8 Memory Management

Although many systems are demand paged (discussed in Chapter 10), there are still many that are not, and in many cases the simpler memory-management strategies may be better, especially for small dedicated systems. We want the student to learn about all of them: resident monitor, swapping, partitions, paging, and segmentation.

Exercises

8.9 A simplified view of thread states is **Ready**, **Running**, **Blocked** where a thread is either ready and waiting to be scheduled, is running on the processor, or is blocked. Consider a thread that may be in one of these three states on a computer with virtual memory. Assuming a thread is running, (1) will the thread change state if it incurs a page fault? (2) what about a TLB-miss but resolved in the page table?
Answer: On a page fault the thread state is set to blocked. On a TLB-miss, the thread continues running.

8.10 Explain the difference between internal and external fragmentation.
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.

8.11 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?
Answer: The linkage editor has to replace unresolved symbolic addresses with the actual addresses associated with the variables in the final program binary. In order to perform this, the modules should keep track of instructions that refer to unresolved symbols. During linking,

each module is assigned a sequence of addresses in the overall program binary and when this has been performed, unresolved references to symbols exported by this binary could be patched in other modules since every other module would contain the list of instructions that need to be patched.

8.12 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:
b. 212K is put in 500K partition
c. 417K is put in 600K partition
d. 112K is put in 288K partition (new partition 288K = 500K − 212K)
e. 426K must wait
f. Best-fit:
g. 212K is put in 300K partition
h. 417K is put in 500K partition
i. 112K is put in 200K partition
j. 426K is put in 600K partition
k. Worst-fit:
l. 212K is put in 600K partition
m. 417K is put in 500K partition
n. 112K is put in 388K partition
o. 426K must wait

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

8.13 Most systems allow programs to allocate more memory to its address space during execution. Data allocated in the heap segments of programs are 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

Answer:

a. contiguous-memory allocation: might require relocation of the entire program since there is not enough space for the program to grow its allocated memory space.

b. pure segmentation: might also require relocation of the segment that needs to be extended since there is not enough space for the segment to grow its allocated memory space.

c. pure paging: incremental allocation of new pages is possible in this scheme without requiring relocation of the program's address space.

8.14 Compare the main 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

Answer: The contiguous memory allocation scheme suffers from external fragmentation as address spaces are allocated contiguously and holes develop as old processes die and new processes are initiated. It also does not allow processes to share code, since a process's virtual memory segment is not broken into noncontiguous finegrained segments. Pure segmentation also suffers from external fragmentation as a segment of a process is laid out contiguously in physical memory and fragmentation would occur as segments of dead processes are replaced by segments of new processes. Segmentation, however, enables processes to share code; for instance, two different processes could share a code segment but have distinct data segments. Pure paging does not suffer from external fragmentation, but instead suffers from internal fragmentation. Processes are allocated in page granularity and if a page is not completely utilized, it results in internal fragmentation and a corresponding wastage of space. Paging also enables processes to share code at the granularity of pages.

8.15 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?

Answer: An address on a paging system is a logical page number and an offset. The physical page is found by searching a table based on the logical page number to produce a physical page number. Because the operating system controls the contents of this table, it can limit a process to accessing only those physical pages allocated to the process. There is no way for a process to refer to a page it does not own because the page will not be in the page table. To allow such access, an operating system simply needs to allow entries for non-process memory to be added to the process's page table. This is useful when two or more processes need to exchange data—they just read and write to the same physical addresses (which may be at varying logical addresses). This makes for very efficient interprocess communication.

8.16 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.

8.17 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 towards lower virtual addresses. What is the significance of the above structure for the following schemes:

a. contiguous-memory allocation

b. pure segmentation

c. pure paging

Answer: 1) Contiguous-memory allocation requires the operating system to allocate the entire extent of the virtual address space to the program when it starts executing. This could be much larger than the actual memory requirements of the process. 2) Pure segmentation gives the operating system flexibility to assign a small extent to each segment at program startup time and extend the segment if required. 3) Pure paging does not require the operating system to allocate the maximum extent of the virtual address space to a process at startup time, but it still requires the operating system to allocate a large page table spanning all of the program's virtual address space. When a program needs to extend the stack or the heap, it needs to allocate a new page but the corresponding page table entry is preallocated.

