Memory Management Strategies

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.

8.1 On a page fault the thread state is set to blocked. On a TLB-miss, the thread continues running.

8.2
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.

8.3
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.

8.4
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.

8.5 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.

8.6 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

8.7 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.

8.8 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.

8.9 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.

8.10 a. A conventional single-level page table?

b. An inverted page table?

8.11 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.

8.12 The major advantage of this scheme is that it is an effective mechanism for code and data sharing. For example, only one copy of an editor or a compiler needs to be kept in memory, and this code can be shared by all processes needing access to the editor or compiler code. Another advantage is protection of code against erroneous modification. The only disadvantage is that the code and data must be separated, which is usually adhered to in a compiler-generated code.

8.13 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.

8.14 a. Logical address: 16 bits

b. Physical address: 15 bits

8.15 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.

8.16 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.

8.17 Since segment tables are a collection of base–limit registers, segments can be shared when entries in the segment table of two different jobs point to the same physical location. The two segment tables must have identical base pointers, and the shared segment number must be the same in the two processes.

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

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

8.19 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.

8.20 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.

8.21 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.

8.22 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

8.23 Both of these problems reduce to a program being able to reference both its own code and its data without knowing the segment or page number associated with the address. MULTICS solved this problem by associating four registers with each process. One register had the address of the current program segment, another had a base address for the stack, another had a base address for the global data, and so on. The idea is that all

references have to be indirect through a register that maps to the current segment or page number. By changing these registers, the same code can execute for different processes without the same page or segment numbers.

8.24 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.