ELECTRICAL & COMPUTER ENGINEERING
TEXAS A&M UNIVERSITY

SHASHANK KAMATH KALASA MOHANDAS
UIN: **627003580**

SRINIVAS PRAHALAD SUMUKHA
UIN: **627008254**

TEAM 6

ASSIGNMENT 3

Computer Communication and Networks
ECEN 602
FALL 2018

Role of SHASHANK KAMATH KALASA MOHANDAS

- tftp_server.c
- README file

Role of SRINIVAS PRAHALAD SUMUKHA

- tftp_server.c
- MAKEFILE
- README file

How to run:

1. Run the Makefile in the terminal by typing the command 'make'
2. One object file named "tftp_server" gets created, execute the server program by typing './tftp_server *ip_address port_number'*. The tftp server gets executed.
3. In a new terminal run the client program by typing 'tftp*'.* The client gets executed, now use standard tftp arguments to send and receive the desired files.

Files included in this assignment:

- tftp_server.c
- makefile
- README

MAKEFILE:

```
########################################################################

#Team 6 Tftp server makefile

#######################################################################
build:tftp_server.c

    gcc tftp_server.c -o tftp_server

clean:
    -rm -rf tftp_server
```

ARCHITECTURE

Server Implementation:

- Firstly, a server is created using socket(), bind() functions.
- The server listens to new clients using a port input by the user. As the new client gets connected, a new port is assigned for further communication between the server and the client. This is achieved by forking the process.
- There are mainly two types of packets exchanged between the server and the client which are RRQ and WRQ. Also ACK packets are exchanged to indicate successful communication.
- There are two modes of communication, octet and netascii.
- When RRQ packet is received firstly, a child process is created and new port is assigned. Then the filename and mode is extracted from the packet. The file is accessed and sent in chunks of 512 bytes. The next chunk of 512 bytes is sent when the first chunk is acknowledged by the client.
- Timeout is also implemented on the server side and a maximum of 10 retries are made. Upon failure of 10 tries, the packet is discarded.
- When WRQ packet is sent, again a new port is created and ACK is sent. Later, the client sends data of 512 bytes and these are added to the new file correspondingly.
- When a client disconnects or the timeout occur, the client socket is closed and all the client process and resources are cleaned up.

SERVER CODE:

