run queue, whereas when each processing core has its own run queue, there must be some sort of load balancing between the different run queues.

**5.15** 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. α = 0 and τ0 = 100 milliseconds

b. α = 0*.*99 and τ0 = 10 milliseconds

A close-up of a text

Description automatically generated

**5.16** A variation of the round-robin scheduler is the **regressive round-robin** scheduler. This scheduler assigns each process a time quantum and a priority. The initial value of a time quantum is 50 milliseconds. However,

every time a process has been allocated the CPU and uses its entire time quantum (does not block for I/O), 10 milliseconds is added to its time quantum, and its priority level is boosted. (The time quantum for a process can be increased to a maximum of 100 milliseconds.) When a process blocks before using its entire time quantum, its time quantum is reduced by 5 milliseconds, but its priority remains the same. What type of process (CPU-bound or I/O-bound) does the regressive round-robin scheduler favor? Explain.

A black text on a white background

Description automatically generated

**5.17** Consider the following set of processes, with the length of the CPU burst given in milliseconds:

|  |  |  |
| --- | --- | --- |
| Process | Burst Time | Priority |
| *P* 1 | 5 | 4 |
| *P* 2 | 3 | 1 |
| *P* 3 | 1 | 2 |
| *P* 4 | 7 | 2 |
| *P* 5 | 4 | 3 |

The processes are assumed to have arrived in the order *P*1, *P*2, *P*3, *P*4, *P*5, all at time 0.

a. Draw four Gantt charts that illustrate the execution of these processes using the following scheduling algorithms: FCFS, SJF, nonpreemptive priority (a larger priority number implies a higher

priority), and RR (quantum = 2).

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 these scheduling algorithms?

d. Which of the algorithms results in the minimum average waiting time (over all processes)?

**5.18** The following processes are being scheduled using a preemptive, priority-based, round-robin scheduling algorithm.

|  |  |  |  |
| --- | --- | --- | --- |
| Process | Priority | Burst | Arrival |
| *P* 1 | 8 | 15 | 0 |
| *P* 2 | 3 | 20 | 0 |
| *P* 3 | 4 | 20 | 20 |
| *P* 4 | 4 | 20 | 25 |
| *P* 5 | 5 | 5 | 45 |
| *P* 6 | 5 | 15 | 55 |

Each process is assigned a numerical priority, with a higher number indicating a higher relative priority. The scheduler will execute the highestpriority process. For processes with the same priority, a round-robin scheduler will be used with a time quantum of 10 units. If a process is preempted by a higher-priority process, the preempted process is placed at the end of the queue.

a. Show the scheduling order of the processes using a Gantt chart.

b. What is the turnaround time for each process?

c. What is the waiting time for each process?

***Step 1: Calculate Completion Time for each process***

* P1: 15 units
* P2: 20 units
* P3: 40 units
* P4: 60 units
* P5: 65 units
* P6: 80 units

***Step 2: Determine the Turnaround Time for each process***

* P1: 15 units
* P2: 20 units
* P3: 20 units
* P4: 40 units
* P5: 20 units
* P6: 25 units

***Step 3: Calculate the Waiting Time for each process***

* P1: 0 units
* P2: 0 units
* P3: 20 units
* P4: 40 units
* P5: 20 units
* P6: 25 units

***Step 4: Create the Gantt Chart***

* P1 (0-15), P2 (15-35), P3 (35-55), P4 (55-75), P5 (75-80), P6 (80-95)

**5.19** The nice command is used to set the nice value of a process on Linux, as well as on other UNIX systems. Explain why some systems may allow any user to assign a process a nice value *>*= 0 yet allow only the root (or administrator) user to assign nice values *<* 0.

**Answer:**

Nice values < 0 are assigned a higher relative priority and such systems may not allow non-root processes to

assign themselves higher priorities.

**5.20** 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.

**5.21** Consider a variant of the RR scheduling algorithm in which 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 two major advantages and two disadvantages of this scheme?

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

A close up of text

Description automatically generated

**5.22** 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. Describe the CPU utilization for a round-robin scheduler when:

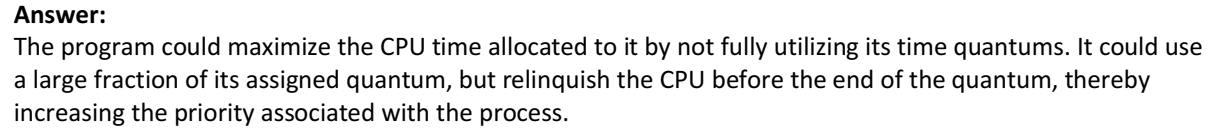
a. The time quantum is 1 millisecond

b. The time quantum is 10 milliseconds

A close-up of a text

Description automatically generated

**5.23** 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?



**5.24** 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 α. When it is running, its priority changes at a rate β. All processes are given a priority of 0 when they enter the ready queue. The parameters α and β can be set to give many different scheduling algorithms.

a. What is the algorithm that results from β *>* α *>* 0?

b. What is the algorithm that results from α *<* β *<* 0?

**Answer:**

a. FCFS

b. LIFO

**5.25** Explain the how the following scheduling algorithms discriminate either in favor of or against short processes:

a. FCFS

b. RR

c. Multilevel feedback queues

A close up of black text

Description automatically generated

**5.26** Describe why a shared ready queue might suffer from performance problems in an SMP environment.

**5.27** Consider a load-balancing algorithm that ensures that each queue has approximately the same number of threads, independent of priority. How effectively would a priority-based scheduling algorithm handle this situation if one run queue had all high-priority threads and a second queue had all low-priority threads?

**5.28** Assume that an SMP system has private, per-processor run queues. When a new process is created, it can be placed in either the same queue as the parent process or a separate queue.

a. What are the benefits of placing the new process in the same queue as its parent?

b. What are the benefits of placing the new process in a different queue?

**5.29** Assume that a thread has blocked for network I/O and is eligible to run again. Describe why a NUMA-aware scheduling algorithm should reschedule the thread on the same CPU on which it previously ran.

**5.30** Using the Windows scheduling algorithm, determine the numeric priority of each of the following threads.

a. A thread in the REALTIME PRIORITY CLASS with a relative priority of NORMAL **(8)**

b. A thread in the ABOVE NORMAL PRIORITY CLASS with a relative priority of HIGHEST **(26)**

c. A thread in the BELOW NORMAL PRIORITY CLASS with a relative priority of ABOVE NORMAL **(14)**

**5.31** Assuming that no threads belong to the REALTIME PRIORITY CLASS and that none may be assigned a TIME CRITICAL priority, what combination of priority class and priority corresponds to the highest possible relative priority in Windows scheduling?

**Answer:**

HIGH priority class and HIGHEST priority within that class. (numeric priority of 15)

**5.32** 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 15? With priority 40?

b. Assume that a thread with priority 50 has used its entire time quantum without blocking. What new priority will the scheduler assign this thread?

c. Assume that a thread with priority 20 blocks for I/O before its time quantum has expired. What new priority will the scheduler assign this thread?

**5.33** Assume that two tasks, *A* and *B*, are running on a Linux system. The nice values of *A* and *B* are -5 and +5, respectively. Using the CFS scheduler as a guide, describe how the respective values of vruntime vary between the two processes given each of the following scenarios:

• Both *A* and *B* are CPU-bound.

• *A* is I/O-bound, and *B* is CPU-bound.

• *A* is CPU-bound, and *B* is I/O-bound.

A close up of a text

Description automatically generated

**5.34** Provide a specific circumstance that illustrates where rate-monotonic scheduling is inferior to earliest-deadline-first scheduling in meeting real-time process deadlines?

**5.35** Consider two processes, *P*1 and *P*2, where *p*1 = 50, *t*1 = 25, *p*2 = 75, and *t*2 = 30.

a. Can these two processes be scheduled using rate-monotonic scheduling? Illustrate your answer using a Gantt chart such as the ones in Figure 5.21–Figure 5.24.

b. Illustrate the scheduling of these two processes using earliestdeadline-first (EDF) scheduling.

A black text on a white background

Description automatically generated

**5.36** Explain why interrupt and dispatch latency times must be bounded in a hard real-time system.

A close-up of a text

Description automatically generated

**5.37** Describe the advantages of using heterogeneous multiprocessing in a mobile system.

**CHAPTER 6: Synchronization Tools**

**Practice Exercises**

**6.1** In Section 6.4, we mentioned that disabling interrupts frequently can affect the system’s clock. Explain why this can occur and how such effects can be minimized.

**Answer:**

The system clock is updated at every clock interrupt. If interrupts were disabled—particularly for a long period of time—the system clock could easily lose the correct time. The system clock is also used for scheduling purposes. For example, the time quantum for a process is expressed as a number of clock ticks. At every clock interrupt, the scheduler determines if the time quantum for the currently running process has expired. If clock interrupts were disabled, the scheduler could not accurately assign time quanta. This effect can be minimized by disabling clock interrupts for only very short periods.

**6.2** 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 satised in a tight loop without relinquishing the processor. One strategy to avoid busy waiting temporarily puts the waiting process to sleep and awakens it when the appropriate program state is reached, but this solution incurs the overhead associated with putting the process to sleep and later waking it up.

**6.3** 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 can 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 rst process to make progress. In a multiprocessor system, other processes execute on other processors and therefore can modify the program state in order to release the rst process from the spinlock.

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

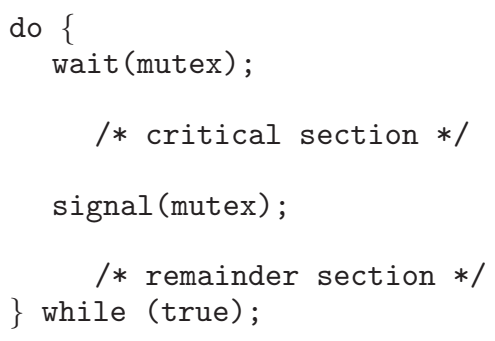
**Answer:**

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 and the operations are not performed atomically, then both operations might decrement the semaphore value, thereby violating mutual exclusion.

**6.5** Illustrate how a binary semaphore can be used to implement mutual exclusion among *n* processes.

**Answer:**

The *n* processes share a semaphore, mutex, initialized to 1. Each process *P i* is organized as follows:



**6.6** Race conditions are possible in many computer systems. Consider a banking system that maintains an account balance with two functions:

deposit(amount) and withdraw(amount).

These two functions are passed the amount that is to be deposited or withdrawn from the bank account balance. Assume that a husband and wife share a bank account. Concurrently, the husband calls the withdraw() function, and the wife calls deposit(). Describe how a race condition is possible and what might be done to prevent the race condition from occurring.

**Answer:**

Assume that the balance in the account is $250.00 and that the husband calls withdraw($50) and the wife calls deposit($100). Obviously, the correct value should be $300.00. Since these two transactions will be serialized, the local value of the balance for the husband becomes $200.00, but before he can commit the transaction, the deposit(100) operation takes place and updates the shared value of the balance to $300.00. We then switch back to the husband, and the value of the shared balance is set to $200.00—obviously an incorrect value.

**Chapter 6 Exercises**

**6.7** The pseudocode of Figure 6.15 illustrates the basic push() and pop() operations of an array-based stack. Assuming that this algorithm could be used in a concurrent environment, answer the following questions:

a. What data have a race condition?

b. How could the race condition be fixed?

a. In a **concurrent environment**, the "top" variable and the "stack" array have a race condition. This is because multiple threads could be accessing and modifying these variables simultaneously, leading to inconsistencies and errors in the **stack operation.**

b. To fix the race condition, we can use synchronization techniques such as locks or semaphores to ensure that only one thread at a time can access and modify the "top" variable and the "stack" array. For example, we could use a **mutex lock** to protect the critical section of code where the "top" variable is updated and the "stack" array is modified. This would ensure that only one thread can access and modify these variables at a time, preventing race conditions and ensuring the correct operation of the stack. Additionally, we could use other synchronization techniques such as condition variables or atomic operations to further protect the stack operation in a concurrent environment.

**6.8** Race conditions are possible in many computer systems. Consider an online auction system where the current highest bid for each item must be maintained. A person who wishes to bid on an item calls the bid(amount) function, which compares the amount being bid to the current highest bid. If the amount exceeds the current highest bid, the highest bid is set to the new amount. This is illustrated below:

A computer code with black text

Description automatically generated

**Figure 6.16** Array-based stack for Exercise 6.12.

A maths ruler with numbers and a line

Description automatically generated with medium confidence

**Figure 6.17** Summing an array as a series of partial sums for Exercise 6.14.

Describe how a race condition is possible in this situation and what might be done to prevent the race condition from occurring.

