Network Centric Programming
Spring 2017 Final Project

Chat Room LED Display

By Vineet Sepaha, Cedric Blake, Brian Chu

Introduction

Our project the Chat Room LED Display is composed of three programs and a hardware LED display. The project displays use inputs from clients as scrolling messages on the LED display. User input is taken in by our client or inputs can even come from other kinds of data formatted into ascii strings from other programs that act as a client. A server program acts as an reciver and middleman between the clients and our third program the printer program. Clients or programs acting as clients will be given the hostname and port of the server application to establish a connection. The server will be listening on this port and receive these connections. Once a connection is established between server and client data from the client can be received by the server. A shared log between the server and printer program is how the server relays the information to be printed to the printer program. The printer program then can read from this shared log and translate the data into hardware signals to cause the inputs to appear on the LED hardware display.

Client Program

Our client program communicates to the server program using TCP stream sockets. Ports are chosen by the hoster of the server application and hence are only used as inputs along with the hostname to allow the client to establish a TCP connection with the server. Our client program will first prompt the user for a name and then read standard input from the terminal to receive a user name. Looping while prompting will then occur to read input strings from standard input. Newline token inherent to standard input are removed by parsing and the chosen username is appended to the front of the message. Now the message is sent directly over the TCP socket to the client as a null terminated ascii string . Messages take the form of username: message and are null terminated unless the message proves be the length of the

internal buffer. All subsequent messages once a username has been chosen are sent in this format. If the user types in exit/ in any form, the message will be forwarded to the server as the last message and termination signal before the client terminates itself. Our client program is a simple terminal based program based on standard input as described above and the server does not send feedback to clients individually. Thus our clients function similar to one way radios that only take and send user messages to the server. Other programs that take place of the client would need to preformat their data and if necessary convert it to null terminate strings before sending it to the server via an established tcp stream socket.

Server Program

The server program is responsible for taking input from multiple clients and writing to a file using a TCP stream sockets. When the server program runs, the user will have to specify the port the server will be listening on. The server will take connections from any IP address as long as they are connected to the specified port. The server will continue to accept client connections as long as the max amount of clients is not exceeded. Similar to how we executed the concurrent proxy, we are using pthreads to connect to multiple clients. The arguments for each thread such as the threadId and the connection file descriptor as passed in a struct called thread_data. Each client has its own port when it is connected to the server. When the server first runs, it truncates the file and receives input from each client that is connected to it. Each input will be appended to a file, or wait for the file lock to be freed before writing to the file. Once a user inputs “/exit” on the client side, the server will close the client connection and continue to wait for more connections while still accepting input from the other clients that are still connected to it.

Print Program

The printing system for the chat room consists of two parts, the hardware used to display each message and the software used to receive each message from the server. As for the Hardware that we are using for this project, we are using a Raspberry Pi 2 as our CPU. The Raspberry Pi is connected to a 32 by 32 LED display which is used for displaying the statements of each client connected to our server. The Raspberry Pi interfaces with the LED display with the use of an LED matrix hat for Raspberry Pi which connects the input pins of the LED matrix to the GPIO pins of the Pi. Finally, The way that we provide power to the display is by USB connection. We have cut a USB cable in half and soldered the power connections of the USB cable to the original power connections of the display itself. From there we can simply plug the USB cable into any USB port which provides enough current to drive the LED matrix, such as a laptop connection.

As for the software of our system. We wrote the display code in Python because the library created by Adafruit Industries was written in Python. The way that the Python script is able to communicate with the server script is through the use of a file in which we have called the Log File. Once the server writes messages to the Log File, the Python script will read from that same file and store that message into local variables in which are used to display the messages. Once we have stored these messages into the local variables, they are passed into a function for displaying the message on the LED display. This fuction also controls font size and color of the message, which we have predefined in our script. After the messages have been displayed, the code will wait for more messages to be written to the Log File by the server. This code is iterative and will run until the user decides to kill the program, in which they do so by pressing ctrl+c.

Example Message Exchange

Client Side

$ ./client localhost 10000
Enter a username:
Vineet
Username: Vineet

Enter Prompt:
Hello
sent
Enter Prompt:

Server Side

$ ./char-room-server 10000
Waiting for a new client connection...
Got a connection from a new client
Waiting for a new client connection...
Vineet: Hello

