ECEN 602
Programming Assignment

Team Number 8

1. Garuda Suma Pranavi (UIN 926009146)
2. Dedeepya Venigalla (UIN 726006161)

We have both equally contributed in implementation of Trivial File Transfer Protocol (TFTP) server.

About the program:

- This code is written in C language.
- In this program, we implement the TFTP server and test it with the in-built TFTP client. TFTP protocol uses UDP protocol (stop and wait) for data transfer.
- This TFTP server is capable of handling multiple clients simultaneously.
- This code allows retrieval of files in the local folder from where the server is executed to the folder where the client is running.
- If the file size is greater than 32MB, the acknowledgement number wraps around to 0 and starts again.
- **Note:** This program has been developed and tested in cloud9 environment.

How to implement the code:

1. First run the attached makefile which is as follows:

```
all: server.cpp
        g++ -o server server.cpp
clean:
        rm server
```

2. Then run the server, by writing:
   ./server server_ip server_port
   (Here we have used 127.0.0.1, loopback address and any port above 1024 for the server)
   For example, ./server 127.0.0.1 22000

3. File transfer from the clients can be done by the following sequence of steps:
   $ tftp
   tftp> connect <server_IP> <server_port>
   tftp>get <filename>

Test Cases

Test Case. Transferring an authentic binary file (ioslinux.bin)

```
opcode =1, Filename =isolinux.bin, mode =netascii
file size =24576, packets=49
file transmitted successfully
```

```
sumag:~/workspace/New Folder.2 $ tftp
tftp> connect 127.0.0.1 22500
tftp> get isolinux.bin
Received 24576 bytes in 0.1 seconds
tftp>
```

Test Case 1) Transferring .bin file with 1's and 0's of size 2048

```
opcode =1, Filename =2048.bin, mode =netascii
file size =2048, packets=5
file transmitted successfully
```

```
tftp> get 2048.bin
Received 2048 bytes in 0.0 seconds
tftp>
```

Test Case 2) Transferring .bin file with 1's and 0's with size 2047

```
opcode =1, Filename =2047.bin, mode =netascii
file size =2047, packets=4
file transmitted successfully
```

```
tftp> get 2047.bin
Received 2047 bytes in 0.0 seconds
tftp>
```

Test Case3) Netascii CR

```
tftp> get testcase3.txt
Received 40 bytes in 0.0 seconds
tftp>
```

```
opcode =1, Filename =testcase3.txt, mode =netascii
file size =40, blocks=1
acknowledgement number 1
block number 1
file transmitted successfully
```

Server side file

```
1  this is hw 3
2  assignment 602
3  server code
```

Client side file

```
1  this is hw 3
2  assignment 602
3  server code
4
```

Test Case 4) File size> 32MB which is the max file size to check if the block number wrap around works

```
sumag:~/workspace $ g++ -o  server  server.cpp
sumag:~/workspace $ ./server 127.0.0.1 22500
opcode =1, Filename =big32.pdf, mode =netascii
file size =34394771, packets=67178
acknowledgement number 1642
packet number 67178
file transmitted successfully
```

Through the wrap around function we can see that the ack num is 1642 i.e. 1642 packets were transmitted after the successful transmission of 65536 packets i.e. 32 MB.

```
tftp> get big32.pdf
Received 34394771 bytes in 1.8 seconds
tftp>
```

Test Case 5) We receive an error message when we request a file that does not exist

```
sumag:~/workspace $ g++ -o  server  server.cpp
sumag:~/workspace $ ./server 127.0.0.1 22500
opcode =1, Filename =assign.pdf, mode =netascii
Could not open requested fileassign.pdf
```

```
sumag:~/workspace/New Folder.2 $ tftp
tftp> connect 127.0.0.1 22500
tftp> get assign.pdf
Error code 1: Could not open requested file
tftp>
```