**6.9** The following program example can be used to sum the array values of size *N* elements in parallel on a system containing *N* computing cores (there is a separate processor for each array element):

for j = 1 to log\_2(N) *{*

for k = 1 to N *{*

if ((k + 1) % pow(2,j) == 0) *{*

values[k] += values[k - pow(2,(j-1))]

*}*

*}*

*}*

This has the effect of summing the elements in the array as a series of partial sums, as shown in Figure 6.16. After the code has executed, the sum of all elements in the array is stored in the last array location. Are there any race conditions in the above code example? If so, identify where they occur and illustrate with an example. If not, demonstrate why this algorithm is free from race conditions.

**6.10** The compare and swap() instruction can be used to design lock-free data structures such as stacks, queues, and lists. The program example shown in Figure 6.17 presents a possible solution to a lock-free stack using CAS instructions, where the stack is represented as a linked list of Node elements with top representing the top of the stack. Is this implementation free from race conditions?

A white paper with black text

Description automatically generated

**Figure 6.18** Lock-free stack for Exercise 6.15.

**6.11** One approach for using compare and swap() for implementing a spinlock is as follows:

void lock spinlock(int \*lock) *{*

while (compare and swap(lock, 0, 1) != 0)

; /\* spin \*/

*}*

A suggested alternative approach is to use the “compare and compareand-swap” idiom, which checks the status of the lock before invoking the compare and swap() operation. (The rationale behind this approach is to invoke compare and swap()only if the lock is currently available.) This strategy is shown below:

void lock spinlock(int \*lock)

*{*

while (true) *{*

if (\*lock == 0) *{*

/\* lock appears to be available \*/

if (!compare and swap(lock, 0, 1))

break;

*}*

*}*

*}*

Does this “compare and compare-and-swap” idiom work appropriately for implementing spinlocks? If so, explain. If not, illustrate how the integrity of the lock is compromised.

**6.12** Some semaphore implementations provide a function getValue() that returns the current value of a semaphore. This function may, for instance, be invoked prior to calling wait() so that a process will only call wait() if the value of the semaphore is *>* 0, thereby preventing blocking while waiting for the semaphore. For example:

if (getValue(&sem) > 0)

wait(&sem);

Many developers argue against such a function and discourage its use. Describe a potential problem that could occur when using the function getValue() in this scenario.

**6.13** The first known correct software solution to the critical-section problem for two processes was developed by Dekker. The two processes, *P*0 and *P* 1, share the following variables:

boolean flag[2]; /\* initially false \*/

int turn;

The structure of process *Pi* (i == 0 or 1) is shown in Figure 6.18. The other process is *Pj* (j == 1 or 0). Prove that the algorithm satisfies all three requirements for the critical-section problem.

**6.14** 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;

while (true) *{*

flag[i] = true;

while (flag[j]) *{*

if (turn == j) *{*

flag[i] = false;

while (turn == j)

; /\* do nothing \*/

flag[i] = true;

*}*

*}*

/\* critical section \*/

turn = j;

flag[i] = false;

/\* remainder section \*/

*}*

**Figure 6.19** The structure of process *Pi* in Dekker’s algorithm.

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 6.19. Prove that the algorithm satisfies all three requirements for the critical-section problem.

**6.15** 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.

**6.16** Consider how to implement a mutex lock using the compare and swap() instruction. Assume that the following structure defining the mutex lock is available:

typedef struct *{*

int available;

*}* lock;

The value (available == 0) indicates that the lock is available, and a value of 1 indicates that the lock is unavailable. Using this struct, illustrate how the following functions can be implemented using the compare and swap() instruction:

• void acquire(lock \*mutex)

• void release(lock \*mutex)

Be sure to include any initialization that may be necessary.

A white paper with black text

Description automatically generated

**Figure 6.20** The structure of process *Pi* in Eisenberg and McGuire’s algorithm.

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

**6.18** The implementation of mutex locks provided in Section 6.5 suffers from busy waiting. Describe what changes would be necessary so that a process waiting to acquire a mutex lock would be blocked and placed

into a waiting queue until the lock became available.

**6.19** Assume that a system has multiple processing cores. For each of the following scenarios, describe which is a better locking mechanism—a spinlock or a mutex lock where waiting processes sleep while waiting for the lock to become available:

• The lock is to be held for a short duration.

• The lock is to be held for a long duration.

• A thread may be put to sleep while holding the lock.

**6.20** Assume that a context switch takes *T* time. Suggest an upper bound (in terms of *T*) for holding a spinlock. If the spinlock is held for any longer, a mutex lock (where waiting threads are put to sleep) is a better alternative.

**6.21** A multithreaded web server wishes to keep track of the number of requests it services (known as ***hits***). Consider the two following strategies to prevent a race condition on the variable hits. The first strategy is to use a basic mutex lock when updating hits:

int hits;

mutex lock hit lock;

hit lock.acquire();

hits++;

hit lock.release();

A second strategy is to use an atomic integer:

atomic t hits;

atomic inc(&hits);

Explain which of these two strategies is more efficient.

**6.22** Consider the code example for allocating and releasing processes shown in Figure 6.20.

a. Identify the race condition(s).

b. Assume you have a mutex lock named mutex with the operations acquire() and release(). Indicate where the locking needs to be placed to prevent the race condition(s).

c. Could we replace the integer variable int number of processes = 0 with the atomic integer atomic t number of processes = 0 to prevent the race condition(s)?

**6.23** 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

A computer code with black text

Description automatically generated

**Figure 6.21** Allocating and releasing processes for Exercise 6.27.

released. Illustrate how semaphores can be used by a server to limit the number of concurrent connections.

**6.24** In Section 6.7, we use the following illustration as an incorrect use of semaphores to solve the critical-section problem:

wait(mutex);

...

critical section

...

wait(mutex);

Explain why this is an example of a liveness failure.

**6.25** Demonstrate that monitors and semaphores are equivalent to the degree that they can be used to implement solutions to the same types of synchronization problems.

**6.26** Describe how the signal() operation associated with monitors differs from the corresponding operation defined for semaphores.

**6.27** Suppose the signal() statement can appear only as the last statement in a monitor function. Suggest how the implementation described in Section 6.7 can be simplified in this situation.

**6.28** Consider a system consisting of processes *P*1, *P*2, ..., *Pn*, each of which has a unique priority number. Write a monitor that allocates three identical printers to these processes, using the priority numbers for deciding the order of allocation.

**6.29** 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.

**6.30** 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 these two different ways in which signaling can be performed?

**6.31** Design an algorithm for 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 function tick() in your monitor at regular intervals.

**6.32** Discuss ways in which the priority inversion problem could be addressed in a real-time system. Also discuss whether the solutions could be implemented within the context of a proportional share scheduler.

**CHAPTER 8: Deadlocks**

**Practice Exercises**

**8.1** List three examples of deadlocks that are not related to a computersystem environment.

**Answer:**

• Two cars crossing a single-lane bridge from opposite directions.

• A person going down a ladder while another person is climbing up the ladder.

• Two trains traveling toward each other on the same track.

**8.2** Suppose that a system is in an unsafe state. Show that it is possible for the threads to complete their execution without entering a deadlocked state.

**Answer:**

An unsafe state may not necessarily lead to deadlock, it just means that we cannot guarantee that deadlock will not occur. Thus, it is possible that a system in an unsafe state may still allow all processes to complete without deadlock occurring. Consider the situation where a system has twelve resources allocated among processes *P*0, *P*1, and *P*2. The resources are allocated according to the following policy:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Max | Current | Need |
| *P* 0 | 10 | 5 | 5 |
| *P* 1 | 4 | 2 | 2 |
| *P* 2 | 9 | 3 | 6 |

Currently, there are two resources available. This system is in an unsafe state. Process *P*1 could complete, thereby freeing a total of four resources, but we cannot guarantee that processes *P*0 and *P*2 can complete. However, it is possible that a process may release resources before requesting any further resources. For example, process *P*2 could release a resource, thereby increasing the total number of resources to ve. This allows pro cess *P*0 to complete, which would free a total of nine resources, thereby allowing process *P*2 to complete as well.

**8.3** Consider the following snapshot of a system:

|  |  |  |  |
| --- | --- | --- | --- |
|  | ***Allocation*** | ***Max*** | ***Available*** |
|  | *A B C D* | *A B C D* | *A B C D* |
| *T*0 | 0 0 1 2 | 0 0 1 2 | 1 5 2 0 |
| *T* 1 | 1 0 0 0 | 1 7 5 0 |  |
| *T*2 | 1 3 5 4 | 2 3 5 6 |  |
| *T*3 | 0 6 3 2 | 0 6 5 2 |  |
| *T*4 | 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 thread *T*1 arrives for (0,4,2,0), can the request be granted immediately?

**Answer:**

a. The values of ***Need*** for processes *P*0 through *P*4, respectively, are (0, 0, 0, 0), (0, 7, 5, 0), (1, 0, 0, 2), (0, 0, 2, 0), and (0, 6, 4, 2).

b. The system is in a safe state. With ***Available*** equal to (1, 5, 2, 0), either process *P*0 or *P*3 could run. Once process *P*3 runs, it releases its resources, which allows all other existing processes to run.

c. The request can be granted immediately. The value of ***Available*** is then (1, 1, 0, 0). One ordering of processes that can nish is *P*0, *P*2, *P*3, *P*1, and *P*4.

**8.4** A possible method 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, and the like.) We can prevent 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 8.5.4.

**Answer:**

This is probably not a good solution because it yields too large a scope. It is better to dene a locking policy with as narrow a scope as possible. The circular wait approach is a reasonable approach to avoiding deadlock, and does not increase the scope of holding a lock.

**8.5** Prove that the safety algorithm presented in Section 8.6.3 requires an order of *m* × *n*2 operations.

**Answer:**

The gure below provides Java code that implements the safety algorithm of the banker’s algorithm (the complete implementation of the banker’s algorithm is available with the source-code download for this text).

A white screen shot of a computer code

Description automatically generated

As can be seen, the nested outer loops—both of which loop through *n* times—provide the *n*2 performance. Within these outer loops are two sequential inner loops that loop *m* times. The Big O of this algorithm is therefore *O*(*m* × *n*2).

**8.6** Consider a computer system that runs 5,000 jobs per month and has no deadlock-prevention or deadlock-avoidance scheme. Deadlocks occur about twice per month, and the operator must terminate and rerun about ten jobs per deadlock. Each job is worth about two dollars (in CPU time), and the jobs terminated tend to be about half done when they are aborted.

A systems programmer has estimated that a deadlock-avoidance algorithm (like the banker’s algorithm) could be installed in the system with an increase of about 10 percent in the average execution time per job. Since the machine currently has 30 percent idle time, all 5,000 jobs per month could still be run, although turnaround time would increase by about 20 percent on average.

a. What are the arguments for installing the deadlock-avoidance algorithm?

b. What are the arguments against installing the deadlock-avoidance algorithm?

**Answer:**

An argument for installing deadlock avoidance in the system is that we could ensure that deadlock would never occur. In addition, despite the increase in turnaround time, all 5,000 jobs could still run.

An argument against installing deadlock-avoidance software is that deadlocks occur infrequently, and they cost little when they do occur.

**8.7** Can a system detect that some of its threads are starving? If you answer “yes,” explain how it can. If you answer “no,” explain how the system can deal with the starvation problem.

**Answer:**

Starvation is a difcult topic to dene, as it may mean different things for different systems. For the purposes of this question, we will dene starvation as the situation in which a process must wait beyond a reasonable period of time—perhaps indenitely—before receiving a requested resource. One way of detecting starvation would be to rst identify a period of time—*T*—that is considered unreasonable. When a process requests a resource, a timer is started. If the elapsed time exceeds *T*, then the process is considered to be starved.

One strategy for dealing with starvation would be to adopt a policy whereby resources are assigned only to the process that has been waiting the longest. For example, if process *Pa* has been waiting longer for resource *X* than process *Pb*, the request from process *Pb* would be deferred until process *Pa*’s request has been satised.

Another strategy would be less strict. In this scenario, a resource might be granted to a process that had waited less than another process, providing that the other process was not starving. However, if another process was considered to be starving, its request would be satised rst.

**8.8** Consider the following resource-allocation policy. Requests for and releases of resources are allowed at any time. If a request for resources cannot be satisfied because the resources are not available, then we check any threads that are blocked waiting for resources. If a blocked thread has the desired resources, then these resources are taken away from it and are given to the requesting thread. The vector of resources for which the blocked thread is waiting is increased to include the resources that were taken away.

