**Lab Assignment-1 Contents**

**Kanti**

**Gnanadeep**

**Usha**

**Program - 1**

**Readme**

1. Create a make file in the directory (Makefile.txt).

2. Use the command "make -f Makefile.txt".

3. Run the output file created from the "make" command.

4. ./<output file name in make file> <time-a.nist.gov/ip address>

5. Capture the output in ctime mode.

6. clean the output file using "make -f Makefile clean".

7. example : ./Daytime time-a.nist.gov

**Makefile**

all:

gcc -o Daytime UDPDaytime.c

clean:

rm -f Daytime

**Source code**

**UDPDaytime client program**

/\* UDPDaytime.c - main

Group-02 Assignment-01

Make file, Output file included

\*/

#define \_\_USE\_BSD 1

#include <stdarg.h>

#include <errno.h>

#include <netinet/in.h>

#include <time.h>

#include <arpa/inet.h>

#include <netdb.h>

#include <sys/socket.h>

#include <sys/types.h>

#include <unistd.h>

#include <stdlib.h>

#include <string.h>

#include <stdio.h>

#define BUFSIZE 64

#define UNIXEPOCH 2208988800 /\* UNIX epoch, in UCT secs : Number of seconds from1st January 1900 \*/

#define MSG "what time is it?\n"

#ifndef INADDR\_NONE

#define INADDR\_NONE 0xffffffff

#endif

extern int errno;

typedef unsigned short u\_short;

extern int errno;

int errexit(const char \*format, ...);

typedef unsigned long u\_long;

extern int errno;

/\*connectUDP - connect to a specified TCP service on a specified host \*/

/\*

\* Arguments:

\* host - name of host to which connection is desired

\* service - service associated with the desired port

\*/

int connectsock(const char \*host, const char \*service, const char \*transport);

int connectUDP(const char \*host, const char \*service);

int errexit(const char \*format, ...);

/\*------------------------------------------------------------------------

\* main - UDP client for TIME service that prints the resulting time

\*------------------------------------------------------------------------

\*/

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

{

char \*host = "localhost"; /\* host to use if none supplied \*/

char \*service = "time"; /\* default service name \*/

time\_t now; /\* 32-bit integer to hold time \*/

int s, n; /\* socket descriptor, read count\*/

switch (argc)

{

case 1:

host = "localhost";

break;

case 3:

service = argv[2];

case 2:

host = argv[1];

break;

default:

fprintf(stderr, "usage: UDPtime [host [port]]\n");

exit(1);

}

s = connectUDP(host, service);

(void) write(s, MSG, strlen(MSG));

/\* Read the time \*/

n = read(s, (char \*)&now, sizeof(now));

if (n < 0)

errexit("read failed: %s\n", strerror(errno));

now = ntohl((u\_long)now); /\* put in host byte order \*/

now -= UNIXEPOCH; /\* convert UCT to UNIX epoch \*/

printf("%s", ctime(&now));

exit(0);

}

/\*------------------------------------------------------------------------

\* connectsock - allocate & connect a socket using TCP or UDP

\*------------------------------------------------------------------------

\*/

int connectsock(const char \*host, const char \*service, const char \*transport )

/\*

\* Arguments:

\* host - name of host to which connection is desired

\* service - service associated with the desired port

\* transport- name of transport protocol to use ("tcp" or "udp")

\*/

{

struct hostent \*phe; /\* pointer to host information entry \*/

struct servent \*pse; /\* pointer to service information entry \*/

struct protoent \*ppe; /\* pointer to protocol information entry\*/

struct sockaddr\_in sin; /\* an Internet endpoint address \*/

int s, type; /\* socket descriptor and socket type \*/

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

sin.sin\_family = AF\_INET;

/\* Map service name to port number \*/

if ( pse = getservbyname(service, transport))

sin.sin\_port = pse->s\_port;

else if ( (sin.sin\_port = htons((u\_short)atoi(service))) == 0 )

errexit("can't get \"%s\" service entry\n", service);