8.18 Assuming a 1 KB page size, what are the page numbers and offsets for the following address references (provided as decimal numbers):

a. 2375

b. 19366

c. 30000

d. 256

e. 16385

Answer:

a. page = 1; offset = 391

b. page = 18; offset = 934

c. page = 29; offset = 304

d. page = 0; offset = 256

e. page = 1; offset = 1

8.19 Consider a logical address space of 32 pages with 1024 words per page; mapped onto a physical memory of 16 frames.

a. How many bits are required in the logical address?

b. How many bits are required in the physical address?

Answer:

a. $2^5 = 32 + 2^10 = 1024 = 15$ bits.

b. $2^4 = 32 + 2^10 = 1024 = 14$ bits.

8.20 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:

a. A conventional single-level page table?

b. An inverted page table?

Answer:

a. A conventional single-level page table?

b. An inverted page table?

8.21 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 $\times$ (200 nanoseconds) + 0.25 $\times$ (400 nanoseconds) = 250 nanoseconds.

8.22 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 page-table 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.

8.23 Explain why it is easier to share a reentrant module using segmentation than it is to do so when pure paging is used.
Answer: Since segmentation is based on a logical division of memory rather than a physical one, segments of any size can be shared with only

one entry in the segment tables of each user. With paging there must be a common entry in the page tables for each page that is shared.

8.24 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

Answer:

a. 219 + 430 = 649

b. 2300 + 10 = 2310

c. illegal reference, trap to operating system

d. 1327 + 400 = 1727

e. illegal reference, trap to operating system

8.25 What is the purpose of paging the page tables?
Answer: In certain situations the page tables could become large enough that by paging the page tables, one could simplify the memory allocation problem (by ensuring that everything is allocated as fixed-size pages as opposed to variable-sized chunks) and also enable the swapping of portions of page table that are not currently used.

8.26 Consider the hierarchical paging scheme used by the VAX architecture. How many memory operations are performed when an user program executes a memory load operation?
Answer: When a memory load operation is performed, there are three memory operations that might be performed. One is to translate the position where the page table entry for the page could be found (since page tables themselves are paged). The second access is to access the page table entry itself, while the third access is the actual memory load operation.

8.27 Compare the segmented paging scheme with the hashed page tables scheme for handling large address spaces. Under what circumstances is one scheme preferable to the other?

Answer: When a program occupies only a small portion of its large virtual address space, a hashed page table might be preferred due to its smaller size. The disadvantage with hashed page tables however is the problem that arises due to conflicts in mapping multiple pages onto the same hashed page table entry. If many pages map to the same entry, then traversing the list corresponding to that hash table entry could incur a significant overhead; such overheads are minimal in the segmented paging scheme where each page table entry maintains information regarding only one page.

8.28 Consider the Intel address-translation scheme shown in Figure 8.22.

a. Describe all the steps that the Intel 80386 takes 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 hardware?

c. Are there any disadvantages to this address-translation system? If so, what are they? If not, why is it not used by every manufacturer?

Answer:

a. The selector is an index into the segment descriptor table. The segment descriptor result plus the original offset is used to produce a linear address with a dir, page, and offset. The dir is an index into a page directory. The entry from the page directory selects the page table, and the page field is an index into the page table. The entry from the page table, plus the offset, is the physical address.

b. Such a page-translation mechanism offers the flexibility to allow most operating systems to implement their memory scheme in hardware, instead of having to implement some parts in hardware and some in software. Because it can be done in hardware, it is more efficient (and the kernel is simpler).

c. Address translation can take longer due to the multiple table lookups it can invoke. Caches help, but there will still be cache misses.