CS 422 (Park) **Assignment I** **Due: Feb. 2 (Wed.), 2005**

Submission instructions: Please type your answers and submit electronic copies using `turnin` *by 5pm 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 or ps format that uses standard fonts (a practical test is to check if the pdf/ps file prints on a CS Department printer without problem). 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. Use* `ps2gif` *to convert a eps/ps plot to gif format (e.g., for inclusion in Word).*

PROBLEM 1 (25 pts)

Read Chapters 1, 2 and 4 from Comer. Access `http://www.greatachievements.com/` and read the parts on Radio and Television, Telephone, and Internet. Give a 1-page summary of the milestones and key events that shaped their evolution, including your own views on what you think were important issues (some not discussed in the articles) that contributed to their state today. Then give a 1-page evaluation of the fundamental differences between the three systems. How does this reconcile with today's "convergence" theme where the Internet is supposed to gobble up different communication media and put them under one roof? Will Radio and Television be push-overs?

PROBLEM 2 (60 pts)

Carry out the experiments specified in `http://www.cs.purdue.edu/∼park/cs422-hw1-05s.html`. Consult the TA notes (`http://www.cs.purdue.edu/homes/wspeirs/422`) for additional comments and information.

PROBLEM 3 (25 pts)

`read`() and `write`() system calls are, by default, *blocking* calls. What does this mean? How can they be made *nonblocking*? Why would one want to make them nonblocking? `sigaction`() and `signal`() are system calls that register a signal handler—a call-back function—with the kernel so that the provided handler is invoked when the corresponding signal is raised. How are `sigaction`() and `signal`() different? Why, all else being equal, is it preferred to use `sigaction`(), especially when considering portability across different UNIX flavours? `read`() and `write`() are also called *slow* system calls. What does this technically mean? What happens to `read`() when it's in the middle of a lengthy read operation and suddenly, in the middle, a signal is raised? Where will the kernel return the user application (i.e., program counter) after the signal handler has finished? Will `read`() have read anything? What signals can be caught—a process can register a handler with the kernel to intercept it—and what signals cannot be caught? (It's easier to list the ones that cannot.) Why would allowing all signals to be catchable be a bad idea? What are the three main IPC (inter-process communication) mechanisms provided by Linux/UNIX? What are their pros/cons? If speed is of the essence, which one is fastest? What happens when `fork`() is executed? How do the child and parent processes know who is who? What properties does the child process inherent from its parent? What properties are not inherited? What does `exec`() do, and why is it a good idea to not lump `fork`() and `exec`() into a single system call? Which one in the `exec`() family is the true system call (the others are library calls)?

PROBLEM 4 (25 pts)

You will find a client/server application under ∼park/pub/cs422; the client program is `client.c` and the server program is `server.c`. Reverse-engineer the code and explain what the server and client are doing. Give a line-by-line explanation of the code—you may group several lines into one unit, as you see fit, and give a modular description. The important criterion is that your answer makes clear that you understand all aspects of the code. If you find inefficiencies, indicate what and why, and suggest possible changes. Include in your answer a clear explanation of what Linux/UNIX system programming trick is being used to direct the output of the server back to the client and how this works. Compile the client/server application and then test it by first running the server process `server.bin` in the background followed by the client process `client.bin` in the foreground. Use the `script` facility to record the run-time session. Submit a copy of the session script along with your write-up.