CMSC 412 Midterm #2 (Fall 2011) - Solution

1.) (20 points) Define and explain the following terms:

a) Rotational Latency

Time for the desired data to spin under the desk head. On average, ½ the time for one complete disk rotation.

b) Super page

Using larger pages (i.e. 4MB on an x86 – 32 bit) to reduce the number of TLB entries required to access a given range of virtual memory and thus reducing TLB misses.

c) Thrashing

When a computer makes little forward progress due to a lack of enough memory to fit the working set of pages into memory.

d) inode

A data structure of the UNIX file system that holds meta data for the file including the modification time, size, and lists of disks blocks holding the actual data. It does not contain the file name.

2.) (20 points) Memory Systems

a) (10 points) Consider an x86-32 like architecture with a hardware TLB (with an access time of 10 ns for a hit or miss), and a two level page directory and page table structure (both of which are pageable). The TLB has a hit rate of 99% and the memory access time to read any word of physical memory is 100ns (i.e. don't consider CPU caches). On average, 0.001% of all memory references result in a page fault which takes an average of 3ms to service. What is the effective access time for the virtual memory?

10 ns + 100 ns + 1/100 (200 ns) + (1/100,000 * 3,000,000 ns) + 2 * 1/100 * 1/100,000 * 3,000,000 ns =

10 ns + 100 ns + 2 ns + 30 ns + 2 * 3 ns = about 148 ns

TLB lookup + memory access time + TLB miss time (access page dir then page table) + page fault time (for page) + page fult time for page table, and page directory

b) (10 points) Explain what the clock algorithm for page aging is and explain why it is often used instead of full LRU.

Periodically a sweep is made of all the page frames with access bit set and they are marked with the current value of a counter (called the clock). The clock is then advanced and all access bits are cleared. This is used to group pages into equivalence classes of ages since last access. Having the hardware set a use bit and the OS make the sweep requires much simpler hardware than maintaining full hardware LRU.

3.) (20 Points) Synchronization: Given an implementation of counting semaphores (semaphores whose values range from 0 to some positive integer), implement binary semaphores (semaphores whose value is either 0 or 1).

```
Int count =1;
countingSemaphore lock=1
countingSemaphote wait=1
```

$P_{binary:}$

```
P(lock)
Count--
V(lock);
P(wait)
```

$V_{binary:}$

```
P(lock)
If (count <= 0) {
    Count++
    V(wait);
}
V(lock)
```

4.) (20 points) File Systems

a) (6 points) Explain why file systems often include a file name cache even if they have a file buffer cache.

Name lookup requires multiple steps (proportional to the number of directories in the path) even if all the disk blocks are in memory and this can be slow. A file name cache turns this into a single hash lookup (on a hit in the name cache).

b) (8 points) Consider a FAT file system (using 32 bit block numbers) and 4KB blocks. If you wanted to read 2^{20} byte of a file, what is the minimum **and maximum** number of disk reads required? Be sure to explain each step, don't just give numbers.

2^{20} byte is in the $2^{20}/2^{12} = 2^8 = 256$th disk block. A single disk block holds up to 1024 FAT entries

Minimum:

Assuming disk blocks for the file are one after the other, takes 2 reads, one for the FAT table and one for the data block.

Maximum:

Each of the 256 blocks are in a different disk block in the FAT, 256 + 1 for the data = 257.

c) (6 points) What is the role of the cleaner in a log structured file system?

The cleaner copies in-use files to the end of the log. Basically it defragments the log to get eliminate holes left by file deletions.

5.) (20 points) Project

a) (6 points) Explain why it is possible to use a single LDT for all user processes in the project once the paging code for project #4 is done.

All processes have the same base address (2GB) and limit (2GB) so they can all share one IDT.

b) (6 points) Why does the function `Alloc_Pageable_Page` need to know the virtual address where the page to be allocated will be mapped?

When paging out a page, the OS needs to know the mapped location so it can clear the page table entry (to prevent a process accessing the page that is no longer in memory).

c) (8 points) Consider the Clone system call. It works like fork (i.e. it creates a new process with the same program as its parent). The difference is that it creates a kernel visible user mode thread (i.e. all of the text and data are shared with the parent). List the steps that would be involved in creating the user mode context to implement Clone. Make sure to describe how the page tables of the new process should be setup.

It starts like a normal create thread, but will need to copy the page directory entries from its parent for the text and data segments. The page tables can be shared for these segments. The page frames for these segments *must* be shared. Would need to either copy the stack or create a new one (the question did not specify is clone took a function to invoke or if the child returns to where the parent was executing).