Practice Exercises

7.1 List three examples of deadlocks that are not related to a computer-system environment.

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

7.3 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 $A \cdots E$, deadlock is possible. (Such synchronization objects may include mutexes, semaphores, condition variables, and the like.) We can prevent the deadlock by adding a sixth object $F$. Whenever a thread wants to acquire the synchronization lock for any object $A \cdots E$, it must first acquire the lock for object $F$. This solution is known as containment: the locks for objects $A \cdots E$ are contained within the lock for object $F$. Compare this scheme with the circular-wait scheme of Section 7.4.4.

7.4 Prove that the safety algorithm presented in Section 7.5.3 requires an order of $m \times n^2$ operations.

7.5 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 10 jobs per deadlock. Each job is worth about $2 (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 in the average execution time per job of about 10 percent. 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?

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

7.7 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 processes that are blocked waiting for resources. If a blocked process has the desired resources, then these resources are taken away from it and are given to the requesting process. The vector of resources for which the blocked process is waiting is increased to include the resources that were taken away.
For example, consider a system with three resource types and the vector $Available$ initialized to $(4,2,2)$. If process $P_0$ asks for $(2,2,1)$, it gets them. If $P_1$ asks for $(1,0,1)$, it gets them. Then, if $P_0$ asks for $(0,0,1)$, it is blocked (resource not available). If $P_2$ now asks for $(2,0,0)$, it gets the available one $(1,0,0)$ and one that was allocated to $P_0$ (since $P_0$ is blocked). $P_0$'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.

7.8 Suppose that you have coded the deadlock-avoidance safety algorithm and now have been asked to implement the deadlock-detection algorithm. Can you do so by simply using the safety algorithm code and redefining $Max_j = Waiting_i + Allocation_j$, where $Waiting_i$ is a vector specifying the resources for which process $i$ is waiting and $Allocation_j$ is as defined in Section 7.5? Explain your answer.

7.9 Is it possible to have a deadlock involving only a single process? Explain your answer.

Exercises

7.10 Consider the traffic deadlock depicted in Figure 7.9.

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

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

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

7.12 In Section 7.4.4, we describe 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.

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

a. Runtime overheads

b. System throughput

7.14 In a real computer system, neither the resources available nor the demands of processes for resources are consistent over long periods
Resources break or are replaced, new processes 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. Increase *Available* (new resources added).

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

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

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

e. Increase the number of processes.

f. Decrease the number of processes.

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

7.16 Consider a system consisting of *m* resources of the same type being shared by *n* processes. A process 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 process is between one resource and *m* resources.

b. The sum of all maximum needs is less than *m* + *n*.
7.17 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.

7.18 Consider again the setting in the preceding question. 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.

7.19 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 multiple-resource-type banker’s scheme by applying the single-resource-type scheme to each resource type individually.

7.20 Consider the following snapshot of a system:

<table>
<thead>
<tr>
<th>Allocation</th>
<th>Max</th>
<th>Available</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B C D</td>
<td>A B C D</td>
<td>A B C D</td>
</tr>
<tr>
<td>P₀ 0 0 1 2</td>
<td>0 0 1 2</td>
<td>1 5 2 0</td>
</tr>
<tr>
<td>P₁ 1 0 0 0</td>
<td>1 7 5 0</td>
<td></td>
</tr>
<tr>
<td>P₂ 1 3 5 4</td>
<td>2 3 5 6</td>
<td></td>
</tr>
<tr>
<td>P₃ 0 6 3 2</td>
<td>0 6 5 2</td>
<td></td>
</tr>
<tr>
<td>P₄ 0 0 1 4</td>
<td>0 6 5 6</td>
<td></td>
</tr>
</tbody>
</table>

Answer the following questions using the banker’s algorithm:

a. What is the content of the matrix Need?

b. Is the system in a safe state?

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

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

7.22 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, design an algorithm that prevents deadlock. Initially, do not be concerned about starvation (the situation in which northbound farmers prevent southbound farmers from using the bridge, or vice versa).

7.23 Modify your solution to Exercise 7.22 so that it is starvation-free.
in memory, however, user memory accesses can be degraded substantially. A TLB can reduce the performance degradation to an acceptable level.

- **Fragmentation.** A multiprogrammed system will generally perform more efficiently if it has a higher level of multiprogramming. For a given set of processes, we can increase the multiprogramming level only by packing more processes into memory. To accomplish this task, we must reduce memory waste, or fragmentation. Systems with fixed-sized allocation units, such as the single-partition scheme and paging, suffer from internal fragmentation. Systems with variable-sized allocation units, such as the multiple-partition scheme and segmentation, suffer from external fragmentation.

- **Relocation.** One solution to the external-fragmentation problem is compaction. Compaction involves shifting a program in memory in such a way that the program does not notice the change. This consideration requires that logical addresses be relocated dynamically, at execution time. If addresses are relocated only at load time, we cannot compact storage.

- **Swapping.** Swapping can be added to any algorithm. At intervals determined by the operating system, usually dictated by CPU-scheduling policies, processes are copied from main memory to a backing store and later are copied back to main memory. This scheme allows more processes to be run than can be fit into memory at one time.

