8.9 Summary

Memory-management algorithms for multiprogrammed operating systems range from the simple single-user system approach to segmentation and paging. The most important determinant of the method used in a particular system is the hardware provided. Every memory address generated by the CPU must be checked for legality and possibly mapped to a physical address. The checking cannot be implemented (efficiently) in software. Hence, we are constrained by the hardware available.

The various memory-management algorithms (contiguous allocation, paging, segmentation, and combinations of paging and segmentation) differ in many aspects. In comparing different memory-management strategies, we use the following considerations:

- **Hardware support.** A simple base register or a base–limit register pair is sufficient for the single- and multiple-partition schemes, whereas paging and segmentation need mapping tables to define the address map.
- **Performance.** As the memory-management algorithm becomes more complex, the time required to map a logical address to a physical address increases. For the simple systems, we need only compare or add to the logical address—operations that are fast. Paging and segmentation can be as fast if the mapping table is implemented in fast registers. If the table is 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. In general, PC operating systems support paging, and operating systems for mobile devices do not.
- **Sharing.** Another means of increasing the multiprogramming level is to share code and data among different processes. 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.

Exercises

8.1 Explain the difference between internal and external fragmentation.

8.2 Consider the following process for generating binaries. A compiler is used to generate the object code for individual modules, and a linkage editor is used to combine multiple object modules into a single program binary. How does the linkage editor change the binding of instructions and data to memory addresses? What information needs to be passed from the compiler to the linkage editor to facilitate the memory-binding tasks of the linkage editor?

8.3 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)? Rank the algorithms in terms of how efficiently they use memory.

8.4 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

8.5 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

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

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

8.8 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?

8.9 Compare paging with segmentation with respect to how much memory the address translation structures require to convert virtual addresses to physical addresses.

8.10 Explain why address space identifiers (ASIDs) are used.

8.11 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

8.12 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

8.13 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

8.14 What is the maximum amount of physical memory?

8.15 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?

8.16 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

8.17 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 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.)

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

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

8.20 Consider the following segment table:

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

What are the physical addresses for the following logical addresses?

a. 0,430

b. 1,10

c. 2,500

d. 3,400

e. 4,112

8.21 What is the purpose of paging the page tables?

8.22 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?

8.23 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?

8.24 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

8.25 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.

Bibliographical Notes

Dynamic storage allocation was discussed by [Knuth (1973)] (Section 2.5), who found through simulation 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. (1961)]. The concept of segmentation was first discussed by [Dennis (1965)]. Paged segmentation was first supported in the GE 645, on which MULTICS was originally implemented ([Organick (1972)] and [Daley and Dennis (1967)]).

Inverted page tables are discussed in an article about the IBM RT storage manager by [Chang and Mergen (1988)].

[Hennessy and Patterson (2012)] explains the hardware aspects of TLBs, caches, and MMUs. [Talluri et al. (1995)] discusses page tables for 64-bit address spaces. [Jacob and Mudge (2001)] describes techniques for managing the TLB. [Fang et al. (2001)] evaluates support for large pages.

http://msdn.microsoft.com/en-us/library/windows/hardware/gg487512.aspx discusses PAE support for Windows systems.

http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html provides various manuals for Intel 64 and IA-32 architectures.

http://www.arm.com/products/processors/cortex-a/cortex-a9.php provides an overview of the ARM architecture.

Bibliography

[Chang and Mergen (1988)] A. Chang and M. F. Mergen, "801 Storage: Architecture and Programming", *ACM Transactions on Computer Systems*, Volume 6, Number 1 (1988), pages 28–50.

[Daley and Dennis (1967)] R. C. Daley and J. B. Dennis, "Virtual Memory, Processes, and Sharing in Multics", *Proceedings of the ACM Symposium on Operating Systems Principles* (1967), pages 121–128.

[Dennis (1965)] J. B. Dennis, "Segmentation and the Design of Multiprogrammed Computer Systems", *Communications of the ACM*, Volume 8, Number 4 (1965), pages 589–602.

