CprE 489, Section 4
Lab Experiment #2: TCP Sockets Programming

Sean Gordon & Noah Thompson

Experience:

This lab was a small introduction to the software workings of network connections, introducing some basic functionality with pre-set lab configurations. While both partners had varying experience with socket connections before, this was a valuable experience with port connections. Neither partner had built a server to send information over a specific port before, with most experience being through http protocols.

The use of ruptime as the focus point was a nice introduction to sending simple but useful information over socket connections. With the use of test servers already set up to facilitate the lab, testing commands for the client interface went smoothly. When building the server, we also learned just how much various information is compiled in the ruptime command, and how long it would take to assemble manually.

Approaches:

The two approaches for compiling server information to send to the client were:

1. Call the various commands that hold information for ruptime, then assemble them into a single string to send back to the client.
2. Use a child process combined with execvp to call the uptime command and send that response back to the client.

Effort:

Both group members provided even levels of effort towards the completion of the lab. The programs were designed separately, but in parallel, with each group member helping the other when something was found or the other was lagging behind.

Sean: 50% || Noah: 50%

Code:

```
//Sample Client-Side Code in TCP Sockets Programming:
//====================================================

#include <string.h>
#include <stdlib.h>
#include <stdio.h>

#include <sys/types.h>
#include <sys/socket.h>

#include <netinet/in.h>



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


    struct sockaddr_in remoteaddr;

    char *SERVER_IP;
    int SERVER_PORT;

    //We're looking for IP address and port only
    if( argc == 3 )
    {
        SERVER_IP = argv[1];
        SERVER_PORT = atoi(argv[2]);
    }
    else{

        printf("Yo chief we got a problem\n");
        printf("You only passed %d arguments\n", argc);
        printf("Falling back to default values\n");

        SERVER_IP = "192.168.254.1";
        SERVER_PORT = 42069;
    }
```

```
    int clisock;
    if ( (clisock = socket(PF_INET, SOCK_STREAM, 0)) < 0 ){
        perror("socket() error:\n");
        exit(1);
    }

    remoteaddr.sin_family = PF_INET;
    remoteaddr.sin_port = htons(SERVER_PORT);
    remoteaddr.sin_addr.s_addr = inet_addr(SERVER_IP);


    connect(clisock, (struct sockaddr *) &remoteaddr,
                                        sizeof(remoteaddr));
    // communication between client and server starts here

    char readBuffer[1000];
    read(clisock, readBuffer, sizeof(readBuffer));

    // communication between client and server ends here
    close(clisock);

    printf("%s\n", readBuffer);

    return 0;
}
```

```
//Sample Server-Side Code in TCP Sockets Programming:
//=====================================================

#include <sys/types.h>
#include <sys/socket.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>

#include <netinet/in.h>

#include <sys/sysinfo.h>
#include <linux/unistd.h>

#include <time.h>


int main(){

    char *IP_ADDRESS = "192.168.254.7";
    int PORT = 40404;

    char *writeBuffer = "Bruuhh";


    struct sockaddr_in serveraddr, clientaddr;
    int len = sizeof(clientaddr);
    int sersock, consock;

    if ( (sersock = socket(PF_INET, SOCK_STREAM, 0)) < 0 ){
        perror("socket() error:\n");
        exit(1);
    }

    serveraddr.sin_family = PF_INET;
    serveraddr.sin_port = htons(PORT);
    serveraddr.sin_addr.s_addr = inet_addr(IP_ADDRESS);
    // OR: serveraddr.sin_addr.s_addr = htonl(INADDR_ANY);
```

```
    bind(sersock, (struct sockaddr *) &serveraddr,
                                        sizeof(serveraddr));
    listen(sersock, 10);


    while(1){
        consock = accept(sersock, (struct sockaddr *) &clientaddr,
                                                        &len);
        // communication between server and client starts here

        //Grabbing uptime information ------------------------------

        //Fork a child to handle connection
        if(fork() == 0){

            char* args[] = {
                "uptime",
                NULL
            };
            //Duplicate the file descriptor so we can send info to
            //the client
            dup2(consock, 1);

            //Execute uptime command and exit, sending it to client
            execvp("uptime", args);
            exit(0);

        }

        // communication between server and client ends here
        close(consock);
    }
    close(sersock);

    return 0;
}
```