/\* Map host name to IP address, allowing for dotted decimal \*/

if ( phe = gethostbyname(host) )

memcpy(&sin.sin\_addr, phe->h\_addr, phe->h\_length);

else if ( (sin.sin\_addr.s\_addr = inet\_addr(host)) == INADDR\_NONE )

errexit("can't get \"%s\" host entry\n", host);

/\* Map transport protocol name to protocol number \*/

if ( (ppe = getprotobyname(transport)) == 0)

errexit("can't get \"%s\" protocol entry\n", transport);

/\* Use protocol to choose a socket type \*/

if (strcmp(transport, "udp") == 0)

type = SOCK\_DGRAM;

else

type = SOCK\_STREAM;

/\* Allocate a socket \*/

s = socket(PF\_INET, type, ppe->p\_proto);

if (s < 0)

errexit("can't create socket: %s\n", strerror(errno));

/\* Connect the socket \*/

if (connect(s, (struct sockaddr \*)&sin, sizeof(sin)) < 0)

errexit("can't connect to %s.%s: %s\n", host, service,strerror(errno));

return s;

}

int connectUDP(const char \*host, const char \*service )

/\*

\* Arguments:

\* host - name of host to which connection is desired

\* service - service associated with the desired port

\*/

{

return connectsock(host, service, "udp");

}

/\*------------------------------------------------------------------------

\* errexit - print an error message and exit

\*------------------------------------------------------------------------

\*/

/\*VARARGS1\*/

int errexit(const char \*format, ...)

{

va\_list args;

va\_start(args, format);

vfprintf(stderr, format, args);

va\_end(args);

exit(1);

}

**Program - 2**

**Readme**

1. To Open the editor:

gedit client.c

2. To Compile to program

gcc client.c -o client

3. To Execute the program

./client time-c.nist.gov time-d.nist.gov

**Makefile**

.PHONY: client

all: client

client:

gcc client.c -o client

clean:

rm client

**Source code**

TCP client program

/\*

TCP Client program by group-02

Make file, code, output are provided

Reference: Internetworking with TCP/IP Vol. 3, Client-Server programming and applications, Comer and Stevens, Linux/POSIX Sockets version, ISBN: 0-13-032071-4, 2001.

\*/

#include <errno.h>

#include <netinet/in.h>

#include <time.h>

#include <unistd.h>

#include <stdarg.h>

#include <stdio.h>

#include <stdlib.h>

#define \_\_USE\_BSD 1

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <netdb.h>

#include <string.h>

#include <stdlib.h>

#ifndef INADDR\_NONE

#define INADDR\_NONE 0xffffffff

#endif /\* INADDR\_NONE \*/

/\* TCPdaytime.c - TCPdaytime, main \*/

#include <unistd.h>

typedef unsigned short u\_short;

extern int errno;

extern int errno;

#define LINELEN 128

int TCPdaytime(const char \*host, const char \*service);

int errexit(const char \*format, ...);

int connectTCP(const char \*host, const char \*service);

int errexit(const char \*format, ...);

/\* connectTCP.c - connectTCP \*/

int connectsock(const char \*host, const char \*service,const char \*transport);

//main - TCP client for DAYTIME service

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

{

char \*host1 = "localhost"; /\* host to use if none supplied \*/

char \*host2 = "localhost"; /\* host to use if none supplied \*/

char \*service = "daytime"; /\* default service port\*/

switch (argc) {

case 1:

host1 = "localhost";

host2 = "localhost";

break;

case 4:

service = argv[3];

/\* FALL THROUGH \*/

case 3:

host1 = argv[1];

host2 = argv[2];

break;

default:

fprintf(stderr, "usage: TCPdaytime [host [port]]\n");

exit(1);

}

int s1=TCPdaytime(host1, service);

int s2=TCPdaytime(host2, service);

printf("%d seconds is the difference between both the Time's",s2-s1);

exit(0);

}

/\* TCPdaytime - invoke Daytime on specified host and print results\*/

int TCPdaytime(const char \*host, const char \*service)