tftp_server.c

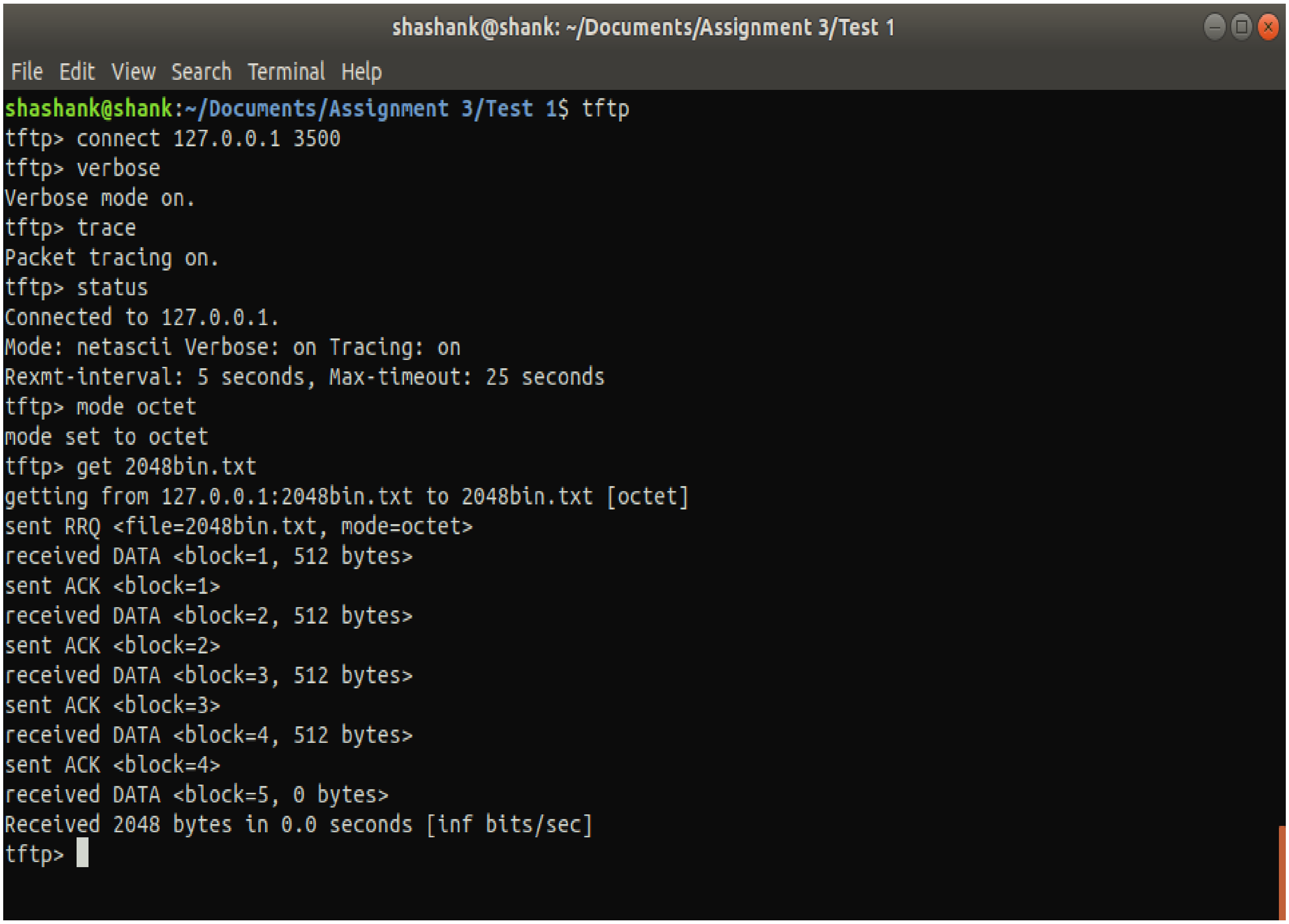
```
/**************************************************************************************************
**
tftp_server.c from TEAM-6
Assignment 3
**************************************************************************************************
***/
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <stdint.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
```

```
#include <netdb.h>
#include <signal.h>
#include <sys/time.h>
#include <sys/select.h>
#include <sys/wait.h>
#include <ctype.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <netdb.h>

#define NONE 0
#define RRQ 1
#define WRQ 2
#define DATA 3
#define ACK 4
#define ERROR 5

#define max_buffer_length 520
int nextchar =-1;

/***************************************************************************************
*******************************
The getaddress is a user defined function which compares the family to IPv4 or IPv6 and
returns the IP address.
****************************************************************************************
******************************/
void *getaddress(struct sockaddr *sa)
{
    if (sa->sa_family == AF_INET)
    {
        return &(((struct sockaddr_in*)sa)->sin_addr);
    }
        return &(((struct sockaddr_in6*)sa)->sin6_addr);
}

/***************************************************************************************
*******************************
The getport is a user defined function which compares the family to IPv4 or IPv6 and
returns the port number.
****************************************************************************************
******************************/
void *getport(struct sockaddr *sa)
{
    if (sa->sa_family == AF_INET)
    {
        return &(((struct sockaddr_in*)sa)->sin_port);
    }
        return &(((struct sockaddr_in6*)sa)->sin6_port);
}


void sigchld_handler(int signum)
{
    int saved_errno = errno;

    while (waitpid( -1, NULL, WNOHANG) > 0) ;

    errno = saved_errno;
}
```

```
int readable_timeo(int fd,int sec)
{
    fd_set rset;
    struct timeval tv;

    FD_ZERO(&rset);
    FD_SET(fd,&rset);

    tv.tv_sec= sec;
    tv.tv_usec=0;
    return (select(fd+1,&rset,NULL,NULL,&tv));/* >0 if descriptor is readable */
}

ssize_t read_file(FILE *fp,char *pointer,uint16_t block_number,char *send_buf)
{
    char c;
    int cnt;
    memset(send_buf,0,strlen(send_buf));
    pointer=send_buf;
    *pointer=0x00;
    pointer++;
    *pointer=0x03;//encoding as DATA
    pointer++;
    if(block_number<=255)
    {
        *pointer=0x00;
        pointer++;
        *pointer=block_number;
        pointer++;
    }
    else
    {
        *pointer=((block_number)&(0xFF00))>>8;
        pointer++;
        *pointer=(block_number)&(0x00FF);
        pointer++;
    }

    for(cnt=0;cnt<512;cnt++)
    {
        if(nextchar>=0)
        {
            *pointer++ =nextchar;
            nextchar=-1;
            continue;
        }
        c=getc(fp);

        if(c==EOF)
        {
            if(ferror(fp))
                printf("read err from getc on local file\n");
            return(cnt+4);
        }

        else if(c=='\n')
        {
            c='\r';
            nextchar='\n';
        }
        else if(c=='\r')
        {
            nextchar='\0';
        }
        else
            nextchar=-1;
        *pointer++=c;
    }
    cnt=516;
    return cnt;
}

ssize_t write_file(FILE *fp2,char *pointer,int recvbytes,char *recv_buf)
{
pointer=recv_buf;
pointer=pointer+4;
char c,ch;
int cnt;
for(cnt=0;cnt<recvbytes-4;cnt++)
{
    c=*pointer;
    pointer++;
    ch=*pointer;
    if((c=='\r' )&& (ch=='\n') )
    {
        putc(c,fp2);

    }
    else if((c=='\r') && (ch=='\0'))
    {
        putc(c,fp2);

    }
    else
    {
    putc(c,fp2);
    }
}
return cnt;
}

int main(int argc, char *argv[])
{
    struct addrinfo addressinfo, *servicelist, *loopvariable;
    struct sockaddr_storage str_addr;
    struct sockaddr_in server_addr;
    socklen_t addr_size;
    struct sigaction sa;

    int socketfd, clientsocketfd;
    int opcode;
    int number_bytes=0;
    int num_bytes_sent=0;
    int num_bytes_rcv=0;
    int yes=1;
    int time_value=0;
    int flag=1;
    int flag1=1;

    unsigned char send_buf[max_buffer_length];
    unsigned char recv_buf[max_buffer_length];
    char buf[max_buffer_length]={0};
    char s[INET6_ADDRSTRLEN];

    pid_t process_id=0;

    //char c;

    memset(&addressinfo,0,sizeof (addressinfo)); // Making the addressinfo struct zero
    addressinfo.ai_family = AF_UNSPEC;// Not defining whether the connection is IPv4 or
IPv6
    addressinfo.ai_socktype = SOCK_DGRAM;

    if(argc !=3)
    {
        printf("Server: Excess Arguments Passed \n");
        exit(1);
    }

    if ((flag = getaddrinfo(argv[1], argv[2], &addressinfo, &servicelist)) != 0)
    {
        printf("GetAddrInfo Error");
        exit(1);
    }
    //printf("Server: Done with getaddrinfo \n");

    // Traversing the linked list for creating the socket
    for(loopvariable = servicelist; loopvariable != NULL; loopvariable = (loopvariable
-> ai_next ))
    {
        if((socketfd = socket(loopvariable -> ai_family, loopvariable -> ai_socktype,
loopvariable -> ai_protocol)) ==  -1 )
        {
            printf("Server: Listener Socket Created.\n");
            continue;
        }
        if (setsockopt(socketfd, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof(int)) ==  -1)
        {
            perror("Server: SetSockOpt\n");
            exit(1);
        }
        //Binding the socket
        if (bind(socketfd, loopvariable->ai_addr, loopvariable->ai_addrlen) ==  -1)
        {
            close(socketfd);
            perror("Server: Listerner Bind Error.\n");
            continue;
        }
        break;
    }
    // Freeing the linked  list
    freeaddrinfo(servicelist);

    if (loopvariable == NULL)
    {
        printf("Server: Listerner failed to bind.\n");
        exit(1);
    }
    printf("Server: Waiting for connections: \n");

    while(1)
    {
        addr_size = sizeof(str_addr);
        if(number_bytes = recvfrom(socketfd,buf,max_buffer_length-1,0,(struct sockaddr
*)&str_addr,&addr_size)== -1)
        {
            perror("Server: recvfrom error");
            exit(1);
        }
        printf("Server: got packet from %s and port : %d
\n",inet_ntop(str_addr.ss_family,getaddress((struct sockaddr *)&str_addr),s, sizeof s),
ntohs(getport((struct sockaddr *)&str_addr)));
        buf[number_bytes]='\0';

        if ((process_id = fork()) ==  -1)
        {
           printf("Server: Fork error \n");
           exit(1);
        }
        else if (process_id == 0)
        {
            opcode=buf[1];
            sa.sa_handler = sigchld_handler;
            sigemptyset(&sa.sa_mask);
            sa.sa_flags = SA_RESTART;
            if (sigaction(SIGCHLD, &sa, NULL) == -1)
            {
                perror("sigaction");
                exit(1);
            }

            server_addr.sin_family = AF_INET;
            server_addr.sin_addr.s_addr = htonl (INADDR_ANY);
            server_addr.sin_port=htons(0); // Assigning a new port

            //Creating a new socket for the client
            if((clientsocketfd = socket(AF_INET,SOCK_DGRAM,0)) == -1)
            {
                perror("Server: New Client Socket error\n");
                exit(1);
            }
            if (setsockopt(clientsocketfd,SOL_SOCKET,SO_REUSEADDR,&yes,sizeof(int)) ==
-1)
            {
                perror("Server: Client SetSockOpt\n");
                exit(1);
            }
            if (bind(clientsocketfd, (struct sockaddr *)&server_addr, sizeof
server_addr) == -1)
            {
                close(clientsocketfd);
                perror("Server_Client Bind error");
                exit(1);
            }
            char filename[30];
            char mode[10];

            //Getting the filename from the input
            strcpy(filename,&buf[2]);//check this 2 in buffer
            filename[strlen(filename)]='\0';
            //printf("Filename %s \n",filename);
            strcpy(mode,&buf[3+strlen(filename)]);
            mode[strlen(mode)]='\0';
            //printf("Mode %s \n",mode);

            //Checking the packet type for RRQ
            if(opcode == RRQ)
            {
                if(!strcmp(mode,"netascii"))
                {
                    FILE *fp;
                    fp=fopen(filename,"r");
                    unsigned char err_msg[520];
                    char print_msg[30]="FIile Does not exist";
                    int len=strlen(print_msg);
                    char *pointer4;
                    if(fp==NULL)
                    {
                        printf("File does not exist\n");
                        memset(err_msg,0,strlen(err_msg));
                        pointer4=err_msg;
                        *pointer4=0x00;
                        pointer4++;
                        *pointer4=0x05;
                        pointer4++;
                        *pointer4=0x00;
                        pointer4++;
                        *pointer4=0x06;
                        pointer4++;
                        strcpy(pointer4,print_msg);
                        pointer4=pointer4+len;
                        *pointer4=0;
                        if((num_bytes_sent=sendto(clientsocketfd,err_msg,24,0,(struct
sockaddr*)&str_addr,addr_size))==-1)
                        {
                            perror("Error Message send error");
                        }
                        printf("Client Disconnected \n");
                        close(clientsocketfd);
                        exit(1);

                    }


                    uint16_t block_number=1;
                    uint16_t block_number_rcv=0;
                    char *pointer1;
                    int count1;
                    while((count1=read_file(fp,pointer1,block_number,send_buf))<=516)
                    {
                        if(count1==0) {break;}
                        int time_count =0;
                        flag1=1;
                        do
                        {

if((num_bytes_sent=sendto(clientsocketfd,send_buf,count1,0,(struct sockaddr
*)&str_addr, addr_size))== -1)
                            {
                                perror("Server: Sendto error\n");
                                break;
                            }
                            //printf("Number of bytes sent: %d \n",num_bytes_sent);
                            while((time_value<=0 || time_value==1) && (time_count<=10))
                            {
                                if((time_value=readable_timeo(clientsocketfd,1))==0)
                                {
                                    printf("Server: TImeout\n");
                                    time_count++;
                                    flag1=1;
                                    break;

                                }
                                else if(time_value==-1)
                                {
                                    printf("Server: Timeout Error\n");
                                }
                                else
                                {
                                    printf("Server: Socket Ready\n");
                                    break;
                                }
                            }
                            if(time_count>10)
                            {
                                printf("Server: Timeout Occured\n");
                                fclose(fp);
                                close(clientsocketfd);
                                flag1=0;
                                exit(1);

                            }
                            memset(recv_buf,0,strlen(recv_buf));

if((num_bytes_rcv=recvfrom(clientsocketfd,recv_buf,max_buffer_length-1,0,(struct
sockaddr *)&str_addr,&addr_size)) == -1)
                            {
                                perror("Server: Client receive error\n");
                                break;
                            }
                            else
                            {
                                if(recv_buf[1]==ACK)
                                {
                                    block_number_rcv=(recv_buf[2]<<8 | recv_buf[3]);
                                    if(block_number_rcv==block_number)
                                    {
                                        printf("Packet %d
acknowledged\n",block_number);

                                        block_number++;
                                        time_value=0;
                                        flag1=0;
                                    }
                                    else
                                    {
                                        flag1=1;
                                        //goto back1;
                                        time_value=0;
                                    }
                                }

                            }
                        }while(flag1);//do while end

                        if(count1>=0 && count1<516)
                        {
                            printf("Final ACK has been received\n");
                            printf("Server: Transfer Done\n");
                            break;
                            flag1=0;
                        }
                    }//end of while(count)
                }//netascii end

                if(!strcmp(mode,"octet"))
                {
                        ssize_t send_bytes;
                        char *pointer1;
                        pointer1=send_buf;
                        uint16_t block_number=1;
                        int fp1;
                        fp1=open(filename,O_RDONLY);
                        unsigned char err_msg[520];
                    char print_msg[30]="FIile Does not exist";
                    int len=strlen(print_msg);
                    char *pointer4;
                    if(fp1==-1)
                    {
                        printf("File does not exist\n");
                        memset(err_msg,0,strlen(err_msg));
                        pointer4=err_msg;
                        *pointer4=0x00;
                        pointer4++;
                        *pointer4=0x05;
                        pointer4++;
