CHAPTER 9

Virtual-Memory Management

Virtual memory can be a very interesting subject since it has so many different aspects: page faults, managing the backing store, page replacement, frame allocation, thrashing, page size. The objectives of this chapter are to explain these concepts and show how paging works.

A simulation is probably the easiest way to allow the students to program several of the page-replacement algorithms and see how they really work. If an interactive graphics display can be used to display the simulation as it works, the students may be better able to understand how paging works. We also present an exercise that asks the student to develop a Java program that implements the FIFO and LRU page-replacement algorithms.

9.1 The following allocation is made by the Buddy system: The 240-byte request is assigned a 256-byte segment. The 120-byte request is assigned a 128-byte segement, the 60-byte request is assigned a 64-byte segment and the 130-byte request is assigned a 256-byte segment. After the allocation, the following segment sizes are available: 64-bytes, 256-bytes, 1K, 2K, 4K, 8K, 16K, 32K, 64K, 128K, 256K, and 512K.

After the releases of memory, the only segment in use would be a 256-byte segment containing 130 bytes of data. The following segments will be free: 256 bytes, 512 bytes, 1K, 2K, 4K, 8K, 16K, 32K, 64K, 128K, 256K, and 512K.

9.2 a. $9EF$ - $0EF$

b. 111 - 211

c. 700 - $D00$

d. $0FF$ - EFF

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

i. Initial value of the counters—0.

ii. Counters are increased—whenever a new page is associated with that frame.

iii. Counters are decreased—whenever one of the pages associated with that frame is no longer required.

iv. 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

9.4 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. Install a faster CPU—No.

b. Install a bigger paging disk—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 get more 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.

9.5 a. Thrashing is occurring.

b. CPU utilization is sufficiently high to leave things alone, and increase degree of multiprogramming.

c. Increase the degree of multiprogramming.

9.6
$$\begin{aligned} \text{effective access time} &= (0.8) \times (1\ \mu\text{sec}) \\ &\quad + (0.1) \times (2\ \mu\text{sec}) + (0.1) \times (5002\ \mu\text{sec}) \\ &= 501.2\ \mu\text{sec} \\ &= 0.5 \text{ millisec} \end{aligned}$$

9.7
- On a page fault the thread state is set to blocked as an I/O operation is required to bring the new page into memory.
- On a TLB-miss, the thread continues running if the address is resolved in the page table.

- The thread will continue running if the address is resolved in the page table.

9.8 For every memory-access operation, the page table needs to be consulted to check whether the corresponding page is resident or not and whether the program has read or write privileges for accessing the page. These checks have to be performed in hardware. A TLB could serve as a cache and improve the performance of the lookup operation.

9.9

Number of frames	LRU	FIFO	Optimal
1	20	20	20
2	18	18	15
3	15	16	11
4	10	14	8
5	8	10	7
6	7	10	7
7	7	7	7

9.10 The program could have a large code segment or use large-sized arrays as data. These portions of the program could be allocated to larger pages, thereby decreasing the memory overheads associated with a page table. The virtual memory system would then have to maintain multiple free lists of pages for the different sizes and also needs to have more complex code for address translation to take into account different page sizes.

9.11 Consider the sequence in a system that holds four pages in memory: 1 2 3 4 4 4 5 1. The most frequently used page replacement algorithm evicts page 4 while fetching page 5, while the LRU algorithm evicts page 1. This is unlikely to happen much in practice. For the sequence "1 2 3 4 4 4 5 1," the LRU algorithm makes the right decision.

9.12 A page fault occurs when an access to a page that has not been brought into main memory takes place. The operating system verifies the memory access, aborting the program if it is invalid. If it is valid, a free frame is located and I/O is requested to read the needed page into the free frame. Upon completion of I/O, the process table and page table are updated and the instruction is restarted.

9.13 The following page faults take place: page fault to access the instruction, a page fault to access the memory location that contains a pointer to the target memory location, and a page fault when the target memory location is accessed. The operating system will generate three page faults with the third page replacing the page containing the instruction. If the instruction needs to be fetched again to repeat the trapped instruction, then the sequence of page faults will continue indefinitely. If the instruction is cached in a register, then it will be able to execute completely after the third page fault.

9.14 A working set for each thread. This is because each kernel thread has its own execution sequence, thus generating its unique sequence of addresses.

9.15 When two processes are accessing the same set of program values (for instance, the code segment of the source binary), then it is useful to map the corresponding pages into the virtual address spaces of the two programs

in a write-protected manner. When a write does indeed take place, then a copy must be made to allow the two programs to individually access the different copies without interfering with each other. The hardware support required to implement is simply the following: on each memory access, the page table needs to be consulted to check whether the page is write protected. If it is indeed write protected, a trap would occur and the operating system could resolve the issue.

9.16 a. 50

b. 5,000

9.17 Consider the following sequence of memory accesses in a system that can hold four pages in memory: 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.

9.18 Thrashing is caused by underallocation of the minimum number of pages required by a process, forcing it to continuously page fault. The system can detect thrashing by evaluating the level of CPU utilization as compared to the level of multiprogramming. It can be eliminated by reducing the level of multiprogramming.