{

char buf[LINELEN+1];

int s, n,a,b,c,sum;

/\* buffer for one line of text socket, read count\*/

s = connectTCP(host, service);

while( (n = read(s, buf, LINELEN)) > 0) {

buf[n] = '\0';

}

//printf("%s\n",buf);

printf("Time from server: ");

for(int i=16;i<=23;i++)

printf("%c",buf[i]);

printf("\n");

a=buf[22]-'0';

a=a\*10+(buf[23]-'0');

b=buf[19]-'0';

b=b\*10+(buf[20]-'0');

b=b\*60; //Convert minutes to seconds

c=buf[16]-'0';

c=c\*10+(buf[17]-'0');

c=c\*3600; //convert hours to seconds

sum=a+b+c; //Total number of seconds

return sum;

}

/\*connectTCP - connect to a specified TCP service on a specified host\*/

int connectTCP(const char \*host, const char \*service )

/\*

\* Arguments:

\* host - name of host to which connection is desired

\* service - service associated with the desired port

\*/

{

return connectsock( host, service, "tcp");

}

/\*connectsock - allocate & connect a socket using TCP or UDP\*/

int connectsock(const char \*host, const char \*service, const char \*transport )

/\*

\* Arguments:

\* host - name of host to which connection is desired

\* service - service associated with the desired port

\* transport - name of transport protocol to use ("tcp" or "udp")

\*/

{

struct hostent \*phe; /\* pointer to host information entry \*/

struct servent \*pse; /\* pointer to service information entry \*/

struct protoent \*ppe; /\* pointer to protocol information entry\*/

struct sockaddr\_in sin; /\* an Internet endpoint address \*/

int s, type; /\* socket descriptor and socket type \*/

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

sin.sin\_family = AF\_INET;

/\* Map service name to port number \*/

if ( pse = getservbyname(service, transport) )

sin.sin\_port = pse->s\_port;

else if ( (sin.sin\_port = htons((u\_short)atoi(service))) == 0 )

errexit("can't get \"%s\" service entry\n", service);

/\* Map host name to IP address, allowing for dotted decimal \*/

if ( phe = gethostbyname(host) )

memcpy(&sin.sin\_addr, phe->h\_addr, phe->h\_length);

else if ( (sin.sin\_addr.s\_addr = inet\_addr(host)) == INADDR\_NONE )

errexit("can't get \"%s\" host entry\n", host);

/\* Map transport protocol name to protocol number \*/

if ( (ppe = getprotobyname(transport)) == 0)

errexit("can't get \"%s\" protocol entry\n", transport);

/\* Use protocol to choose a socket type \*/

if (strcmp(transport, "udp") == 0)

type = SOCK\_DGRAM;

else

type = SOCK\_STREAM;

/\* Allocate a socket \*/

s = socket(PF\_INET, type, ppe->p\_proto);

if (s < 0)

errexit("can't create socket: %s\n", strerror(errno));

/\* Connect the socket \*/

if (connect(s, (struct sockaddr \*)&sin, sizeof(sin)) < 0)

errexit("can't connect to %s.%s: %s\n", host, service,strerror(errno));

return s;

}

/\*errexit - print an error message and exit\*/

int errexit(const char \*format, ...)

{

va\_list args;

va\_start(args, format);

vfprintf(stderr, format, args);

va\_end(args);

exit(1);

}

**Output:**

Time from server: 01:34:34  
Time from server: 01:34:34  
  
0 seconds is the difference between both the Time's

**Program - 3**

**Read Me**

Steps to execute the program:

1. Open the terminal

2. Navigate to the program directory

3. Build targets

4. make all

If you wish to build individual executables:

Build server alone:

make server

Build client alone:

make client

5. Run server and client on two different terminals:

a. ./server <port>

ex: ./server 10000

b. ./client <server\_IP> <port>

ex: ./client 127.0.0.1 10000

6. Enter "quit" when you wish to exit the client.

7. Enter ^C to quit running the server.

**Makefile**

# Make file targets to build TCP client & TCP server (Assignment 1, problem #3)

# Format

# target: dependencies

# action

.PHONY: server client

all: server client

server:

gcc tcp\_server\_3.c -o server

client:

gcc tcp\_client\_3.c -o client

clean:

rm server client

**Source Code**

**Client Program**

// CMPE\_297 Assignment\_1 Program #3

// TCP Client

// Author - Team #2

// This TCP client program connects to the server

// sends an integer value. Receives an integer value

// back from the server which is one decremented from

// the value sent by the client. Client then prints

// the value to its output screen.