[Fang et al. (2001)] Z. Fang, L. Zhang, J. B. Carter, W. C. Hsieh, and S. A. McKee, "Reevaluating Online Superpage Promotion with Hardware Support", *Proceedings of the International Symposium on High-Performance Computer Architecture*, Volume 50, Number 5 (2001).

[Hennessy and Patterson (2012)] J. Hennessy and D. Patterson, *Computer Architecture: A Quantitative Approach*, Fifth Edition, Morgan Kaufmann (2012).

[Howarth et al. (1961)] D. J. Howarth, R. B. Payne, and F. H. Sumner, "The Manchester University Atlas Operating System, Part II: User's Description", *Computer Journal*, Volume 4, Number 3 (1961), pages 226–229.

[Jacob and Mudge (2001)] B. Jacob and T. Mudge, "Uniprocessor Virtual Memory Without TLBs", *IEEE Transactions on Computers*, Volume 50, Number 5 (2001).

[Kilburn et al. (1961)] T. Kilburn, D. J. Howarth, R. B. Payne, and F. H. Sumner, "The Manchester University Atlas Operating System, Part I: Internal Organization", *Computer Journal*, Volume 4, Number 3 (1961), pages 222–225.

[Knuth (1973)] D. E. Knuth, *The Art of Computer Programming, Volume 1: Fundamental Algorithms*, Second Edition, Addison-Wesley (1973).

[Organick (1972)] E. I. Organick, *The Multics System: An Examination of Its Structure*, MIT Press (1972).

[Talluri et al. (1995)] M. Talluri, M. D. Hill, and Y. A. Khalidi, "A New Page Table for 64-bit Address Spaces", *Proceedings of the ACM Symposium on Operating Systems Principles* (1995), pages 184–200.

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 us to 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 page locking, and other issues.

Exercises

9.1 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

9.2 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). This is illustrated in Figure 9.30. Assuming a thread is in the Running state, answer the following questions, and explain your answer:

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?

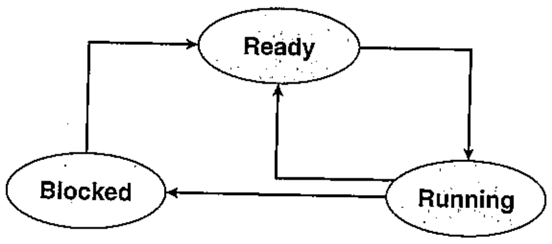

Figure 9.30 Thread state diagram for Exercise 9.2.

9.3 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.

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

9.5 A certain computer provides its users with a virtual memory space of 2^{32} bytes. The computer has 2^{22} 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.

9.6 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?

9.7 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 one 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.

9.8 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

9.9 The page table shown in Figure 9.31 is for a system with 16-bit virtual and physical addresses and with 4,096-byte pages. The reference bit is

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

Figure 9.31 Page table for Exercise 9.9.

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 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.

- `0xE12C`
- `0x3A9D`
- `0xA9D9`
- `0x7001`
- `0xACA1`

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?

9.10 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.

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

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

9.13 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 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 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?

9.14 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.

9.15 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?

9.16 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?

9.17 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?

9.18 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?

9.19 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?

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

9.21 Consider the parameter Δ used to define the working-set window in the working-set model. When Δ is set to a small 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?

9.22 In a 1,024-KB segment, memory is allocated using the buddy system. Using Figure 9.26 as a guide, draw a tree illustrating how the following memory requests are allocated:

- Request 6-KB
- Request 250 bytes
- Request 900 bytes
- Request 1,500 bytes
- Request 7-KB

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

- Release 250 bytes
- Release 900 bytes
- Release 1,500 bytes

9.23 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

9.24 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?

9.25 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

9.26 Write a program that implements the FIFO, LRU, and optimal 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.

9.27 Repeat Exercise 3.22, this time using Windows shared memory. In particular, using the producer—consumer strategy, design two programs that communicate with shared memory using the Windows API as outlined in Section 9.7.2. The producer will generate the numbers specified in the Collatz conjecture and write them to a shared memory object

Network file systems, such as NFS, use client–server methodology to allow users to access files and directories from remote machines as if they were on local file systems. System calls on the client are translated into network protocols and retranslated into file-system operations on the server. Networking and multiple-client access create challenges in the areas of data consistency and performance.