                        *pointer4=0x00;
                        pointer4++;
                        *pointer4=0x06;
                        pointer4++;
                        strcpy(pointer4,print_msg);
                        pointer4=pointer4+len;
                        *pointer4=0;
                        if((num_bytes_sent=sendto(clientsocketfd,err_msg,24,0,(struct
sockaddr*)&str_addr,addr_size))==-1)
                            {
                                perror("Error Message send error");
                            }
                        printf("Client Disconnected \n");
                        close(clientsocketfd);
                        exit(1);
                    }
                    flag1=1;
                    while(1)
                    {
                        memset(send_buf,0,strlen(send_buf));
                        pointer1=send_buf;
                        *pointer1=0x00;
                        pointer1++;
                        *pointer1=0x03;
                        pointer1++;

                        if(block_number<=255)
                        {

                            *pointer1=0x00;
                            pointer1++;
                            *pointer1=block_number;
                            pointer1++;
                        }
                        else
                        {
                            *pointer1=((block_number)&(0xFF00))>>8;
                            pointer1++;
                            *pointer1=(block_number)&(0x00FF);
                            pointer1++;
                        }
                        send_bytes=read(fp1,pointer1,512);

                        int time_count=0;
                        do
                        {


if(num_bytes_sent=sendto(clientsocketfd,send_buf,send_bytes+4,0,(struct sockaddr
*)&str_addr, addr_size) == -1)
                        {
                            perror("Server: Error in sending in octet mode\n");
                        }



                        while((time_value<=0 || time_value==1) && (time_count<=10))
                        {
                            if((time_value=readable_timeo(clientsocketfd,1))==0)
                            {
                                printf("Server: TImeout\n");
                                time_count++;
                                flag1=1;
                                //break;

                            }
                            else if(time_value==-1)
                            {
                                printf("Server: Timeout Error\n");
                            }
                            else
                            {
                                //printf("Server: Socket Ready\n");
                                break;
                            }
                        }//timeout while end

                        if(time_count>10)
                        {
                            printf("Server: Timeout Occured\n");
                            //fp1=0;
                            close(clientsocketfd);
                            flag1=0;
                            exit(1);

                        }

                        memset(recv_buf,0,strlen(recv_buf));

if(num_bytes_rcv=recvfrom(clientsocketfd,recv_buf,max_buffer_length-1,0,(struct
sockaddr *)&str_addr,&addr_size)== -1)
                        {
                            perror("Receive error");
                        }
                        else
                        {
                            if(recv_buf[1]==ACK)
                            {
                                uint16_t
block_number_rcv=(recv_buf[2]<<8)|(recv_buf[3]);
                                if(block_number_rcv == block_number)
                                {
                                    printf("Packet %d acknowledged\n",block_number);
                                    block_number++;
                                    flag1=0;

                                }
                                else
                                {
                                    flag1=1;
                                    time_value=0;
                                }
                            }//ACK end
                        }
                    }while(flag1);//end do here

                    if(send_bytes>=0 && send_bytes<512)
                        {
                            printf("Final ACK received\n");
                            printf("File transfer done\n");
                            break;
                        }
                }//end of while
                }//octet end
            close(clientsocketfd);
            }//RRQ end

            if(opcode == WRQ)
            {
                if(!strcmp(mode,"netascii"))
                {
                    char *pointer3, *pointer4;
                    unsigned char err_msg[520];
                    char print_msg[30]="FIile Exists";
                    uint16_t block_number =1;
                    int len=strlen(print_msg);
                    uint16_t block_number_rcv;
                    int written_count;
                    pointer3=send_buf;
                    *pointer3=0x00;
                    pointer3++;
                    *pointer3=ACK;
                    pointer3++;
                    *pointer3=0x00;
                    pointer3++;
                    *pointer3=0x00;
                    pointer3++;

                    if((num_bytes_sent=sendto(clientsocketfd,send_buf,4,0,(struct
sockaddr*)&str_addr,addr_size))==-1)
                    {
                        perror("Server-Client: Send to error");
                    }
                    FILE *fp2;
                    fp2=fopen(filename,"wx");
                    if(fp2==NULL)
                    {
                        printf("File already exists");
                        memset(err_msg,0,strlen(err_msg));
                        pointer4=err_msg;
                        *pointer4=0x00;
                        pointer4++;
                        *pointer4=0x05;
                        pointer4++;
                        *pointer4=0x00;
                        pointer4++;
                        *pointer4=0x06;
                        pointer4++;
                        strcpy(pointer4,print_msg);
                        pointer4=pointer4+len;
                        *pointer4=0;
                        if((num_bytes_sent=sendto(clientsocketfd,err_msg,24,0,(struct
sockaddr*)&str_addr,addr_size))==-1)
                        {
                            perror("Error Message send error");
                        }
                        close(clientsocketfd);
                        exit(1);
                    }
                    flag1=1;
                    do
                    {
                        memset(recv_buf,0,strlen(recv_buf));

if((num_bytes_rcv=recvfrom(clientsocketfd,recv_buf,max_buffer_length-4,0,(struct
sockaddr *)&str_addr,&addr_size))== -1)
                        {
                            perror("Client receive error\n");
                        }
                        recv_buf[num_bytes_rcv]='\0';
                        if(recv_buf[1]==DATA)
                        {
                            block_number_rcv=(recv_buf[2]<<8)| (recv_buf[3]);
                            pointer3=recv_buf;
                            if(block_number_rcv==block_number)
                            {

written_count=write_file(fp2,pointer3,num_bytes_rcv,recv_buf);
                                memset(send_buf,0,strlen(send_buf));
                                pointer3=send_buf;
                                *pointer3=0x00;
                                pointer3++;
                                *pointer3=ACK;
                                pointer3++;

                                if(block_number_rcv<=255)
                                {

                                    *pointer3=0x00;
                                    pointer3++;
                                    *pointer3=block_number_rcv;
                                    pointer3++;
                                }
                                else
                                {
                                    *pointer3=((block_number_rcv)&(0xFF00))>>8;
                                    pointer3++;
                                    *pointer3=(block_number_rcv)&(0x00FF);
                                    pointer3++;
                                }


if((num_bytes_sent=sendto(clientsocketfd,send_buf,4,0,(struct sockaddr *)&str_addr,
addr_size)) == -1)
                                {
                                    perror("server_child_sendto error");
                                }
                                block_number++;
                                if(num_bytes_rcv<516)
                                {
                                flag1=0;
                                close(clientsocketfd);
                                exit(1);
                                }

                                else if(num_bytes_rcv==516)
                                {
                                flag1=1;
                                }
                            }
                        }
                    }while(flag1);
                }//netascii end

                if(!strcmp(mode,"octet"))
                {
                    char *pointer3, *pointer4;
                    unsigned char err_msg[520];
                    char print_msg[30]="FIile Exists";
                    uint16_t block_number =1;
                    int len=strlen(print_msg);
                    uint16_t block_number_rcv;
                    int written_count;
                    pointer3=send_buf;
                    *pointer3=0x00;
                    pointer3++;
                    *pointer3=ACK;
                    pointer3++;
                    *pointer3=0x00;
                    pointer3++;
                    *pointer3=0x00;
                    pointer3++;

                    if((num_bytes_sent=sendto(clientsocketfd,send_buf,4,0,(struct
sockaddr*)&str_addr,addr_size))==-1)
                    {
                        perror("Server-Client: Send to error");
                    }
                    int fp3;
                    fp3=open(filename,O_WRONLY|O_CREAT|O_EXCL,0644);
                    if(fp3==-1)
                    {
                        if(errno==EEXIST)
                        {
                            printf("File already exists");
                            memset(err_msg,0,strlen(err_msg));
                            pointer4=err_msg;
                            *pointer4=0x00;
                            pointer4++;
                            *pointer4=0x05;
                            pointer4++;
                            *pointer4=0x00;
                            pointer4++;
                            *pointer4=0x06;
                            pointer4++;
                            strcpy(pointer4,print_msg);
                            pointer4=pointer4+len;
                            *pointer4=0;

if((num_bytes_sent=sendto(clientsocketfd,err_msg,24,0,(struct
sockaddr*)&str_addr,addr_size))==-1)
                            {
                                perror("Error Message send error");
                            }
                            close(clientsocketfd);
                            exit(1);
                        }
                    }
                    flag1=1;
                    do
                    {
                        memset(recv_buf,0,strlen(recv_buf));

if((num_bytes_rcv=recvfrom(clientsocketfd,recv_buf,max_buffer_length-4,0,(struct
sockaddr *)&str_addr,&addr_size))== -1)
                        {
                            perror("Client receive error\n");
                        }
                        recv_buf[num_bytes_rcv]='\0';
                        ////print here if needed
                        if(recv_buf[1]==DATA)
                        {
                            block_number_rcv=(recv_buf[2]<<8)| (recv_buf[3]);

                            if(block_number_rcv==block_number)
                            {
                                write(fp3,&recv_buf[4],num_bytes_rcv-4);
                                memset(send_buf,0,strlen(send_buf));
                                pointer3=send_buf;
                                *pointer3=0x00;
                                pointer3++;
                                *pointer3=ACK;
                                pointer3++;

                                if(block_number_rcv<=255)
                                {
                                    *pointer3=0x00;
                                    pointer3++;
                                    *pointer3=block_number_rcv;
                                    pointer3++;
                                }
                                else
                                {
                                    *pointer3=((block_number_rcv)&(0xFF00))>>8;
                                    pointer3++;
                                    *pointer3=(block_number_rcv)&(0x00FF);
                                    pointer3++;
                                }


if((num_bytes_sent=sendto(clientsocketfd,send_buf,4,0,(struct sockaddr *)&str_addr,
addr_size)) == -1)
                                {
                                    perror("server_child_sendto error");
                                }
                                block_number++;
                                if(num_bytes_rcv<516)
                                {
                                printf("File transfer Complete\n");
                                flag1=0;
                                close(clientsocketfd);
                                exit(1);
                                }

                                else if(num_bytes_rcv==516)
                                {
                                flag1=1;
                                }
                            }
                        }
                    }while(flag1);
                    }//octet end
            }//end WRQ
        }//Process_id creation
    }//Infinite While loop
    close(socketfd);
    return 0;
}//Main
```