For example, a system has three resource types, and the vector ***Available*** is initialized to (4,2,2). If thread *T*0 asks for (2,2,1), it gets them. If *T* 1 asks for (1,0,1), it gets them. Then, if *T*0 asks for (0,0,1), it is blocked (resource not available). If *T*2 now asks for (2,0,0), it gets the available one (1,0,0), as well as one that was allocated to *T*0 (since *T*0 is blocked). *T*0 ’s ***Allocation*** vector goes down to (1,2,1), and its ***Need*** vector goes up to (1,0,1).

a. Can deadlock occur? If you answer “yes,” give an example. If you answer “no,” specify which necessary condition cannot occur.

b. Can indefinite blocking occur? Explain your answer.

**Answer:**

a. Deadlock cannot occur, because preemption exists.

b. Yes. A process may never acquire all the resources it needs if they are continuously preempted by a series of requests such as those of process *C*.

**8.9** Consider the following snapshot of a system:

|  |  |  |
| --- | --- | --- |
|  | ***Allocation*** | ***Max*** |
|  | *A B C D* | *A B C D* |
| *T*0 | 3 0 1 4 | 5 1 1 7 |
| *T* 1 | 2 2 1 0 | 3 2 1 1 |
| *T*2 | 3 1 2 1 | 3 3 2 1 |
| *T*3 | 0 5 1 0 | 4 6 1 2 |
| *T*4 | 4 2 1 2 | 6 3 2 5 |

Using the banker’s algorithm, determine whether or not each of the following states is unsafe. If the state is safe, illustrate the order in which the threads may complete. Otherwise, illustrate why the state is unsafe.

a. ***Available*** = (0, 3, 0, 1)

b. ***Available*** = (1, 0, 0, 2)

**Answer:**

a. Not safe. Processes *P*2, *P*1, and *P*3 are able to nish, but no remaining processes can finish.

b. Safe. Processes *P*1, *P*2, and *P*3 are able to nish. Following this, processes *P*0 and *P*4 are also able to finish.

**8.10** Suppose that you have coded the deadlock-avoidance safety algorithm that determines if a system is in a safe state or not, and now have been asked to implement the deadlock-detection algorithm. Can you do so by

simply using the safety algorithm code and redefining ***Max****i* = ***Waiting****i* + ***Allocation****i*, where ***Waiting****i* is a vector specifying the resources for which thread *i* is waiting and ***Allocation****i* is as defined in Section 8.6?

Explain your answer.

**Answer:**

Yes. The *Max* vector represents the maximum request a process may make. When calculating the safety algorithm, we use the *Need* matrix, which represents *Max — Allocation*. Another way to think of this is *Max = Need + Allocation*. According to the question, the *Waiting* matrix fullls a role similar to the *Need* matrix; therefore, *Max = Waiting + Allocation*.

**8.11** Is it possible to have a deadlock involving only one single-threaded

process? Explain your answer

**Answer:**

No. This follows directly from the hold-and-wait condition.

**Chapter 8 Exercises**

**8.12** Consider the traffic deadlock depicted in Figure 8.12.

A map of cars with different colors

Description automatically generated with medium confidence

**Figure 8.11** Traffic deadlock for Exercise 8.12.

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

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

**8.13** Draw the resource-allocation graph that illustrates deadlock from the program example shown in Figure 8.1 in Section 8.2.

**8.14** In Section 6.8.1, we described a potential deadlock scenario involving processes *P*0 and *P*1 and semaphores S and Q. Draw the resourceallocation graph that illustrates deadlock under the scenario presented

in that section.

**8.15** Assume that a multithreaded application uses only reader–writer locks for synchronization. Applying the four necessary conditions for deadlock, is deadlock still possible if multiple reader–writer locks are used?

**8.16** The program example shown in Figure 8.1 doesn’t always lead to deadlock. Describe what role the CPU scheduler plays and how it can contribute to deadlock in this program.

**8.17** In Section 8.5.4, we described a situation in which we prevent deadlock by ensuring that all locks are acquired in a certain order. However, we also point out that deadlock is possible in this situation if two threads

simultaneously invoke the transaction() function. Fix the transaction() function to prevent deadlocks.

**8.18** Which of the six resource-allocation graphs shown in Figure 8.12 illustrate deadlock? For those situations that are deadlocked, provide the cycle of threads and resources. Where there is not a deadlock situation, illustrate the order in which the threads may complete execution.

**8.19** Compare the circular-wait scheme with the various deadlock-avoidance schemes (like the banker’s algorithm) with respect to the following issues:

a. Runtime overhead

b. System throughput

**8.20** In a real computer system, neither the resources available nor the demands of threads for resources are consistent over long periods (months). Resources break or are replaced, new processes and threads come and go, and 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 collage of diagrams

Description automatically generated

**Figure 8.12** Resource-allocation graphs for Exercise 8.18.

a. Increase ***Available*** (new resources added).

b. Decrease ***Available*** (resource permanently removed from system).

c. Increase ***Max*** for one thread (the thread needs or wants more resources than allowed).

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

e. Increase the number of threads.

f. Decrease the number of threads.

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 such that the system does not enter an unsafe state.

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

**8.21** Consider the following snapshot of a system:

|  |  |  |
| --- | --- | --- |
|  | ***Allocation*** | ***Max*** |
|  | *A B C D* | *A B C D* |
| *T*0 | 2 1 0 6 | 6 3 2 7 |
| *T* 1 | 3 3 1 3 | 5 4 1 5 |
| *T*2 | 2 3 1 2 | 6 6 1 4 |
| *T*3 | 1 2 3 4 | 4 3 4 5 |
| *T*4 | 3 0 3 0 | 7 2 6 1 |

What are the contents of the ***Need*** matrix?

**8.22** Consider a system consisting of four resources of the same type that are shared by three threads, each of which needs at most two resources. Show that the system is deadlock free.

**8.23** Consider a system consisting of *m* resources of the same type being shared by *n* threads. A thread can request or release only one resource at a time. Show that the system is deadlock free if the following two conditions hold:

a. The maximum need of each thread is between one resource and *m* resources.

* This means that no process can request more than m resources at any given time.
* This ensures that there will always be enough resources available to satisfy the needs of all processes.

b. The sum of all maximum needs is less than *m* + *n.*

* This means that the total number of resources needed by all processes is less than the total number of resources available (m) plus the total number of processes (n).
* This ensures that there will always be some resources available for each process to request, even if all processes request their maximum number of resources at the same time.
* Together, these two conditions guarantee that there will never be a situation in which all processes are waiting for resources that are held by other processes, thus avoiding deadlock.

**8.24** Consider the version of the dining-philosophers problem in which the chopsticks are placed at the center of the table and any two of them can 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 can be satisfied without causing deadlock given the current allocation of chopsticks to philosophers.

**8.25** Consider again the setting in the preceding exercise. Assume now that each philosopher requires three chopsticks to eat. Resource requests are still issued one at a time. Describe some simple rules for determining whether a particular request can be satisfied without causing deadlock given the current allocation of chopsticks to philosophers.

**8.26** 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 we cannot implement the multipleresource-type banker’s scheme by applying the single-resource-type

scheme to each resource type individually.

**8.27** Consider the following snapshot of a system:

|  |  |  |
| --- | --- | --- |
|  | ***Allocation*** | ***Max*** |
|  | *A B C D* | *A B C D* |
| *T*0 | 1 2 0 2 | 4 3 1 6 |
| *T* 1 | 0 1 1 2 | 2 4 2 4 |
| *T*2 | 1 2 4 0 | 3 6 5 1 |
| *T*3 | 1 2 0 1 | 2 6 2 3 |
| *T*4 | 1 0 0 1 | 3 1 1 2 |

Using the banker’s algorithm, determine whether or not each of the following states is unsafe. If the state is safe, illustrate the order in which the threads may complete. Otherwise, illustrate why the state is unsafe.

a. ***Available*** = (2, 2, 2, 3)

b. ***Available*** = (4, 4, 1, 1)

c. ***Available*** = (3, 0, 1, 4)

d. ***Available*** = (1, 5, 2, 2)

A screenshot of a computer code

Description automatically generated

A close-up of a document

Description automatically generated

A white sheet with black text

Description automatically generated

**8.28** Consider the following snapshot of a system:

|  |  |  |  |
| --- | --- | --- | --- |
|  | ***Allocation*** | ***Max*** | ***Available*** |
|  | *A B C D* | *A B C D* | *A B C D* |
| *T*0 | 3 1 4 1 | 6 4 7 3 | 2 2 2 4 |
| *T* 1 | 2 1 0 2 | 4 2 3 2 |  |
| *T*2 | 2 4 1 3 | 2 5 3 3 |  |
| *T*3 | 4 1 1 0 | 6 3 3 2 |  |
| *T*4 | 2 2 2 1 | 5 6 7 5 |  |

Answer the following questions using the banker’s algorithm:

a. Illustrate that the system is in a safe state by demonstrating an order in which the threads may complete.

b. If a request from thread *T*4 arrives for (2, 2, 2, 4), can the request be granted immediately?

c. If a request from thread *T*2 arrives for (0, 1, 1, 0), can the request be granted immediately?

d. If a request from thread *T*3 arrives for (2, 2, 1, 2), can the request be granted immediately?

a. The system is in a **safe state**, and the threads can complete in the following order: T2, T1, T3, T0, T4.

b. Yes, the request from thread T4 for (2, 2, 2, 4) can be granted immediately.

c. Yes, the request from thread T2 for (0, 1, 1, 0) can be granted immediately.

d. Yes, the **request** from thread T3 for (2, 2, 1, 2) can be granted immediately.

How to answer the questions

a. The**safe sequence** is T2, T1, T3, T0, T4.

b. If a request from thread T4 arrives for (2,2,2,4), can the request be granted immediately?

To check if the request can be granted, we need to compare the request (2 2 2 4) with the available resources (2 2 2 4) and the need of thread T4 (3 4 5 4). Since the request is less than or equal to both the available resources and the need, the request can be granted immediately.

c. If a request from thread T2 arrives for (0,1,1,0), can the request be granted immediately?

To check if the request can be granted, we need to compare the request (0 1 1 0) with the **available resources**(2 2 2 4) and the need of thread T2 (0 1 2 0). Since the request is less than or equal to both the available resources and the need, the request can be granted immediately.

d. If a request from**thread T3** arrives for (2,2,1,2), can the request be granted immediately?

To check if the request can be granted, we need to compare the request (2 2 1 2) with the available resources (2 2 2 4) and the need of thread T3 (2 2 2 2). Since the request is less than or equal to both the available resources and the need, the request can be granted immediately.

**8.29** What is the optimistic assumption made in the deadlock-detection algorithm? How can this assumption be violated?

**8.30** 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 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 and/or mutex locks, design an algorithm in pseudocode 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).

**8.31** Modify your solution to Exercise 8.30 so that it is starvation-free.

**CHAPTER 9: Main Memory**

**Practice Exercises**

**9.1** Name two differences between logical and physical addresses.

|  |
| --- |
| **Answer:** |

A logical address does not refer to an actual physical address; rather, it refers to an abstract address in an abstract address space. A physical address refers to an actual physical address in memory. Alogical address is generated by the CPU and is translated into a physical address by the memory management unit(MMU). Therefore, physical addresses are generated by the MMU.

**9.2** Why are page sizes always powers of 2?

|  |
| --- |
| **Answer:** |

Recall that paging is implemented by breaking up an address into a page and offset number. It is most efcient to break the address into ***X*** page bits and ***Y*** offset bits, rather than perform arithmetic on the address to calculate the page number and offset. Because each bit position represents a power of 2, splitting an address between bits results in a page size that is a power of 2.

**9.3** Consider a system in which a program can be separated into two parts: code and data. The CPU knows whether it wants an instruction (instruction fetch) or data (data fetch or store). Therefore, two base–limit register pairs are provided: one for instructions and one for data. The instruction

base–limit register pair is automatically read-only, so programs can be shared among different users. Discuss the advantages and disadvantages of this scheme.

**Answer:**

The major advantage of this scheme is that it is an effective mechanism for code and data sharing. For example, only one copy of an editor or a compiler needs to be kept in memory, and this code can be shared by all processes needing access to the editor or compiler code. Another advantage is protection of code against erroneous modication. The only disadvantage is that the code and data must be separated, which is usually adhered to in a compiler-generated code.