// To compile: gcc tcp\_client\_3.c -o client

// To run: ./client server\_address server\_port

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<sys/socket.h>

#include<sys/types.h>

#include<netinet/in.h>

#include<string.h>

#include<ctype.h>

#include<arpa/inet.h>

#include<netdb.h>

#define ERROR -1

#define BUFFER 1024

// function to convert the given

// string to lower case.

char \*conv\_to\_lower(char str[],char \*res)

{

int i = 0;

while(str[i])

{

res[i] = tolower(str[i]);

i++;

}

return res;

}

int connectsock(const char \*host, const char \*service, const char \*transport)

{

struct hostent \*phe;

struct servent \*pse;

struct protoent \*ppe;

struct sockaddr\_in remote\_server;

int sock, type;

//reset socket address structure.

memset(&remote\_server,0,sizeof(remote\_server));

remote\_server.sin\_family = AF\_INET;

//map service name to port number

if(pse = getservbyname(service, transport))

remote\_server.sin\_port = pse->s\_port;

else if((remote\_server.sin\_port = htons((u\_short)atoi(service))) == 0)

//perror("Cant get \"%s\" service entry\n", service);

perror("Cant get service entry\n");

//map host name to IP address allowing dotted decimal

if(phe = gethostbyname(host))

memcpy(&remote\_server.sin\_addr, phe->h\_addr, phe->h\_length);

else if ((remote\_server.sin\_addr.s\_addr = inet\_addr(host) == INADDR\_NONE))

//perror("Cant get \"%s\" host entry\n", host);

perror("Cant get host entry\n");

bzero(&remote\_server.sin\_zero, 8);

//map transport protocol name to protocol number

ppe = getprotobyname(transport);

if((ppe == NULL))

//perror("cant get \"%s\" protocol entry\n", transport);

perror("Cant get protocol entry\n");

//use protocol to chose a socket type

if(strcmp(transport, "udp") == 0)

type = SOCK\_DGRAM;

else

type = SOCK\_STREAM;

//allocate a socket

sock = socket(PF\_INET, type, ppe->p\_proto);

if(sock < 0)

//perror("Cant create socket: %s\n", strerror(errno));

perror("Cant create socket\n");

//connect the socket

if(connect(sock, (struct sockaddr \*)&remote\_server, sizeof(remote\_server)) < 0)

//perror("Cant connect to %s %s: %s\n", host, service, sererror(errno));

perror("Cant connect to the remote host\n");

return sock;

}

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

{

int sock;

char input[BUFFER];

char output[BUFFER];

char \*res\_str;

int len;

int res;

if(argc < 3)

{

printf("Usage: ./client <server IP> <server port> \n");

exit(0);

} else

{

printf("\n>> Send an integer. Server responds with a decremented value.\n");

printf(">> Enter quit when you wish to exit the client.\n\n");

sock = connectsock(argv[1], argv[2], "tcp");

strcpy(input, "1");

while(1)

{

// Take user input.

memset(input, 0, BUFFER);

printf("User input:");

fgets(input, BUFFER, stdin);

len = strlen(input);

input[len-1] = '\0';

res\_str = (char \*) malloc(len);

// Check if user wishes to exit the client

if((res = strcmp(conv\_to\_lower(input,res\_str), "quit")) == 0)

{

free(res\_str);

break;

}

free(res\_str);

// Send user input to the server.

send(sock, input, strlen(input), 0);

memset(output, 0, BUFFER);

// receive server's response and print.

len = recv(sock, output, BUFFER, 0);

output[len] = '\0';

printf("Server output:%s\n\n", output);

}

//close client socket.

close(sock);

}

}

**Server Program**

// CMPE\_207 Assignment\_1 Program #3

// Author - Team #2

// TCP server program

// This server program listens for active clients.