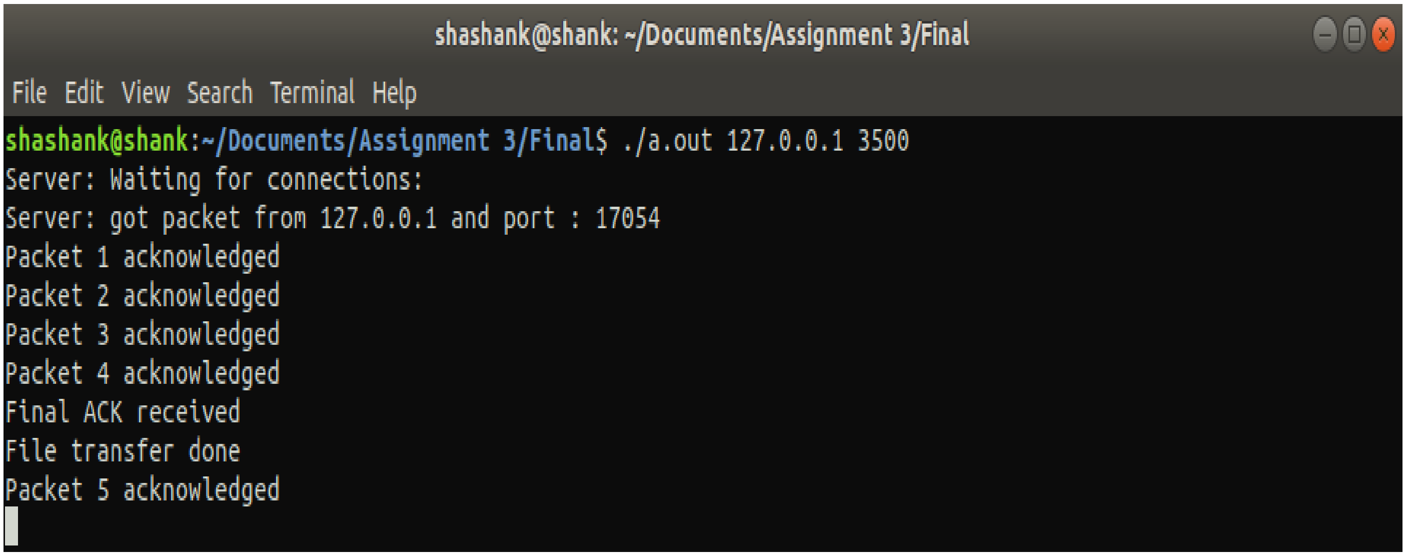
TEST CASES:

