CS 536 Fall 2019

Lab 1: System Programming Review [200 pts]

Due: 09/16/2019 (Mon), 11:59 PM

Objective

The objective of this introductory lab is to review and practice C system programming which is the foundation for network/socket programming. If you are rusty, please use this practice period to bring yourself up-to-speed on system level C programming. Utilize the PSOs and office hours to resolve any questions. The goal of this lab is to understand the structure and implementation of concurrent client/server apps which comprise the bulk of networked applications. This lab is an individual effort. Please note the assignment submission policy specified on the course home page.

Reading

Read chapter 1 from Peterson & Davie (textbook).

Problem 1 [40 pts]

The directory

/homes/park/pub/cs536/lab1 (currently pointing to /u/riker/u3/park/pub/cs536/lab1)

accessible from our lab machines in LWSN B148 and HAAS G056 contains simsh.c which implements a prototypical concurrent server: a simple, minimalist shell. A concurrent server receives and parses a client request -- in this case, from stdin (by default keyboard) -- then delegates the actual execution of a requested task to a worker process or thread. Create a directory, lab1, somewhere under your home directory, and a subdirectory v1/ therein. Copy simsh.c to v1/, compile, run and check that you understand how it works. Then modify simsh.c, call it simsh-v1.c, that accepts command-line arguments. That is, whereas simsh.c can handle "ls" and "ps", it cannot handle "ls -l -a" and "ps -gaux". Your modified code, simsh-v1.c, handles command-line arguments by parsing "ls -l -a" into string tokens "ls", "-l", "-a" and passing them to execvp() as its second argument (an array of pointers to the parsed tokens). Compile, test, and verify that your code works correctly. Annotate your code so that a TA can understand what you are aiming to do.

Problem 2 [60 pts]

Create a subdirectory v2/ under lab1. Modify your code, simsh-v1.c, so that instead of accepting client requests from stdin (i.e., human typing on keyboard in v1/) the request is communicated from a client process running program fifocli-v1.c. The server process running fifoser-v1.c creates a communication channel in the form of a FIFO (i.e., named pipe) using mkfifo(). Name the FIFO created by mkfifo() "server_queue". The client process, started after running the server process (run the two processes in separate windows for ease of observation), uses open() to open the FIFO created by the server and sends its request using write(). For example, the bytes of the string "ls -l -a". To indicate end-of-request, the client transmits EOS (end-of-string) character '\0' (ASCII NUL). The server, fifoser-v1.c, reads the byte stream using read() and upon seeing '\0' treats it as end of the client

request. The bytes are stored in a char array and parsed analogously to simsh-v1.c in Problem 1. The server forks a child and delegates the work of executing the client's request via execvp(). Unlike in simsh-v1.c, the FIFO server does not call waitpid() to block until its child terminates. Instead, it returns to the beginning of the while-loop to wait for the next client request. The client program, fifocli-v1.c, reads the request to send to the server from stdin. For simplicity, the client process accepts one request via stdin then terminates after forwarding the request to the server. Compile, test, and verify that your code works correctly. Make sure to annotate your code.

Problem 3 [50 pts]

Create a subdirectory v3/. In Problem 2, a client's request to execute a legacy binary (e.g., "ls") is performed by a worker process (i.e., child) and the result output to stdout. With server and client processes running in different windows, the output is displayed in the server's window. Modify the code of Problem 2 using dup2() so that the output of the legacy app is shown in the client process's window. That is, legacy apps such as "ps" and "date" are coded so that their output is directed to stdout which, by default, is the display (in windowing systems the window associated with a foreground process). Your modified server, fifoser-v2.c, opens a new FIFO, "client_queue" created by the client fifocli-v2.c. The client process uses its own FIFO created using mkfifo() to receive the response to its request from the server. The server uses dup2() to redirect stdout in the child process to the client's FIFO so that the output of "ls" is sent to the client process. The client process reads from its FIFO byte-by-byte using read() and after the last byte has been read and stored in a char array, outputs the content to its stdout which outputs the server's response in the client's window. Inspect the output written to stdout by legacy apps such as "ls" and "ps" to determine if a special character is written at the end that fifocli-v2.c can use to determine termination of the response message from the server. Otherwise, the client might indefinitely block on read() from its FIFO. Explain your solution for detecting end-of-transmission by the server by adding comments at the header of fifocli-v2.c. For simplicity, the client still handles a single request and terminates upon output of the server response to stdout. Compile, test, and verify that your code works correctly. Make sure to annotate your code.

Problem 4 [50 pts]

Create a subdirectory v4/. Modify the server of Problem 3, call it fifoser-v3.c, so that upon receiving a client request, parsing it, but before spawning a child process, the parent process throws a coin and if it comes up heads ignores the request (i.e., drops the client request) and goes on to wait for the next request. When it decides to ignore a request, it outputs to stdout a message saying that it did so along with the client request (i.e., string). The modified client, fifocli-v3.c, expects the server to be unreliable and retransmits a request if a response has not been received within 2 seconds. The client does so by registering a SIGALRM handler, void retxreq(int), using signal() at start-up and calling alarm() before writing the request to the server's FIFO. If the server's response arrives prior to the alarm going off (i.e., SIGALRM event being raised) then the client cancels the alarm. If not, retxreq() retransmits the request. Patience is not the client's strong suit. It keeps track of how many requests have been sent (inclusive the initial request), and if the count exceeds 3, gives up by printing a suitable message to stdout and terminating. Compile, test, and verify that your code works correctly. In this version, split the functions into multiple files and provide a Makefile to compile your client/server app. Make sure to annotate your code.

Bonus Problem [20 pts]

Explain the three meanings of "bandwidth" used in networking: bps, bandwidth of signal, bandwidth of communication medium (e.g., different make/quality copper wire, optical fiber). Visit the TA during the PSOs or office hours and use the laptop with the RTL-SDR USB dongle/antenna and CubicSDR software to tune into a FM radio frequency of your choice. Vary the cutoff bandwidth through the interface provided by CubicSDR and

gauge its impact on the resultant audio quality. Why are cutoff bandwidths that are too small or too large detrimental to the audio quality decoded from the electromagnetic waves used to carry analog data? See if you can do the same for aviation frequencies in the AM modulated aviation band (about 122 MHz to 136 MHz). How can SDR software such as CubicSDR using FFT spectrum output (in our case graphical waterfall) help determine a good cutoff bandwidth? Provide your answers in Lab1Answers.pdf under v5/. *Note: Using the RTL-SDR and CubicSDR software, discussing its workings and results, can be done as a group. If this is the case, please specify with whom you have collaborated on the Bonus Problem in Lab1Answers.pdf. The write-up should be in your own words.*

The Bonus Problem is completely optional and serves to provide additional exercises to help understand material discussed in class. The bonus points count toward reaching the 45% contribution of the lab component to the course grade.

Turn-in Instructions

Electronic turn-in instructions:

We will use turnin to manage lab assignment submissions. Go to the parent directory of the directory lab1/ where you deposited the submissions and type the command

turnin -v -c cs536 -p lab1 lab1

This lab is an individual effort. Please note the assignment submission policy specified on the course home page.

Back to the CS 536 web page