Due to the fundamental role that file systems play in system operation, their performance and reliability are crucial. Techniques such as log structures and caching help improve performance, while log structures and RAID improve reliability. The WAFL file system is an example of optimization of performance to match a specific I/O load.

Exercises

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

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

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

c. Extents can be of a few fixed sizes, and these sizes are predetermined.

11.2 Contrast the performance of the three techniques for allocating disk blocks (contiguous, linked, and indexed) for both sequential and random file access.

11.3 What are the advantages of the variant of linked allocation that uses a FAT to chain together the blocks of a file?

11.4 Consider a system where free space is kept in a free-space list.

a. Suppose that the pointer to the free-space list is lost. Can the system reconstruct the free-space list? Explain your answer.

b. Consider a file system similar to the one used by UNIX with indexed allocation. How many disk I/O operations might be required to read the contents of a small local file at /a/b/c? Assume that none of the disk blocks is currently being cached.

c. Suggest a scheme to ensure that the pointer is never lost as a result of memory failure.

11.5 Some file systems allow disk storage to be allocated at different levels of granularity. For instance, a file system could allocate 4 KB of disk space as a single 4-KB block or as eight 512-byte blocks. How could we take advantage of this flexibility to improve performance? What modifications would have to be made to the free-space management scheme in order to support this feature?

11.6 Discuss how performance optimizations for file systems might result in difficulties in maintaining the consistency of the systems in the event of computer crashes.

11.7 Consider a file system on a disk that has both logical and physical block sizes of 512 bytes. Assume that the information about each file is already in memory. For each of the three allocation strategies (contiguous, linked, and indexed), answer these questions:

a. How is the logical-to-physical address mapping accomplished in this system? (For the indexed allocation, assume that a file is always less than 512 blocks long.)

b. If we are currently at logical block 10 (the last block accessed was block 10) and want to access logical block 4, how many physical blocks must be read from the disk?

11.8 Consider a file system that uses inodes to represent files. Disk blocks are 8 KB in size, and a pointer to a disk block requires 4 bytes. This file system has 12 direct disk blocks, as well as single, double, and triple indirect disk blocks. What is the maximum size of a file that can be stored in this file system?

11.9 Fragmentation on a storage device can be eliminated by recompaction of the information. Typical disk devices do not have relocation or base registers (such as those used when memory is to be compacted), so how can we relocate files? Give three reasons why recompacting and relocation of files are often avoided.

11.10 Assume that in a particular augmentation of a remote-file-access protocol, each client maintains a name cache that caches translations from file names to corresponding file handles. What issues should we take into account in implementing the name cache?

11.11 Explain why logging metadata updates ensures recovery of a file system after a file-system crash.

11.12 Consider the following backup scheme:

- **Day 1.** Copy to a backup medium all files from the disk.
- **Day 2.** Copy to another medium all files changed since day 1.
- **Day 3.** Copy to another medium all files changed since day 1.

This differs from the schedule given in Section 11.7.4 by having all subsequent backups copy all files modified since the first full backup. What are the benefits of this system over the one in Section 11.7.4? What are the drawbacks? Are restore operations made easier or more difficult? Explain your answer.

Programming Problems

The following exercise examines the relationship between files and inodes on a UNIX or Linux system. On these systems, files are repre sented with inodes. That is, an inode is a file (and vice versa). You can complete this exercise on the Linux virtual machine that is provided with this text. You can also complete the exercise on any Linux, UNIX, or Mac OS X system, but it will require creating two simple text files named `file1.txt` and `file3.txt` whose contents are unique sentences.

11.13 In the source code available with this text, open `file1.txt` and examine its contents. Next, obtain the inode number of this file with the command

```
ls -li file1.txt
```

This will produce output similar to the following:

```
16980 -rw-r--r-- 2 os os 22 Sep 14 16:13 file1.txt
```

where the inode number is boldfaced. (The inode number of `file1.txt` is likely to be different on your system.)