1. **Transferring a binary file of 2048 bytes and checking that it matches source file.**



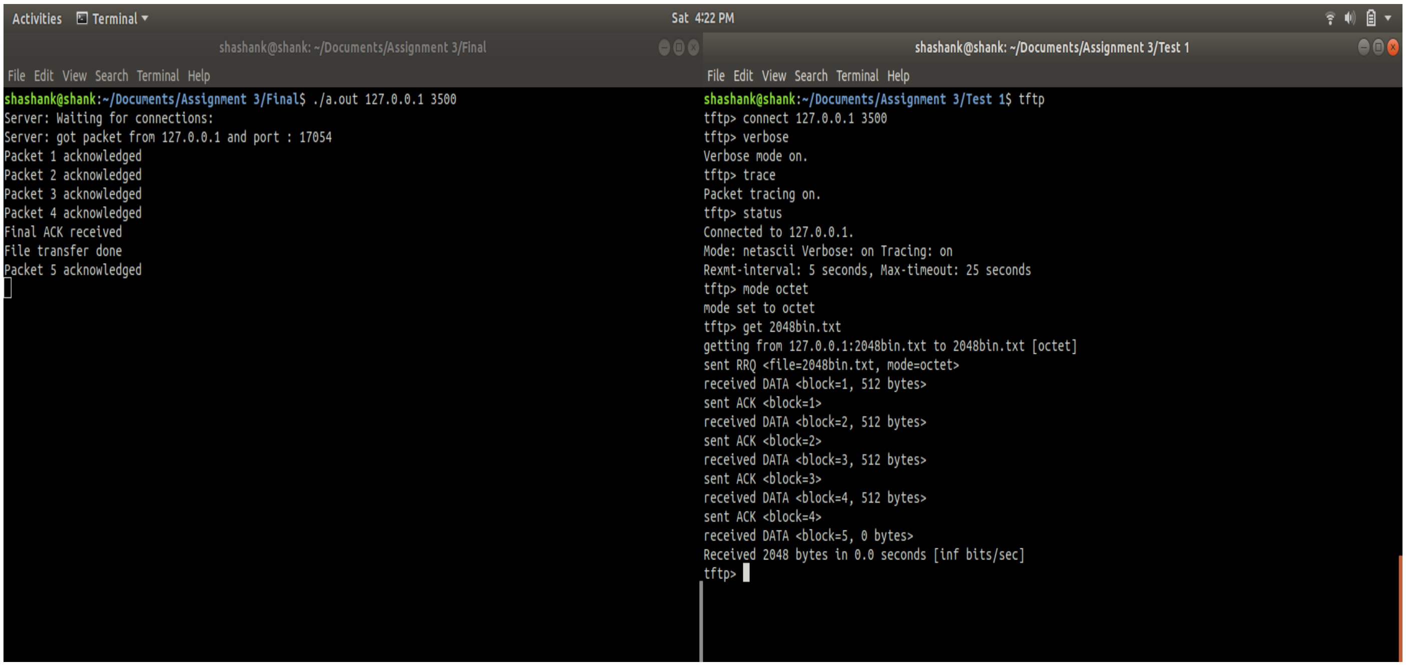
shashank@shank: ~/Documents/Assignment 3/Test 1
File Edit View Search Terminal Help

```
shashank@shank:~/Documents/Assignment 3/Test 1$ tftp
tftp> connect 127.0.0.1 3500
tftp> verbose
Verbose mode on.
tftp> trace
Packet tracing on.
tftp> status
Connected to 127.0.0.1.
Mode: netascii Verbose: on Tracing: on
Rexmt-interval: 5 seconds, Max-timeout: 25 seconds
tftp> mode octet
mode set to octet
tftp> get 2048bin.txt
getting from 127.0.0.1:2048bin.txt to 2048bin.txt [octet]
sent RRQ <file=2048bin.txt, mode=octet>
received DATA <block=1, 512 bytes>
sent ACK <block=1>
received DATA <block=2, 512 bytes>
sent ACK <block=2>
received DATA <block=3, 512 bytes>
sent ACK <block=3>
received DATA <block=4, 512 bytes>
sent ACK <block=4>
received DATA <block=5, 0 bytes>
Received 2048 bytes in 0.0 seconds [inf bits/sec]
tftp> 
```



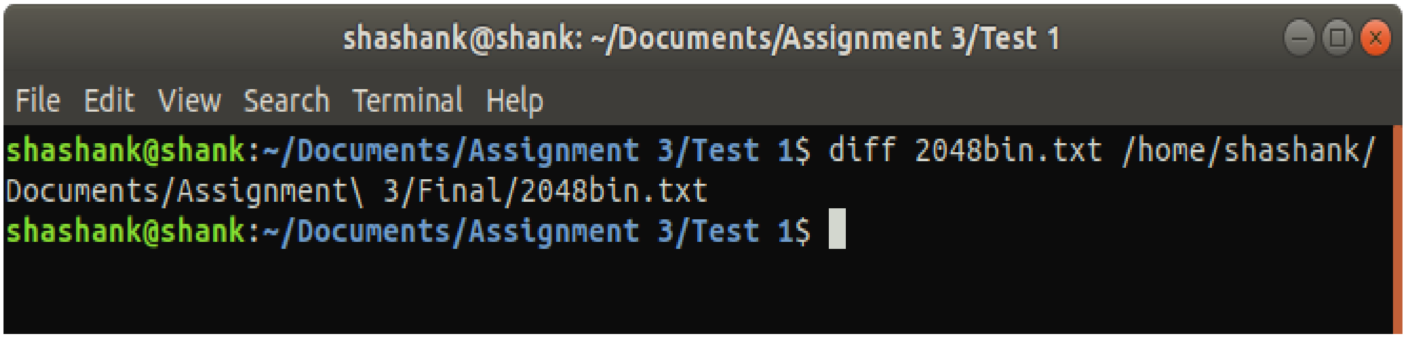
shashank@shank: ~/Documents/Assignment 3/Final
File Edit View Search Terminal Help

```
shashank@shank:~/Documents/Assignment 3/Final$ ./a.out 127.0.0.1 3500
Server: Waiting for connections:
Server: got packet from 127.0.0.1 and port : 17054
Packet 1 acknowledged
Packet 2 acknowledged
Packet 3 acknowledged
Packet 4 acknowledged
Final ACK received
File transfer done
Packet 5 acknowledged
```