Test Case 6) Three clients connect to the TFTP server simultaneously, requesting for files

```
sumag:~/workspace $ g++ -o  server  server.cpp
sumag:~/workspace $ ./server 127.0.0.1 22500
opcode =1, Filename =less32.pdf, mode =netascii
file size =30267863, packets=59117
file transmitted successfully
opcode =1, Filename =less32.pdf, mode =netascii
file size =30267863, packets=59117
file transmitted successfully
opcode =1, Filename =less32.pdf, mode =netascii
file size =30267863, packets=59117
file transmitted successfully
```

```
sumag:~/workspace/New Folder.2 $ tftp
tftp> connect 127.0.0.1 22500
tftp> get less32.pdf
Received 30267863 bytes in 1.5 seconds
tftp>
```

```
sumag:~/workspace/New Folder.3 $ tftp
tftp> connect 127.0.0.1 22500
tftp> get less32.pdf
Received 30267863 bytes in 1.5 seconds
tftp>
```

```
sumag:~/workspace/New Folder.1 $ tftp
tftp> connect 127.0.0.1 22500
tftp> get less32.pdf
Received 30267863 bytes in 1.5 seconds
tftp>
```

Data being successfully transmitted from each server to the client. (We can't see all the messages in the window, hence the last screenshot).

Test Case 7) Terminating the client in the middle of transmission (we are transferring the file of size ~ 28MB)

```
sumag:~/workspace $ ./server 127.0.0.1 22500
opcode =1, Filename =less32.pdf, mode =netascii
file size =30267863, packets=59117
retransmitting attempt1for block54254
retransmitting attempt2for block54254
retransmitting attempt3for block54254
retransmitting attempt4for block54254
retransmitting attempt5for block54254
retransmitting attempt6for block54254
retransmitting attempt7for block54254
retransmitting attempt8for block54254
retransmitting attempt9for block54254
retransmitting attempt10for block54254
request timedout
```

The whole file was not sent, data transmission stopped at the specific block number being used when the client was terminated.

