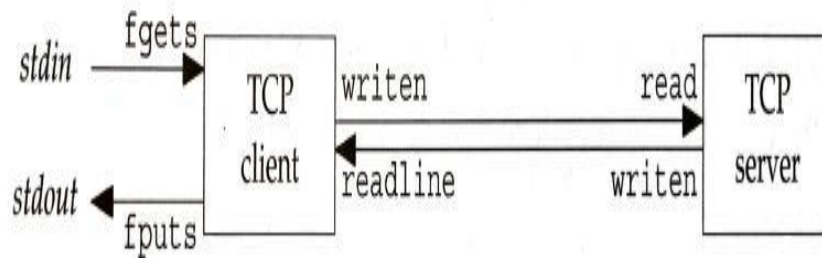


ECEN 602 Network Programming Assignment 1

This document consists of the README, Makefile, Test Cases and Codes

TCP echo Server and Client

We have implemented a Client and Server system, where the client reads input from the console and writes it onto the server and also returns the number of bytes written. The server then reads the received message and writes it back to the client, thus echoing it. On receipt of the echoed message, the client reads it character by character and displays it on the console.



Prerequisites

As we were living off campus, the VPN connection seemed to have some sort of communication depot error. So, we used the online architecture-Cloud9, which replicates the Linux server (Ubuntu).

Getting the Development Environment Running / Deployment

```
bash - "ubuntu@ ×
sumag:~/workspace $ make
gcc -o echos server.c
gcc -o echo client.c
```

This is the **Makefile** which is compiling the client and server code.

After make, we write 2 statements:

`./echos 4000` (where 4000 is port number, can be any above 1024)
`./echo 127.0.0.1 4000` (where 127.0.0.1 is the loopback address, since server and client are running on the same physical machine and 4000 is the server port we want to connect to)

For multiple clients:

```
gcc -o echo2 client2.c
gcc -o echo3 client3.c
./echo2 127.0.0.1 4000
./echo3 127.0.0.1 4000
```

Methodology

1. First, we implemented a simple echo server where the client reads from console and sends it to the server which then reads the message and echoes it back to the client.
2. Then we implemented the fork function , where the server accepted multiple requests from clients at the same time and echoed back the message correctly. (Shown in test case).
3. Then we tried to resolve the problem of zombie processes. (reaping them through sighandler).
4. Then we checked error returns from each function, which set an errno on error and also displayed the error message.
5. The last and most important step was making the written and readline functions which is implemented character by character i.e. the message is read character by character and also written the same way. Moreover, if the whole message is not written at once, we use a for loop to write the remaining bytes until all of them have been read. Many error such as EINTR, EOF, end of line etc. have also been checked in the readline function.

Running the Test Cases

1. Output when client 1 connects and takes a string from console as input and it is correctly echoed back to the client

Server

```
sumag:~/workspace $ gcc -o echos server.c
sumag:~/workspace $ gcc -o echo client.c
sumag:~/workspace $ ./echos 7500
socket created
new connection accepted
Child created

Echoing back the message received from client : suma
length of message received 4
length of message written 4
```

Client

```
sumag:~/workspace $ ./echo 127.0.0.1 7500
socket created
connected suma
length of message received 4
length of message written 4
suma
```