Appendix

Client Code - Written by Brian Chu

```
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/socket.h>
#include <errno.h>
#include <string.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <pthread.h>
#include <netinet/in.h>
#include <ctype.h>
#include <sys/time.h>
```

```
int  main(int argc, char **argv)
{

int socketfd;

if(argc!=3) //requires user hostname and port hence 3 args
   {

   fprintf(stderr,"Please give a  hostname port\n");
   exit(1);
   }


struct addrinfo info;
struct addrinfo * ptr;

memset(&info,0,sizeof(info));

info.ai_family = AF_INET;
info.ai_socktype = SOCK_STREAM;

if(getaddrinfo(argv[1],argv[2],&info,&ptr)!=0) //get our information of our server connection
   {
   perror("getaddrinfo");
   exit(1);

   }
if(ptr == NULL) //if its null its unable to connect
   {
   fprintf(stderr,"UNABLE TO CONNECT TO HOST!\n");
   exit(1);
   }

socketfd = socket(ptr->ai_family,ptr->ai_socktype,ptr->ai_protocol); //make a socket

if(socketfd == -1) //opps socket issues
   {
   perror("socket");
   exit(1);
   }
if(connect(socketfd,ptr->ai_addr,ptr->ai_addrlen)==-1) //connect to the server
   {
         close(socketfd);
         perror("connect");
         exit(1);

   }
```

```
freeaddrinfo(ptr); //clean up
int num;
int isize;
char buff[2000];
memset(buff,0,2000);
int nsize;
char namebuff[2000];
char bigbuff[5000];
memset(bigbuff,0,5000);
memset(namebuff,0,2000);
while(1)
{

   printf("Enter a username: \n");
   nsize = read(STDIN_FILENO,namebuff,sizeof(namebuff)); //check the input size
   if(nsize <= 0)
   {
         printf("we have a read error for your name please reenter it\n"); //continue looping on
error
         continue;
   }
   printf("Username: %s\n",namebuff);
   break;//breakin and leavin


}
strtok(namebuff,"\n");//remove newline
while(1)
{
   printf("Enter Prompt:\n");//promtping
   memset(buff,0,sizeof(buff));
   memset(bigbuff,0,sizeof(bigbuff));
   isize = read(STDIN_FILENO,buff,sizeof(buff)); //promtping
   if(isize == -1) //error test
   {
         fprintf(stderr,"an error occured while reading please see below and input your message
again");
         perror("read");
         continue;
   }

   strcat(bigbuff,namebuff);//add name
   strcat(bigbuff,": "); //add token
   strcat(bigbuff,buff);//add message
   strcat(bigbuff,"\n"); //add a new line at the end
   num = send(socketfd,bigbuff,strlen(bigbuff),0);//sending
   //num = send(socketfd,buff,isize,0);
```

```
    if(num == -1)//error
    {
        fprintf(stderr,"ERROR SENDING TO CHAT HOST\n");
        perror("send");
    }
    printf("sent\n");
    if(strstr(bigbuff,"exit/")!=NULL) //exit code
    {
        printf("detected exit now leaving\n");
        break;
    }
}
}
```

Server Code - Written by Vineet Sepaha

```
/* Server code */
#include "csapp.h"

#define OUTPUT_FILE "logfile.txt"
#define NUM_THREADS 25
#define BUFFER_SIZE 160

// Thread data structure to keep track of arguments passed to each thread
typedef struct thread_data {
        int connfd;
        int threadId;
} thread_data;

pthread_mutex_t file_lock;

//This will be used to thread the program once it runs iteratively
void* processRequest(void* t){

        //Unpack the data
        thread_data* t_data = (thread_data*)t;
        int connfd = t_data->connfd;
        int threadId = t_data->threadId;

        // Read from the client connection
        char textBuf[BUFFER_SIZE];
        int readLen;

        while ( (readLen=Read(connfd, textBuf, sizeof(textBuf))) > 0){
                if(!strstr(textBuf, "/exit")){
```

```
                        // Open the file we will be writing to (a+ means to append instead of w
which overwrites)
                        pthread_mutex_lock(&file_lock);
                FILE* fp = Fopen(OUTPUT_FILE, "a+");
                        printf("%s",textBuf);
                        // Write to the output file
                        Fwrite(textBuf, sizeof(char), readLen, fp);
                        bzero(&textBuf, sizeof(textBuf));
                        Fclose(fp);
                        pthread_mutex_unlock(&file_lock);
                } else {
                        printf("Finished reading from the client %d\n", threadId+1);
                        break;
                }
        }
        Close(connfd);
        return 0;
}

int main(int argc, char* argv[]){

        //Declarations
        int port; // Listening port
        int sockfd; // The socket file descriptor
        int enable = 1; // Used for setsockopt
        int threadId = 0; // How to reference elements in the thread array

        pthread_t threads[NUM_THREADS]; // Array that stores the threads
        thread_data t_data[NUM_THREADS]; // Corresponding struct that hold thread data

        pthread_mutex_init(&file_lock, NULL); // Initialize the file lock

        // Empty out the file
        FILE* reset = Fopen(OUTPUT_FILE, "w");
        Fclose(reset);


        /* Check arguments */
    if (argc != 2) {
       fprintf(stderr, "Usage: %s <port number>\n", argv[0]);
       exit(0);
    }

    // Get the port the server will be listening on
    port = atoi(argv[1]);

        // Socket
        sockfd = Socket(AF_INET, SOCK_STREAM, 0);
```

```
	// This allows for constant reuse of the port
	if (setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &enable, sizeof(int)) < 0)
	perror("setsockopt: ");

	// Bind
	struct sockaddr_in servaddr;
	bzero(&servaddr,sizeof(servaddr));
	servaddr.sin_family=AF_INET;
	servaddr.sin_addr.s_addr=htonl(INADDR_ANY);
	servaddr.sin_port=htons(port);

	Bind(sockfd, (struct sockaddr*)&servaddr, sizeof(servaddr));

	// Listen
	Listen(sockfd, LISTENQ);

	for(;;threadId++){
		// Accept a connection
		printf("Waiting for a new client connection... \n");
		int connfd;
		connfd = Accept(sockfd, NULL, 0);
		printf("Got a connection from a new client\n");

		if(threadId <= NUM_THREADS){
			t_data[threadId].connfd = connfd;
			t_data[threadId].threadId = threadId;
			Pthread_create(&threads[threadId], NULL, processRequest,
(void*)&t_data[threadId]);
		} else {
			printf("Thread Limit Reached\n");
			threadId = 0;
			int i;
			for(i = 0; i < NUM_THREADS; ++i){
				Pthread_join(threads[i], NULL);
			}
		}
	}
	Close(sockfd);

	return 0;
}
```

