CSE 551
Problem Set #1

Due: 4:30pm, Thursday, October 22, 2009

1. Write pseudo-code in Java, Python, C++ or C, for a (very) simple UNIX shell capable of executing a command through a pipe. The focus of this question is on the concurrency, so you do not need to parse command line arguments, set environment variables or the like. Feel free to assume any convenient string parsing package. For example, if the user types "ls | wc", your program should fork off the two programs, which together will calculate the # of files in the directory. For this, you will need the UNIX system calls for fork, execve, open, close, pipe, dup, and wait. It is important to note that these system calls have subtly different semantics than Mesa (particularly fork and wait), so you'll need to read the man pages and/or the UNIX paper carefully. pipe(int fdes[2]) returns a pipe such that bytes written on fdes[1] can be read from fdes[0]. One tricky step concerns how to replace standard in and out (file descriptors 0 and 1); this is the role of "dup". Please note: The code does not need to compile or run; we are interested in the algorithm, not the syntax. However, it is cool to see it run, as it shows the power the UNIX model gives the developer. Please add enough comments so that the TA can understand your intent.

2. Write pseudo-code for a highly concurrent, multithreaded file buffer cache. A buffer cache stores recently used (that is, likely to be used soon) disk blocks in memory for improved latency and throughput. These blocks can be file data blocks, indirect blocks, inodes, and directories – that is, any persistent storage in the file system. Assume that the file system is to run on a system with dozens of CPUs, and so must be able to do many file operations in parallel. For simplicity, assume file operations are in terms of complete, block-aligned block reads and writes, int readblock(x[], block #) and int writeblock(x[], block #). Blocks can be in cache or on disk, and cache misses may cause blocks to be evicted from the cache to be written back to disk. Disk operations have the same interface: diskblockread(x[], block #) and diskblockwrite(x[], block #). Multiple threads can call readblock and writeblock concurrently, and to the maximum degree possible, those operations should be allowed to complete in parallel (consider that you are running on a multicore). Disk operations must be done only one at a time and are synchronous, in that the calling thread blocks until the disk operation completes.