Activities
Terminal
Sat 4:22 PM
shashank@shank: ~/Documents/Assignment 3/Final
File Edit View Search Terminal Help
shashank@shank: ~/Documents/Assignment 3/Test 1
File Edit View Search Terminal Help

```
shashank@shank:~/Documents/Assignment 3/Final$ ./a.out 127.0.0.1 3500
Server: Waiting for connections:
Server: got packet from 127.0.0.1 and port : 17054
Packet 1 acknowledged
Packet 2 acknowledged
Packet 3 acknowledged
Packet 4 acknowledged
Final ACK received
File transfer done
Packet 5 acknowledged
```

```
shashank@shank:~/Documents/Assignment 3/Test 1$ tftp
tftp> connect 127.0.0.1 3500
tftp> verbose
Verbose mode on.
tftp> trace
Packet tracing on.
tftp> status
Connected to 127.0.0.1.
Mode: netascii Verbose: on Tracing: on
Rexmt-interval: 5 seconds, Max-timeout: 25 seconds
tftp> mode octet
mode set to octet
tftp> get 2048bin.txt
getting from 127.0.0.1:2048bin.txt to 2048bin.txt [octet]
sent RRQ <file=2048bin.txt, mode=octet>
received DATA <block=1, 512 bytes>
sent ACK <block=1>
received DATA <block=2, 512 bytes>
sent ACK <block=2>
received DATA <block=3, 512 bytes>
sent ACK <block=3>
received DATA <block=4, 512 bytes>
sent ACK <block=4>
received DATA <block=5, 0 bytes>
Received 2048 bytes in 0.0 seconds [inf bits/sec]
tftp>
```


shashank@shank: ~/Documents/Assignment 3/Test 1
File Edit View Search Terminal Help

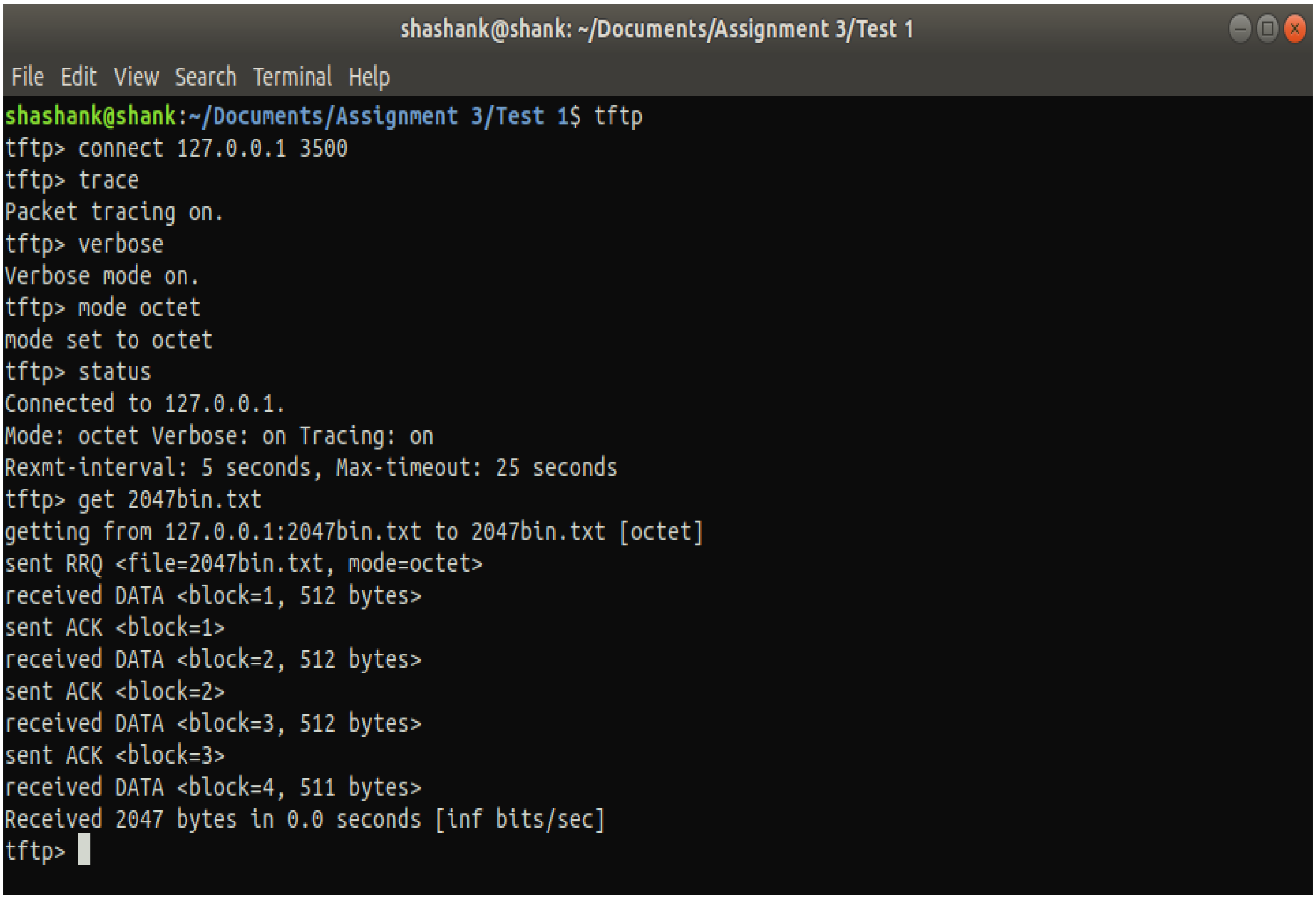
```
shashank@shank:~/Documents/Assignment 3/Test 1$ diff 2048bin.txt /home/shashank/
Documents/Assignment\ 3/Final/2048bin.txt
shashank@shank:~/Documents/Assignment 3/Test 1$
```

This test case showcases the ability of the server to handle a download request of binary file (size 2048 bytes) by the client. Initially the server has the binary file in this case it is "2048bin.txt". The client connects to the server using the ip. To enable the octet mode on the client side the command "mode octet" is used. Later the file is downloaded using the command "get "2048bin.txt", the server gets the request from the client and services it and everytime a the server gets an acknowledgment it prints the ack and at then end of the transfer the server also prints the status to indicate the completion of the file transfer. Note: The client used in this test case is a standard ubuntu tftp client. For more clarity verbose and trace are enabled in the above test case using the command "verbose" and "trace".