**9.4** Consider a logical address space of 64 pages of 1,024 words each, mapped onto a physical memory of 32 frames.

a. How many bits are there in the logical address?

b. How many bits are there in the physical address?

a. Logical address: 16 bits

b. Physical address: 15 bits

**9.5** What is the effect of allowing two entries in a page table to point to the same page frame in memory? Explain how this effect could be used to decrease the amount of time needed to copy a large amount of memory

from one place to another. What effect would updating some byte on one page have on the other page?

**Answer:**

By allowing two entries in a page table to point to the same page frame in memory, users can share code and data. If the code is reentrant, much memory space can be saved through the shared use of large programs such as text editors, compilers, and database systems. “Copying” large amounts of memory could be effected by having different page tables point to the same memory location.

However, sharing of nonreentrant code or data means that any user having access to the code can modify it, and these modications would be reected in the other user’s “copy.”

**9.6** Given six memory partitions of 300 KB, 600 KB, 350 KB, 200 KB, 750 KB, and 125 KB (in order), how would the first-fit, best-fit, and worst-fit algorithms place processes of size 115 KB, 500 KB, 358 KB, 200 KB, and 375 KB (in order)?

**Answer:**

a. **First fit**:

- 115 KB is put in 300-KB partition, leaving 185 KB, 600 KB, 350 KB, 200 KB, 750 KB, 125 KB

- 500 KB is put in 600-KB partition, leaving 185 KB, 100 KB, 350 KB, 200 KB, 750 KB, 125 KB

- 358 KB is put in 750-KB partition, leaving 185 KB, 100 KB, 350 KB, 200 KB, 392 KB, 125 KB

- 200 KB is put in 350-KB partition, leaving 185 KB, 100 KB, 150 KB, 200 KB, 392 KB, 125 KB

- 375 KB is put in 392-KB partition, leaving 185 KB, 100 KB, 150 KB, 200 KB, 17 KB, 125 KB

b. **Best fit**:

- 115 KB is put in 125-KB partition, leaving 300 KB, 600 KB, 350 KB, 200KB, 750 KB, 10 KB

- 500 KB is put in 600-KB partition, leaving 300 KB, 100 KB, 350 KB, 200 KB, 750 KB, 10 KB

- 358 KB is put in 750-KB partition, leaving 300 KB, 100 KB, 350 KB, 200 KB, 392 KB, 10 KB

- 200 KB is put in 200-KB partition, leaving 300 KB, 100 KB, 350 KB, 0 KB, 392 KB, 10 KB

- 375 KB is put in 392-KB partition, leaving 300 KB, 100 KB, 350 KB, 0 KB, 17 KB, 10 KB

c. **Worst fit**:

- 115 KB is put in 750-KB partition, leaving 300 KB, 600 KB, 350 KB, 200 KB, 635 KB, 125 KB

- 500 KB is put in 635-KB partition, leaving 300 KB, 600 KB, 350 KB, 200 KB, 135 KB, 125 KB

- 358 KB is put in 600-KB partition, leaving 300 KB, 242 KB, 350 KB, 200 KB, 135 KB, 125 KB

- 200 KB is put in 350-KB partition, leaving 300 KB, 242 KB, 150 KB, 200 KB, 135 KB, 125 KB

- 375 KB must wait

**9.7** Assuming a 1-KB page size, what are the page numbers and offsets for the following address references (provided as decimal numbers):

a. 3085

b. 42095

c. 215201

d. 650000

e. 2000001

**Answer:**

a. page = 3; offset = 13

b. page = 41; offset = 111

c. page = 210; offset = 161

d. page = 634; offset = 784

e. page = 1953; offset = 129

**9.8** The BTV operating system has a 21-bit virtual address, yet on certain embedded devices, it has only a 16-bit physical address. It also has a 2-KB page size. How many entries are there in each of the following?

a. A conventional, single-level page table

b. An inverted page table

What is the maximum amount of physical memory in the BTV operating system?

**Answer:**

Conventional, single-level page table will have 210 = 1024 entries. Inverted page table will have 25 = 32 entries. The maximum amount of physical memory is 216 = 65536 (or 64 KB.)

**9.9** Consider a logical address space of 256 pages with a 4-KB page size, mapped onto a physical memory of 64 frames.

a. How many bits are required in the logical address?

b. How many bits are required in the physical address?

**Answer:**

a. 12 + 8 = 20 bits.

b. 12 + 6 = 18 bits.

**9.10** Consider a computer system with a 32-bit logical address and 4-KB page size. The system supports up to 512 MB of physical memory. How many entries are there in each of the following?

a. A conventional, single-level page table

b. An inverted page table

**Answer:**

a. 220 entries.

b. 512 K K/4K = 128K entries.

**Chapter 9 Exercises**

**9.11** Explain the difference between internal and external fragmentation.

**9.12** Consider the following process for generating binaries. A compiler is used to generate the object code for individual modules, and a linker is used to combine multiple object modules into a single program binary.

How does the linker change the binding of instructions and data to memory addresses? What information needs to be passed from the compiler to the linker to facilitate the memory-binding tasks of the linker?

**9.13** Given six memory partitions of 100 MB, 170 MB, 40 MB, 205 MB, 300 MB, and 185 MB (in order), how would the first-fit, best-fit, and worst-fit algorithms place processes of size 200 MB, 15 MB, 185 MB, 75 MB, 175

MB, and 80 MB (in order)? Indicate which—if any—requests cannot be satisfied. Comment on how efficiently each of the algorithms manages memory.

Given memory spaces P1=200MB, P2= 15MB, P3=185MB, P4=75MB, P5=175MB, and P6=80MB.

Available spaces F1=100MB, F2=170MB, F3=40MB, F4205MB, F5=300MB, and F6=185MB.

Three proposed algorithms were used to define memory space and properly labelled processes.

**First-fit algorithm**

P1 allocated to F4 with remaining space of 205-200 = 5MB.

P2 allocated to F1 with remaining space of 100-15= 85MB

P3 allocated to F5 with remaining space of 300-185= 115MB.

P4 allocated to F1 with remaining space of 85-75= 10MB

P5 allocated to F6 with remaining space of 185-175=10MB

P6 allocated to F2 with remaining space of 170-80= 90MB

**Best-fit algorithm**

P1 allocated to F4 with remaining space of 205-200 = 5MB.

P2 allocated to F3 with remaining space of 40-15= 85MB

P3 was allocated to F6 but is entirely occupied, so no remaining space will be there

P4 allocated to F1 with remaining space of 100-75= 25MB

P5 allocated to F5 with remaining space of 300-175= 125MB

P6 allocated to F5 with remaining space of 125-80=45MB.

**Worst-fit Algorithm**

P1 allocated to F5 with remaining space of 300-200 = 100MB.

P2 allocated to F4 with remaining space of 205-15=190MB P3 allocated to F4 with the remaining space of 190-185= 5MB.

P4 allocated to F6 with remaining space of 185-75=110MB.

P5 does not contain any memory available spaces.

P6 allocated to F2 with remaining space of 170-80= 90MB.

According to this, the best-fit algorithm located and manages the memory very well whereas the worst-fit algorithm does not allocate the memory properly.

**9.14** Most systems allow a program to allocate more memory to its address space during execution. Allocation of data 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. Paging

**9.15** Compare the memory organization schemes of contiguous memory allocation and paging with respect to the following issues:

a. External fragmentation

b. Internal fragmentation

c. Ability to share code across processes

Comparison based on memory organization schemes concerning external fragmentation, internal fragmentation and ability to share code across processes.

**Contiguous memory**

1. External fragmentation:

* **Address space-** gaps and holes will constantly be growing while initiating a new process.
* **Variable size partition**- will suffer
* **Fixed-size partition**- Will not suffer

2. Internal fragmentation:

* **Address space-**gaps and holes will be constant while initiating a new process.
* **Variable size partition**- will not suffer
* **Fixed-size partition**- will suffer

3. Sharing code- Due to the virtual segment, contiguous memory will be not supported.

**9.16** On a system with paging, a process cannot access memory that it does not own. Why? How could the operating system allow access to additional memory? Why should it or should it not?

**9.17** Explain why mobile operating systems such as iOS and Android do not support swapping.

**9.18** Although Android does not support swapping on its boot disk, it is possible to set up a swap space using a separate SD nonvolatile memory card. Why would Android disallow swapping on its boot disk yet allow it on a secondary disk?

**9.19** Explain why address-space identifiers (ASIDs) are used in TLBs.

**9.20** 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, which 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 toward lower virtual addresses. What is the significance of this structure for the following schemes?

a. Contiguous memory allocation

b. Paging

**9.21** Assuming a 1-KB page size, what are the page numbers and offsets for the following address references (provided as decimal numbers)?

a. 21205

b. 164250

c. 121357

d. 16479315

e. 27253187

To determine the page numbers and offsets for the given address references, we need to consider the page size of 1-kilobyte (1KB).

To find the page number, we divide the decimal address by the page size. The quotient gives us the page number, and the remainder gives us the offset within that page.

Let's calculate the page numbers and offsets for each address reference:

(a) Address 21205:

Page Number = 21205 / 1024 = 20

Offset = 21205 % 1024 = 205

Therefore, for address 21205, the page number is 20 and the offset is 205.

(b) Address 164250:

Page Number = 164250 / 1024 = 160

Offset = 164250 % 1024 = 362

For address 164250, the page number is 160 and the offset is 362.

(c) Address 121357:

Page Number = 121357 / 1024 = 118

Offset = 121357 % 1024 = 389

So, for address 121357, the page number is 118 and the offset is 389.

(d) Address 16479315:

Page Number = 16479315 / 1024 = 16077

Offset = 16479315 % 1024 = 931

For address 16479315, the page number is 16077 and the offset is 931.

(e) Address 27253187:

Page Number = 27253187 / 1024 = 26613

Offset = 27253187 % 1024 = 619

Finally, for address 27253187, the page number is 26613 and the offset is 619.

To summarize:

(a) Page number: 20, Offset: 205

(b) Page number: 160, Offset: 362

(c) Page number: 118, Offset: 389

(d) Page number: 16077, Offset: 931

(e) Page number: 26613, Offset: 619

By using the page size and performing the division and modulo operations, we can find the page numbers and offsets for the given address references.

**9.22** The MPV operating system is designed for embedded systems and has a 24-bit virtual address, a 20-bit physical address, and a 4-KB page size. How many entries are there in each of the following?

a. A conventional, single-level page table

b. An inverted page table

What is the maximum amount of physical memory in the MPV operating system?

**a. Conventional, single-level page table:**

The number of entries in a conventional, single-level page table is equal to the number of pages that can be addressed in the virtual memory.

Given that the virtual address is 24-bit and the page size is 4-KB (which is 212212 bytes), the number of pages in the virtual memory is 2(24-12) = 2122(24−12)=212.

So, there are **4096 entries** in a conventional, single-level page table.

**b. Inverted page table:**

The number of entries in an inverted page table is equal to the number of frames in the physical memory.

Given that the physical address is 20-bit and the page size is 4-KB (which is 212212 bytes), the number of frames in the physical memory is 2(20-12) = 282(20−12)=28.

So, there are **256 entries** in an inverted page table.

**Maximum amount of physical memory in the MPV operating system:**

The maximum amount of physical memory that can be addressed is determined by the size of the physical address.

Given that the physical address is 20-bit, the maximum amount of physical memory that can be addressed is

220220 bytes, which is **1 Megabyte (MB)**.

Please note that these calculations assume that every bit in the address can be used and that there are no reserved addresses or other constraints on the address space. If such constraints exist, the actual amount of addressable memory could be less.

**9.23** Consider a logical address space of 2,048 pages with a 4-KB page size, mapped onto a physical memory of 512 frames.

a. How many bits are required in the logical address?

b. How many bits are required in the physical address?

a. The number of bits required in the logical address can be determined by calculating the total number of pages in the logical address space. In this case, the logical address space has 2,048 pages. To find the number of bits required, we can calculate the logarithm base 2 of 2,048.

log2(2048) = 11

Therefore, 11 bits are required in the logical address.

b. The number of bits required in the physical address can be determined by calculating the total number of frames in the physical memory. In this case, the physical memory has 512 frames. To find the number of bits required, we can calculate the logarithm base 2 of 512.

log2(512) = 9

Therefore, 9 bits are required in the physical address.

c. The maximum amount of physical memory in this system can be determined by multiplying the number of frames by the page size. In this case, the page size is 4 KB.

Maximum amount of physical memory = 512 frames \* 4 KB/frame

= 512 \* 4 KB

= 2048 KB

Therefore, the maximum amount of physical memory in this system is 2048 KB.

