that enables us to map a large logical address space onto a smaller physical memory. Virtual memory allows us to run extremely large processes and to raise the degree of multiprogramming, increasing CPU utilization. Further, it frees application programmers from worrying about memory availability. In addition, with virtual memory, several processes can share system libraries and memory. Virtual memory also enables us to use an efficient type of process creation known as copy-on-write, wherein parent and child processes share actual pages of memory.

Virtual memory is commonly implemented by demand paging. Pure demand paging never brings in a page until that page is referenced. The first reference causes a page fault to the operating system. The operating-system kernel consults an internal table to determine where the page is located on the backing store. It then finds a free frame and reads the page in from the backing store. The page table is updated to reflect this change, and the instruction that caused the page fault is restarted. This approach allows a process to run even though its entire memory image is not in main memory at once. As long as the page-fault rate is reasonably low, performance is acceptable.

We can use demand paging to reduce the number of frames allocated to a process. This arrangement can increase the degree of multiprogramming (allowing more processes to be available for execution at one time) and—in theory, at least—the CPU utilization of the system. It also allows processes to be run even though their memory requirements exceed the total available physical memory. Such processes run in virtual memory.

If total memory requirements exceed the capacity of physical memory, then it may be necessary to replace pages from memory to free frames for new pages. Various page-replacement algorithms are used. FIFO page replacement is easy to program but suffers from Belady’s anomaly. Optimal page replacement requires future knowledge. LRU replacement is an approximation of optimal page replacement, but even it may be difficult to implement. Most page-replacement algorithms, such as the second-chance algorithm, are approximations of LRU replacement. In addition to a page-replacement algorithm, a frame-allocation policy is needed. Allocation can be fixed, suggesting local page replacement; or dynamic, suggesting global replacement. The working-set model assumes that processes execute in localities. The working set is the set of pages in the current locality. Accordingly, each process should be allocated enough frames for its current working set. If a process does not have enough memory for its working set, it will thrash. Providing enough frames to each process to avoid thrashing may require process swapping and scheduling.

Most operating systems provide features for memory mapping files, thus allowing file I/O to be treated as routine memory access. The WiredAP implements shared memory through memory mapping files.

Kernel processes typically require memory to be allocated using pages that are physically contiguous. The buddy system allocates memory to kernel processes in units sized according to a power of 2, which often results in fragmentation. Slab allocators assign kernel data structures to caches associated with slabs, which are made up of one or more physically contiguous pages. With slab allocation, no memory is wasted due to fragmentation, and memory requests can be satisfied quickly.

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 page mapping, 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 \( p \); \( p \) 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 these 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 $A$:

```java
int A[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 page 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 contrasted 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., 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.)
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 forresident 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 ofresident 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 pre-aging 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 to 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 processors. 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 9. Assume that demand paging is used.

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

\[
C_n = \frac{1}{(n + 1)} \binom{2n}{n} = \frac{(2n)!}{n!(n + 1)!}
\]

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

In this instance, the producer process will be passed an integer parameter on the command line specifying how many Catalan numbers to produce (for example, providing 5 on the command line means the producer process will generate the first five Catalan numbers).

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

Demand paging was first used in the Atlas system, implemented on the Manchester University MUSE computer around 1960 (Kilburn et al. [1961]). Another early demand-paging system was MULTICS, implemented on the GE 645 system (Organick [1972]).

Belady et al. [1969] were the first researchers to observe that the FIFO replacement strategy may produce the anomaly that bears Belady’s name. Mottow et al. [1970] demonstrated that stack algorithms are not subject to Belady’s anomaly.

The optimal replacement algorithm was presented by Belady [1966] and was proved to be optimal by Mattson et al. [1970]. Belady’s optimal algorithm is for a fixed allocation; Prieve and Fabry [1976] presented an optimal algorithm for situations in which the allocation can vary.

The enhanced clock algorithm was discussed by Carr and Herrosey [1981].

The working-set model was developed by Denning [1968]. Discussions concerning the working-set model were presented by Denning [1980].

The scheme for monitoring the page-fault rate was developed by Wulf [1969], who successfully applied this technique to the Burroughs 8900 computer system.