Operating Systems Week 2 – Solutions

Q1. Briefly explain why paging suffers from internal fragmentation but not external fragmentation.

Answer: Internal fragmentation occurs as a full frame is allocated to a page, even though only a few bytes may be needed by the process to store the data it requested memory for. There is no external fragmentation as there are no gaps between pages: each frame is exactly one page in size, so they are stored adjacently in memory.

Q2. 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:
– First-fit:
  212K is put in 500K partition (leaving a 288K partition)
  417K is put in 600K partition
  112K is put in 288K partition
  426K must wait (cannot be allocated)

– Best-fit:
  212K is put in 300K partition
  417K is put in 500K partition
  112K is put in 200K partition
  426K is put in 600K partition

– Worst-fit:
  212K is put in 600K partition (leaving a 388K partition)
  417K is put in 500K partition
  112K is put in 388K partition
  426K must wait (cannot be allocated)

Q3.

Size of the objects: 4 bytes per variable x 3 variables = 12 bytes
Memory used after the loop (which creates 16 objects)
= 16 * 12
= 192 bytes

Using a contiguous memory allocation model: 16 holes were created, i.e. one per object. These are 16 bytes each (it says in the question). 16 * 16 = 256 bytes for memory control blocks.

*NB. The importance of the 10000 was just to make sure it was a very large hole. If the hole was 192 + 15 * 16 = 432 bytes, it still would have been **just** large enough, and in fact, only 15 memory control blocks would have been created: after storing the 16th Coordinate, there would be literally 0 memory left, so no space for another memory control block.*

Q4. 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 (200 nanoseconds) + 0.25 (400 nanoseconds) = 250 nanoseconds.

Q5

The object pool is of size 10, so for the 16 coordinates, **2** pools are created. Thus:

a) Memory for memory control blocks = one per pool = 2 * 16 bytes = 32 bytes
b) Coordinate objects that are in use = 16 * 12 bytes = 192 bytes, as before
c) Spare coordinate objects: 20 were created, 16 were used, so 4 spare 4 * 12 = 48 bytes
d) Single-linked list nodes: one for each spare = 4 * 8 = 32 bytes

e) Total overhead = 32 + 32 + 48 = 112 bytes

(Compare this to Q3 where the overheads were 256 bytes.)