Virtual Memory

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.

Exercises

9.14 Assume a program has just referenced an address in virtual memory. Describe a scenario how each of the following can occur: (If a scenario cannot 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

Answer:

- TLB miss with no page fault page has been brought into memory, but has been removed from the TLB
- TLB miss and page fault page fault has occurred
- TLB hit and no page fault page is in memory and in the TLB. Most likely a recent reference
- 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.

9.15 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. is 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 include an explanation of your answer.)

a. Will the thread change state if it incurs a page fault? If so, to what new state?

b. Will the thread change state if it generates a TLB miss that is resolved in the page table? If so, to what new state?

c. Will the thread change state if an address reference is resolved in the page table? If so, to what new state?

Answer:

- 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.

9.16 Consider a system that uses pure demand paging:

a. When a process first start 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 a process changes its locality and the size of the new working set is too large to be stored into available free memory. What are some options system designers could choose from to handle this situation?

Answer:

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.

9.17 Give an example that illustrates the problem with restarting the block move instruction (MVC) on the IBM 360/370 when the source and destination regions are overlapping.

Answer: 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.

9.18 Discuss the hardware support required to support demand paging.
Answer: 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.

9.19 What is the copy-on-write feature and under what circumstances is it beneficial to use this feature? What is the hardware support required to implement this feature?
Answer: 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.

9.20 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 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.

9.21 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:

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

9.22 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 there exists a process with five user-level threads where the mapping of user threads to kernel threads is **many: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 (i.e., would they also have to wait for the faulting page to be brought into memory?) Explain.
Answer: 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.

9.23 Table 9.1 is a 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.)

Page	Page Frame
0	-
1	2
2	C
3	A
4	–
5	4
6	3
7	–
8	B
9	0

Table 9.1 Page Table

Given the following virtual addresses, convert them to their equivalent physical addresses in hexadecimal. All numbers are given in hexadecimal. (A dash for a page frame indicates the page is not in memory.)

- $9EF$
- 111
- 700
- $0FF$

Answer:

- $9EF$ - $0EF$

- 111 - 211
- 700 - $D00$
- $0FF$ - EFF

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

Answer: 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.

9.25 Discuss situations under which the least frequently used page-replacement algorithm generates fewer page faults than the least recently used page-replacement algorithm. Also discuss under what circumstance the opposite holds.
Answer: 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.

9.26 Discuss situations under which the most frequently used page-replacement algorithm generates fewer page faults than the least recently used page-replacement algorithm. Also discuss under what circumstance the opposite holds.
Answer: 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.

9.27 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 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 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?

Answer:

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.

9.28 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, 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 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. 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.

9.29 Suppose that a machine provides instructions that can access memory locations using the one-level indirect addressing scheme. What is the sequence of page faults incurred when all of the pages of a program are currently non resident 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?
Answer: 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.

9.30 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?

Answer: 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).

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 the initial value of the counters is
   ii. When counters are increased
   iii. When counters are decreased
   iv. 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:
   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

9.32 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 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?

Answer:

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

9.33 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?

Answer: 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.

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

Answer: 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.

9.35 Consider the parameter Δ used to define the working-set window in the working-set model. What is the effect of setting Δ to a small value on the page fault frequency and the number of active (non-suspended) processes currently executing in the system? What is the effect when Δ is set to a very high value?

Answer: 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.

9.36 Assume there is an initial 1024 KB segment where memory is allocated using the Buddy system. Using Figure 9.27 as a guide, draw the 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

Answer: 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.

9.37 Consider a system that provides support for user and kernel-level threads where the mapping is one: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
Answer: A working set for each thread. This is because each kernel thread has its own execution sequence, thus generating its unique sequence of addresses.

9.38 The slab allocation algorithm uses a separate cache for each different object type. Assuming there is one cache per object type, explain why this doesn't scale well with multiple CPUs. What could be done to address this scalability issue?
Answer: 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.

9.39 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?
Answer: 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.