**CS 422 (Park)** **Assignment II** **Due: Feb. 16 (Wed.), 2005**

*Submission instructions: Please type your answers and submit electronic copies using* `turnin` *by 11:59pm 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** (15 pts)

Read Chapters 5, 6, and 7 from Comer. Pick one problem from each chapter (except Problems 5.4, 6.9 and 7.1) and solve them.

**PROBLEM 2** ($70 = 40 + 15 + 15$ pts)

**(a)** As a continuation of Problem 4, Assignment I, extend the client/server application such that any Linux command can be requested by the client to be executed by the server and its output returned to the client. The request format should be of the form

*process-ID # command # argument-*1 *# argument-*2 *# ⋯ # argument-n*

where *argument-1, argument-2, ...*, *argument-n* represent command-line arguments (possibly empty), and "#" is a delimiter symbol. The server, before parsing the request message from the client, should first do a length check and determine if the total request size is bigger than 22 bytes. If so, the client's request is rejected and the command

`echo` 'your request is too long'

executed instead. Test your program by first running the server application in the background, then executing `client.bin` multiple times with `date`, `host`, `who`, `ps`, `ls -l ps -gaux`, `ls -a -l -i`, `echo whatsmymaxlen?`, `ls -a -c -d -i -l -q -r -t -u -l`, and `terve`. Use `script` to record the run-time session. Submit your code and output. You must provide adequate documentation in your code. (*Remark: Make use of string processing library functions to minimize the parsing effort.*)

**(b)** Why should the length of arguments from client requests be first checked before they are copied to anywhere else in user space? In part (a), a length check of 22 bytes is instituted. For your favorite shell (if you don't have one, pick `bash` or `tcsh`), find out what the maximum input length is that it will support (note that a shell is just another server program, albeit a very useful one). What happens if the maximum length is exceeded?

**(c)** A concurrent server is one that receives a client request, interpretes it, then spawns a child process to handle the actual request. An iterative server is one that does all the chores, including execution of a client request, itself. Can the remote command execution server be designed as an iterative server? If so, how? If not, why not? What is a *daemon* (there is an "a") and how would we make the server process into one?

**PROBLEM 3** (20 pts)

Assume you have a very noisy channel (e.g., wireless) where the probability of a bit flipping is given by $\varepsilon \geq 0$. The source is a binary (i.e., $\Sigma = \{0, 1\}$) with $p_0 = 0.5$, $p_1 = 0.5$. Assume that to achieve reliable transmission you employ a straightforward redundancy scheme or code $F$ where $F(a) = a^n$ for $a \in \{0, 1\}$ where "$a^n$" stands for the $n$-fold repetition of the symbol $a$. Here $n$ is odd. Upon receiving a code word $w$ of length $n$, decoding (i.e., $F^{-1}$) is performed by taking a majority vote of the 0s and 1s in $w$. That is, $F^{-1}(w) = 0$ iff the number of 0s is larger than the number of 1s in $w$ (and vice versa for 1). Assume the *stringency of reliable transmission*—the probability of a symbol being correctly decoded at the receiver—is given by a parameter $0 < \delta < 1$. Is $\delta = 1$ (perfect reliability) achievable? Explain your reasoning. In general, to achieve reliability $\delta$, how large does $n$ need to be? Show your derivation and estimation. Compute $n$ for $\delta = 0.9, 0.999, 0.999999, 0.999999999$. What is the per-bit reliability—increase in reliability gained by incrementing $n$ by 1—of this coding scheme? Would you say this code is efficient or desirable?

**PROBLEM 4** (20 pts)

Consider the broadband communication example discussed in class where there are 5 sender-receiver pairs, each sender transmits a single bit, and the users are assigned frequencies 1 Hz, 2 Hz, 3 Hz, 4 Hz, and 5 Hz to perform their transmissions concurrently. We assume the sender and receiver have each a clock that are sufficiently synchronized, and tick at the granularity of 1 second. Each user aims to achieve a data rate of 1 bps. Suppose amplitude modulation (AM) is used to represent a bit, with amplitude 5 representing a 0 and amplitude 10 representing a 1. Each sender uses a sine curve whose value is 0 at time 0. Using `gnuplot`, plot for each sender what he/she transmits assuming senders 1–5 wish to transmit the bits 1, 0, 0, 1, 0, respectively. Plot what each receiver receives—note that they all "see" the same "mixed" signal. First, can you, if you were a receiver, by visual inspection alone guess what your bit might be? Test your visual powers by asking a friend to randomly choose a different 5-bit sequence, then without knowing what they are, infer the bits being transmitted. Then, without relying on eyesight and the limited powers of your brain, nor resorting to calculating the spectrum using Fourier analysis, can you devise a hands-on scheme—you won't find it in any book—that may allow each user to retrieve their bit from the "jumbled mess," even if there were 100 users each using a different frequency? Put your idea into action and show how it works using `gnuplot`.