**client.c Code**

///////////////////////////////////////////////////////

//To Compile: gcc -o client client.c -lnsl -lsocket

//To Run: ./client ServerName PortNumber

// eg., ./client bgunix 45656

//Programmer: Douglas G Calmes

///////////////////////////////////////////////////////

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <netdb.h>

////////////////////////////////////////

// This is the same function as in

// the server.

////////////////////////////////////////

void error(const char \*msg)

{

perror(msg);

exit(0);

}

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

{

int sockfd, portno, n;

struct sockaddr\_in serv\_addr; // serv\_addr - Address of the server we want to connect to

struct hostent \*server; // pointer to the hostent structure

char buffer[256];

if (argc < 3) {

fprintf(stderr, "usage %s hostname port\n", argv[0]);

exit(0);

}

// This is the same lines of code from the server!

portno = atoi(argv[2]);

sockfd = socket(AF\_INET, SOCK\_STREAM, 0);

if (sockfd < 0)

error("ERROR opening socket\n");

// End of the same code!

// Set the server to the host on the Internet.

// If it returns NULL: Tell user they cannon find the host

server = gethostbyname(argv[1]);

if (server == NULL) {

fprintf(stderr, "ERROR, no such host\n");

exit(0);

}

// Set the fields in serv\_addr

// The majority of the fields will be the same as the server.

// Use bcopy() because server->h\_addr is a character string.

// This will copy the length bytes from s1 to s2.

bzero((char \*) &serv\_addr, sizeof(serv\_addr));

serv\_addr.sin\_family = AF\_INET;

bcopy((char \*)server->h\_addr,

(char \*)&serv\_addr.sin\_addr.s\_addr,

server->h\_length);

serv\_addr.sin\_port = htons(portno);

// connect() is called by the client in order to connect to the server.

// Arguements in connect(): socket file descriptor

// address of the host (connection)

// size of the address

// If the connect() succeedes it will return 0. If not it will

// return -1 and print out that it had an error connecting.

if (connect(sockfd,(struct sockaddr \*) &serv\_addr, sizeof(serv\_addr)) < 0)

error("ERROR connecting\n");

// Have the user enter a message.

printf("Please enter the message: ");

bzero(buffer, 256);

// Read the message from stdin.

fgets(buffer, 255, stdin);

// Read the reply from the socket and display it on the screen.

n = write(sockfd, buffer, strlen(buffer));

if (n < 0)

error("ERROR writing to socket\n");

bzero(buffer, 256);

n = read(sockfd, buffer, 255);

if (n < 0)

error("ERROR reading from socket\n");

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

close(sockfd);

return 0;

}

**server.c Code**

Script started on Thu Dec 08 22:13:03 2016

sh-4.3$ cat server.c

///////////////////////////////////////////////////////////////////

// To Comiple: gcc -o server server.c -lsocket

// To Run: ./server PortNumber

// (Portnumber can be any number between 2000 and 65535)

// e.g., ./server 45656

//

// Programmer: Douglas G Calmes

///////////////////////////////////////////////////////////////////

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

/////////////////////////////////////////////

// This function will be called if the

// system ends up failing. It will display

// display an error message then abort

// the program.

/////////////////////////////////////////////

void error(const char \*msg)

{

perror(msg);

exit(1);

}

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

{

////////////////////////////////////////////////////

// Variables:

// stockfd & newsockfd - file descriptors

// portno - stores the port number

// clilen - size of the address of the client

// n - return value for read() & write()

////////////////////////////////////////////////////

int sockfd, newsockfd, portno;

socklen\_t clilen;

char buffer[256]; // Buffer that contains characters from the socket connection

struct sockaddr\_in serv\_addr, cli\_addr;

int n;

// Print an error if there are less than two arguments

if (argc < 2) {

fprintf(stderr,"ERROR, no port provided\n");

exit(1);

}

// set sockfd and print an error if it is less than 0

sockfd = socket(AF\_INET, SOCK\_STREAM, 0);

if (sockfd < 0)

error("ERROR opening socket");

// Use two arguments, pointer to the buffer and size of the buffer

// to set all of the values in the buffer to 0. aka. initialize

bzero((char \*) &serv\_addr, sizeof(serv\_addr));

// Convert the string of digits into an integer

portno = atoi(argv[1]);

// Set sin\_family to AF\_INET

serv\_addr.sin\_family = AF\_INET;

// Contains the IP address of the host.

// This will always be the IP address, so we set it to INADDR\_ANY to get

// this address.

serv\_addr.sin\_addr.s\_addr = INADDR\_ANY;

// sin\_port contains the port number.

// Uses htons() to convert a port number from host byte order to network byter order.

serv\_addr.sin\_port = htons(portno);

// Report an error if these are less than 0.

// The most common failure is when the socket

// is already being used on this machine.

if (bind(sockfd, (struct sockaddr \*) &serv\_addr, sizeof(serv\_addr)) < 0)

error("ERROR on binding");

// This call will allow the process to listen on the socket for connections.

// It contains two arguments: socket file descriptor & size of backlog queue.

// The size of the backlog should be set to the maximum size.

// If the first argument is valid, it wont fail or check for errors.

listen(sockfd,5);

// We first set clilen to the size of cli\_addr.

// accept() system call is what causes the process to block. This will block until

// a client is connected to the server.

// Once connected, the process will be woken up.

// This call will return a new file descriptor.

// All communication on this connection will now use this new file descriptor.

// The other two arguments are a reference pointer to the address of the client on

// the other end of the connection & the size of the structure.

clilen = sizeof(cli\_addr);

newsockfd = accept(sockfd,

(struct sockaddr \*) &cli\_addr,

&clilen);

if (newsockfd < 0)

error("ERROR: on accept");

// This next step will happen once the client has connected to our server.

// The buffer will then be initialized using the bzero() function.

// It will then read from the socket.

// The updated file descriptor will be used here.

// read() will block until something for it to read shows up.

// It will continue to read until 255, or until it reads the total

// number of characters in the socket.

// It will then return the number of characters that were read.

bzero(buffer,256);

n = read(newsockfd, buffer, 255);

if (n < 0)

error("ERROR: reading from socket");

printf("Here is the message: %s\n", buffer);

// Write a message to the client of everything written by the server

n = write(newsockfd,"Message received!", 18);

if (n < 0)

error("ERROR: writing to socket");

close(newsockfd);

close(sockfd);

return 0;

}

