

CSE 411

Fall 1999

Solution for Homework #3

Nan Huo, Yiqiong Wu and Anirudh Modi

November 1, 1999

1. Such system would pre-allocate a file space at the creation time, then allow the file to grow dynamically. Since the simplest file organization would be to make the disk image match the virtual address space identically, it means that the file offset is equal to the virtual address. Therefore it would not be necessary for page table to translate virtual address to file offset. The file system will allocate file blocks in a random order as needed so that there isn't any special code to be written in the file system and just use regular interface. For those virtual address page that will never be used at all, page table would mark them as invalid backing store address. When a page fault occurs to such a virtual page, the system would not allocate the disk block to it because the page table indicate it doesn't actually exist. When a modified page comes and the file block has not yet been allocated, the file system would allocate a new block in the backing store at random. And since there is only one backing store, the page table should store the actual allocation information for the file, which means the file is mapped by some sort of associative table.
2. The disk rotates at 750 rpm, which is equivalent to $750/60 = 12.5$ rotations per second. Thus, a single rotation requires $1000/12.5 = 80$ ms. The average rotational latency is half of the time required for a single rotation, hence it is $80/2 = 40$ ms. For cases B, C and D, a diagram of the disk allocated with 4-block clusters with two-way disk interleaving is shown in the figure below:
 - A. Time required to service a page fault if the page size is 1 K
= Seek time + Rotational latency + Data transfer time + Cache to VM time
= $60 + 40 + 2 \times 1 + 0.5 \times 1$
= 102.5 ms
 - B. Time required to service a page fault if the page size is 1 K when the disk is allocated with 4-block clusters (see figure)
= Seek time + Rotational latency + Data transfer time + Cache to VM time

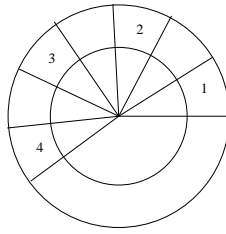


Figure 1: 4-block clusters with two-way disk interleaving

$$= 60 + 40 + 2 \times 7 + 0.5 \times 4$$

$$= 116.0 \text{ ms}$$

- C. Time required to service the 1 K pagefault
 = Seek time + Rotational latency + Data transfer time + Cache to VM time
 = $60 + 40 + 2 \times 7 + 0.5 \times 1$
 = 114.5 ms
- D. Time required to service the page fault
 = Seek time + Rotational latency + Data transfer time + Cache to VM time
 = $60 + 40 + 2 \times 7 + 0.5 \times 4$
 = 116.0 ms

If the replaced pages in the case above are modified, the page fault time would typically double. Here are the exact times required for each case:

- A: $T = (60 + 40 + 2 \times 1 + 0.5 \times 1) \times 2 = 205.0 \text{ ms}$
 B: $T = (60 + 40 + 2 \times 7 + 0.5 \times 4) \times 2 = 232.0 \text{ ms}$
 C: $T = 114.5 + (60 + 40 + 2 \times 7 + 0.5 \times 4) = 230.5 \text{ ms}$
 D: $T = (60 + 40 + 2 \times 7 + 0.5 \times 4) \times 2 = 232.0 \text{ ms}$

Advantages and disadvantages of these four approaches with respect to each other:

Case D may allocate pages to a process that may never be used. Thus, some in-use pages may be replaced unnecessarily/prematurely. On the other hand, if those pages really are needed in the near future, it assures that they will be in the memory. Allocating 4-block pages (case B) leads to a lesser degree of internal fragmentation, which allows for a better use of the main memory. When unwanted pages are allocated (as in cases C and D), those pages can be replaced as and when needed.

- The potential problem is that consistency of the sharing information in this implementation. For example, one process modify the executable it is running so that the program recompile it. When the next page fault occurs, it loads a portion of the new version of the program. While other active processes didn't change the program, the program in memory would be internally inconsistent. If each process allocate and copy the executable into its own backing store, it might not run the new version of the program or at least run a consistent one.