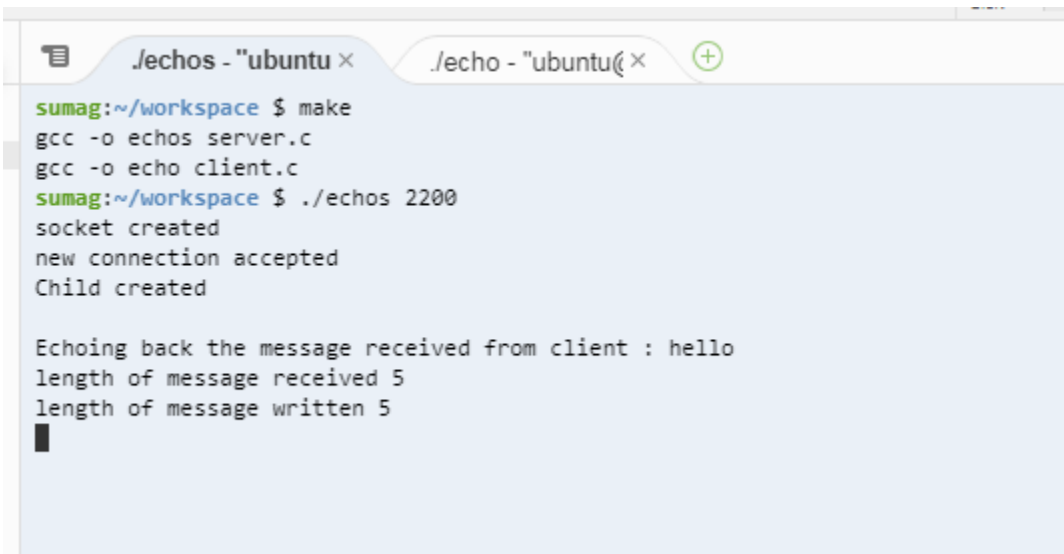
2. EOF implemented by entering ctrl+D twice, after the message is entered. So the current message is echoed back but the next message entered does not receive an echo as the connection has been terminated from the client.

Client Side



```
sumag:~/workspace $ ./echo 127.0.0.1 2200
socket created
connected hello
length of message received 5
length of message written 5
hello
xyzlength of message received 2
length of message written 2
test
```

Server Side



```
sumag:~/workspace $ make
gcc -o echos server.c
gcc -o echo client.c
sumag:~/workspace $ ./echos 2200
socket created
new connection accepted
Child created

Echoing back the message received from client : hello
length of message received 5
length of message written 5
```

3. Client terminated after entering text: We enter the string hi and then we terminate the client using Ctrl+C instead of enter, hence the socket cannot echo back.

Client

```
bash - "ubuntu@ x bash - "ubuntu@ x +
sumag:~/workspace $ ./echo 127.0.0.1 3000
socket created
connected hi^C
sumag:~/workspace $ █
```

Server

```
bash - "ubuntu@ x bash - "ubuntu@ x +
sumag:~/workspace $ gcc -o echos server.c
sumag:~/workspace $ gcc -o echo client.c
sumag:~/workspace $ ./echos 3000
socket created
Child created

sumag:~/workspace $ █
```

4. Three clients connected to the server at the same port at the same time. The server echoes back to all the three clients as shown:

```
./echo1 - "ubuntu x ./echos x ./echo2 x
sumag:~/workspace $ ./echo1 127.0.0.1 5000
suma child 1
suma child 1
█
```

```
./echo1 x ./echos x ./echo2 - "ubuntu x ./e
sumag:~/workspace $ ./echo2 127.0.0.1 5000
suma child 2
suma child 2
█
```

```
./echo1x  ./echosx  ./echo2x  ./echo3 - "ubuntu x
sumag:~/workspace $ ./echo3 127.0.0.1 5000
suma child 3
suma child 3
█
```

```
sumag:~/workspace $ gcc -o echos server.c
sumag:~/workspace $ gcc -o echo1 client.c
sumag:~/workspace $ gcc -o echo2 client2.c
sumag:~/workspace $ gcc -o echo3 client3.c
sumag:~/workspace $ ./echos 5000
Child created

Echoing back - suma child 1
Child created

Echoing back - suma child 2
Child created

Echoing back - suma child 3
█
```

Authors

1. Garuda Suma Pranavi – 926009146
2. Dedeepya Venigalla – 726006161

We have both equally contributed in developing the code for the assignment.