The UNIX `ln` command creates a link between a source and target file. This command works as follows:

```
ln [-s] <source file> <target file>
```

UNIX provides two types of links: (1) **hard links** and (2) **soft links.** A hard link creates a separate target file that has the same inode as the source file. Enter the following command to create a hard link between `file1.txt` and `file2.txt`:

```
ln file1.txt file2.txt
```

What are the inode values of `file1.txt` and `file2.txt`? Are they the same or different? Do the two files have the same—or different—contents?

Next, edit `file2.txt` and change its contents. After you have done so, examine the contents of `file1.txt`. Are the contents of `file1.txt` and `file2.txt` the same or different?

Next, enter the following command which removes `file1.txt`:

```
rm file1.txt
```

Does `file2.txt` still exist as well?

Now examine the man pages for both the `rm` and `unlink` commands. Afterwards, remove `file2.txt` by entering the command

```
strace rm file2.txt
```

The `strace` command traces the execution of system calls as the command `rm file2.txt` is run. What system call is used for removing `file2.txt`?

A soft link (or symbolic link) creates a new file that "points" to the name of the file it is linking to. In the source code available with this text, create a soft link to `file3.txt` by entering the following command:

```
ln -s file3.txt file4.txt
```

After you have done so, obtain the inode numbers of `file3.txt` and `file4.txt` using the command

```
ls -li file*.txt
```

Are the inodes the same, or is each unique? Next, edit the contents of `file4.txt`. Have the contents of `file3.txt` been altered as well? Last, delete `file3.txt`. After you have done so, explain what happens when you attempt to edit `file4.txt`.

Bibliographical Notes

The MS-DOS FAT system is explained in [Norton and Wilton (1988)]. The internals of the BSD UNIX system are covered in full in [McKusick and Neville-Neil (2005)]. Details concerning file systems for Linux can be found in [Love (2010)]. The Google file system is described in [Ghemawat et al. (2003)]. FUSE can be found at http://fuse.sourceforge.net.

Log-structured file organizations for enhancing both performance and consistency are discussed in [Rosenblum and Ousterhout (1991)], [Seltzer et al. (1993)], and [Seltzer et al. (1995)]. Algorithms such as balanced trees (and much more) are covered by [Knuth (1998)] and [Cormen et al. (2009)]. [Silvers (2000)] discusses implementing the page cache in the NetBSD operating system. The ZFS source code for space maps can be found at http://src.opensolaris.org/source/xref/onnv/onnv-gate/usr/src/uts/common/fs/zfs/space_map.c.

The network file system (NFS) is discussed in [Callaghan (2000)]. NFS Version 4 is a standard described at http://www.ietf.org/rfc/rfc3530.txt. [Ousterhout (1991)] discusses the role of distributed state in networked file systems. Log-structured designs for networked file systems are proposed in [Hartman and Ousterhout (1995)] and [Thekkath et al. (1997)]. NFS and the UNIX file system (UFS) are described in [Vahalia (1996)] and [Mauro and McDougall (2007)]. The NTFS file system is explained in [Solomon (1998)]. The Ext3 file system used in Linux is described in [Mauerer (2008)] and the WAFL file system is covered in [Hitz et al. (1995)]. ZFS documentation can be found at http://www.opensolaris.org/os/community/ZFS/docs.

Bibliography

[Callaghan (2000)] B. Callaghan, *NFS Illustrated*, Addison-Wesley (2000).

[Cormen et al. (2009)] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, Third Edition, MIT Press (2009).

[Ghemawat et al. (2003)] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google File System", *Proceedings of the ACM Symposium on Operating Systems Principles* (2003).

[Hartman and Ousterhout (1995)] J. H. Hartman and J. K. Ousterhout, "The Zebra Striped Network File System", *ACM Transactions on Computer Systems*, Volume 13, Number 3 (1995), pages 274–310.

a block is corrupted, the system must have a way to lock out that block or to replace it logically with a spare.

Because an efficient swap space is a key to good performance, systems usually bypass the file system and use raw-disk access for paging I/O. Some systems dedicate a raw-disk partition to swap space, and others use a file within the file system instead. Still other systems allow the user or system administrator to make the decision by providing both options.