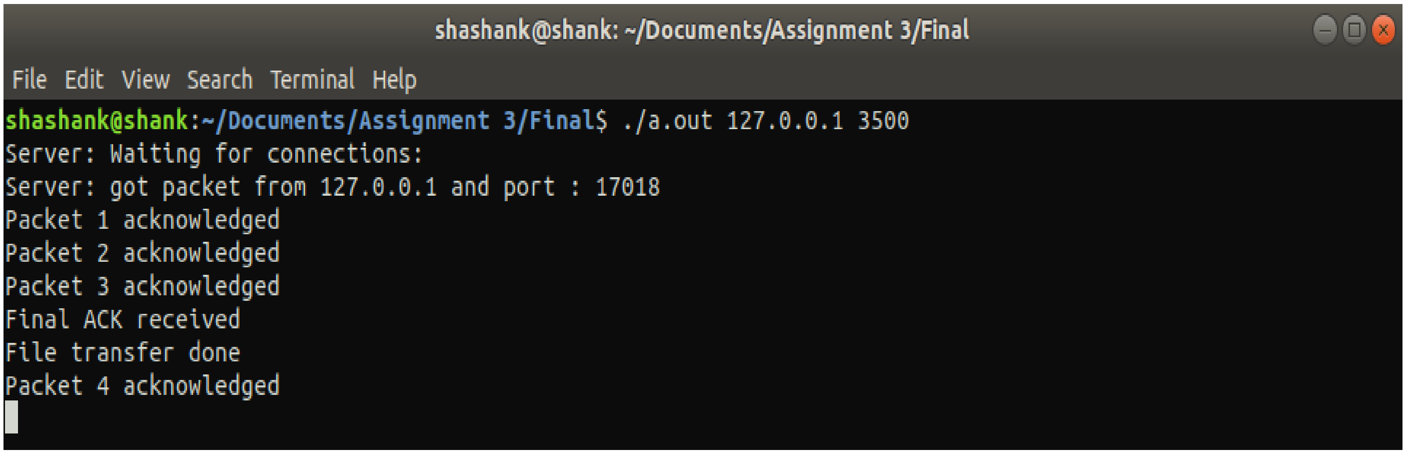
The first screenshot represents the client side of the interaction and the second screenshot represents the server side of the tftp interaction. The ip address of the server is 127.0.0.1 and the port number is 3500, the client connects to the server using the ip 127.0.0.1 and port number 3500 using the command "connect 127.0.0.1 3500".

The third screenshot is a combined screenshot and the fourth screenshot shows the result of the tftp. The bin file which was initially at the server is transferred to the client using tftp protocol.

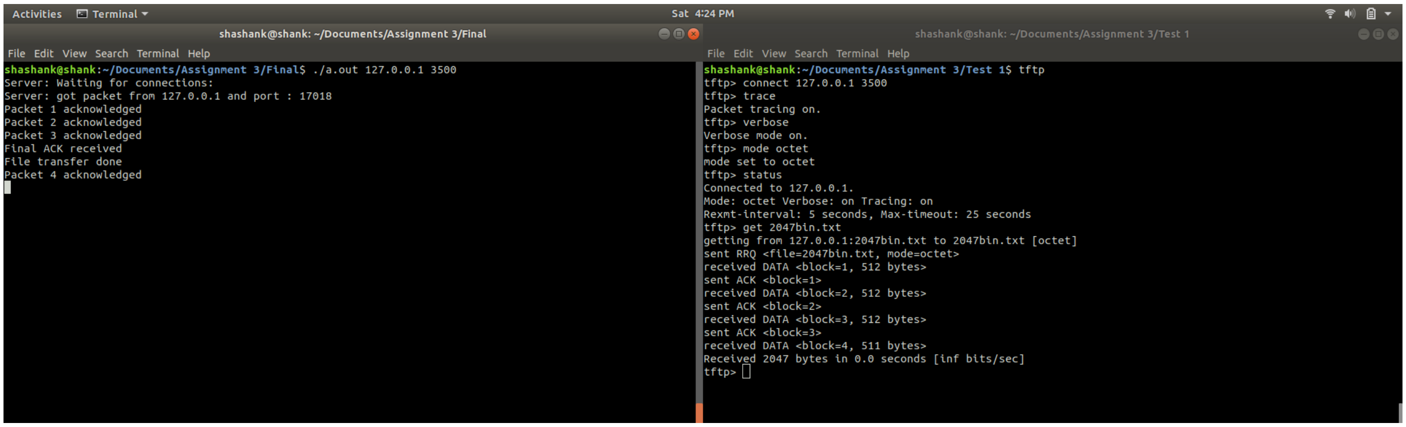
2. Transferring a binary file of 2047 bytes and checking that it matches source file.


shashank@shank: ~/Documents/Assignment 3/Test 1
File Edit View Search Terminal Help

```
shashank@shank:~/Documents/Assignment 3/Test 1$ tftp
tftp> connect 127.0.0.1 3500
tftp> trace
Packet tracing on.
tftp> verbose
Verbose mode on.
tftp> mode octet
mode set to octet
tftp> status
Connected to 127.0.0.1.
Mode: octet Verbose: on Tracing: on
Rexmt-interval: 5 seconds, Max-timeout: 25 seconds
tftp> get 2047bin.txt
getting from 127.0.0.1:2047bin.txt to 2047bin.txt [octet]
sent RRQ <file=2047bin.txt, mode=octet>
received DATA <block=1, 512 bytes>
sent ACK <block=1>
received DATA <block=2, 512 bytes>
sent ACK <block=2>
received DATA <block=3, 512 bytes>
sent ACK <block=3>
received DATA <block=4, 511 bytes>
Received 2047 bytes in 0.0 seconds [inf bits/sec]
tftp> 
```


shashank@shank: ~/Documents/Assignment 3/Final
File Edit View Search Terminal Help

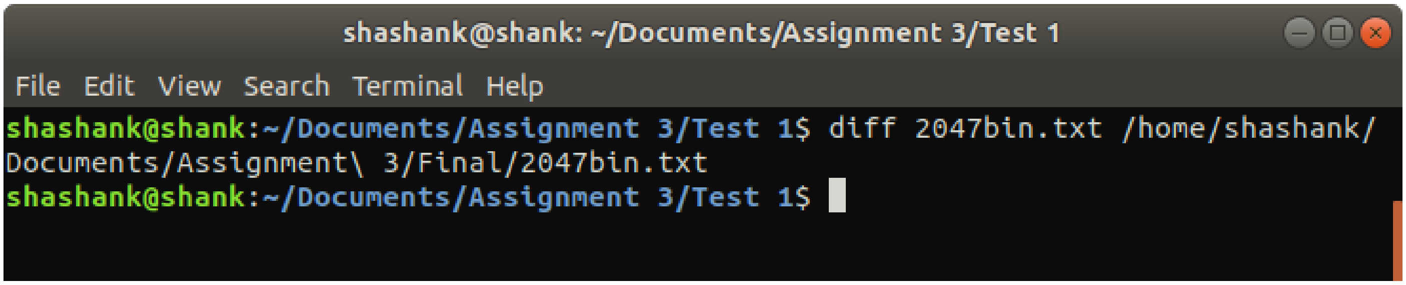
```
shashank@shank:~/Documents/Assignment 3/Final$ ./a.out 127.0.0.1 3500
Server: Waiting for connections:
Server: got packet from 127.0.0.1 and port : 17018
Packet 1 acknowledged
Packet 2 acknowledged
Packet 3 acknowledged
Final ACK received
File transfer done
Packet 4 acknowledged
```