Print Program Code - Written by Cedric Blake

```
#!/usr/bin/env python
# Display a runtext with double-buffering.
from samplebase import SampleBase
from rgbmatrix import graphics
import time

fileName = "logfile.txt" #name of file written by server

messages = 3 #number of messages that are displayed at once

#function for comparing the length of 3 strings
def threeLenCmp(len1, len2, len3):
    if(len1 > len2):
        if(len1 > len3):
            return 1;
        else:
            return 3
    elif(len2 > len3):
        return 2
    else:
        return 3

class RunText(SampleBase):
    def __init__(self, *args, **kwargs):
        super(RunText, self).__init__(*args, **kwargs)

    def Run(self):
        #initialize message position
        offscreenCanvas = self.matrix.CreateFrameCanvas()
        #initialize and set font size
        font = graphics.Font()
        font.LoadFont("../../fonts/7x13.bdf")
        #set prefered text colors
        textColor1 = graphics.Color(255, 155, 0)
        textColor2 = graphics.Color(255, 0, 155)
        textColor3 = graphics.Color(0, 155, 255)
        pos = offscreenCanvas.width
        fd = open(fileName, 'r') #open the log file
        #print(self.args)

        logLine1 = fd.readline() #attempt to read line, if end of file, "" is given
        logLine1 = logLine1[:-1]
        logLine1 = logLine1[:-1]
```

```
        logLine2 = fd.readline() #attempt to read line, if end of file, "" is given
        logLine2 = logLine2[:-1]
        logLine2 = logLine2[:-1]

        logLine3 = fd.readline() #attempt to read line, if end of file, "" is given
        logLine3 = logLine3[:-1]
        logLine3 = logLine3[:-1]

        #this while loop will continuously check for new messages and will display the
        #new messages as they are written to the log file by the server connection
        while True:

            offscreenCanvas.Clear()
            len1 = graphics.DrawText(offscreenCanvas, font, pos, 10, textColor1, logLine1)
            len2 = graphics.DrawText(offscreenCanvas, font, pos, 20, textColor2, logLine2)
            len3 = graphics.DrawText(offscreenCanvas, font, pos, 30, textColor3, logLine3)

            longLen = threeLenCmp(len1, len2, len3);
            pos -= 1
            #this if/else statement is used to manage when we stop attempting to display to the
matrix
            #it tests for the message with the longest length, then prints up to that many characters
            #this is so that messages do not get truncated during display
            if (longLen == 1):
                if (pos + len1 < 0):
                    pos = offscreenCanvas.width
                    logLine1 = fd.readline() #read another line
                    logLine1 = logLine1[:-1]
                    logLine1 = logLine1[:-1]

                    logLine2 = fd.readline()
                    logLine2 = logLine2[:-1]
                    logLine2 = logLine2[:-1]

                    logLine3 = fd.readline()
                    logLine3 = logLine3[:-1]
                    logLine3 = logLine3[:-1]
            elif (longLen == 2):
                if (pos + len2 < 0):
                    pos = offscreenCanvas.width
                    logLine1 = fd.readline() #read another line
                    logLine1 = logLine1[:-1]
                    logLine1 = logLine1[:-1]

                    logLine2 = fd.readline()
                    logLine2 = logLine2[:-1]
                    logLine2 = logLine2[:-1]
```

```
                logLine3 = fd.readline()
                logLine3 = logLine3[:-1]
                logLine3 = logLine3[:-1]
            else:
                if (pos + len3 < 0):
                    pos = offscreenCanvas.width
                    logLine1 = fd.readline() #read another line
                    logLine1 = logLine1[:-1]
                    logLine1 = logLine1[:-1]

                    logLine2 = fd.readline()
                    logLine2 = logLine2[:-1]
                    logLine2 = logLine2[:-1]

                    logLine3 = fd.readline()
                    logLine3 = logLine3[:-1]
                    logLine3 = logLine3[:-1]

            time.sleep(0.05)
            #reset message position bach to right side of display
            offscreenCanvas = self.matrix.SwapOnVSync(offscreenCanvas)
        fd.close()


# Main function
if __name__ == "__main__":
    parser = RunText()
    if (not parser.process()):
        parser.print_help()
```