Because of the amount of storage required on large systems, disks are frequently made redundant via RAID algorithms. These algorithms allow more than one disk to be used for a given operation and allow continued operation and even automatic recovery in the face of a disk failure. RAID algorithms are organized into different levels; each level provides some combination of reliability and high transfer rates.

Exercises

12.1 None of the disk-scheduling disciplines, except FCFS, is truly fair (starvation may occur).

a. Explain why this assertion is true.

b. Describe a way to modify algorithms such as SCAN to ensure fairness.

c. Explain why fairness is an important goal in a time-sharing system.

d. Give three or more examples of circumstances in which it is important that the operating system be unfair in serving I/O requests.

12.2 Explain why SSDs often use an FCFS disk-scheduling algorithm.

12.3 Suppose that a disk drive has 5,000 cylinders, numbered 0 to 4,999. The drive is currently serving a request at cylinder 2,150, and the previous request was at cylinder 1,805. The queue of pending requests, in FIFO order, is:

2,069, 1,212, 2,296, 2,800, 544, 1,618, 356, 1,523, 4,965, 3,681

Starting from the current head position, what is the total distance (in cylinders) that the disk arm moves to satisfy all the pending requests for each of the following disk-scheduling algorithms?

a. FCFS

b. SSTF

c. SCAN

d. LOOK

e. C-SCAN

f. C-LOOK

12.4 Elementary physics states that when an object is subjected to a constant acceleration a, the relationship between distance d and time t is given by $d = \frac{1}{2}at^2$. Suppose that, during a seek, the disk in Exercise 12.3 accelerates the disk arm at a constant rate for the first half of the seek, then decelerates the disk arm at the same rate for the second half of the seek. Assume that the disk can perform a seek to an adjacent cylinder in 1 millisecond and a full-stroke seek over all 5,000 cylinders in 18 milliseconds.

a. The distance of a seek is the number of cylinders over which the head moves. Explain why the seek time is proportional to the square root of the seek distance.

b. Write an equation for the seek time as a function of the seek distance. This equation should be of the form $t = x + y\sqrt{L}$, where t is the time in milliseconds and L is the seek distance in cylinders.

c. Calculate the total seek time for each of the schedules in Exercise 12.3. Determine which schedule is the fastest (has the smallest total seek time).

d. The **percentage speedup** is the time saved divided by the original time. What is the percentage speedup of the fastest schedule over FCFS?

12.5 Suppose that the disk in Exercise 12.4 rotates at 7,200 RPM.

a. What is the average rotational latency of this disk drive?

b. What seek distance can be covered in the time that you found for part a?

12.6 Describe some advantages and disadvantages of using SSDs as a caching tier and as a disk-drive replacement compared with using only magnetic disks.

12.7 Compare the performance of C-SCAN and SCAN scheduling, assuming a uniform distribution of requests. Consider the average response time (the time between the arrival of a request and the completion of that request's service), the variation in response time, and the effective bandwidth. How does performance depend on the relative sizes of seek time and rotational latency?

12.8 Requests are not usually uniformly distributed. For example, we can expect a cylinder containing the file-system metadata to be accessed more frequently than a cylinder containing only files. Suppose you know that 50 percent of the requests are for a small, fixed number of cylinders.

a. Would any of the scheduling algorithms discussed in this chapter be particularly good for this case? Explain your answer.

b. Propose a disk-scheduling algorithm that gives even better performance by taking advantage of this "hot spot" on the disk.

imply overhead from several sources: context switching to cross the kernel's protection boundary, signal and interrupt handling to service the I/O devices, and the load on the CPU and memory system to copy data between kernel buffers and application space.

Exercises

13.1 When multiple interrupts from different devices appear at about the same time, a priority scheme could be used to determine the order in which the interrupts would be serviced. Discuss what issues need to be considered in assigning priorities to different interrupts.