Source Code:

```
//mode is netascii(for text) or octet(for binary) depending on the file
#include<iostream>
#include<fstream>
#include<string>
#include<stdio.h> // for file related functions
#include<sys/socket.h> //sendto and recvfrom functions
#include<netinet/in.h> // for htons
#include<sys/select.h> //for select()
#include<sys/wait.h>
#include<stdlib.h>
#include<netdb.h> //hret
#include<string.h> //memset
#include<sys/socket.h>// for socklen_t
#include<arpa/inet.h> //ntop
#include<sys/types.h>
#include<unistd.h>
#include<fcntl.h>
#include<signal.h>
#include<cstdlib>

using namespace std;

//opcode rrq 1 data 3 ack 4 error 5
#define rrq 1
#define data 3
#define ack 4
#define error 5

//max block size 32MB, 2^16*512, 512 byte bocks each
#define block_size 512

//defining tftp block structure
struct tftp
{
    char * file_name;
    uint16_t opcode;
    uint16_t block_num;
    char * mode;
};

char temp[512];
```

```
/*void cr(FILE *src, FILE *dest)
{

char c;

while (is.get(c)!=EOF)
{
    if (c=='\n')
    {
        fputc('\r', dest);
        fputc('\n', dest);
    }

    else if (c=='\r')
    {
        fputc('\r', dest);
        fputc('\0', dest);
    }

    else
    fputc (c,dest);
}
}
*/

//converting from host byte order to network byte order for 16 bits
void hton_16 (char *buffer, unsigned short int k)
{
    k=htons(k);
    memcpy(buffer, &k, 2);
}

//converting from network byte order to host byte order for 16 bits
unsigned short int ntoh_16 (char * buffer)
{
    unsigned short int k;
    memcpy(&k, buffer, 2);
    k=ntohs(k);
    return k;
}

//to reap dead processes and avoid zombie processes
void sigchld_handler (int s)
{
    int errno;
    int saved_errno=errno;
    while (waitpid(-1, NULL, WNOHANG)>0);
    errno=saved_errno;
}


void *get_inaddr (struct sockaddr *sa)  //get sockaddr
{
    if (sa->sa_family == AF_INET)
    {
        return &(((struct sockaddr_in*)sa)->sin_addr); //IPv4
    }
     return &(((struct sockaddr_in6*)sa)->sin6_addr); //IPv6
}

struct tftp *decoding (char * block)
{
    struct tftp *result;
    result=(struct tftp *)malloc(sizeof(struct tftp));
```

```
    result->opcode=ntoh_16(block);

    if (result->opcode==rrq)
    {
    char temp[512];
    int m=2, n=0;

    while (block[m]!='\0')
    temp[n++]=block[m++];
    temp[n]='\0';

    result->file_name=(char *)malloc(strlen(temp)*sizeof(char));
    strcpy(result->file_name,temp);

    while(block[m]=='\0')
    m++;

    n=0;
    while(block[m]!='\0')
    temp[n++]=block[m++];

    temp[n]='\0';
    result->mode =(char *)malloc(strlen(temp)*sizeof(char));
    strcpy(result->mode, temp);
    }

else if(result->opcode==ack)
{
    result->block_num=ntoh_16(block+2);
}
return result;
}

char *encoding(uint16_t opcode, uint16_t block_num, char *msg, int length)
{
    char *block;
    if (opcode==data)
    {
        block= (char *)malloc((length+4)*sizeof(char));
        hton_16(block,opcode);
        hton_16(block+2, block_num);
        memcpy(block+4, msg, length);
        return block;
    }

    else if (opcode ==error)
    {
        block= (char *)malloc((length+5)*sizeof(char));
        hton_16(block,opcode);
        hton_16(block+2, block_num);
        memcpy(block+4, msg, length);
        memset(block+4+length,'\0', 1);
        return block;
    }
    return NULL;
}

int main (int argc, char *argv[])
{
    if (argc!=3)
    {
        fprintf (stderr, "usage: ./server IP port \n");
        return 1;
    }
```

```
    struct sigaction sa;
    struct addrinfo hints, *servinfo, *p;
    struct sockaddr_storage address_client, temp_address;
    socklen_t address_length, temp_length;

    memset(&hints, 0, sizeof(hints));
    hints.ai_family=AF_INET;
    hints.ai_socktype=SOCK_DGRAM;

    int rv;
    if ((rv=getaddrinfo(argv[1], argv[2], &hints, &servinfo))!=0)
    {
        fprintf(stderr, "getaddrinfo: %s \n",gai_strerror(rv));
        return 1;
    }

    int sock_fd;
    for (p=servinfo; p!=NULL; p=p->ai_next)
    {
        if ((sock_fd=socket(p->ai_family, p->ai_socktype, p->ai_protocol))==-1)
        {
            perror ("socket error");
            continue;
        }

        if (bind(sock_fd, p->ai_addr, p->ai_addrlen)==-1)
        {
            close(sock_fd);
            perror ("bind error");
            continue;
        }
        break;
    }

   char buffer[1024];
   int nbytes;

struct sockaddr new_sa;
new_sa =*(p->ai_addr);
struct sockaddr_in* new_address;
new_address = (struct sockaddr_in*) &new_sa;
new_address->sin_port= htons(0);

sa.sa_handler=sigchld_handler;
sigemptyset(&sa.sa_mask);
sa.sa_flags=SA_RESTART;
if (sigaction(SIGCHLD, &sa, NULL)==-1)
{
    perror("sigaction");
    exit(1);
}

fd_set master;
fd_set read_fds;
int fdmax;

FD_ZERO(&master);
