5CCS2OSC Tutorial – Week 2

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

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 memory allocation requests of 212 KB, 417 KB, 112 KB, and 426 KB (arriving in this order)? *(Assume there are no allocation overheads due to memory control blocks.)*

Which algorithm makes the most efficient use of memory, in terms of managing to meet the most allocation requests?

Q3. For this question, assume the size of a memory control block is 16 bytes, and that you are using a contiguous memory allocation model. Consider the following class:

```
class Coordinate {
public:
  int x;
  int y;
  int z;
};
```

- If int variables are 32 bits, what is the size of object of this class, in bytes?

The following code is then executed:

```
for (int i = 0; i < 16; ++i) { new Coordinate(); }
```

- At this point, how much memory is used to store Coordinate objects?
- If we are using a contiguous memory allocation model, starting with one hole of size 10000 bytes, how much of this memory has now been used to store Memory Control Blocks?

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 (a TLB), 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.)

Q5. Consider an 'object pool' where:

- The objects being stored in the pool are Coordinate objects (from question 3)
- The object pool reserves space for Coordinate objects ten at a time
- These blocks of ten Coordinates are taken from some hole in a heap using a contiguous memory allocation model (initially, with a single hole of size 10000).
- Unused Coordinate objects are placed on a list; and the size of the single-linked-list nodes used to store this list are 8 bytes.

Suppose again, the code:

```
for (int i = 0; i < 16; ++i) { new Coordinate(); }
```

...is then executed, but 'new' requests the memory from the object pool.

a) How much memory has now been used to store Memory Control Blocks, for the Coordinates in the object pool? (Assuming again each of these is 16 bytes)
b) At this point, how much memory is used to store Coordinate objects *that are actually in use*?
c) How much memory is used to store Coordinate objects that have been created for the object pool, but that are *not* currently in use?
d) How much memory in total is used to store the single-linked-list nodes for these objects?

Assume the total memory overheads for an object pool, at a given point is:

- Memory used for memory control blocks; plus
- Memory used for single-linked-list nodes; plus
- Memory used for objects that are not presently in use

e) What is the total memory overhead in the example discussed in this question?