13.2 What are the advantages and disadvantages of supporting memory-mapped I/O to device control registers?

13.3 Consider the following I/O scenarios on a single-user PC:

a. A mouse used with a graphical user interface

b. A tape drive on a multitasking operating system (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.

13.4 In most multiprogrammed systems, user programs access memory through virtual addresses, while the operating system uses raw physical addresses to access memory. What are the implications of this design for the initiation of I/O operations by the user program and their execution by the operating system?

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

13.6 Describe three circumstances under which blocking I/O should be used. Describe three circumstances under which nonblocking I/O should be used. Why not just implement nonblocking I/O and have processes busy-wait until their devices are ready?

13.7 Typically, at the completion of a device I/O, a single interrupt is raised and appropriately handled by the host processor. In certain settings, however, the code that is to be executed at the completion of the I/O can be broken into two separate pieces. 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?

13.8 Some DMA controllers support direct virtual memory access, where the targets of I/O operations are specified as virtual addresses and a translation from virtual to physical address is performed during the DMA. How does this design complicate the design of the DMA controller? What are the advantages of providing such functionality?

13.9 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.

13.10 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.

13.11 Discuss the advantages and disadvantages of guaranteeing reliable transfer of data between modules in the STREAMS abstraction.

Bibliographical Notes

[Vahalia (1996)] provides a good overview of I/O and networking in UNIX. [McKusick and Neville-Neil (2005)] detail the I/O structures and methods employed in FreeBSD. The use and programming of the various interprocess-communication and network protocols in UNIX are explored in [Stevens (1992)]. [Hart (2005)] covers Windows programming.

[Intel (2011)] provides a good source for Intel processors. [Rago (1993)] provides a good discussion of STREAMS. [Hennessy and Patterson (2012)] describe multiprocessor systems and cache-consistency issues.

Bibliography

[Hart (2005)] J. M. Hart, *Windows System Programming*, Third Edition, Addison-Wesley (2005).

[Hennessy and Patterson (2012)] J. Hennessy and D. Patterson, *Computer Architecture: A Quantitative Approach*, Fifth Edition, Morgan Kaufmann (2012).

[Intel (2011)] *Intel 64 and IA-32 Architectures Software Developer's Manual, Combined Volumes: 1, 2A, 2B, 3A, and 3B*. Intel Corporation (2011).

[McKusick and Neville-Neil (2005)] M. K. McKusick and G. V. Neville-Neil, *The Design and Implementation of the FreeBSD UNIX Operating System*, Addison Wesley (2005).

[Rago (1993)] S. Rago, *UNIX System V Network Programming*, Addison-Wesley (1993).

[Stevens (1992)] R. Stevens, *Advanced Programming in the UNIX Environment*, Addison-Wesley (1992).

[Vahalia (1996)] U. Vahalia, *Unix Internals: The New Frontiers*, Prentice Hall (1996).

protection domain:	untrusted applet	URL loader	networking
socket permission:	none	*.lucent.com:80, connect	any

class:	gui:
	...
	get(url);
	open(addr);
	...
		get(URL u):
		...
		doPrivileged {
		open('proxy.lucent.com:80');
		}
		<request u from proxy>
		...
			open(Addr a):
			...
			checkPermission
			(a, connect);
			connect (a);
			...

Figure 14.9 Stack inspection.

well-defined interfaces. Compliance is enforced through a sophisticated collection of load-time and run-time checks. As a result, an object cannot manipulate its run-time stack, because it cannot get a reference to the stack or other components of the protection system.

More generally, Java's load-time and run-time checks enforce **type safety** of Java classes. Type safety ensures that classes cannot treat integers as pointers, write past the end of an array, or otherwise access memory in arbitrary ways. Rather, a program can access an object only via the methods defined on that object by its class. This is the foundation of Java protection, since it enables a class to effectively **encapsulate** and protect its data and methods from other classes loaded in the same JVM. For example, a variable can be defined as `private` so that only the class that contains it can access it or `protected` so that it can be accessed only by the class that contains it, subclasses of that class, or classes in the same package. Type safety ensures that these restrictions can be enforced.