Acknowledgements

1. Beej's Guide
2. [www.cs.dartmouth.edu](http://www.cs.dartmouth.edu/socketprogramming) /socketprogramming

Code

Server.c

```
#include <sys/socket.h>
#include <sys/types.h>
#include <netdb.h>
#include <stdio.h>
#include <strings.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <errno.h>

void handler(int s)
{
    int saved_errno=errno;
    while ( waitpid (-1,NULL,WNOHANG) >0) ;
    errno=saved_errno;
}

int writen(int sockfd,char sendline[])
{
    ssize_t actualsize;
    int i;
    int readsize=strlen(sendline);
    printf("%s","length of message received ");
    printf("%i \n",readsize-1);
    actualsize=write(sockfd,sendline,strlen(sendline));
    printf("%s","length of message written ");
    printf("%zu \n",actualsize-1);

    if (actualsize<readsize)
    {
        for (i=actualsize;i<=readsize;i++)
        {
            write(sockfd,(void*)&sendline[i],1);
        }
    }
    return actualsize;
}

ssize_t read_line(int newfd,void *buffer, size_t n)
```

```

{
    ssize_t read_bytes=0;
    size_t total_read=0;
    char *buf=NULL;
    char ch;
    buf = buffer;

    while(1)
    {   memset(&ch,' ',1);

        read_bytes = read(newfd, &ch, 1);

        if (read_bytes > 0)
        {
            if (total_read < n-1)
            {
                *buf= ch;
                buf++;
                total_read++;
                // printf("buf1=%s", buf-1);
            }
        }

        if (read_bytes == -1)
        {
            if (errno == EINTR)
                continue;
            else
                return (-1);
        }

        if (read_bytes == 0)
        {
            exit(1); //end of file
        }

        if (ch == '\n')
            break;
    }

    *buf = '\0';
    //printf("buf=%s", buf-total_read);
    return total_read+1;
}

```

```

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

    char str[100];
    int listenfd;
    int newfd;
    pid_t childpid;
    int n;
    int port;
    struct sockaddr_in sadr;
    struct sigaction s;

    port = atoi ( argv[1] );
    if ((listenfd = socket( PF_INET, SOCK_STREAM, 0 ))<0)
    {
        perror("socket");
        exit(1);
    }
    else
    {
        fputs("socket created \n", stdout);
    }

    s.sa_flags=SA_RESTART;
    s.sa_handler=handler;
    sigemptyset(&s.sa_mask);

    if(sigaction(SIGCHLD,&s,NULL)==-1)
    {
        perror("sigaction error");
        exit(1);
    }

    memset(&sadr,0,sizeof sadr);

    sadr.sin_family = AF_INET;
    sadr.sin_port = htons(port);
    sadr.sin_addr.s_addr = htons(INADDR_ANY);

    if ( bind (listenfd, (struct sockaddr *) &sadr, sizeof sadr) <0)
    {
        close(listenfd);
        perror("server bind error");
        exit(1);
    }

```



```

}

if (listen (listenfd, 20) <0)
{
    perror ( " listen error ");
    exit(1);
}

for(;;)
{
    newfd = accept(listenfd, (struct sockaddr*) NULL, NULL); //accepting new connections from listen
    fputs("new connection accepted ",stdout);
    fputs("\n",stdout);

    if ((childpid = fork()) == 0 ) //child process id = 0 when fork is called
    {
        fputs("Child created \n",stdout);
        fputs("\n",stdout);
        close (listenfd); //closing the listening socket of the child
        memset(str,0,100);
    }

    while ((n = read_line(newfd,str,100)) > 0)
    {
        fputs("Echoing back the message received from client : ",stdout);
        fputs(str,stdout);
        writen(newfd,str);
        memset(str,0,100);
    }

    if (n < 0)
    fputs("Read error", stdout);
    exit(0);
}

//closing the server socket
close(newfd);
}

```

Client1.c

```

#include <sys/socket.h>
#include <sys/types.h>
#include <netdb.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <strings.h>
#include <unistd.h>
#include <strings.h>
#include <arpa/inet.h>
#include <errno.h>

