CSCI 4320 (Principles of Operating Systems), Fall 2002

Homework 4

Assigned: November 15, 2002.

Due: November 21, 2002, at 5pm.

Credit: 50 points.

1 Reading

Be sure you have read chapters 3 and 4.

2 Problems

Answer the following questions. You may write out your answers by hand or using a word processor or other program, but please submit hard copy, either in class or in my mailbox in the department office.

1. (4 points) Figure 3-1(b) on p. 162 of the textbook illustrates using semaphores to control access to two resources. Each process using this pseudocode first acquires resource 1, then acquires resource 2, then uses both resources, and finally releases them, in the reverse of the order in which they were acquired. Would it work just as well to release the resources in the opposite order (i.e., in the same order in which they were acquired), or could this lead to deadlock? Assume at least two processes are running this code at the same time.

2. (6 points) Suppose you are designing an electronic funds transfer system, in which there will be many identical processes that work as follows: Each process accepts as input an amount of money to transfer, the account to be credited, and the account to be debited. It then locks both accounts (one at a time), transfers the money, and releases the locks when done. Many of these processes could be running at the same time. A friend proposes a simple scheme for locking the accounts: First lock the account to be credited; then lock the account to be debited. Can this scheme lead to deadlock?

   If you think it cannot, briefly explain why not. If you think it can, first give an example of a possible deadlock situation, and then design a scheme that avoids deadlocks, but in such a way that once an account is locked, it is not released until the funds transfer is complete (i.e., a design that relies on repeatedly locking one account, trying the other, and releasing the first is not allowed).

3. (6 points) Consider a computer system that uses variable partitions and swapping to manage memory (as described in chapter 4 pages 197–198); suppose it has 10,000 bytes of main memory, currently allocated as shown by the following table. (In this problem, amounts and addresses are in decimal, and unrealistically small, to keep the arithmetic simple.)

Locations	Contents
7500 – 9999	free
4500 – 7499	process B
4000 – 4499	free
3000 – 3999	process A
2000 – 2999	free
0 – 1999	operating system

Answer the following questions:

(a) Suppose we now need to allocate space first for process C, which needs 400 bytes, and then for process D, which needs 200 bytes. At what addresses (locations) will memory be allocated for these two processes if we use

i. a best-fit strategy for allocating memory?

ii. a worst-fit strategy for allocating memory?

iii. a first-fit strategy for allocating memory?

(b) Suppose the MMU for this system uses the simple base register / limit register scheme described in chapter 1 of the textbook (pages 26–27).

i. What value would need to be loaded into the base register if we performed a context switch to restart process B?

ii. What memory locations would correspond to the following virtual (program) addresses in process B?

- 100
- 4000

4. (10 points) Now consider a computer system using paging to manage memory; suppose it has 64K (2^{16}) bytes of memory and a page size of 4K bytes, and suppose the page table for some process (call it process A) looks like the following.

Page number	Present/absent bit	Page frame number
0	1	4
1	1	5
2	1	2
3	0	?
4	0	?
5	1	7

Answer the following questions about this system.

(a) If the only processes in main memory were process A and an operating system using page frame 0, how many free page frames would there be?

(b) How many additional page table entries would be required to give process A an address space of 64K bytes?

(c) How many bits are required to represent physical addresses (memory locations) on this system? If each process has a maximum address space of 64K bytes, how many bits are required to represent virtual (program) addresses?

(d) What memory locations would correspond to the following virtual (program) addresses for process A? (Here, the addresses will be given in hexadecimal, i.e., base 16, to make the needed calculations simpler. Your answers should also be in hexadecimal.)

- 0x1420
- 0x2ff0
- 0x4008
- 0x0010

(e) If we want to guarantee that this system could support 16 concurrent processes and give each an address space of 64K bytes, how much disk space would be required for storing out-of-memory pages? Justify your answer. (If your answer depends on making additional assumptions, state what they are — e.g., you might assume that the operating system will always use the first page frame of memory and will never be paged out.)

(f) Would it be possible on this system to give each process a maximum address space of 1M (2^{20}) bytes? Briefly explain your answer.

5. (6 points) Consider a small computer system with only four page frames and address spaces consisting of eight pages. Suppose we start out with all page frames empty (pure demand paging) and run a program that references pages in the following order:

$$0, 1, 7, 2, 3, 2, 7, 1, 0, 3$$

(That is, its first reference to memory is in page number 0, its second reference to memory is in page number 1, etc.) If FIFO page replacement is being used, how many page faults will occur during the running of this program?

6. (6 points) Consider another small computer system with only four page frames. Suppose you have implemented the aging algorithm for page replacement, using 4-bit counters and updating the counters after every clock tick, and suppose the R bits for the four pages are as follows after the first four clock ticks.

Time	R bit (page 0)	R bit (page 1)	R bit (page 2)	R bit (page 3)
after tick 1	0	1	1	1
after tick 2	1	0	1	1
after tick 3	1	0	1	0
after tick 4	1	1	0	1

What are the values of the counters (in binary) for all pages after these four clock ticks? If a page needed to be removed at that point, which page would be chosen for removal?

7. (8 points) Now consider a much bigger computer system, one in which addresses (both physical and virtual) are 32 bits and the system has 2^{32} bytes of memory. (You can express your answers in terms of powers of 2, if that is convenient.)

(a) What is the maximum size in bytes of a process's address space on this system?

(b) Is there a limit to how much main memory this system can make use of?

(c) If page size is 4K (2^{12}) and each page table entry consists of a page frame number and four additional bits (present/absent, referenced, modified, and read-only), how much space is required for each process's page table? If we allow a maximum of 64 (2^6) processes,

how much space in total is required for all their page tables? (You should express the size of each page table entry in bits, not bytes, assuming 8 bits per byte and rounding up if necessary.)

(If 64 processes seems like a lot, log into one of the department Linux machines, execute the command `top`, and note the total number of processes it reports.)

(d) Suppose instead we use a single inverted page table (as described in chapter 4, pages 213–214), in which each entry consists of a page number, a process ID, and four additional bits (free/in-use, referenced, modified, and read-only), and we allow at most 64 processes, how much space is needed for this inverted page table? (You should express the size of each page table entry in bits, not bytes, assuming 8 bits per byte and rounding up if necessary.)

8. (4 points) Can a page (i.e., the contents of a page frame) be in two working sets at the same time? Why or why not?