CS 422 (Park) **Assignment II** **Due: Sep. 25 (Sun.), 2011**

Submission instructions: Please type your answers and submit electronic copies using `turnin` *by 11pm on the due date. You may use any number of word processing software (e.g., Framemaker, Word, LaTeX), but the final output must be in pdf format that uses standard fonts (a practical test is to check if the pdf file prints on a CS Department printer). For experiments and programming assignments that involve output to terminal, please use* `script` *to record the output and submit the output file.*

PROBLEM 1 (25 pts)

Read Chapters 8, 10 and 11 from Comer. Solve Problems 6.7, 6.9, 7.14, 11.6 and 11.10.

PROBLEM 2 (40 pts)

(a) Suppose you are hired as CTO in your uncle's start-up company that aims to build a handheld device that can do everything that cell phones, smart phones, tablets and laptops can do w.r.t. network connectivity variety (e.g., data, voice, GPS, music streaming). Even just for voice, your phone should work when traveling to different parts of Europe where GSM is the norm, and parts of Asia where CDMA is used (in the U.S. we use both). For WiFi, keep in mind that there are three different systems deployed today. How many different types of network interfaces would you propose to your uncle (the CEO) as being needed? What frequency ranges will the devices operate in? Ignoring the cost factor of equipping a device with all the proposed air interfaces, what design and performance issues should your uncle be made aware of so that he can decide whether to plunge ahead or not? If you were tasked to trim down the number of proposed interfaces to a smaller, more meaningful number, what would they be? Explain your choices.

(b) Suppose your uncle is stubborn and decides to plow ahead with his original plan with the interfaces you have identified. Suppose a Linux based platform (e.g., Android) is selected to program the services provided by the handheld. From a system programming perspective, since a network connection could be established over any of the air interfaces, how do you select in Linux which interface is to be used to transmit a bit stream? How does this relate to the often used quote, "everthing in UNIX is a file"? Each air interface will come with a hardware address provided by the manufacturer. Should Linux configure each of the interfaces with their own IP addresses? If IP addresses are assigned, does it make sense for some of them to be private? Discuss your reasoning.

PROBLEM 3 (80 pts)

(a) You will find a client/server application under /u/u3/park/pub/; the client program is `client.c` and the server program is `server.c`. Compile, run (both client and server must run on the same host), and check their behavior. The server binary should be executed first (execute the server and client from different windows). Use `script` to record the output. Reverse-engineer the code and explain what the server and client are doing. List all the system calls used and explain their role. Which of the "system calls" is not a true system call—a system call traps to the kernel—but a wrapper library function that then makes a system call?

(b) The system call library in Linux is the lowest level interface to the kernel that invokes its services on behalf of processes and threads. Check the version of Linux that the lab machine you are using is running and determine where the system call library is defined. Suppose you don't like how one of the system calls is implemented (say `read`()) and you would like to modify the code and customize it to how you like it. To do so, do you have to recompile the kernel or is there a simpler way of doing so? What will happen to legacy apps (i.e., binaries) that have been compiled with the old system library that contains the old `read`() system call? Would they have to be recompiled? Explain your reasoning.

(c) Perl, PHP, shell scripts, Python, and Java, among many others, are interpreted languages where code written in these languages is not compiled and run as binaries. Instead, they are interpreted—i.e., processed—by software (essentially a virtual machine or simplified variants thereof) that then carry out the requested function in accordance with the language's semantics and syntax. For example, in the case of Perl, the Perl interpreter is software written in C that is compiled into machine code, that then takes as input Perl code (just text), interpretes it, and executes machine instructions in accordance with the interpretation. To implement the provided client/server C code in an interpreted environment, what features would be needed in the interpreted language? What are some benefits of implementing the client/server code in an interpreted environment? What are the negatives? Does using interpreted languages bypass system call libraries when interacting with a kernel? Explain your reasoning.

(d) Pick any of the interpreted languages indicated above. Can the client/server C code be ported to the interpreted platform so that it behaves the same as the compiled C counterpart? Explain your reasoning.

PROBLEM 4 (60 = 20 + 40 pts)

(a) The `read()` systems calls in both server and client code of Problem 3 implement synchronous, blocking I/O. This means that if the FIFO IPC (inter-process communication) buffer is empty then the calling process is checkpointed, context-switched out, a different ready process is context-switched in, and only when the FIFO becomes non-empty is the context-switched out process returned to ready state and context-switched back in by the kernel at a future time. Modify both client and server code so that the `read()` system calls implement synchronous, nonblocking I/O. Rerun the test of Problem 3(a) and show that its behavior is equivalent to the synchronous, blocking I/O version with respect to function. With respect to performance, which is preferable and why? Can you think of hardware/software environments where the synchronous, nonblocking version makes sense, or perhaps, even necessary?

(b) Modify the server and client of Problem 3(a) such that the server listens to requests on two FIFOs (call the second FIFO `.oracle_fifo_two`) and have a second client process send its request to the second FIFO of the server process. Make the FIFO name a command-line argument of the client so that the same binary can be executed for different destination FIFOs. Going back to synchronous, blocking I/O, the server code needs to be modified since there are two FIFO descriptors that it needs to monitor, but calling `read()` to block on one of the two FIFOs won't do since a request may be arriving on the other FIFO. Use the `select()` system call in Linux to extend the server's synchronous, blocking I/O to multiple—in out case, two—FIFOs. Rerun the tests twice, once running the client process that sends a request to the old FIFO first, followed by the client that sends its request to the second FIFO, and in the second test run, reversing their order. Submit the modified client/server code and trace of run-time tests recorded by `script`.