**9.24** Consider a computer system with a 32-bit logical address and 8-KB page size. The system supports up to 1 GB of physical memory. How many entries are there in each of the following?

a. A conventional, single-level page table

b. An inverted page table

A conventional, single-level page table in a system with a 32-bit logical address and 8-KB page size would have 8388608 entries. An inverted page table in the same system would have 2097152 entries.

Explanation:

The logical address space in a paging system is divided into fixed-size units called pages. The number of entries in a single-level page table corresponds with the number of pages that can be addressed.

To calculate this, we can use the formula: Number of pages = Size of logical address space / Page size. Here, the logical address is 32-bit, and the page size is 8-KB (or 8192 bytes, since 1 KB = 1024 bytes). Given a 32-bit logical address, the maximum addressable space is 2^32 bytes. Hence, the Number of pages = 2^32 / 8192 = 2^23 = 8388608 entries. So, a conventional, single-level page table would have 8388608 entries.

An inverted page table, however, has one entry per frame in physical memory, rather than per page of logical address space. Given that there is 1 GB (or 2^30 bytes) of physical memory and a page size of 8-KB, the number of frames is 2^30 / 8192 = 2^21 = 2097152 entries. So, an inverted page table would have 2097152 entries.

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

a. If a memory reference takes 50 nanoseconds, how long does a paged memory reference take?

b. If we add TLBs, and if 75 percent of all page-table references are found in the TLBs, what is the effective memory reference time? (Assume that finding a page-table entry in the TLBs takes 2 nanoseconds, if the entry is present.)

a. 100 ns : 50 ns for accessing page table , 50 ns for accessing the target data in the memory.

b. timeavg = 0.75∗(50ns+2ns) + 0.25∗(2ns+50ns+50ns) = 64.5ns

**9.26** What is the purpose of paging the page tables?

Paging the page tables is a memory management technique used in computer operating systems to efficiently manage virtual memory. It is designed to overcome the limitations of storing all page table entries in contiguous memory, especially when dealing with large address spaces.

In a paging system, the memory is divided into fixed-size blocks called "pages," and the virtual memory space of a process is divided into fixed-size blocks called "page frames." The page table is a data structure that keeps track of the mapping between the virtual pages and physical page frames in memory.

Paging the page tables involves dividing the page table itself into smaller units called "page table entries" (PTEs). The page table entries contain information about the mapping of virtual page numbers to physical page frame numbers. The purpose of paging the page tables is as follows:

1. **Efficient Memory Usage**: By dividing the page table into smaller units (PTEs), the operating system can allocate memory for each process more efficiently. Instead of allocating memory for the entire page table, it only needs to allocate memory for the required PTEs, reducing memory wastage.
2. **Reduced Memory Overhead**: Paging the page tables reduces the memory overhead associated with managing the page table itself. With large address spaces, storing the entire page table in contiguous memory would be impractical and consume a significant amount of memory. Paging the page tables allows for a more compact and scalable approach.
3. **Page Table Swapping**: Paging the page tables enables the operating system to swap page table entries in and out of memory as needed. When a process is not actively using all of its page table entries, the OS can swap out some of the PTEs to free up memory for other processes or system tasks.
4. **Increased Address Space**: Paging the page tables enables the operating system to support larger address spaces since it does not need to allocate contiguous memory for the entire page table. This is particularly useful in modern systems with vast amounts of virtual memory.

Overall, paging the page tables is a crucial technique in virtual memory management, enabling efficient memory usage, reducing overhead, and supporting large address spaces in modern computer systems. It contributes to the effective utilization of resources and helps maintain system performance and stability.

**9.27** Consider the IA-32 address-translation scheme shown in Figure 9.22.

a. Describe all the steps taken by the IA-32 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?

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

**CHAPTER 10: Virtual Memory**

**Practice Exercises**

**10.1** Under what circumstances do page faults occur? Describe the actions taken by the operating system when a page fault occurs.

**Answer:**

A page fault occurs when an access to a page that has not been brought into main memory takes place. The operating system veries the memory access, aborting the program if it is invalid. If it is valid, a free frame is located and I/O is requested to read the needed page into the free frame. Upon completion of I/O, the process table and page table are updated, and the instruction is restarted.

**10.2** Assume that you have a page-reference string for a process with *m* frames (initially all empty). The page-reference string has length *p*, and *n* distinct page numbers occur in it. Answer these questions for any

page-replacement algorithms:

a. What is a lower bound on the number of page faults?

b. What is an upper bound on the number of page faults?

**Answer:**

a. *n*

b. *p*

**10.3** Consider the following page-replacement algorithms. Rank these algorithms on a five-point scale from “bad” to “perfect” according to their page-fault rate. Separate those algorithms that suffer from Belady’s

anomaly from those that do not.

a. LRU replacement

b. FIFO replacement

c. Optimal replacement

d. Second-chance replacement

**Answer:**

Rank Algorithm Suffer from Belady’s anomaly

1 Optimal no

2 LRU no

3 Second-chance yes

4 FIFO yes

**10.4** An operating system supports a paged virtual memory. The central processor has a cycle time of 1 microsecond. It costs an additional 1 microsecond to access a page other than the current one. Pages have

1,000 words, and the paging device is a drum that rotates at 3,000 revolutions per minute and transfers 1 million words per second. The following statistical measurements were obtained from the system:

• One percent of all instructions executed accessed a page other than the current page.

• Of the instructions that accessed another page, 80 percent accessed a page already in memory.

• When a new page was required, the replaced page was modified 50 percent of the time.

Calculate the effective instruction time on this system, assuming that the system is running one process only and that the processor is idle during drum transfers.

**Answer:**

effective access time = 0.99 × (1 μsec + 0.008 × (2 μsec) + 0.002 × (10,000 μsec + 1,000 μsec) + 0.001 × (10,000 μsec + 1,000 μsec)

= (0.99 + 0.016 + 22.0 + 11.0) μsec

= 34.0 μsec

**10.5** Consider the page table for a system with 12-bit virtual and physical addresses and 256-byte pages.

|  |  |
| --- | --- |
| Page | Page Frame |
| 0 | – |
| 1 | 2 |
| 2 | C |
| 3 | A |
| 4 | – |
| 5 | 4 |
| 6 | 3 |
| 7 | – |
| 8 | B |
| 9 | 0 |

The list of free page frames is *D*, *E*, *F* (that is, *D* is at the head of the list, *E* is second, and *F* is last). A dash for a page frame indicates that the page is not in memory.

Convert the following virtual addresses to their equivalent physical addresses in hexadecimal. All numbers are given in hexadecimal.

• 9EF

• 111

• 700

• 0FF

**Answer:**

• 9*EF* → 0*EF*

• 111 → 211

• 700 → *D*00

• 0*FF* → *EFF*

**10.6** Discuss the hardware functions required to support demand paging.

**Answer:**

For every memory-access operation, the page table must be consulted to check whether the corresponding page is resident and whether the program has read or write privileges for accessing the page. These checks must be performed in hardware. A TLB could serve as a cache and improve the performance of the lookup operation.

**10.7** Consider the two-dimensional array A:

int A[][] = new int[100][100];

where A[0][0] is at location 200 in a paged memory system with pages of size 200. A small process that manipulates the matrix resides in page 0 (locations 0 to 199). Thus, every instruction fetch will be from page 0. For three page frames, how many page faults are generated by the following array-initialization loops? Use LRU replacement, and assume that page frame 1 contains the process and the other two are initially empty.

a. for (int j = 0; j < 100; j++)

for (int i = 0; i < 100; i++)

A[i][j] = 0;

b. for (int i = 0; i < 100; i++)

for (int j = 0; j < 100; j++)

A[i][j] = 0;

**Answer:**

a. 5,000

b. 50

**10.8** Consider the following page reference string:

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

How many page faults would occur for the following replacement algorithms, assuming one, two, three, four, five, six, and seven frames? Remember that all frames are initially empty, so your first unique pages will cost one fault each.

• LRU replacement

• FIFO replacement

• Optimal replacement

**Answer:**

A close-up of a calendar

Description automatically generated

**10.9** Consider the following page reference string:

7, 2, 3, 1, 2, 5, 3, 4, 6, 7, 7, 1, 0, 5, 4, 6, 2, 3, 0, 1.

Assuming demand paging with three frames, how many page faults would occur for the following replacement algorithms?

• LRU replacement

• FIFO replacement

• Optimal replacement

**Answer:**

• 18

• 17

• 13

**10.10** Suppose that you want to use a paging algorithm that requires a reference bit (such as second-chance replacement or working-set model), but the hardware does not provide one. Sketch how you could simulate a reference bit even if one were not provided by the hardware, or explain why it is not possible to do so. If it is possible, calculate what the cost would be.

**Answer:**

You can use the valid/invalid bit supported in hardware to simulate the reference bit. Initially set the bit to invalid. On rst reference, a trap to the operating system is generated. The operating system will set a software bit to 1 and reset the valid/invalid bit to valid.

**10.11** You have devised a new page-replacement algorithm that you think may be optimal. In some contorted test cases, Belady’s anomaly occurs. Is the new algorithm optimal? Explain your answer.

**Answer:**

No. An optimal algorithm will not suffer from Belady’s anomaly because—by denition—an optimal algorithm replaces the page that will not be used for the longest time. Belady’s anomaly occurs when a page-replacement algorithm evicts a page that will be needed in the immediate future. An optimal algorithm would not have selected such a page.

**10.12** Segmentation is similar to paging but uses variable-sized “pages.” Define two segment-replacement algorithms, one based on the FIFO page-replacement scheme and the other on the LRU page-replacement scheme. Remember that since segments are not the same size, the segment that is chosen for replacement may be too small to leave enough consecutive locations for the needed segment. Consider strategies for systems where segments cannot be relocated and strategies for systems where they can.

**Answer:**

a. **FIFO**. Find the rst segment large enough to accommodate the incoming segment. If relocation is not possible and no one segment is large enough, select a combination of segments whose memories are contiguous, which are “closest to the rst of the list,” and which can accommodate the new segment. If relocation is possible, rearrange the memory so that the first *N* segments large enough for the incoming segment are contiguous in memory. Add any leftover space to the free-space list in both cases.

b. **LRU**. Select the segment that has not been used for the longest time and that is large enough, adding any leftover space to the free-space list. If no one segment is large enough, and if relocation is not available, select a combination of the “oldest” segments that are contiguous in memory and are large enough. If relocation is

available, rearrange the oldest *N* segments to be contiguous in memory and replace those with the new segment.

**10.13** Consider a demand-paged computer system where the degree of multiprogramming is currently fixed at four. The system was recently measured to determine utilization of the CPU and the paging disk. Three alternative results are shown below. For each case, what is happening? Can the degree of multiprogramming be increased to increase the CPU utilization? Is the paging helping?

a. CPU utilization 13 percent; disk utilization 97 percent

b. CPU utilization 87 percent; disk utilization 3 percent

c. CPU utilization 13 percent; disk utilization 3 percent

**Answer:**

a. Thrashing is occurring.

b. CPU utilization is sufciently high to leave things alone and increase the degree of multiprogramming.

c. Increase the degree of multiprogramming.

**10.14** We have an operating system for a machine that uses base and limit registers, but we have modified the machine to provide a page table. Can the page table be set up to simulate base and limit registers? How can it be, or why can it not be?

**Answer:**

The page table can be set up to simulate base and limit registers provided that the memory is allocated in xed-size segments. The base of a segment can be entered into the page table and the valid/invalid bit used to indicate that portion of the segment as resident in the memory. There will be some problem with internal fragmentation.

**Chapter 10 Exercises**

**10.15** Assume that a program has just referenced an address in virtual memory. Describe a scenario in which each of the following can occur. (If no such scenario can occur, explain why.)

• TLB miss with no page fault

• TLB miss with page fault

• TLB hit with no page fault

• TLB hit with page fault

• TLB miss with no page fault.Possible. Page has been brought into memory, but has been removed from the TLB.

• TLB miss and page fault.Possible. Page fault has occurred

• TLB hit and no page fault.Possible. Page is in memory and in the TLB. Most likely a recent reference

• TLB hit and page fault .Not possible. The TLB is a cache of the page table. If an entry is not in the page table, it will not be in the TLB.

**10.16** A simplified view of thread states is ***ready***, ***running***, and ***blocked***, where a thread is either ready and waiting to be scheduled, is running on the processor, or is blocked (for example, waiting for I/O).

A diagram of a process

Description automatically generated

Assuming a thread is in the ***running*** state, answer the following questions, and explain your answers:

a. Will the thread change state if it incurs a page fault? If so, to what state will it change?

b. Will the thread change state if it generates a TLB miss that is resolved in the page table? If so, to what state will it change?

c. Will the thread change state if an address reference is resolved in the page table? If so, to what state will it change?