```

```

int writen(int sockfd, char sendline[])
{
    ssize_t actualsize;
    int i;
    int readsize = strlen(sendline);
    printf("%s", "length of message received ");
    printf("%i \n", readsize-1);
    actualsize = write(sockfd, sendline, strlen(sendline));
    printf("%s", "length of message written ");
    printf("%zu \n", actualsize-1);

    if (actualsize < readsize)
    {
        for (i = actualsize; i <= readsize; i++)
        {
            write(sockfd, (void*)&sendline[i], 1);
        }
    }
    return actualsize;
}

```

```

ssize_t read_line(int newfd, void *buffer, size_t n)
{
    ssize_t read_bytes = 0;
    size_t total_read = 0;
    char *buf = NULL;
    char ch;
    buf = buffer;

    while(1)
    {
        memset(&ch, '\0', 1);

```

```

read_bytes = read(newfd, &ch, 1);

if (read_bytes > 0)
{
    if (total_read < n-1)
    {
        *buf= ch;
        buf++;
        total_read++;
        // printf("buf1=%s", buf-1);
    }
}

if (read_bytes == -1)
{
    if (errno == EINTR)
        continue;
    else
        return (-1);
}

if (read_bytes == 0)
{
    exit(1); //end of file
}

    if (ch == '\n')
        break;
}

*buf = '\0';
//printf("buf=%s", buf-total_read);
return total_read+1;
}

int main(int argc, char *argv[])
{
    int sockfd;
    char writeline[100];
    char read[100];
    struct sockaddr_in saddr;
    int port;
    port=atoi(argv[2]);

```

```

sockfd=socket(PF_INET,SOCK_STREAM,0);

if (sockfd==-1)
{
    perror("socket");
    exit(-1);
}
else
{
    fputs("socket created \n", stdout);
}

memset(&sadr,0,sizeof sadr);

sadr.sin_family = AF_INET;
sadr.sin_port = htons(port);
inet_pton ( AF_INET, argv[2], &(sadr.sin_addr));

if ( connect ( sockfd ,(struct sockaddr *)&sadr, sizeof(sadr))==-1)
{
    close(sockfd);
    perror("connection failed");
    return 1;
}
else
fputs ("connected ", stdout);

while(1)
{
    memset(writeline,0,100);
    memset(read,0,100);
    fgets(writeline,100,stdin);

    if (writen(sockfd,writeline)<0)
    {
        printf("%s", "write error");
    }

    read_line(sockfd,read,100);
    fputs(read,stdout);
}
close(sockfd);

```

```
}
```

Client2.c

```
#include <sys/socket.h>
#include <sys/types.h>
#include <netdb.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <strings.h>
#include <unistd.h>
#include <strings.h>
#include <arpa/inet.h>
#include <errno.h>
```

```
int writen(int sockfd,char sendline[])
{
    ssize_t actualsize;
    int i;
    int readsize=strlen(sendline);
    printf("%s","length of message received ");
    printf("%i \n",readsize-1);
    actualsize=write(sockfd,sendline,strlen(sendline));
    printf("%s","length of message written ");
    printf("%zu \n",actualsize-1);
```

```
    if (actualsize<readsize)
    {
        for (i=actualsize;i<=readsize;i++)
        {
            write(sockfd,(void*)&sendline[i],1);
        }
    }
    return actualsize;
}
```

```
ssize_t read_line(int newfd,void *buffer, size_t n)
{
    ssize_t read_bytes=0;
    size_t total_read=0;
    char *buf=NULL;
    char ch;
```

```

buf = buffer;

while(1)
{  memset(&ch,' ',1);

    read_bytes = read(newfd, &ch, 1);

    if (read_bytes > 0)
    {
        if (total_read < n-1)
        {
            *buf= ch;
            buf++;
            total_read++;
            // printf("buf1=%s", buf-1);
        }
    }

    if (read_bytes == -1)
    {
        if (errno == EINTR)
            continue;
        else
            return (-1);
    }

    if (read_bytes == 0)
    {
        exit(1); //end of file
    }

    if (ch == '\n')
        break;
    }

    *buf = '\0';
    //printf("buf=%s", buf-total_read);
    return total_read+1;
}

int main(int argc,char *argv[])
{
    int sockfd;
    char writeline[100];

