Additional Exercises 1

Preliminaries

These exercises are *not* an assignment. They provide extra work that may help your understanding of some issues in operating systems. You will not, ordinarily, receive credit for this work. However, I would be happy to discusss solutions with you and I reserve the right to give some kind of bonus credit to any student who shows excellence in solving these problems.

Written Exercises

w1. A low-end microprocessor has 16KB of processor cache (of which 8KB is used to cache program data, and 8KB is used to cache program instructions) and can address 4MB of memory. Cache references take 10ns, whereas references to main memory take 100ns. It takes 5ns to determine whether a memory location is contained in the cache. If the location is cached, it is read from the cache, if it is not, it is read from main memory (after it is read, it is symultaneously used be the instuction and stored in the cache for future use).

(a) If 90% of memory accesses come from the cache, how long does it take, on average, to fetch the value stored in a memory location?

(b) In certain scenarios, the performance of a program is worsened by having a cache. Give such a scenario involving the

i. Instruction cache

ii. Data cache

Tip: You do not need to read up on microprocessor caches to answer this question.

w2. IBM's OS/390 UNIX System Services™ allow UNIX programs to be run on computers running IBM's OS/390 operating system. Although IBM may use the UNIX trademark and and describe these facilities as OS/390 UNIX, few people think of OS/390 as a UNIX system. Explain this situation, and give examples of two other operating systems which, with appropriate changes, would be able to call themselves UNIX.

Tip: You do not need to read up on OS/390 to answer this question.

w3. When is batch processing the preferred strategy for work to be done by the computer? When is timesharing the preferred strategy?

w4. The *degree of multiprogramming* is the maximum number of processes that may be supported by a uniprocessor machine at any given time. Discuss some of the factors that must be considered in determining the degree of multiprogramming for a particular system. You may assume that the operating system is a batch system, rather than a timesharing system.

Programming Exercises

For each of the programming exercises below, develop the program as indicated in the question. Before you begin, you may want to familiarize yourself with the following system calls: fork, execve, dup, pipe, wait.

You may program in either C or C++, but you should try to make your programs as simple as possible.

P1. Read the UNIX manual page for the system program tee. Write your own version, called mytee, that performs the basic functionality of tee (i.e., it does not need to take option switches, just a single output filename, as its argument). Your program must not call any library functions and may only use the following system calls: open, close, read, write, and exit.

Tip: Standard input is file descriptor zero. Standard output is file descriptor one.

P2. Write a program, harness, that executes another program with that program's standard input set up to read from the file test.in and its output set up to write to the file test.out. Your program must not call any library functions and may only use the following system calls: open, close, dup, execve and exit.

P3. Write a program, spew, that produces a stream of 1000 integers (one per line), beginning at some random (possibly negative) integer, where each successive integer is greater than the last. You may not directly make any system calls, but must instead use system library functions (such as puts or printf). Find and use a random number generator from the standard library (there may be more than one available). You may store the numbers in an array before outputting them.

P4. Write a program, merge, that forks two copies of your spew program, and outputs (on standard output) the result of merging the output of both programs (such that the resulting output is also in ascending order).

Your program should

- Create two pipes
- Fork two child processes, which set up the input ends of the pipes and then execve the spew program
- Read from the pipes and produce the output
- After both pipes run out of integers, check the exit status of the children to be sure they executed correctly

You may not rely on spew producing exactly 1000 lines, and your program should begin to produce its output after reading one line from each spew process (i.e., it should not store all its input data before producing any output).

Tip: You may find the fdopen library function useful for converting the output file descriptor of the pipe into a file stream suitable for use with the fgets or fscanf library functions.

(For fun, you may want to generalize your program so that it can run an arbitrary number of spew processes, rather than just two).