9.19 If the pointer is moving fast, then the program is accessing a large number of pages simultaneously. It is most likely that during the period between the point at which the bit corresponding to a page is cleared and it is checked again, the page is accessed again and therefore cannot be replaced. This results in more scanning of the pages before a victim page is found. If the pointer is moving slow, then the virtual memory system is finding candidate pages for replacement extremely efficiently, indicating that many of the resident pages are not being accessed.

9.20 a. When a page fault occurs and if the page does not exist in the free-frame pool, then one of the pages in the free-frame pool is evicted to disk, creating space for one of the resident pages to be moved to the free-frame pool. The accessed page is then moved to the resident set.

b. When a page fault occurs and if the page exists in the free-frame pool, then it is moved into the set of resident pages, while one of the resident pages is moved to the free-frame pool.

c. When the number of resident pages is set to one, then the system degenerates into the page replacement algorithm used in the free-frame pool, which is typically managed in a LRU fashion.

d. When the number of pages in the free-frame pool is zero, then the system degenerates into a FIFO page-replacement algorithm.

9.21 This has long been a problem with the slab allocator—poor scalability with multiple CPUs. The issue comes from having to lock the global cache when it is being access. This has the effect of serializing cache accesses on multiprocessor systems. Solaris has addressed this by introducing a per-CPU cache, rather than a single global cache.

9.22
$$\begin{aligned} 0.2\ \mu\text{sec} &= (1-P) \times 0.1\ \mu\text{sec} + (0.3P) \times 8\ \text{millisec} + (0.7P) \times 20\ \text{millisec} \\ 0.1 &= -0.1P + 2400\ P + 14000\ P \\ 0.1 &\simeq 16{,}400\ P \\ P &\simeq 0.000006 \end{aligned}$$

9.23

a. **FIFO**. Find the first segment large enough to accommodate the incoming segment. If relocation is not possible and no one segment is large enough, select a combination of segments whose memories are contiguous, which are "closest to the first of the list" and which can accommodate the new segment. If relocation is possible, rearrange the memory so that the first N segments large enough for the incoming segment are contiguous in memory. Add any leftover space to the free-space list in both cases.

b. **LRU**. Select the segment that has not been used for the longest period of time and that is large enough, adding any leftover space to the free space list. If no one segment is large enough, select a combination of the "oldest" segments that are contiguous in memory (if relocation is not available) and that are large enough. If relocation is available, rearrange the oldest N segments to be contiguous in memory and replace those with the new segment.

9.24

a. Stack—good.

b. Hashed symbol table—not good.

c. Sequential search—good.

d. Binary search—not good.

e. Pure code—good.

f. Vector operations—good.

g. Indirection—not good.

9.25 Yes, because there is only one kernel thread for all user threads, that kernel thread blocks while waiting for the page fault to be resolved. Since there are no other kernel threads for available user threads, all other user threads in the process are thus affected by the page fault.

9.26

a. Initially quite high as needed pages are not yet loaded into memory.

b. It should be quite low as all necessary pages are loaded into memory.

c. (1) Ignore it; (2) get more physical memory; (3) reclaim pages more aggressively due to the high page fault rate.

9.27

a. TLB miss with no page fault page has been brought into memory, but has been removed from the TLB

b. TLB miss and page fault page fault has occurred

c. TLB hit and no page fault page is in memory and in the TLB. Most likely a recent reference

d. TLB hit and page fault cannot occur. The TLB is a cache of the page table. If an entry is not in the page table, it will not be in the TLB.

9.28 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 low-order 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.

9.29 The costs are additional hardware and slower access time. The benefits are good utilization of memory and larger logical address space than physical address space.

9.30 Assume that the page boundary is at 1024 and the move instruction is moving values from a source region of 800:1200 to a target region of 700:1100. Assume that a page fault occurs while accessing location 1024. By this time the locations of 800:923 have been overwritten with the new values and therefore restarting the block move instruction would result in copying the new values in 800:923 to locations 700:823, which is incorrect.

9.31 When Δ is set to a small value, then the set of resident pages for a process might be underestimated, allowing a process to be scheduled even though all of its required pages are not resident. This could result in a large number of page faults. When Δ is set to a large value, then a process's resident set is overestimated and this might prevent many processes from being scheduled even though their required pages are resident. However, once a process is scheduled, it is unlikely to generate page faults since its resident set has been overestimated.

9.32 Yes, in fact many processors provide two TLBs for this very reason. As an example, the code being accessed by a process may retain the same working set for a long period of time. However, the data the code accesses may change, thus reflecting a change in the working set for data accesses.

9.33 Such an algorithm could be implemented with the use of a reference bit. After every examination, the bit is set to zero; set back to one if the page is referenced. The algorithm would then select an arbitrary page for replacement from the set of unused pages since the last examination.

The advantage of this algorithm is its simplicity—nothing other than a reference bit need be maintained. The disadvantage of this algorithm is that it ignores locality by using only a short time frame for determining whether to evict a page or not. For example, a page may be part of the working set of a process, but may be evicted because it was not referenced since the last examination (that is, not all pages in the working set may be referenced between examinations).