Sure, let’s go through these scenarios:

**a. Will the thread change state if it incurs a page fault? If so, to what state will it change?**

Yes, the thread will change state if it incurs a page fault. It will move to the **blocked state** because it has to wait for the required page to be loaded into memory, causing it to be unable to proceed with execution until this operation is complete.

**b. Will the thread change state if it generates a TLB miss that is resolved in the page table? If so, to what state will it change?**

No, the thread will not change state if it generates a TLB miss that is resolved in the page table. The TLB miss would cause a slight delay as the system retrieves the required mapping from the page table, but this does not block the thread’s execution; hence it remains in the **running state**.

**c. Will the thread change state if an address reference is resolved in the page table? If so, to what state will it change?**

No, the thread will not change state if an address reference is resolved in the page table. Address resolution in the page table is a normal operation that occurs during execution and does not cause the thread to be blocked or moved into another state; thus, it remains in the **running state**.

These answers are based on the typical behavior of threads in an operating system. However, the exact behavior can vary depending on the specific implementation of the operating system. Please refer to the documentation of the specific operating system for more accurate information.

**10.17** Consider a system that uses pure demand paging.

a. When a process first starts execution, how would you characterize the page-fault rate?

b. Once the working set for a process is loaded into memory, how would you characterize the page-fault rate?

c. Assume that a process changes its locality and the size of the new working set is too large to be stored in available free memory.

Identify some options system designers could choose from to handle this situation.

**a. When a process first starts execution, how would you characterize the page-fault rate?**

[In a system that uses pure demand paging, when a process first starts execution, the page-fault rate would be high](https://www.geeksforgeeks.org/what-is-demand-paging-in-operating-system/)[1](https://www.geeksforgeeks.org/what-is-demand-paging-in-operating-system/). [This is because in pure demand paging, no pages are initially loaded into memory when the program starts, and all pages are initially marked as being on disk](https://www.geeksforgeeks.org/what-is-demand-paging-in-operating-system/)[1](https://www.geeksforgeeks.org/what-is-demand-paging-in-operating-system/). Therefore, as the process starts to execute and references pages, it will incur page faults for each unique page it references, until those pages are loaded into memory[1](https://www.geeksforgeeks.org/what-is-demand-paging-in-operating-system/).

**b. Once the working set for a process is loaded into memory, how would you characterize the page-fault rate?**

Once the working set (the set of pages that a process is actively using) for a process is loaded into memory, the page-fault rate would be low or even zero[2](https://phoenixnap.com/kb/paging)[3](https://matthews.sites.truman.edu/files/2017/01/chapter9.pdf). [This is because most of the page references generated by the process can be satisfied from the memory, resulting in a lower page fault rate](https://www.geeksforgeeks.org/what-is-demand-paging-in-operating-system/)[4](https://questions.llc/questions/517861).

**c. Assume that a process changes its locality and the size of the new working set is too large to be stored in available free memory. Identify some options system designers could choose from to handle this situation.**

If a process changes its locality and the size of the new working set is too large to fit into available free memory, system designers have several options[5](https://quizlet.com/403705316/chapter-9-operating-systems-flash-cards/)[6](https://www.geeksforgeeks.org/working-set-in-paging/):

1. **Increase the size of the physical memory**: If there is enough physical memory capacity, the system designer can choose to add more memory to accommodate the larger working set[5](https://quizlet.com/403705316/chapter-9-operating-systems-flash-cards/).
2. **Use a paging or swapping mechanism**: The operating system could move some pages that are not in the current working set out to disk, freeing up memory for the new working set[6](https://www.geeksforgeeks.org/working-set-in-paging/).
3. **Suspend some processes**: The operating system could suspend some less important processes, freeing up their memory for use by the process with the large working set[5](https://quizlet.com/403705316/chapter-9-operating-systems-flash-cards/).
4. **Optimize the memory management algorithms**: The operating system could use more sophisticated memory management algorithms to optimize the use of available memory[6](https://www.geeksforgeeks.org/working-set-in-paging/).

These strategies could be used individually or in combination, depending on the specific requirements and constraints of the system.

**10.18** The following is a page table for a system with 12-bit virtual and physical addresses and 256-byte pages. Free page frames are to be allocated in the order 9, F, D. A dash for a page frame indicates that the page is not in memory.

|  |  |
| --- | --- |
| Page | Page Frame |
| 0 | 0 x 4 |
| 1 | 0 x B |
| 2 | 0 x A |
| 3 | – |
| 4 | – |
| 5 | 0 x 2 |
| 6 | – |
| 7 | 0 x 0 |
| 8 | 0 x C |
| 9 | 0 x 1 |

Convert the following virtual addresses to their equivalent physical addresses in hexadecimal. All numbers are given in hexadecimal. In the case of a page fault, you must use one of the free frames to update the page table and resolve the logical address to its corresponding physical address.

• 0x2A1

• 0x4E6

• 0x94A

• 0x316

Given that the page size is 256 bytes, the lower 8 bits of the virtual address are used as the offset within the page, and the upper 4 bits are used as the page number.

The free frames are to be allocated in the order 9, F, D.

**1. Virtual Address: 0x2A1**

* Page number = 2 (from the upper 4 bits of the virtual address)
* Offset = A1 (from the lower 8 bits of the virtual address)

The page table shows that page 2 is in frame A. So, the physical address is A1.

**2. Virtual Address: 0x4E6**

* Page number = 4 (from the upper 4 bits of the virtual address)
* Offset = E6 (from the lower 8 bits of the virtual address)

The page table shows that page 4 is not in memory (indicated by a dash). This is a page fault. We allocate the first free frame (9) to this page. So, the physical address is 9E6.

**3. Virtual Address: 0x94A**

* Page number = 9 (from the upper 4 bits of the virtual address)
* Offset = 4A (from the lower 8 bits of the virtual address)

The page table shows that page 9 is in frame 1. So, the physical address is 14A.

**4. Virtual Address: 0x316**

* Page number = 3 (from the upper 4 bits of the virtual address)
* Offset = 16 (from the lower 8 bits of the virtual address)

The page table shows that page 3 is not in memory (indicated by a dash). This is a page fault. We allocate the next free frame (F) to this page. So, the physical address is F16.

So, the equivalent physical addresses for the given virtual addresses are:

* 0x2A1 -> 0xA1
* 0x4E6 -> 0x9E6
* 0x94A -> 0x14A
* 0x316 -> 0xF16

Please note that these calculations assume that every bit in the address can be used and that there are no reserved addresses or other constraints on the address space. If such constraints exist, the actual amount of addressable memory could be less.

A white rectangular object with black lines

Description automatically generated

**10.19** What is the copy-on-write feature, and under what circumstances is its use beneficial? What hardware support is required to implement this feature?

**10.20** A certain computer provides its users with a virtual memory space of 232 bytes. The computer has 222 bytes of physical memory. The virtual memory is implemented by paging, and the page size is 4,096 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 212, the page table size is 220. 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.

**10.21** Assume that 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 frame is available or if 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?

0.2 \_sec = (1 − P) × 0.1 \_sec + (0.3P) × 8 millisec + (0.7P) × 20 millisec

0.1 = −0.1P + 2400 P + 14000 P

0.1 \_ 16,400 P

P \_ 0.000006

**10.22** Consider the page table for a system with 16-bit virtual and physical addresses and 4,096-byte pages.

|  |  |  |
| --- | --- | --- |
| Page | Page Frame | Reference Bit |
| 0 | 9 | 0 |
| 1 | – | 0 |
| 2 | 10 | 0 |
| 3 | 15 | 0 |
| 4 | 6 | 0 |
| 5 | 13 | 0 |
| 6 | 8 | 0 |
| 7 | 12 | 0 |
| 8 | 7 | 0 |
| 5 | 10 | 0 |
| 4 | 11 | 0 |
| 1 | 12 | 0 |
| 0 | 13 | 0 |
| 2 | 15 | 0 |
| 9 | 0 | – |
| 14 | 0 | – |

The reference bit for a page is set to 1 when the page has been referenced. Periodically, a thread zeroes out all values of the reference bit. A dash for a page frame indicates that the page is not in memory. The page-replacement algorithm is localized LRU, and all numbers are provided in decimal.

a. Convert the following virtual addresses (in hexadecimal) to the

equivalent physical addresses. You may provide answers in either

hexadecimal or decimal. Also set the reference bit for the appropriate entry in the page table.

• 0x621C

• 0xF0A3

• 0xBC1A

• 0x5BAA

• 0x0BA1

b. Using the above addresses as a guide, provide an example of a logical address (in hexadecimal) that results in a page fault.

c. From what set of page frames will the LRU page-replacement algorithm choose in resolving a page fault?

Let’s solve this step by step:

**a. Convert the following virtual addresses (in hexadecimal) to the equivalent physical addresses:**

Given that the page size is 4,096 bytes, the lower 12 bits of the virtual address are used as the offset within the page, and the upper 4 bits are used as the page number.

Here’s how we can do it:

**1. Virtual Address: 0x621C**

* Page number = 6 (from the upper 4 bits of the virtual address)
* Offset = 21C (from the lower 12 bits of the virtual address)

The page table shows that page 6 is in frame 8. So, the physical address is 821C.

**2. Virtual Address: 0xF0A3**

* Page number = 15 (from the upper 4 bits of the virtual address)
* Offset = 0A3 (from the lower 12 bits of the virtual address)

The page table shows that page 15 is not in memory (indicated by a dash). This is a page fault. We need to allocate a free frame to this page.

**3. Virtual Address: 0xBC1A**

* Page number = 11 (from the upper 4 bits of the virtual address)
* Offset = C1A (from the lower 12 bits of the virtual address)

The page table does not have an entry for page 11. This is a page fault. We need to allocate a free frame to this page.

**4. Virtual Address: 0x5BAA**

* Page number = 5 (from the upper 4 bits of the virtual address)
* Offset = BAA (from the lower 12 bits of the virtual address)

The page table shows that page 5 is in frame 13. So, the physical address is DBAA.

**5. Virtual Address: 0x0BA1**

* Page number = 0 (from the upper 4 bits of the virtual address)
* Offset = BA1 (from the lower 12 bits of the virtual address)

The page table shows that page 0 is in frame 13. So, the physical address is DBA1.

**b. Using the above addresses as a guide, provide an example of a logical address (in hexadecimal) that results in a page fault.**

Any logical address with a page number that is not in the page table or is marked as not in memory (with a dash) will result in a page fault. For example, a logical address starting with F (such as 0xF123) would result in a page fault because page 15 is not in memory according to the page table.

**c. From what set of page frames will the LRU page-replacement algorithm choose in resolving a page fault?**

The LRU (Least Recently Used) page-replacement algorithm will choose the page frame that has not been used for the longest time. This is determined by the reference bit in the page table. A page with a reference bit of 0 is a candidate for replacement. If all pages have a reference bit of 1, then the LRU algorithm will need additional information (such as a timestamp of the last reference) to determine which page to replace. In this case, since all reference bits are 0, any page could be a candidate for replacement.

**10.23** When a page fault occurs, the process requesting the page must block while waiting for the page to be brought from disk into physical memory. Assume that there exists a process with five user-level threads and that the mapping of user threads to kernel threads is many to one. If one user thread incurs a page fault while accessing its stack, would the other user threads belonging to the same process also be affected by the page fault—that is, would they also have to wait for the faulting page to be brought into memory? Explain.

**10.24** Apply the (1) FIFO, (2) LRU, and (3) optimal (OPT) replacement algorithms for the following page-reference strings:

• 2, 6, 9, 2, 4, 2, 1, 7, 3, 0, 5, 2, 1, 2, 9, 5, 7, 3, 8, 5

• 0, 6, 3, 0, 2, 6, 3, 5, 2, 4, 1, 3, 0, 6, 1, 4, 2, 3, 5, 7

• 3, 1, 4, 2, 5, 4, 1, 3, 5, 2, 0, 1, 1, 0, 2, 3, 4, 5, 0, 1

• 4, 2, 1, 7, 9, 8, 3, 5, 2, 6, 8, 1, 0, 7, 2, 4, 1, 3, 5, 8

• 0, 1, 2, 3, 4, 4, 3, 2, 1, 0, 0, 1, 2, 3, 4, 4, 3, 2, 1, 0

Indicate the number of page faults for each algorithm assuming demand paging with three frames.

To determine the number of page faults for each algorithm (FIFO, LRU, and OPT) with three frames, we need to simulate the page replacement process for each page-reference string.