14.10 Summary

Computer systems contain many objects, and they need to be protected from misuse. Objects may be hardware (such as memory, CPU time, and I/O devices) or software (such as files, programs, and semaphores). An access right is permission to perform an operation on an object. A domain is a set of access rights. Processes execute in domains and may use any of the access rights in the domain to access and manipulate objects. During its lifetime, a process may be either bound to a protection domain or allowed to switch from one domain to another.

The access matrix is a general model of protection that provides a mechanism for protection without imposing a particular protection policy on the system or its users. The separation of policy and mechanism is an important design property.

The access matrix is sparse. It is normally implemented either as access lists associated with each object or as capability lists associated with each domain. We can include dynamic protection in the access-matrix model by considering domains and the access matrix itself as objects. Revocation of access rights in a dynamic protection model is typically easier to implement with an access-list scheme than with a capability list.

Real systems are much more limited than the general model and tend to provide protection only for files. UNIX is representative, providing read, write, and execution protection separately for the owner, group, and general public for each file. MULTICS uses a ring structure in addition to file access. Hydra, the Cambridge CAP system, and Mach are capability systems that extend protection to user-defined software objects. Solaris 10 implements the principle of least privilege via role-based access control, a form of the access matrix.

Language-based protection provides finer-grained arbitration of requests and privileges than the operating system is able to provide. For example, a single Java JVM can run several threads, each in a different protection class. It enforces the resource requests through sophisticated stack inspection and via the type safety of the language.

Exercises

14.1 Consider the ring-protection scheme in MULTICS. If we were to implement the system calls of a typical operating system and store them in a segment associated with ring 0, what should be the values stored in the ring field of the segment descriptor? What happens during a system call when a process executing in a higher-numbered ring invokes a procedure in ring 0?

14.2 The access-control matrix can be used to determine whether a process can switch from, say, domain A to domain B and enjoy the access privileges of domain B. Is this approach equivalent to including the access privileges of domain B in those of domain A?

14.3 Consider a computer system in which computer games can be played by students only between 10 P.M. and 6 A.M., by faculty members between 5 P.M. and 8 A.M., and by the computer center staff at all times. Suggest a scheme for implementing this policy efficiently.

14.4 What hardware features does a computer system need for efficient capability manipulation? Can these features be used for memory protection?

14.5 Discuss the strengths and weaknesses of implementing an access matrix using access lists that are associated with objects.

14.6 Discuss the strengths and weaknesses of implementing an access matrix using capabilities that are associated with domains.

14.7 Explain why a capability-based system such as Hydra provides greater flexibility than the ring-protection scheme in enforcing protection policies.

14.8 Discuss the need for rights amplification in Hydra. How does this practice compare with the cross-ring calls in a ring-protection scheme?

14.9 What is the need-to-know principle? Why is it important for a protection system to adhere to this principle?

14.10 Discuss which of the following systems allow module designers to enforce the need-to-know principle.

a. The MULTICS ring-protection scheme

b. Hydra's capabilities

c. JVM's stack-inspection scheme

14.11 Describe how the Java protection model would be compromised if a Java program were allowed to directly alter the annotations of its stack frame.

14.12 How are the access-matrix facility and the role-based access-control facility similar? How do they differ?

14.13 How does the principle of least privilege aid in the creation of protection systems?

14.14 How can systems that implement the principle of least privilege still have protection failures that lead to security violations?

Bibliographical Notes

The access-matrix model of protection between domains and objects was developed by [Lampson (1969)] and [Lampson (1971)]. [Popek (1974)] and [Saltzer and Schroeder (1975)] provided excellent surveys on the subject of protection. [Harrison et al. (1976)] used a formal version of the access-matrix model to enable them to prove properties of a protection system mathematically.

The concept of a capability evolved from Iliffe's and Jodeit's *codewords*, which were implemented in the Rice University computer ([Iliffe and Jodeit (1962)]). The term ***capability*** was introduced by [Dennis and Horn (1966)].

The Hydra system was described by [Wulf et al. (1981)]. The CAP system was described by [Needham and Walker (1977)]. [Organick (1972)] discussed the MULTICS ring-protection system.

