Week 14 Lab 1:

1. Updated timeServer.c and timeClient.c to take in port numbers as arguments

Text

Description automatically generated

1. Selected a port number, 31651, and tested if it’s free using netstat. Used that port number as an argument to timeServer.c and timeClient.c.
2. Lastly, used ps and kill to check and terminate server

Text

Description automatically generated

1. I wasn’t sure if we were supposed to do the above, or actually code it such that the code will find a free port number between 10000 and 50000, and then use that port number to do stuff, so I did both. I attached the listings (timeServer\_.c and timeClient\_.c) for better clarification of what I mean.

Graphical user interface, text

Description automatically generated

timeServer.c:

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <errno.h>

#include <string.h>

#include <sys/types.h>

#include <time.h>

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

{

**int** listenfd = 0, connfd = 0;

**struct** sockaddr\_in serv\_addr;

**char** sendBuff[1025];

time\_t ticks;

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

memset(&serv\_addr, '0', **sizeof**(serv\_addr));

memset(sendBuff, '0', **sizeof**(sendBuff));

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

serv\_addr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

serv\_addr.sin\_port = htons(atoi(argv[1]));

bind(listenfd, (**struct** sockaddr\*)&serv\_addr, **sizeof**(serv\_addr));

listen(listenfd, 10);

**while**(1)

{

connfd = accept(listenfd, (**struct** sockaddr\*)**NULL**, **NULL**);

ticks = time(**NULL**);

snprintf(sendBuff, **sizeof**(sendBuff), "%.24s\r\n", ctime(&ticks));

write(connfd, sendBuff, strlen(sendBuff));

close(connfd);

sleep(1);

}

}

timeClient.c:

#include <sys/socket.h>

#include <sys/types.h>

#include <netinet/in.h>

#include <netdb.h>

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <unistd.h>

#include <errno.h>

#include <arpa/inet.h>

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

{

**int** sockfd = 0, n = 0;

**char** recvBuff[1024];

**struct** sockaddr\_in serv\_addr;

**if**(argc != 3)

{

printf("\n Usage: %s <ip of server> <port #>\n",argv[0]);

**return** 1;

}

memset(recvBuff, '0',**sizeof**(recvBuff));

**if**((sockfd = socket(AF\_INET, SOCK\_STREAM, 0)) < 0)

{

printf("\n Error : Could not create socket \n");

**return** 1;

}

memset(&serv\_addr, '0', **sizeof**(serv\_addr));

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

serv\_addr.sin\_port = htons(atoi(argv[2]));

**if**(inet\_pton(AF\_INET, argv[1], &serv\_addr.sin\_addr)<=0)

{

printf("\n inet\_pton error occured\n");

**return** 1;

}

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

{

printf("\n Error : Connect Failed \n");

**return** 1;

}

**while** ( (n = read(sockfd, recvBuff, **sizeof**(recvBuff)-1)) > 0)

{

recvBuff[n] = 0;

**if**(fputs(recvBuff, stdout) == EOF)

{

printf("\n Error : Fputs error\n");

}

}

**if**(n < 0)

{

printf("\n Read error \n");

}

**return** 0;

}

timeServer\_.c

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <errno.h>

#include <string.h>

#include <sys/types.h>

#include <time.h>

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

{

**int** freePort;

**int** listenfd = 0, connfd = 0;

**struct** sockaddr\_in serv\_addr;

**char** sendBuff[1025];

time\_t ticks;

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

memset(&serv\_addr, '0', **sizeof**(serv\_addr));

memset(sendBuff, '0', **sizeof**(sendBuff));

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

serv\_addr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

**for**(**int** i = 10000; i < 50000; i++) {

FILE \*fp;

**char** path[1035];

**char** s[100];

**char** port[30];

sprintf(port, "%d", i);

strcpy(s, "netstat -aont | grep \"`hostname -i`:");

strcat(s, port);

strcat(s, "\" ");

fp = popen(s, "r");

**if**(fp == **NULL**) {

printf("Failed to run command.\n");

exit(1);

}

**if** (fgets(path, **sizeof**(path), fp) == **NULL**) {

freePort = i;

**break**;

}

pclose(fp);

}

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

bind(listenfd, (**struct** sockaddr\*)&serv\_addr, **sizeof**(serv\_addr));

listen(listenfd, 10);

**while**(1)

{

connfd = accept(listenfd, (**struct** sockaddr\*)**NULL**, **NULL**);

ticks = time(**NULL**);

snprintf(sendBuff, **sizeof**(sendBuff), "%.24s\r\n", ctime(&ticks));

write(connfd, sendBuff, strlen(sendBuff));

close(connfd);

sleep(1);

}

}

timeClient\_.c

#include <sys/socket.h>

#include <sys/types.h>

#include <netinet/in.h>

#include <netdb.h>

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <unistd.h>

#include <errno.h>

#include <arpa/inet.h>

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

{

**int** sockfd = 0, n = 0;

**char** recvBuff[1024];

**struct** sockaddr\_in serv\_addr;

**int** freePort;

**if**(argc != 3)

{

printf("\n Usage: %s <ip of server> <port #>\n",argv[0]);

**return** 1;

}

memset(recvBuff, '0',**sizeof**(recvBuff));

**if**((sockfd = socket(AF\_INET, SOCK\_STREAM, 0)) < 0)

{

printf("\n Error : Could not create socket \n");

**return** 1;

}

memset(&serv\_addr, '0', **sizeof**(serv\_addr));

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

**for**(**int** i = 10000; i < 50000; i++) {

FILE \*fp;

**char** path[1035];

**char** s[100];

**char** port[30];

sprintf(port, "%d", i);

strcpy(s, "netstat -aont | grep \"`hostname -i`:");

strcat(s, port);

strcat(s, "\" ");

fp = popen(s, "r");

**if**(fp == **NULL**) {

printf("Failed to run command.\n");

exit(1);

}

**if** (fgets(path, **sizeof**(path), fp) == **NULL**) {

freePort = i;

**break**;

}

pclose(fp);

}

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

**if**(inet\_pton(AF\_INET, argv[1], &serv\_addr.sin\_addr)<=0)

{

printf("\n inet\_pton error occured\n");

**return** 1;

}

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

{

printf("\n Error : Connect Failed \n");

**return** 1;

}

**while** ( (n = read(sockfd, recvBuff, **sizeof**(recvBuff)-1)) > 0)

{

recvBuff[n] = 0;

**if**(fputs(recvBuff, stdout) == EOF)

{

printf("\n Error : Fputs error\n");

}

}

**if**(n < 0)

{

printf("\n Read error \n");

}

**return** 0;

}