// Receives an integer from the client. Decrements

// the int by one and sends it back to the client.

// To compile: gcc tcp\_server\_3 -o server

// To run: ./server port\_number

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<sys/socket.h>

#include<sys/types.h>

#include<netinet/in.h>

#include<string.h>

#include<arpa/inet.h>

#include<netdb.h>

#define ERROR 0

#define MAX\_CLIENTS 5

#define MAX\_DATA 1024

int passivesock(const char \*service, const char \*transport, int qlen, int \*sock)

{

struct servent \*pse;

struct protoent \*ppe;

struct sockaddr\_in server;

int res = 1;

int type;

//reset socket address structure.

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

server.sin\_family = AF\_INET;

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

//map service name to port number

if(pse = getservbyname(service, transport))

server.sin\_port = htons(ntohs((u\_short)pse->s\_port));

else if((server.sin\_port = htons((u\_short)atoi(service))) == 0)

//perror("Cant get \"%s\" service entry\n", service);

perror("Cant get service entry\n");

bzero(&server.sin\_zero, 8);

//map transport protocol name to protocol number

ppe = getprotobyname(transport);

if((ppe == NULL))

//perror("cant get \"%s\" protocol entry\n", transport);

perror("Cant get protocol entry\n");

//use protocol to chose a socket type

if(strcmp(transport, "udp") == 0)

type = SOCK\_DGRAM;

else

type = SOCK\_STREAM;

//allocate a socket

\*sock = socket(PF\_INET, type, ppe->p\_proto);

if (\*sock < 0){

res = -1;

perror("Cant create socket\n");

}

//Bind the socket

if(bind(\*sock, (struct sockaddr \*)&server, sizeof(server)) < 0)

{

res = -1;

perror("Cant bind to specified port\n");

}

//listen for incoming client connections

if(type == SOCK\_STREAM && (listen(\*sock, qlen) < 0))

{

res = -1;

perror("Listen error\n");

}

return res;

}

int validate\_data(char data[], int len)

{

int result = 1;

int in\_value;

if(data[0] == '-')

in\_value = 1;

else

in\_value = 0;

for(int i = in\_value; i < len;i++)

{

if(data[i] >= 48 && data[i] <=57)

continue;

else{

result = 0;

break;

}

}

return result;

}

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

{

struct sockaddr\_in client; //client data structure

int sock, result;

int new\_cli\_sock;

socklen\_t sockaddr\_len = sizeof(struct sockaddr\_in);

int data\_len;

char data[MAX\_DATA];

int num;

if(argc != 2)

{

printf("Usage: ./server <port>\n");

exit(0);

}

result = passivesock(argv[1], "tcp", MAX\_CLIENTS, &sock);

if (result == -1)

exit(-1);

printf("\n\nTCP Server listening for active connections...\n");

//Accept client connection

while(1)

{

if((new\_cli\_sock = accept(sock, (struct sockaddr \*)&client, &sockaddr\_len)) == ERROR)

{

perror("Accept Error");

exit(-1);

}

printf("New client connected from port no: %d & IP: %s\n", ntohs(client.sin\_port), inet\_ntoa(client.sin\_addr));

data\_len = 1;

while(data\_len)

{

memset(data, 0, MAX\_DATA);

data\_len = recv(new\_cli\_sock, data, MAX\_DATA, 0);

if(data\_len)

{

// validate the received data

printf("\nReceived: %s\n", data);

result = validate\_data(data, strlen(data));

if(result)

{

num = atoi(data);

// decrement the integer data by 1.

num = num - 1;

snprintf(data,MAX\_DATA, "%d",num);

} else

{

strcpy(data, "Sorry, no valid integer sent.");

data\_len = strlen(data);

