**ECEN 602**

**NETWORK SIMULATION ASSIGNMENT – 01**

**TEAM 17**

**Mohammad Faizal Khan**

**Amiya Ranjan Panda**

**INTRODUCTION**

We have created a basic TCP Server-Client model system for the assignment wherein the server part was built by Faizal Khan and the client part was built by

Amiya Panda.

**ARCHITECTURE**

In this model, the server is capable of catering request from five client at max, which is scalable by changing “listen (…, 5)” from the code. This is implemented using the fork() command. The maximum reading/writing buffer capacity is limited to 32 Bytes, which is scalable by changing MAXLINE from code. The clients sends data bits to the server and if the server is unable to write the entire bitstream, the client sends again the remaining bitstreams. This flow is implemented using the readline(), written(), str\_echo() and str\_cli() functions.

In this implementation, we have created function as mentioned below :

1. writen
2. readline
3. my\_read
4. readline\_buffer
5. Str\_echo
6. str\_cli

USAGE:

1. Copy and paste the files makefile, echo.c and echos.c in the system.

2. In the linux terminal, type "make" to compile and build the executables echo and echos.

3. For running the server, open a terminal and type "./echos <port>".

4. For the client, open another terminal and use command "./echo <ip> <port>". Here we are using our

loopback ip 127.0.0.1 for convenience.

ECHO SOURCE CODE :

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <errno.h>

#include <string.h>

#include <time.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <netdb.h>

#include <sys/wait.h>

#include <signal.h>

static int read\_cnt;

static char \*read\_ptr;

static char read\_buf[256];

static ssize\_t my\_read(int fd, char \*ptr)

{

if (read\_cnt <= 0) {

again:

if ( (read\_cnt = read(fd, read\_buf, sizeof(read\_buf))) < 0) {

if (errno == EINTR)

goto again;

return (-1);

} else if (read\_cnt == 0)

return (0);

read\_ptr = read\_buf;

}

read\_cnt--;

\*ptr = \*read\_ptr++;

return (1);

}

ssize\_t readline(int fd, char \*vptr, size\_t maxlen)

{

ssize\_t n, rc;

char c;

char \*ptr;

ptr = vptr;

for (n = 1; n < maxlen; n++) {

if ( (rc = my\_read(fd, &c)) == 1) {

\*ptr++ = c;

if (c == '\n')

break; /\* newline is stored, like fgets() \*/

} else if (rc == 0) {

\*ptr = 0;

return (n - 1); /\* EOF, n - 1 bytes were read \*/

} else

return (-1); /\* error, errno set by read() \*/

}

\*ptr = 0; /\* null terminate like fgets() \*/

return (n);

}

ssize\_t

readlinebuf(void \*\*vptrptr)

{

if (read\_cnt)

\*vptrptr = read\_ptr;

return (read\_cnt);

}

ssize\_t writen(int fd, const char \*vptr, size\_t n)

{

size\_t nleft;

ssize\_t nwritten;

const char \*ptr;

ptr = vptr;

nleft = n;

while (nleft > 0) {

if ( (nwritten = write(fd, ptr, nleft)) <= 0) {

if (nwritten < 0 && errno == EINTR)

nwritten = 0; /\* and call write() again \*/

else

return (-1); /\* error \*/

}

nleft -= nwritten;

ptr += nwritten;

}

return (n);

}

void str\_cli(FILE \*fp, int sockfd)

{

char sendline[256], recvline[256];

while (fgets(sendline, 256, fp) != NULL) {

writen(sockfd, sendline, strlen (sendline));

if (readline(sockfd, recvline, 256) == 0){

printf("str\_cli: server terminated prematurely");

exit;

}

fputs(recvline, stdout);

}

}

int

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

{

int sockfd;

struct sockaddr\_in servaddr;

if (argc != 2){

printf("usage: tcpcli <IPaddress>");

exit;

}

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

bzero(&servaddr, sizeof(servaddr));

servaddr.sin\_family = AF\_INET;

servaddr.sin\_port = htons(50001);

inet\_pton(AF\_INET, argv[1], &servaddr.sin\_addr);

connect(sockfd, (struct sockaddr\*) &servaddr, sizeof(servaddr));

str\_cli(stdin, sockfd); /\* do it all \*/

exit(0);

}

ECHOS SOURCE CODE :

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <errno.h>

#include <string.h>

#include <time.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <netdb.h>

#include <sys/wait.h>

#include <signal.h>

ssize\_t writen(int fd, const void \*vptr, size\_t n) /\* Write "n" bytes to a descriptor. \*/

{

size\_t nleft;

ssize\_t nwritten;

const char \*ptr;

ptr = vptr;

nleft = n;

while (nleft > 0) {

if ( (nwritten = write(fd, ptr, nleft)) <= 0) {

if (nwritten < 0 && errno == EINTR)

nwritten = 0; /\* and call write() again \*/

else

return (-1); /\* error \*/

}

nleft -= nwritten;

ptr += nwritten;

}

return (n);

}

void str\_echo(int sockfd)

{

ssize\_t n;

char buf[256];

again:

while ( (n = read(sockfd, buf, 256)) > 0)

writen(sockfd, buf, n);

if (n < 0 && errno == EINTR)

goto again;

else if (n < 0){

printf("str\_echo: read error");

exit;}

}

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

{

int listenfd, connfd;

pid\_t childpid;

socklen\_t clilen;

struct sockaddr\_in cliaddr, servaddr;

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

int SERV\_PORT = argv[1];

bzero(&servaddr, sizeof(servaddr));

servaddr.sin\_family = AF\_INET;

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

servaddr.sin\_port = htons (SERV\_PORT);

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

listen(listenfd, 5);

for ( ; ; ) {

clilen = sizeof(cliaddr);

connfd = accept(listenfd, (struct sockaddr\*) &cliaddr, &clilen);

if ( (childpid = fork()) == 0) { /\* child process \*/

close(listenfd); /\* close listening socket \*/

str\_echo(connfd); /\* process the request \*/

exit (0);

}

close(connfd); /\* parent closes connected socket \*/

}

}

MAKE FILE :

output: echo.C echos.c

gcc echo.o -o echo

gcc echos.o -o echos

echos.o:echos.c

gcc -c echos.c

echo.0: echo.C

gcc -c echo.C

clean:

rm \*.o core