FD_ZERO(&read_fds);

while (1)
{
    address_length=sizeof(address_client);
    if ((nbytes=recvfrom(sock_fd, buffer, 1023, 0, (struct sockaddr *)&address_client,
&address_length))==-1)
    {
```

```
        perror ("recvfrom");
        exit(2);
    }

    struct sockaddr_in* new_client = (struct sockaddr_in*)&address_client;
    struct tftp *rcv;
    rcv = decoding (buffer);
    cout<<"opcode ="<<rcv->opcode<<", Filename ="<<rcv->file_name<<", mode ="<<rcv-
>mode<<endl;


    if (rcv->opcode !=rrq)
    {
        cout<<"invalid opcode received"<<endl;
        continue;
    }

    if (!(fork()))
    {
        close(sock_fd);
        if((sock_fd=socket(p->ai_family, p->ai_socktype, p->ai_protocol))==-1)
        {
            perror ("socket error");
        }

        if ((rv=bind(sock_fd, &new_sa, sizeof(new_sa)))==-1)
        {
            perror ("bind error");
        }

        socklen_t length= sizeof(new_sa);
        if (getsockname(sock_fd, &new_sa, &length)==-1)
        {
            perror ("getsockname error");
        }

        ifstream file;
        file.open(rcv->file_name, ios::in|ios::binary);
        if(file.is_open()==false)
        {
            string msg="Could not open requested file";
            cout << msg << rcv->file_name << endl;

            char *errmsg=(char *)malloc(1024*sizeof(char));
            errmsg=strcpy(errmsg,msg.c_str());
            char *errblock=encoding(error,1,errmsg,(strlen(errmsg)));
            free(errmsg);
            if ((nbytes=sendto(sock_fd,errblock, (strlen(errmsg)+5),0,(struct sockaddr
*)&address_client, address_length))==-1)
            {
                perror ("sendto");
                exit(2);
            }
            exit(0);
        }

        streampos begin_file,end_file;
        begin_file= file.tellg();
        file.seekg(0,ios::end);
        end_file=file.tellg();
        file.seekg(0,ios::beg);

        unsigned long file_size= end_file-begin_file;
        unsigned long nblocks= (file_size/512) +1;
```

```
        cout<<"file size ="<<file_size<<", blocks="<<nblocks<<endl;

        char file_buffer[513];
        int ack_no=0, retransmit =0, msg_length;
        int block_num=0;
        unsigned long net_ack_no=0;

        free(rcv->file_name);
        free (rcv->mode);
        free(rcv);

        FD_SET(sock_fd, &master);
        fdmax=sock_fd;
        struct timeval tv;
        tv.tv_sec= 0;
        tv.tv_usec=100000;

        while(net_ack_no<nblocks)
        {
           if (ack_no==block_num)
           {
              file.read(file_buffer, 512);

              msg_length=file.gcount();
              block_num=(block_num+1)%65536;
              retransmit=0;
           }

           if (retransmit>0)
           {
              if(retransmit==11)
              {
                 cout<<"request timedout"<<endl;
                 break;
              }

           cout<<"retransmitting attempt"<<retransmit<<"for block"<<block_num<<endl;
           }
           char *block=encoding (data, block_num, file_buffer, msg_length);
           if ((nbytes=sendto(sock_fd, block, msg_length+4, 0, (struct sockaddr *)&address_client,
address_length))==-1)
           {
              perror("sendto error");
              exit(1);
           }
           free(block);

           tv.tv_usec=100000;
           read_fds=master;
           if(select(fdmax+1, &read_fds, NULL, NULL, &tv)==-1)
           {
              perror("select error");
              exit(1);
           }

           if (FD_ISSET(sock_fd, &read_fds))
           {
              temp_length=sizeof(temp_address);
              if ((nbytes=recvfrom(sock_fd, buffer, 4, 0, (struct sockaddr *)&temp_address,
&temp_length))==-1)
              {
                 perror ("recvfrom error");
                 exit(1);
              }
```

```
            struct sockaddr_in* temp_client_1 = (struct sockaddr_in*) &address_client;
            if(temp_client_1->sin_addr.s_addr==new_client->sin_addr.s_addr)
            {
               rcv=decoding(buffer);
               ack_no=rcv->block_num;
               net_ack_no++;
               free(rcv);
            }
         }
         retransmit++;
      }

      file.close();
      close(sock_fd);
      if (net_ack_no == nblocks)
      {
         cout<<"acknowledgement number "<<ack_no<<endl;
         cout<<"block number "<<nblocks<<endl;
         cout<<"file transmitted successfully "<<endl;
      }
      return 0;
      exit(0);
   }
      free(rcv->file_name);
      free(rcv->mode);
      free(rcv);
}
close(sock_fd);
freeaddrinfo(servinfo);
return 0;
}
```

Acknowledgements

1. Beej's Guide
2. www.tutorialspoint.com

Multiple references from GeeksforGeeks, and other online coding helpline resources.