Let's go through each page-reference string and calculate the number of page faults for each algorithm:

1. Page-reference string: 2, 6, 9, 2, 4, 2, 1, 7, 3, 0, 5, 2, 1, 2, 9, 5, 7, 3, 8, 5

* FIFO:
* Frames: [2, *, ], [2, 6, ], [2, 6, 9], [, 6, 9], [, 4, 9], [*, 4, 2], [1, 4, 2], [1, 7, 2], [1, 7, 3], [0, 7, 3], [0, 5, 3], [0, 5, 2], [1, 5, 2], [1, 9, 2], [1, 9, 5], [7, 9, 5], [7, 3, 5], [8, 3, 5], [8, 3, \_], [8, 5, \_]
* Number of page faults: 14
* LRU:
* Frames: [2, *, ], [2, 6, ], [2, 6, 9], [, 6, 9], [, 4, 9], [*, 4, 2], [1, 4, 2], [1, 7, 2], [1, 7, 3], [0, 7, 3], [0, 5, 3], [0, 5, 2], [1, 5, 2], [1, 9, 2], [1, 9, 5], [7, 9, 5], [7, 3, 5], [8, 3, 5], [8, 3, \_], [8, 5, \_]
* Number of page faults: 14
* OPT:
* Frames: [2, *, ], [2, 6, ], [2, 6, 9], [, 6, 9], [, 4, 9], [*, 4, 2], [1, 4, 2], [1, 7, 2], [1, 7, 3], [0, 7, 3], [0, 5, 3], [0, 5, 2], [1, 5, 2], [1, 9, 2], [1, 9, 5], [7, 9, 5], [7, 3, 5], [8, 3, 5], [8, 3, \_], [8, 5, \_]
* Number of page faults: 14

1. Page-reference string: 0, 6, 3, 0, 2, 6, 3, 5, 2, 4, 1, 3, 0, 6, 1, 4, 2, 3, 5, 7

* FIFO:
* Frames: [0, *, ], [0, 6, ], [0, 6, 3], [, 6, 3], [, 2, 3], [*, 2, 6], [\_, 3, 6], [5, 3, 6], [5, 2, 6], [5, 2, 4], [1, 2, 4], [1, 3, 4], [1, 3, 0], [6, 3, 0], [6, 1, 0], [6, 1, 4], [2, 1, 4], [2, 3, 4], [2, 3, 5], [7, 3, 5]
* Number of page faults: 15
* LRU:
* Frames: [0, *, ], [0, 6, ], [0, 6, 3], [, 6, 3], [, 2, 3], [*, 2, 6], [\_, 3, 6], [5, 3, 6], [5, 2, 6], [5, 2, 4], [1, 2, 4], [1, 3, 4], [1, 3, 0], [6, 3, 0], [6, 1, 0], [6, 1, 4], [2, 1, 4], [2, 3, 4], [2, 3, 5], [7, 3, 5]
* Number of page faults: 15
* OPT:
* Frames: [0, *, ], [0, 6, ], [0, 6, 3], [, 6, 3], [, 2, 3], [*, 2, 6], [\_, 3, 6], [5, 3, 6], [5, 2, 6], [5, 2, 4], [1, 2, 4], [1, 3, 4], [1, 3, 0], [6, 3, 0], [6, 1, 0], [6, 1, 4], [2, 1, 4], [2, 3, 4], [2, 3, 5], [7, 3, 5]
* Number of page faults: 15

1. Page-reference string: 3, 1, 4, 2, 5, 4, 1, 3, 5, 2, 0, 1, 1, 0, 2, 3, 4, 5, 0, 1

* FIFO:
* Frames: [3, *, ], [3, 1, ], [3, 1, 4], [, 1, 4], [, 2, 4], [*, 2, 5], [\_, 4, 5], [1, 4, 5], [1, 3, 5], [1, 3, 2], [0, 3, 2], [0, 1, 2], [0, 1, *], [*, 1, *], [*, 0, \_], [2, 0, \_], [2, 3, \_], [2, 3, 4], [5, 3, 4], [5, 0, 4]
* Number of page faults: 14
* LRU:
* Frames: [3, *, ], [3, 1, ], [3, 1, 4], [, 1, 4], [, 2, 4], [*, 2, 5], [\_, 4, 5], [1, 4, 5], [1, 3, 5], [1, 3, 2], [0, 3, 2], [0, 1, 2], [0, 1, *], [*, 1, *], [*, 0, \_], [2, 0, \_], [2, 3, \_], [2, 3, 4], [5, 3, 4], [5, 0, 4]
* Number of page faults: 14
* OPT:
* Frames: [3, *, ], [3, 1, ], [3, 1, 4], [, 1, 4], [, 2, 4], [*, 2, 5], [\_, 4, 5], [1, 4, 5], [1, 3, 5], [1, 3, 2], [0, 3, 2], [0, 1, 2], [0, 1, *], [*, 1, *], [*, 0, \_], [2, 0, \_], [2, 3, \_], [2, 3, 4], [5, 3, 4], [5, 0, 4]
* Number of page faults: 14

1. Page-reference string: 4, 2, 1, 7, 9, 8, 3, 5, 2, 6, 8, 1, 0, 7, 2, 4, 1, 3, 5, 8

* FIFO:
* Frames: [4, *, ], [4, 2, ], [4, 2, 1], [, 2, 1], [, 7, 1], [*, 7, 9], [\_, 8, 9], [3, 8, 9], [3, 5, 9], [3, 5, 2], [6, 5, 2], [6, 8, 2], [6, 8, 1], [0, 8, 1], [0, 7, 1], [0, 7, 4], [2, 7, 4], [2, 1, 4], [2, 1, 3], [5, 1, 3]
* Number of page faults: 15
* LRU:
* Frames: [4, *, ], [4, 2, ], [4, 2, 1], [, 2, 1], [, 7, 1], [*, 7, 9], [\_, 8, 9], [3, 8, 9], [3, 5, 9], [3, 5, 2], [6, 5, 2], [6, 8, 2], [6, 8, 1], [0, 8, 1], [0, 7, 1], [0, 7, 4], [2, 7, 4], [2, 1, 4], [2, 1, 3], [5, 1, 3]
* Number of page faults: 15
* OPT:
* Frames: [4, *, ], [4, 2, ], [4, 2, 1], [, 2, 1], [, 7, 1], [*, 7, 9], [\_, 8, 9], [3, 8, 9], [3, 5, 9], [3, 5, 2], [6, 5, 2], [6, 8, 2], [6, 8, 1], [0, 8, 1], [0, 7, 1], [0, 7, 4], [2, 7, 4], [2, 1, 4], [2, 1, 3], [5, 1, 3]
* Number of page faults: 15

1. Page-reference string: 0, 1, 2, 3, 4, 4, 3, 2, 1, 0, 0, 1, 2, 3, 4, 4, 3, 2, 1, 0

* FIFO:
* Frames: [0, *, ], [0, 1, ], [0, 1, 2], [0, 1, 3], [0, 1, 4], [, 1, 4], [, 3, 4], [*, 3, 2], [*, 1, 2], [*, 1, 0], [*, 0, 0], [1, 0, 0], [1, 2, 0], [1, 2, 3], [1, 2, 4], [*, 2, 4], [*, 3, 4], [*, 3, 2], [*, 1, 2], [*, 1, 0]
* Number of page faults: 14
* LRU:
* Frames: [0, *, ], [0, 1, ], [0, 1, 2], [0, 1, 3], [0, 1, 4], [, 1, 4], [, 3, 4], [*, 3, 2], [*, 1, 2], [*, 1, 0], [*, 0, 0], [1, 0, 0], [1, 2, 0], [1, 2, 3], [1, 2, 4], [*, 2, 4], [*, 3, 4], [*, 3, 2], [*, 1, 2], [*,

A white background with black text

Description automatically generated

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

a. Pointer is moving fast.

b. Pointer is moving slow.

**10.26** Discuss situations in which the least frequently used (LFU) pagereplacement algorithm generates fewer page faults than the least recently used (LRU) page-replacement algorithm. Also discuss under what circumstances the opposite holds.

**10.27** Discuss situations in which the most frequently used (MFU) pagereplacement algorithm generates fewer page faults than the least recently used (LRU) page-replacement algorithm. Also discuss under what circumstances the opposite holds.

**10.28** The KHIE (pronounced “k-hi”) operating 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 LRU replacement policy. Answer the following questions:

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

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

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

A close-up of a page

Description automatically generated

**10.29** Consider a demand-paging system with the following time-measured utilizations:

CPU utilization 20%

Paging disk 97.7%

Other I/O devices 5%

For each of the following, indicate whether it will (or is likely to) improve CPU utilization. Explain your answers.

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.

**10.30** Explain why minor page faults take less time to resolve than major page faults.

**10.31** Explain why compressed memory is used in operating systems for mobile devices.

**10.32** Suppose that a machine provides instructions that can access memory locations using the one-level indirect addressing scheme. What sequence of page faults is incurred when all of the pages of a program are currently nonresident 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?

**10.33** Consider the page references:

What pages represent the locality at time (*X*)?

A graph with lines and numbers

Description automatically generated

**10.34** Suppose that your replacement policy (in a paged system) is to examine each page regularly and to discard 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?

**10.35** A page-replacement algorithm should minimize the number of page faults. We can achieve 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 associated with that frame. Then,

to replace a page, we can search for the page frame with the smallest counter.

a. Define a page-replacement algorithm using this basic idea. Specifically address these problems:

• What is the initial value of the counters?

• When are counters increased?

• When are counters decreased?

• How is the page to be replaced selected?

b. How many page faults occur for your algorithm for the following reference string with 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 pagereplacement strategy for the reference string in part b with four page frames?

1. a.Define a page-replacement algorithm addressing the problem of:
   1. initial value of the counters- 0
   2. Counters are increased - whenever a nee 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.
2. 14 page faults
3. 11 page faults

**10.36** 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 those remaining, 10 percent (or 2 percent of the total) cause page faults. What is the effective memory access time?

Access Time = (0.8) \* (1 microsecond) + (0.18) \* (2 microsecond) + (0.02) \* (20002 microsecond)  
            = 401.2 microsecond  
            = 0.4 millisecond

**10.37** 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?

**10.38** Is it possible for a process to have two working sets, one representing data and another representing code? Explain.

**10.39** Consider the parameter Δ used to define the working-set window in the working-set model. When Δ is set to a low value, what is the effect on the page-fault frequency and the number of active (nonsuspended) processes currently executing in the system? What is the effect when Δ is set to a very high value?

**10.40** In a 1,024-KB segment, memory is allocated using the buddy system. Using Figure 10.26 as a guide, draw a tree illustrating how the following memory requests are allocated:

• Request 5-KB

• Request 135 KB.

• Request 14 KB.

• Request 3 KB.

• Request 12 KB.

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

• Release 3 KB.

• Release 5 KB.

• Release 14 KB.

• Release 12 KB.

**10.41** A system provides support for user-level and kernel-level threads. The mapping in this system is one to one (there is a corresponding kernel thread for each user thread). Does a multithreaded process consist of (a) a working set for the entire process or (b) a working set for each thread? Explain

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

**10.43** 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 would be needed to provide this functionality?

**CHAPTER 12: I/O Systems**

**Practice Exercises**

**12.1** State three advantages of placing functionality in a device controller, rather than in the kernel. State three disadvantages.

**Answer:**

Three advantages:

a. Bugs are less likely to cause an operating system crash

b. Performance can be improved by utilizing dedicated hardware and hard-coded algorithms

c. The kernel is simplified by moving algorithms out of it

Three disadvantages:

a. Bugs are harder to fix—a new firmware version or new hardware is needed

b. Improving algorithms likewise require a hardware update rather than just a kernel or device-driver update

c. Embedded algorithms could conflict with application’s use of the device, causing decreased performance.

**12.2** The example of handshaking in Section 12.2 used two bits: a busy bit and a command-ready bit. Is it possible to implement this handshaking with only one bit? If it is, describe the protocol. If it is not, explain why

one bit is insufficient.

**Answer:**

It is possible, using the following algorithm. Let’s assume we simply use the busy-bit (or the command-ready bit; this answer is the same regardless). When the bit is off, the controller is idle. The host writes to data-out and sets the bit to signal that an operation is ready (the equivalent of setting the command-ready bit). When the controller is finished, it clears the busy bit. The host then initiates the next operation. This solution requires that both the host and the controller have read and write access to the same bit, which can complicate circuitry and increase the cost of the controller.

**12.3** Why might a system use interrupt-driven I/O to manage a single serial port and polling I/O to manage a front-end processor, such as a terminal concentrator?

