CS 422 (Park) **Assignment II** **Due: Feb. 7 (Mon.), 2011**

Submission instructions: Please type your answers and submit electronic copies using `turnin` *by 4pm 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. Use* `gnuplot` *to plot graphs.*

PROBLEM 1 (30 pts)

Read Chapters 5–8 and 10 and 11 from Comer. Solve Problems 6.3, 7.7, 7.10, 8.7, 11.4, and 11.10.

PROBLEM 2 (30 pts)

Suppose you work at a venture capital firm and you are asked to evaluate a business plan by a start-up that proposes to use TDM using a 1 THz sinusoid with amplitude modulation to provide high-speed wireless Internet access to housing subdivisions in suburbia. What are all the essential issues on your checklist that the business plan must satisfactorily address—focus on those that are technical in nature—before you would consider making a positive recommendation? Note that even focusing on technical issues, the list won't be short. For each issue, briefly explain why you consider it essential for potential success.

PROBLEM 3 (30 pts)

What are Fourier's key insight and technical accomplishment as they relate to modern broadband communication? Why is the original 18th century framework insufficient for realizing FDM-based multi-user communication? What is OFDM and why is it considered superior to old style guard bands for making FDM work?

PROBLEM 4 (50 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) FIFO is but one IPC (inter-process communication) mechanism in Linux/UNIX that can be used to facilitate communication between client and server processes. What are alternative methods that are feasible? What are their pros/cons compared to FIFO?

(c) Why is it essential for the server to use `dup2` to do its job correctly and efficiently? Is there a way to achieve the same outcome without using `dup2`?

(d) The example server is a *concurrent* server that forkes child processes to carry out the actual task that clients have requested. An *iterative* server carries out a task itself without delegating it to a child process. What are the pros/cons of the two server designs? Give two examples of servers (describe the services they provide) that may be implemented as iterative servers. Is it meaningful to make `server.c` into an iterative server? Explain.

PROBLEM 5 (40 pts)

Rewrite the client/server application (currently running on a single host communicating using FIFO; we will change it later to run on different hosts communicating over TCP/IP sockets) such that the client can request any command (not just `finger`) to be executed by the server and the result returned to the client. The client program, call it,

`client.bin`, accepts the command to be executed by the server as a command-line argument. That is,

% `client.bin <command>`

where `<command>` may itself include one or more command-line arguments (e.g., `ls -l -a`). The client's request "packet" should be backward-compatible (a key requirement in most real-world systems), meaning that when the client is invoked without a command then a request sent to the original server (i.e., the one where `finger` is hard-coded) works as before. When a command is specified, define an extended request format that builds on the legacy format such that it is correctly parsed by the new server that handles arbitrary command requests. You will also need to consider technical details such as what function in the `exec` family (`execlp` may not be the most convenient call) to invoke. Perform benchmark runs with `ls -l -a`, `ps -g -u -x`, and three other commands of your choice each with four or more command-line arguments. Log the interaction using `script`.

PROBLEM 6 (40 pts)

An extension of Problem 5, critically examine your server code and evaluate whether it is DoS (denial of service) attack-proof. DoS, in our context, will mean for a client to send a request to the server that makes it behave "irregularly" such as exhibiting incorrect run-time behavior including crashing. If your server code is deemed not DoS attack-proof, discuss why, and modify it so that it is. Demonstrate using test cases that it is able to withstand DoS attacks that you deem relevant. Conversely, show how it misbehaves without the DoS protection features. The TAs will be performing their own DoS attacks when evaluating your server code. Those that fail especially miserably may be subject to special penalties.