CS4410 Homework 4

Due 9am Thursday March 12

100 points as shown below. Very short answers, please.

1. (10 points). What do we mean when we say that a virtual memory system is thrashing ?

Thrashing refers to a situation in which the I/O rate associated with page faults has become annoyingly high. One way to express this is to say that the VM system is unable to keep the working set resident, and hence tends to page out pages that will soon be referenced again. Thrashing is very much a subjective problem: on a slow computer, one might not think of a given policy as thrashing, but when the same workload is moved to a faster machine, the faster execution may cause it to page fault more frequently and suddenly, merely because the CPU sped up, the same application seems to be threashing.

2. (10 points). Do threads in the same process share page table entries? What about TLB entries?

Yes: threads in a single process share the same address space. The page table is a representation of the address space and the TLB is a cache of page table entries.

3. (10 points). Assuming that a process is launched with no pages in memory. On a virtual memory system, what is the minimum number of page faults that it could occur before it terminates?

At best we will incur one page fault for each page actually referenced (e.g. code that actually gets executed or data that actually is touched). In general this will be a function of the program being run, the input it receives, etc.

4. (10 points). Suppose that an operating system supports very large virtual address spaces composed of paged variable-length segments that may be widely separated by gaps. Why would a traditional page table get large in this case?

The gaps need to be represented in the page table by PTEs that explicitly say that the corresponding pages are not in use. Thus a "missing page" (a gap) still has an overhead of one PTE per page.

5. (10 points). Explain the idea of an inverted page table. Would it suffer from the same issues that you mentioned in responding to question 4?

*In an inverted page table, rather than having one page table entry per logical page, we have one per physical page. Thus the size of the IPT is always proportional to the amount of memory on the computer. It would **not** suffer from the problem discussed in (4) because the gaps in the virtual address space don't correspond to real pages, hence there is no IPT entry associated with them. On the other hand, the OS still needs to maintain a conventional page table somewhere on the side (perhaps in paged memory) to keep track of the contents of memory, and that other conventional page table would need some way of dealing with gaps.*

6. (10 points). Describe a situation in which "file-block prefetching" can harm performance.

A block prefetch algorithm guesses the next file block that an application will access. If it guesses wrong, the disk ends up doing extra work: the work the application requested, plus extra work to handle the prefetches.

7. (10 points). Suppose that you create a file named /tmp/xyz.jog and then create a true link (on Linux) to it named "my_file.jpg". This file now has two "types". What will happen if you try and open my_file? Assume that the file is really a JPEG (a photo) and that ".jog" is a typo, not a file system extension for any existing application.

It should open properly in the photo viewer application. The fact that there is a second name with a different extension won't matter.

8. (10 points). Same scenario as 7. You realize that the original file name was just a mistake, so you rename /tmp/xyz.jog as /tmp/xyz.jpg. But then you edit the photo, and the photo editing program deletes /tmp/xyz.jpg and then writes a new file, which it renames /tmp/xyz.jpg. What would you see if you print the photo under its other name: my_file.jpg?

You'll see the old copy – the file from before it was edited.

9. (10 points). Same scenario as 8, but now assume that the link is a symbolic link.

If the symbolic link pointed to the old (incorrectly spelled) name, you won't be able to open it: you'll get "file doesn't exist". However if the link had the correct name, you'll see the new (edited) version of the file.

10. (10 points). You measure the file access performance of your new computer and discover that reading a random byte from a random file seems to have more than one "cost". In fact, the values cluster: there is one set of files for which this cost is usually 11ms. A second set of files cost more to access: 18ms. A third set of files cost even more to access: 25ms. Explain why this sort of "three costs model" isn't surprising. What accounts for the extra costs in the second and third group?

In Linux, small files are represented very efficiently: the data blocks are listed right in the inode. Medium files have a second representation involving one or more "indirection" blocks. Large files have double indirection blocks. Thus a random access to a small file will typically involve two disk I/O read requests (one to read the inode, one to read the data). A medium file will often need three (one for the inode, one for the indirection block, one to read the data). A large file on average will need four (inode, double indirection block, single indirection block, data).

From the numbers we can see that the cost of a disk read is apparently 7ms on this computer.