}

// send response to the client

printf("Sent: %s\n",data);

send(new\_cli\_sock, data, strlen(data), 0);

}

}

printf("Client disconnected\n");

// close client connection

close(new\_cli\_sock);

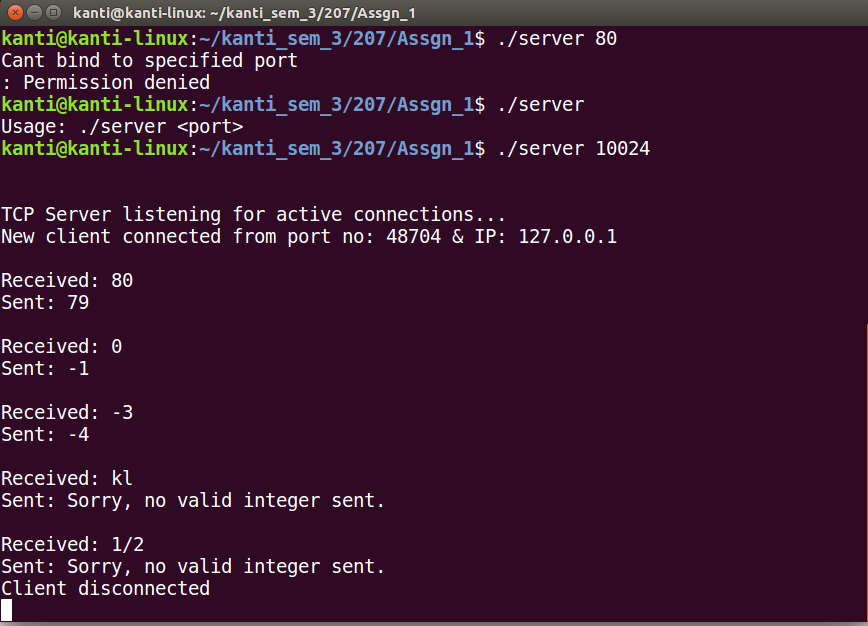
}

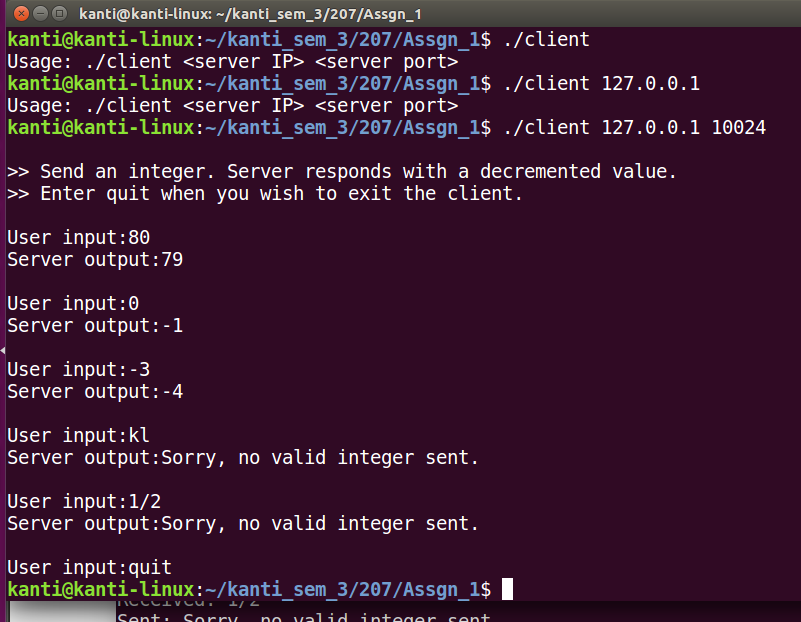
//close the server socket

close(sock);

}

**Output:**

****

****

References:

1. Internetworking with TCP/IP Vol. 3, Client-Server programming and applications, Comer and Stevens, Linux/POSIX Sockets version, ISBN: 0-13-032071-4, 2001