Fall 2014
CSE 421/521 – Operating Systems
Homework Assignment #3

*** SOLUTION KEY ***

Problem 1:

Explain the difference between internal and external fragmentation. Suggest ways to prevent each.

Answer: Internal Fragmentation is the area in a region or a page that is not used by the job occupying that region or page. This space is unavailable for use by the system until that job is finished and the page or region is released. This can be addressed by dynamically determining the size of the allocation which would exactly fit the request, i.e. segmentation. In case of external fragmentation, the total memory space available is sufficient to satisfy a request, but it is not contiguous and for that reason it cannot satisfy the request. This can be addressed by reallocation of memory space (compaction) or via a non-contiguous memory allocation technique, i.e. segmentation.

Problem 2:

Given five memory partitions of 100 KB, 500 KB, 200 KB, 300 KB, and 600 KB (in order), how would each of 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?

Answer:

a. First-fit:

- 212K is put in 500K partition
- 417K is put in 600K partition
- 112K is put in 288K partition (new partition 288K = 500K - 212K)
- 426K must wait

b. Best-fit:

- 212K is put in 300K partition
- 417K is put in 500K partition
- 112K is put in 200K partition
- 426K is put in 600K partition

c. Worst-fit:

- 212K is put in 600K partition
- 417K is put in 500K partition
- 112K is put in 388K partition
- 426K must wait

In this example, Best-fit turns out to be the best.

Problem 3:

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.

Answer: Paging requires more memory overhead to maintain the translation structures. Segmentation requires just two registers per segment: one to maintain the base of the segment and the other to maintain the extent of the segment. Paging on the other hand requires one entry per page, and this entry provides the physical address in which the page is located.

Problem 4:

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 associative registers, and 75 percent of all page-table references are found in the associative registers, what is the effective memory reference time? (Assume that finding a page-table entry in the associative registers takes zero time, if the entry is there.)

Answer:

a. 400 nanoseconds; 200 nanoseconds to access the page table and 200 nanoseconds to access the word in memory.

b. Effective access time = 0.75 x (200 nanoseconds) + 0.25 x (400 nanoseconds) = 250 nanoseconds.

Problem 5:

Why are segmentation and paging sometimes combined into one scheme?

Answer: Segmentation and paging are often combined in order to improve upon each other. Segmented paging is helpful when the page table becomes very large. A large contiguous section of the page table that is unused can be collapsed into a single segment table entry with a pagetable address of zero. Paged segmentation handles the case of having very long segments that require a lot of time for allocation. By paging the segments, we reduce wasted memory due to external fragmentation as well as simplify the allocation.

Problem 6:

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 4096 bytes. A user process generates the virtual address 11123456. Explain how the system establishes the corresponding physical location. Distinguish between software and hardware operations.

Answer: The virtual address in binary form is

0001 0001 0001 0010 0011 0100 0101 0110

Since the page size is 2^{12}, the page table size is 2^{20}. Therefore the loworder 12 bits "0100 0101 0110" are used as the displacement into the page, while the remaining 20 bits "0001 0001 0001 0010 0011" are used as the displacement in the page table.

Problem 7:

Assume 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 page is available or 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?

Answer:
0.2 microsec = $(1 - P)$ x 0.1 microsec + $(0.3P)$ x 8 millisec + $(0.7P)$ x 20 millisec
$0.1 = -0.1P + 2400\ P + 14000\ P$
$0.1 = 16{,}400\ P$
$P = 0.000006$

Problem 8:

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

Answer: Consider the following sequence of memory accesses in a system that can hold four pages in memory. Sequence: 1 1 2 3 4 5 1. When page 5 is accessed, the least frequently used page-replacement algorithm would replace a page other than 1, and therefore would not incur a page fault when page 1 is accessed again. On the other hand, for the sequence "1 2 3 4 5 2," the least recently used algorithm performs better.

Problem 9:

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

- CPU utilization 20%
- Paging disk 97.7%
- Other I/O devices 5%

Which (if any) of the following will (probably) improve CPU utilization? Explain your answer.

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
a) hard disks.
g. Add prepaging to the page fetch algorithms.
h. Increase the page size.

Answer: The system obviously is spending most of its time paging, indicating over-allocation of memory. If the level of multiprogramming is reduced resident processes would page fault less frequently and the CPU utilization would improve. Another way to improve performance would be to get more physical memory or a faster paging drum.

a. Get a faster CPU—No.

b. Get a bigger paging drum—No.

c. Increase the degree of multiprogramming—No.

d. Decrease the degree of multiprogramming—Yes.

e. Install more main memory—Likely to improve CPU utilization as more pages can remain resident and not require paging to or from the disks.

f. Install a faster hard disk, or multiple controllers with multiple hard disks—Also an improvement, for as the disk bottleneck is removed by faster response and more throughput to the disks, the CPU will get more data more quickly.

g. Add prepaging to the page fetch algorithms—Again, the CPU will getmore data faster, so it will be more in use. This is only the case if the paging action is amenable to prefetching (i.e., some of the access is sequential).

h. Increase the page size—Increasing the page size will result in fewer page faults if data is being accessed sequentially. If data access is more or less random, more paging action could ensue because fewer pages can be kept in memory and more data is transferred per page fault. So this change is as likely to decrease utilization as it is to increase it.

Problem 10:

A page-replacement algorithm should minimize the number of page faults. We can do 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 that are associated with that frame. Then, to replace a page, we search for the page frame with the smallest counter.

a. Define a page-replacement algorithm using this basic idea. Specifically address the problems of (1) what the initial value of the counters is, (2) when counters are increased, (3) when counters are decreased, and (4) how the page to be replaced is selected.

b. How many page faults occur for your algorithm for the following reference string, for 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?

Answer:

a. Define a page-replacement algorithm addressing the problems of:

1. Initial value of the counters—0.
2. Counters are increased—whenever a new page is associated with that frame.
3. Counters are decreased—whenever one of the pages associated with that frame is no longer required.
4. How the page to be replaced is selected—find a frame with the smallest counter. Use FIFO for breaking ties.

b. 14 page faults

c. 11 page faults