```

```

char read[100];
struct sockaddr_in saddr;
int port;
port=atoi(argv[2]);

sockfd=socket(PF_INET,SOCK_STREAM,0);

if (sockfd==-1)
{
    perror("socket");
    exit(-1);
}
else
{
    fputs("socket created \n", stdout);
}

memset(&saddr,0,sizeof saddr);

saddr.sin_family = AF_INET;
saddr.sin_port = htons(port);
inet_pton ( AF_INET, argv[2], &(saddr.sin_addr));

if ( connect ( sockfd ,(struct sockaddr *)&saddr, sizeof(saddr))==-1)
{
    close(sockfd);
    perror("connection failed");
    return 1;
}
else
fputs ("connected ", stdout);

while(1)
{
    memset(writeline,0,100);
    memset(read,0,100);
    fgets(writeline,100,stdin);

    if (writen(sockfd,writeline)<0)
    {
        printf("%s", "write error");
    }

    read_line(sockfd,read,100);

```

```

        fputs(read,stdout);

    }
    close(sockfd);
}

```

Client3.c

```

#include <sys/socket.h>
#include <sys/types.h>
#include <netdb.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <strings.h>
#include <unistd.h>
#include <strings.h>
#include <arpa/inet.h>
#include <errno.h>

int writen(int sockfd,char sendline[])
{
    ssize_t actualsize;
    int i;
    int readsize=strlen(sendline);
    printf("%s","length of message received ");
    printf("%i \n",readsize-1);
    actualsize=write(sockfd,sendline,strlen(sendline));
    printf("%s","length of message written ");
    printf("%zu \n",actualsize-1);

    if (actualsize<readsize)
    {
        for (i=actualsize;i<=readsize;i++)
        {
            write(sockfd,(void*)&sendline[i],1);
        }
    }
    return actualsize;
}

ssize_t read_line(int newfd,void *buffer, size_t n)
{
    ssize_t read_bytes=0;

```



```

size_t total_read=0;
char *buf=NULL;
char ch;
buf = buffer;

while(1)
{  memset(&ch,' ',1);

    read_bytes = read(newfd, &ch, 1);

    if (read_bytes > 0)
    {
        if (total_read < n-1)
        {
            *buf= ch;
            buf++;
            total_read++;
            // printf("buf1=%s", buf-1);
        }
    }

    if (read_bytes == -1)
    {
        if (errno == EINTR)
            continue;
        else
            return (-1);
    }

    if (read_bytes == 0)
    {
        exit(1); //end of file
    }

    if (ch == '\n')
        break;
    }

    *buf = '\0';
    //printf("buf=%s", buf-total_read);
    return total_read+1;
}

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

```

```

{
    int sockfd;
    char writeline[100];
    char read[100];
    struct sockaddr_in saddr;
    int port;
    port=atoi(argv[2]);

    sockfd=socket(PF_INET,SOCK_STREAM,0);

    if (sockfd==-1)
    {
        perror("socket");
        exit(-1);
    }
    else
    {
        fputs("socket created \n", stdout);
    }

    memset(&saddr,0,sizeof saddr);

    saddr.sin_family = AF_INET;
    saddr.sin_port = htons(port);
    inet_pton ( AF_INET, argv[2], &(saddr.sin_addr));

    if ( connect ( sockfd ,(struct sockaddr *)&saddr, sizeof(saddr))==-1)
    {
        close(sockfd);
        perror("connection failed");
        return 1;
    }
    else
    fputs ("connected ", stdout);

    while(1)
    {
        memset(writeline,0,100);
        memset(read,0,100);
        fgets(writeline,100,stdin);

        if (written(sockfd,writeline)<0)
        {
            printf("%s", "write error");

```

```
    }  
  
    read_line(sockfd,read,100);  
    fputs(read,stdout);  
  
}  
close(sockfd);  
}
```