**Answer:**

Polling can be more efficient than interrupt-driven I/O. This is the case when the I/O is frequent and of short duration. Even though a single serial port will perform I/O relatively infrequently and should thus use interrupts, a collection of serial ports such as those in a terminal concentrator can produce a lot of short I/O operations, and interrupting for each one could create a heavy load on the system. A well-timed polling loop could alleviate that load without wasting many resources through looping with no I/O needed.

**12.4** Polling for an I/O completion can waste a large number of CPU cycles if the processor iterates a busy-waiting loop many times before the I/O completes. But if the I/O device is ready for service, polling can be much

more efficient than catching and dispatching an interrupt. Describe a hybrid strategy that combines polling, sleeping, and interrupts for I/O device service. For each of these three strategies (pure polling, pure interrupts, hybrid), describe a computing environment in which that strategy is more efficient than either of the others.

**Answer:**

A hybrid approach could switch between polling and interrupts depending on the length of the I/O operation wait. For example, we could poll and loop N times, and if the device is still busy at N+1, we could set an interrupt and sleep. This approach would avoid long busy-waiting cycles. This method would be best for very long or very short busy times. It would be inefficient it the I/O completes at N+T (where T is a small number of cycles) due to the overhead of polling plus setting up and catching interrupts.

Pure polling is best with very short wait times. Interrupts are best with known long wait times.

**12.5** How does DMA increase system concurrency? How does it complicate hardware design?

**Answer:**

DMA increases system concurrency by allowing the CPU to perform tasks while the DMA system transfers data via the system and memory buses. Hardware design is complicated because the DMA controller must be integrated into the system, and the system must allow the DMA controller to be a bus master. Cycle stealing may also be necessary to allow the CPU and DMA controller to share use of the memory bus.

**12.6** Why is it important to scale up system-bus and device speeds as CPU speed increases?

**Answer:**

Consider a system which performs 50% I/O and 50% computes. Doubling the CPU performance on this system would increase total system performance by only 50%. Doubling both system aspects would increase performance by 100%. Generally, it is important to remove the current system bottleneck, and to increase overall system performance, rather than blindly increasing the performance of individual system components.

**12.7** Distinguish between a driver end and a stream module in a STREAMS operation.

**Answer:**

The STREAMS driver controls a physical device that could be involved in a STREAMS operation. The STREAMS module modifies the flow of data between the head (the user interface) and the driver.

**Chapter 12 Exercises**

**12.8** 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.

*When creating the priority plan, several factors must be taken. Device interrupts should be prioritized above* *traps created by user applications, as they might interfere with system call handling code. Second, because such operations can always be postponed, interrupts that merely perform tasks, such as transferring data served up by a device to user/kernel buffers, may be prioritized lower than interrupts that control devices. Third, devices that must deal with data in real time should take precedence over others. Additionally, since the data would only be available for a few times, devices without any type of data buffering would need to be given higher priority.*

**12.9** What are the advantages and disadvantages of supporting memorymapped I/O to device-control registers?

*It is useful to provide memory-mapped I/O to device control registers because it reduces the need for specialist I/O instructions in the instruction set, which also means that protection rules preventing user applications from using these I/O instructions are not necessary. The drawback of this increased flexibility is that it needs to be handled carefully; for security purposes, memory translation units must make sure that user applications cannot access memory regions linked to device control registers.*

**12.10** 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 (with no device preallocation 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 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.

*Ans:*

*a) Although buffering maybe used to record mouse movements during times when high priority operations are taking place, it is relatively infrequent and typically need to be handles immediately to ensure responsiveness. Caching or spooling is not applicable. Hence, the best fit for this scenario is interrupt-driven I/O as it is more efficient and responsive. Typically, the CPU doesn't have to continuously check the status of the mouse but generates an interrupt signal to the CPU when it has an event to be processed.*

*b) Spooling can be used to transfer data to a device when multiple users wish to read or write data. It is being used for tape driver as it allows the system to queue up jobs for the tape drive to handle in sequence, hence optimising the usage of the tape drive with slow I/O operations. Caching isn't used very much with tape drives, as they are sequential access devices. Polled I/O, on the other hand, could result in the CPU wasting cycles checking the status of the tape drive. However, interrupt-driven I/O is best preferred as it allows the CPU to perform other tasks while the tape is busy.*

*c) Disk drives typically use interrupt-driven I/O, allowing the CPU to perform other tasks while the disk drive is busy. It interrupts the CPU when the drive completes a read or write, to process the data. Buffering can also be used as it helps in managing data flow or to hold data when transferring from user space to disk and vice versa. Caching is also commonly used to save disk- resident data to improve performance. Spooling isn't used for disk drives since it is more associated with sequential access device as opposed to disk drive with random access devices.*

*d) In this scenario, with graphics card having direct connection to the CPU, spooling isn't usually used as it doesn't need devices to queue up jobs to be processed in sequence. Caching could be used to store frequently accessed texture or other graphical data for quicker access. This helps in the performance. Buffering can be used to handle large amounts of data for graphics processing. This greatly help in smoothing out data flow and managing the workload of the graphics card. Typically, this scenario uses a combination of polled and interrupt-driven I/O.Interrupt-driven I/O can be used for tasks that are not time-critical and where we don't want the CPU to continuously check the status of the device. Polled I/O, on the other hand, can be used for time- critical tasks where the overhead of handling interrupts could cause performance issues.*

**12.11** 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 for the initiation of I/O operations by the user program and their execution by the operating system?

*A buffer is often specified by the user software for data being transmitted to or from a device. This buffer is identified by a virtual address and is located in user space. The I/O operation must be initiated by the kernel, and either before or after the operation, data must be copied from the user buffer to the kernel buffer. The kernel must convert the virtual address provided by the user program to the equivalent physical address in the context of the user program's virtual address space in order to access the user buffer. Typically, software is used to carry out this translation, adding overhead. Additionally, if the user buffer isn't already in physical memory, the necessary page(s) must be obtained from the swap space. The data copy operation may need to be handled carefully, which could cause a delay.*

**12.12** What are the various kinds of performance overhead associated with servicing an interrupt?

Servicing an interrupt involves several steps that can introduce performance overhead. Here are some of the main types of overhead associated with servicing an interrupt:

1. **Context Switching Overhead**: When an interrupt occurs, the CPU needs to save the current process state, including the program counter, register contents, and other relevant information before it can start executing the interrupt service routine[1](https://hsi.web.cern.ch/dshs/publications/wotug21/dsnic/html/node22.html). Once the interrupt has been handled, the original process state must be restored[1](https://hsi.web.cern.ch/dshs/publications/wotug21/dsnic/html/node22.html). This process of saving and restoring the process state is known as context switching and can introduce significant overhead[1](https://hsi.web.cern.ch/dshs/publications/wotug21/dsnic/html/node22.html).
2. **Instruction Pipeline Flushing**: Modern CPUs use a technique called pipelining, where multiple instructions are processed simultaneously at different stages of execution. When an interrupt occurs, the CPU may need to flush (discard) the contents of the instruction pipeline before it can start executing the interrupt service routine[2](https://www.quesba.com/questions/various-kinds-performance-overhead-associated-servicing-interrupt-12-13-424936)[3](https://www.transtutors.com/questions/what-are-the-various-kinds-of-performance-overhead-associated-with-servicing-an-inte-3970519.htm). After the interrupt has been handled, the pipeline must be refilled, which can take several clock cycles[2](https://www.quesba.com/questions/various-kinds-performance-overhead-associated-servicing-interrupt-12-13-424936)[3](https://www.transtutors.com/questions/what-are-the-various-kinds-of-performance-overhead-associated-with-servicing-an-inte-3970519.htm).
3. **Interrupt Handling Routine Execution**: The execution of the interrupt handling routine itself also introduces overhead. This includes the time taken to execute the instructions in the interrupt service routine and any additional operations required to handle the interrupt[2](https://www.quesba.com/questions/various-kinds-performance-overhead-associated-servicing-interrupt-12-13-424936)[3](https://www.transtutors.com/questions/what-are-the-various-kinds-of-performance-overhead-associated-with-servicing-an-inte-3970519.htm).
4. **Increased Memory Access Latency**: If the interrupt service routine needs to access data that is not currently in the CPU’s cache, this can lead to cache misses, which increase memory access latency[4](https://zhuanlan.zhihu.com/p/424679602).
5. **Nested Interrupts**: If interrupts can be nested (i.e., an interrupt can occur while another interrupt is being serviced), this can introduce additional overhead due to the increased complexity of managing multiple interrupt service routines[5](https://www2.keil.com/docs/default-source/default-document-library/01_shore_arm.pdf).

Please note that the exact overhead can vary depending on the specific hardware and software configuration of the system. For example, some systems may use techniques such as interrupt coalescing or prioritized interrupt handling to reduce the overhead associated with servicing interrupts[4](https://zhuanlan.zhihu.com/p/424679602)[5](https://www2.keil.com/docs/default-source/default-document-library/01_shore_arm.pdf).

**12.13** 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 devices are ready?

***Answer:***

*Generally, blocking I/O is appropriate when the process will only be waiting for one spe-cific 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 asynchronous 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.*

**12.14** 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. The first piece 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?

This **strategy** of using a two-part design in interrupt handlers, with an immediate **execution** and a deferred procedure call (DPC), serves a few key purposes:

**The Purposes**

**Responsiveness**: The immediate execution part ensures quick response to critical **I/O operations,**reducing latency. This is especially important in real-time systems or when dealing with time-sensitive tasks.

**Minimizing Blocking:** Lengthy operations or resource-intensive tasks can block the main processing thread if executed immediately. Deferring less critical tasks to the DPC helps prevent such blocking.

**Priority Management:**By separating immediate and deferred tasks, priority levels can be assigned. Immediate tasks may run at a higher priority, ensuring time-critical actions are handled promptly.

**Efficiency**: The strategy optimizes resource usage, as not all tasks require immediate attention. It balances responsiveness with efficient system resource allocation, resulting in more predictable and responsive system behavior.

**12.15** 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 functionality?

Direct Virtual Memory Access (DVMA) allows I/O operations to be specified as virtual addresses, with the DMA controller performing the translation from virtual to physical addresses. This design introduces several complexities but also offers certain advantages.

**Complexities:**

1. **Address Translation**: The DMA controller needs to be capable of performing address translation from virtual to physical addresses[1](https://www.geeksforgeeks.org/direct-memory-access/). This requires the DMA controller to have access to the same page tables used by the CPU, or a similar mechanism for address translation[1](https://www.geeksforgeeks.org/direct-memory-access/).
2. **Memory Management**: The DMA controller needs to be aware of the memory management policies of the operating system, including page replacement algorithms and handling of page faults[1](https://www.geeksforgeeks.org/direct-memory-access/). This can significantly increase the complexity of the DMA controller design[1](https://www.geeksforgeeks.org/direct-memory-access/).
3. **Synchronization**: The DMA controller needs to ensure that it does not access a page that is being swapped out by the CPU, or vice versa[1](https://www.geeksforgeeks.org/direct-memory-access/). This requires careful synchronization between the CPU and the DMA controller[1](https://www.geeksforgeeks.org/direct-memory-access/).

**Advantages:**

1. **Efficiency**: DVMA can reduce CPU overhead by allowing data to be transferred directly between memory-mapped I/O and the application’s virtual address space, without the need for data to be copied between kernel and user space[1](https://www.geeksforgeeks.org/direct-memory-access/)[2](https://www.knowcomputing.com/direct-memory-access-dma-in-operating-system/).
2. **Flexibility**: With DVMA, I/O operations can be performed on data wherever it resides in the virtual memory, without the need for data to be contiguous or aligned on page boundaries[1](https://www.geeksforgeeks.org/direct-memory-access/).
3. **Performance**: By reducing the involvement of the CPU in data transfers, DVMA can lead to improved system performance[2](https://www.knowcomputing.com/direct-memory-access-dma-in-operating-system/). The CPU is free to perform other tasks while the data transfer is being handled by the DMA controller[2](https://www.knowcomputing.com/direct-memory-access-dma-in-operating-system/).

Please note that the exact benefits and complexities can vary depending on the specific hardware and software configuration of the system.

**12.16** UNIX coordinates the activities of the kernel I/O components by manipulating shared in-kernel data structures, whereas Windows uses object oriented message passing between kernel I/O components. Discuss three pros and three cons of each approach.

**12.17** Write (in pseudocode) an implementation of virtual clocks, including the queueing and management of timer requests for the kernel and applications. Assume that the hardware provides three timer channels.

**12.18** Discuss the advantages and disadvantages of guaranteeing reliable transfer of data between modules in the STREAMS abstraction.