- **Sharing.** Another means of increasing the multiprogramming level is to share code and data among different users. Sharing generally requires that either paging or segmentation be used to provide small packets of information (pages or segments) that can be shared. Sharing is a means of running many processes with a limited amount of memory, but shared programs and data must be designed carefully.

- **Protection.** If paging or segmentation is provided, different sections of a user program can be declared execute-only, read-only, or read-write. This restriction is necessary with shared code or data and is generally useful in any case to provide simple run-time checks for common programming errors.

### Practice Exercises

8.1 Name two differences between logical and physical addresses.

8.2 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.

8.3 Why are page sizes always powers of 2?
8.4 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?

8.5 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 the one page have on the other page?

8.6 Describe a mechanism by which one segment could belong to the address space of two different processes.

8.7 Sharing segments among processes without requiring that they have the same segment number is possible in a dynamically linked segmentation system.
   a. Define a system that allows static linking and sharing of segments without requiring that the segment numbers be the same.
   b. Describe a paging scheme that allows pages to be shared without requiring that the page numbers be the same.

8.8 In the IBM/370, memory protection is provided through the use of keys. A key is a 4-bit quantity. Each 2-K block of memory has a key (the storage key) associated with it. The CPU also has a key (the protection key) associated with it. A store operation is allowed only if both keys are equal or if either is zero. Which of the following memory-management schemes could be used successfully with this hardware?
   a. Bare machine
   b. Single-user system
   c. Multiprogramming with a fixed number of processes
   d. Multiprogramming with a variable number of processes
   e. Paging
   f. Segmentation

8.9 Explain the difference between internal and external fragmentation.

8.10 Consider the following process for generating binaries. A compiler is used to generate the object code for individual modules, and a linkage editor is used to combine multiple object modules into a single program binary. How does the linkage editor change the binding of instructions and data to memory addresses? What information needs to be passed from the compiler to the linkage editor to facilitate the memory-binding tasks of the linkage editor?
8.11 Given five memory partitions of 100 KB, 500 KB, 200 KB, 300 KB, and 600 KB (in order), how would the first-fit, best-fit, and worst-fit algorithms place processes of 212 KB, 417 KB, 112 KB, and 426 KB (in order)? Which algorithm makes the most efficient use of memory?

8.12 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. Pure segmentation
   c. Pure paging

8.13 Compare the memory organization schemes of contiguous memory allocation, pure segmentation, and pure paging with respect to the following issues:
   a. External fragmentation
   b. Internal fragmentation
   c. Ability to share code across processes

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

8.15 Compare paging with segmentation with respect to the amount of memory required by the address translation structures in order to convert virtual addresses to physical addresses.

8.16 Program binaries in many systems are typically structured as follows. Code is stored starting with a small, fixed virtual address, such as 0. The code segment is followed by the data segment that is used for storing the program variables. When the program starts executing, the stack is allocated at the other end of the virtual address space and is allowed to grow toward lower virtual addresses. What is the significance of this structure for the following schemes?
   a. Contiguous memory allocation
   b. Pure segmentation
   c. Pure paging

8.17 Assuming a 1-KB page size, what are the page numbers and offsets for the following address references (provided as decimal numbers):
   a. 2375
   b. 19366
   c. 30000
   d. 256
   e. 16385
8.18 Consider a logical address space of 32 pages with 1,024 words per page, mapped onto a physical memory of 16 frames.
   a. How many bits are required in the logical address?
   b. How many bits are required in the physical address?

8.19 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

8.20 Consider a paging system with the page table stored in memory.
   a. If a memory reference takes 200 nanoseconds, how long does a paged memory reference take?
   b. If we add TLBs, and 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 zero time, if the entry is there.)

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

8.22 Explain why sharing a reentrant module is easier when segmentation is used than when pure paging is used.

8.23 Consider the following segment table:

<table>
<thead>
<tr>
<th>Segment</th>
<th>Base</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>219</td>
<td>600</td>
</tr>
<tr>
<td>1</td>
<td>2300</td>
<td>14</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>90</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1327</td>
<td>580</td>
</tr>
<tr>
<td>4</td>
<td>1952</td>
<td>96</td>
</tr>
</tbody>
</table>

What are the physical addresses for the following logical addresses?
   a. 0,430
   b. 1,10
   c. 2,500
   d. 3,400
   e. 4,112

8.24 What is the purpose of paging the page tables?

8.25 Consider the hierarchical paging scheme used by the VAX architecture. How many memory operations are performed when a user program executes a memory-load operation?
8.26 Compare the segmented paging scheme with the hashed page table scheme for handling large address spaces. Under what circumstances is one scheme preferable to the other?

8.27 Consider the Intel address-translation scheme shown in Figure 8.22.
   a. Describe all the steps taken by the Intel Pentium 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?

Programming Problems

8.28 Assume that a system has a 32-bit virtual address with a 4-KB page size. Write a C program that is passed a virtual address (in decimal) on the command line and have it output the page number and offset for the given address. As an example, your program would run as follows:

```
./a.out 19986
```

Your program would output:

```
The address 19986 contains:
  page number = 4
  offset = 3602
```

Writing this program will require using the appropriate data type to store 32 bits. We encourage you to use unsigned data types as well.

