Solution Assignment 3

Question No. 1
Given memory partitions of 100KB, 500KB, 200KB, 300KB, and 600KB (in order), how would each of the first-fit, best-fit, and worst-fit algorithms place processes of 212KB, 417KB, 112KB and 426KB (in order)? Which algorithm makes the most efficient use of memory?

Solution:

In first-fit:

212KB process will be placed in 500KB partition
417KB process will be placed in 600KB partition
112KB process will be placed in the new 288KB partition
There will be no big enough partition for 426KB process

In best-fit:

212KB process will be placed in 300KB partition
417KB process will be placed in 500KB partition
112KB process will be placed in 200KB partition
426KB process will be placed in 600KB partition

In worst-fit:

212KB process will be placed in 600KB partition
417KB process will be placed in 500KB partition
112KB process will be placed in the new 388KB partition
There will be no big enough partition for 426KB process

As can be seen, best-fit is making the most efficient use of memory.

Question No. 2
Why page sizes are usually powers of 2?

Solution:

Paging is implemented by breaking up an address into a page and offset number. It is most efficient to break the address into X page bits and Y offset bits, rather than perform arithmetic on the address to calculate the page number and offset. Because each bit position represents a power of 2, splitting an address between bits results in a page size that is a power of 2.

Question No. 3
Consider a logical address space of 16 pages of 4096 words each, mapped onto a physical memory of 64 frames?

a. How many bits are there in the logical address?
b. How many bits are there in the physical address?

Solutions:

# of bits in the logical address = # of bits in page number + # of bits in offset
= 4 + 12 = 16 bits
# of bits in the physical address = # of bits in frame number + # of bits in offset
= 6 + 12 = 18 bits

Question No. 4
Consider a paging system with the page table stored in memory.

a. If a memory reference takes 300 nanoseconds, how long does a paged memory reference take?
b. If we add associative registers, and 85 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.)

Solution:

a) Paged memory reference here will take 600 ns, 300 ns to access the page table and 300 ns to access the actual memory.

b) Since 85 percent of all page-table references are found in the associative registers, therefore, effective memory reference time = 0.85 * 300 ns + 0.15 * 600 ns = 255 + 90 = 345 ns.

Question No. 5
Explain why it is easier to share a reentrant module using segmentation than it is to do so when pure paging is used.

Solution:

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.

Question No. 6
Assume that we have a demand-paged memory. The page table is held in registers. It takes 10 milliseconds to service a page fault if an empty page is available or if the replaced page is not modified, and 30 milliseconds if the replaced page is modified. Memory access time is 200 nanoseconds.
Assume that the page to be replaced is modified 60 percent of the time. What is the maximum acceptable page-fault rate for an effective access time of no more than 500 nanoseconds?

Solution:

Let x be the rate of occurrence of page fault
Then effective access time can be equated with page fault rate as follows.

500 ns $= (1 – x) * 200$ ns $+ x * (0.60 * 30$ ms $+ 0.40 * 10$ ms$)$
500 ns $– 200$ ns $= -x * 200$ ns $+ x * 18$ ms $+ x * 4$ ms
300 ns $= x * (22$ milliseconds $- 200$ nanoseconds$)$
$x = 300$ ns $/ 21999800$ ns
$x = 0.0000136$
Hence, the maximum acceptable page-fault rate is 0.0000136

Question No. 7
Consider the following page reference string:

1, 2, 3, 4, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6, 2, 1, 5, 6, 2.

How many page faults would occur for the following replacement algorithms, assuming one, two, three, four, five, six, or seven frames? Remember all frames are initially empty, so your first unique pages will all cost one fault each.

a. LRU replacement
b. FIFO replacement
c. Optimal replacement

Solution:

Number of Frames	LRU replacement	FIFO replacement	Optimal replacement
1	20	20	20
2	19	19	15
3	15	18	10
4	8	12	7
5	7	10	7
6	7	7	7
7	7	7	7

Question No. 8

Consider the two dimensional array A:

```
int A[ ] [ ] = new int [50] [50]
```

where A[0] [0] is at location 400, in a paged system with pages of size 400. A small process is in page 0 (locations 0 to 399) for manipulating the matrix; thus, every instruction fetch will be from page 0.

For three page frames, how many page faults are generated by the following array-initialization loops, using LRU replacement, and assuming page frame 1 has the process in it, and the other two are initially empty:

```
for (int j = 0; j < 50; j++)
    for (int i = 0; i < 50; i++)
        A[i][j] = 0;
```

Solution:

There is a correction made in the question. Both the inner and outer loops go for 50 iterations rather than 100.

Let int be of 4 bytes. As page size is 400, each page will contain 100 integers.

Here, starting with a page fault, after every two iterations of the inner for loop a page fault will be generated. So, for each iteration of the outer loop 25 page faults will be generated. Hence, total page faults generated will be 50 * 25 = 1250.