Revocation was discussed by [Redell and Fabry (1974)], [Cohen and Jefferson (1975)], and [Ekanadham and Bernstein (1979)]. The principle of separation of policy and mechanism was advocated by the designer of Hydra ([Levin et al. (1975)]). The confinement problem was first discussed by [Lampson (1973)] and was further examined by [Lipner (1975)].

The use of higher-level languages for specifying access control was suggested first by [Morris (1973)], who proposed the use of the seal and unseal operations discussed in Section 14.9. [Kieburtz and Silberschatz (1978)], [Kieburtz and Silberschatz (1983)], and [McGraw and Andrews (1979)] proposed various language constructs for dealing with general dynamic-resource-management schemes. [Jones and Liskov (1978)] considered how a static access-control scheme can be incorporated in a programming language that supports abstract data types. The use of minimal operating-system support to enforce protection was advocated by the Exokernel Project ([Ganger et al. (2002)], [Kaashoek et al. (1997)]). Extensibility of system code through language-based protection mechanisms was discussed in [Bershad et al. (1995)]. Other techniques for enforcing protection include sandboxing ([Goldberg et al.

(1996)]) and software fault isolation ([Wahbe et al. (1993)]). The issues of lowering the overhead associated with protection costs and enabling user-level access to networking devices were discussed in [McCanne and Jacobson (1993)] and [Basu et al. (1995)].

More detailed analyses of stack inspection, including comparisons with other approaches to Java security, can be found in [Wallach et al. (1997)] and [Gong et al. (1997)].

Bibliography

[Basu et al. (1995)] A. Basu, V. Buch, W. Vogels, and T. von Eicken, "U-Net: A User-Level Network Interface for Parallel and Distributed Computing", *Proceedings of the ACM Symposium on Operating Systems Principles* (1995).

[Bershad et al. (1995)] B. N. Bershad, S. Savage, P. Pardyak, E. G. Sirer, M. Fiuczynski, D. Becker, S. Eggers, and C. Chambers, "Extensibility, Safety and Performance in the SPIN Operating System", *Proceedings of the ACM Symposium on Operating Systems Principles* (1995), pages 267–284.

[Cohen and Jefferson (1975)] E. S. Cohen and D. Jefferson, "Protection in the Hydra Operating System", *Proceedings of the ACM Symposium on Operating Systems Principles* (1975), pages 141–160.

[Dennis and Horn (1966)] J. B. Dennis and E. C. V. Horn, "Programming Semantics for Multiprogrammed Computations", *Communications of the ACM*, Volume 9, Number 3 (1966), pages 143–155.

[Ekanadham and Bernstein (1979)] K. Ekanadham and A. J. Bernstein, "Conditional Capabilities", *IEEE Transactions on Software Engineering*, Volume SE-5, Number 5 (1979), pages 458–464.

[Ganger et al. (2002)] G. R. Ganger, D. R. Engler, M. F. Kaashoek, H. M. Briceno, R. Hunt, and T. Pinckney, "Fast and Flexible Application-Level Networking on Exokernel Systems", *ACM Transactions on Computer Systems*, Volume 20, Number 1 (2002), pages 49–83.

[Goldberg et al. (1996)] I. Goldberg, D. Wagner, R. Thomas, and E. A. Brewer, "A Secure Environment for Untrusted Helper Applications", *Proceedings of the 6th Usenix Security Symposium* (1996).

[Gong et al. (1997)] L. Gong, M. Mueller, H. Prafullchandra, and R. Schemers, "Going Beyond the Sandbox: An Overview of the New Security Architecture in the Java Development Kit 1.2", *Proceedings of the USENIX Symposium on Internet Technologies and Systems* (1997).

[Harrison et al. (1976)] M. A. Harrison, W. L. Ruzzo, and J. D. Ullman, "Protection in Operating Systems", *Communications of the ACM*, Volume 19, Number 8 (1976), pages 461–471.

[Iliffe and Jodeit (1962)] J. K. Iliffe and J. G. Jodeit, "A Dynamic Storage Allocation System", *Computer Journal*, Volume 5, Number 3 (1962), pages 200–209.