Wiley Plus

Visit Wiley Plus for
- Source code
- Solutions to practice exercises
- Additional programming problems and exercises
- Labs using an operating system simulator

Bibliographical Notes

Dynamic storage allocation was discussed by Knuth [1973] (Section 2.5), who found through simulation results that first fit is generally superior to best fit. Knuth [1973] also discussed the 50-percent rule.

The concept of paging can be credited to the designers of the Atlas system, which has been described by Kilburn et al. [1961] and by Howarth et al.
In addition to requiring that we solve the major problems of page replacement and frame allocation, the proper design of a paging system requires that we consider prepaging, page size, TLB reach, inverted page tables, program structure, I/O interlock, and other issues.

Practice Exercises

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

9.2 Assume that you have a page-reference string for a process with \( m \) frames (initially all empty). The page-reference string has length \( p \); \( n \) 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?

9.3 Which of the following programming techniques and structures are "good" for a demand-paged environment? Which are "not good"? Explain your answers.
   a. Stack
   b. Hashed symbol table
   c. Sequential search
   d. Binary search
   e. Pure code
   f. Vector operations
   g. Indirection

9.4 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

9.5 When virtual memory is implemented in a computing system, there are certain costs associated with the technique and certain benefits. List the costs and the benefits. Is it possible for the costs to exceed the benefits? If it is, what measures can be taken to ensure that this does not happen?

9.6 An operating system supports a paged virtual memory, using a central processor with 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.

9.7 Consider the two-dimensional array \( A \):

\[
\text{int } A[][] = \text{new int[100][100]};
\]

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, using LRU replacement and assuming 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 \( j = 0; j < 100; j++ \))
   \( A[i][j] = 0; \)

9.8 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

9.9 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.

9.10 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.

9.11 Segmentation is similar to paging but uses variable-sized “pages.” Define two segment-replacement algorithms based on FIFO and LRU page-replacement schemes. Remember that since segments are not the same size, the segment that is chosen to be replaced may not be big enough 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.

9.12 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. The results are one of the following alternatives. 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

9.13 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 tables be set up to simulate base and limit registers? How can they be, or why can they not be?

Exercises

9.14 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 and page fault
   • TLB hit and no page fault
   • TLB hit and page fault

9.15 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 (i.e. is waiting for I/O.) This is illustrated in Figure 9.31. Assuming a thread is in the Running state, answer the following questions: (Be sure to explain your answer.)
a. Will the thread change state if it incurs a page fault? If so, to what new state?

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

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

9.16 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.

9.17 Give an example that illustrates the problem with restarting the move character instruction (MVC) on the IBM 360/370 when the source and destination regions are overlapping.

9.18 Discuss the hardware support required to support demand paging.

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

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

9.21 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?

9.22 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.

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

<table>
<thead>
<tr>
<th>Page</th>
<th>Page Frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>–</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>C</td>
</tr>
<tr>
<td>3</td>
<td>A</td>
</tr>
<tr>
<td>4</td>
<td>–</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>–</td>
</tr>
<tr>
<td>8</td>
<td>B</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
</tr>
</tbody>
</table>

Convert the following virtual addresses to their equivalent physical addresses in hexadecimal. All numbers are given in hexadecimal. (A dash for a page frame indicates that the page is not in memory.)

- 9EF
- 111
- 700
- 0FF

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

a. Pointer is moving fast.

b. Pointer is moving slow.
9.25 Discuss situations in which the least frequently used page-replacement algorithm generates fewer page faults than the least recently used page-replacement algorithm. Also discuss under what circumstances the opposite holds.

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

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

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

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

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

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

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

<table>
<thead>
<tr>
<th>Utilization</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU utilization</td>
<td>20%</td>
</tr>
<tr>
<td>Paging disk</td>
<td>97.7%</td>
</tr>
<tr>
<td>Other I/O devices</td>
<td>5%</td>
</tr>
</tbody>
</table>

For each of the following, say 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.

9.29 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?

9.30 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?

9.31 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:
      i. What is the initial value of the counters?
      ii. When are counters increased?
      iii. When are counters decreased?
      iv. 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 page-replacement strategy for the reference string in part b with four page frames?

9.32 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?

9.33 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?

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

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

9.36 Assume there is a 1,024-KB segment where memory is allocated using the buddy system. Using Figure 9.27 as a guide, draw a tree illustrating how the following memory requests are allocated:

- Request 240 bytes
- Request 120 bytes
- Request 60 bytes
- Request 130 bytes

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

- Release 240 bytes
- Release 60 bytes
- Release 120 bytes

9.37 Consider a system that 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

9.38 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?

9.39 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 provide this functionality?

### Programming Problems

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

9.41 The Catalan numbers are an integer sequence \( C_n \) that appear in tree-enumeration problems. The first Catalan numbers for \( n = 1, 2, 3, \ldots \) are 1, 2, 5, 14, 42, 132, \ldots . A formula generating \( C_n \) is