Activities
Terminal
Sat 4:24 PM
shashank@shank: ~/Documents/Assignment 3/Final
File Edit View Search Terminal Help
shashank@shank: ~/Documents/Assignment 3/Test 1
File Edit View Search Terminal Help

```
shashank@shank:~/Documents/Assignment 3/Final$ ./a.out 127.0.0.1 3500
Server: Waiting for connections:
Server: got packet from 127.0.0.1 and port : 17018
Packet 1 acknowledged
Packet 2 acknowledged
Packet 3 acknowledged
Final ACK received
File transfer done
Packet 4 acknowledged
```

```
shashank@shank:~/Documents/Assignment 3/Test 1$ tftp
tftp> connect 127.0.0.1 3500
tftp> trace
Packet tracing on.
tftp> verbose
Verbose mode on.
tftp> mode octet
mode set to octet
tftp> status
Connected to 127.0.0.1.
Mode: octet Verbose: on Tracing: on
Rexmt-interval: 5 seconds, Max-timeout: 25 seconds
tftp> get 2047bin.txt
getting from 127.0.0.1:2047bin.txt to 2047bin.txt [octet]
sent RRQ <file=2047bin.txt, mode=octet>
received DATA <block=1, 512 bytes>
sent ACK <block=1>
received DATA <block=2, 512 bytes>
sent ACK <block=2>
received DATA <block=3, 512 bytes>
sent ACK <block=3>
received DATA <block=4, 511 bytes>
Received 2047 bytes in 0.0 seconds [inf bits/sec]
tftp> 
```



shashank@shank: ~/Documents/Assignment 3/Test 1
File Edit View Search Terminal Help

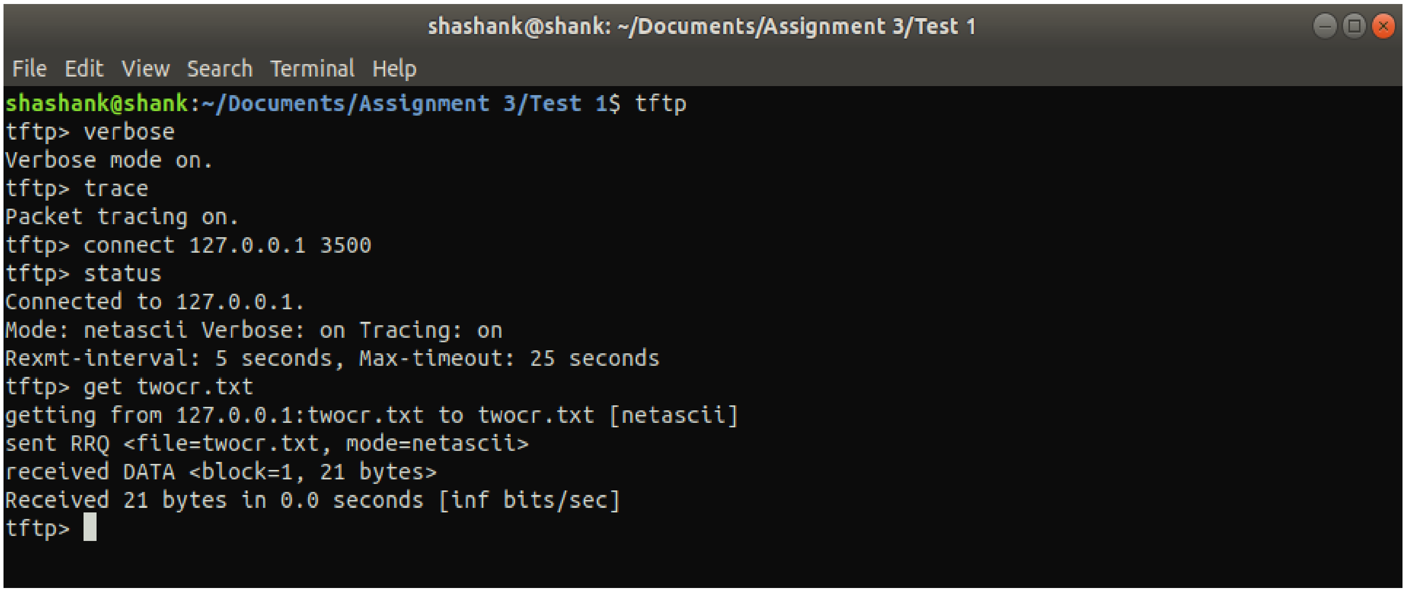
```
shashank@shank:~/Documents/Assignment 3/Test 1$ diff 2047bin.txt /home/shashank/
Documents/Assignment\ 3/Final/2047bin.txt
shashank@shank:~/Documents/Assignment 3/Test 1$ 
```

This test case showcases the ability of the server to handle a download request of binary file (size 2047 bytes) by the client. Initially the server has the binary file in this case it is “2047bin.txt”. The client connects to the server using the ip and the port number. To enable the octet mode on the client side the command “mode octet” is used. Later the file is downloaded using the command “get 2047bin.txt”, the server gets the request from the client and services it and every time the server gets an acknowledgment it prints the ack and at end of the transfer the server also prints the status to indicate the completion of the file transfer. Note: The client used in this test case is a standard ubuntu tftp client. For more clarity verbose and trace are enabled in the above test case using the command “verbose” and “trace”.

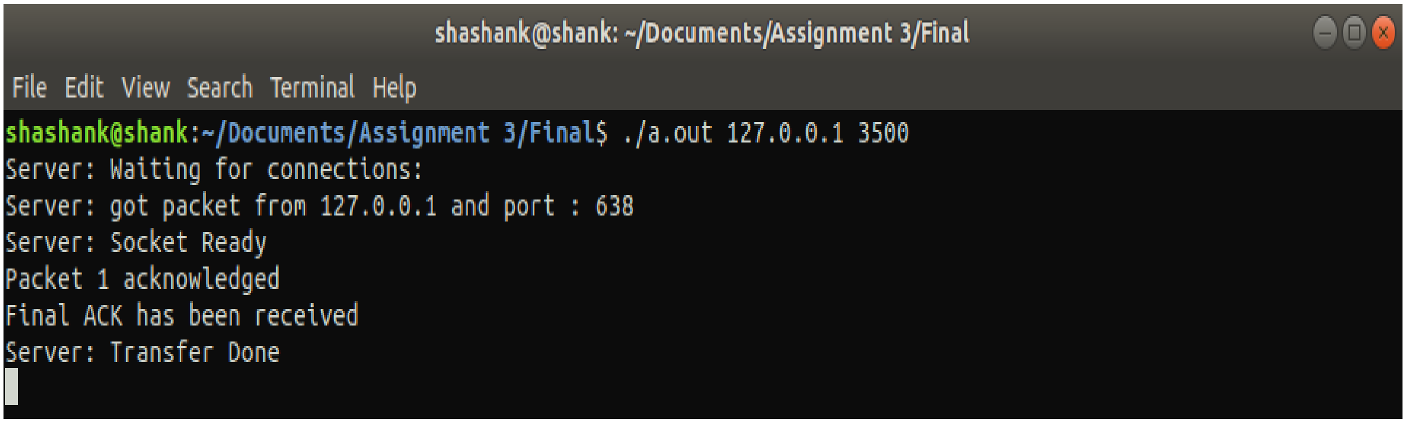
The first screenshot represents the client side of the interaction and the second screenshot represents the server side of the tftp interaction. The ip address of the server is 127.0.0.1 and the port number is 3500, the client connects to the server using the ip 127.0.0.1 and port number 3500 using the command “connect 127.0.0.1 3500”.

The third screenshot is a combined screenshot and the fourth screenshot shows the result of the tftp. The bin file which was initially at the server is transferred to the client using tftp protocol.

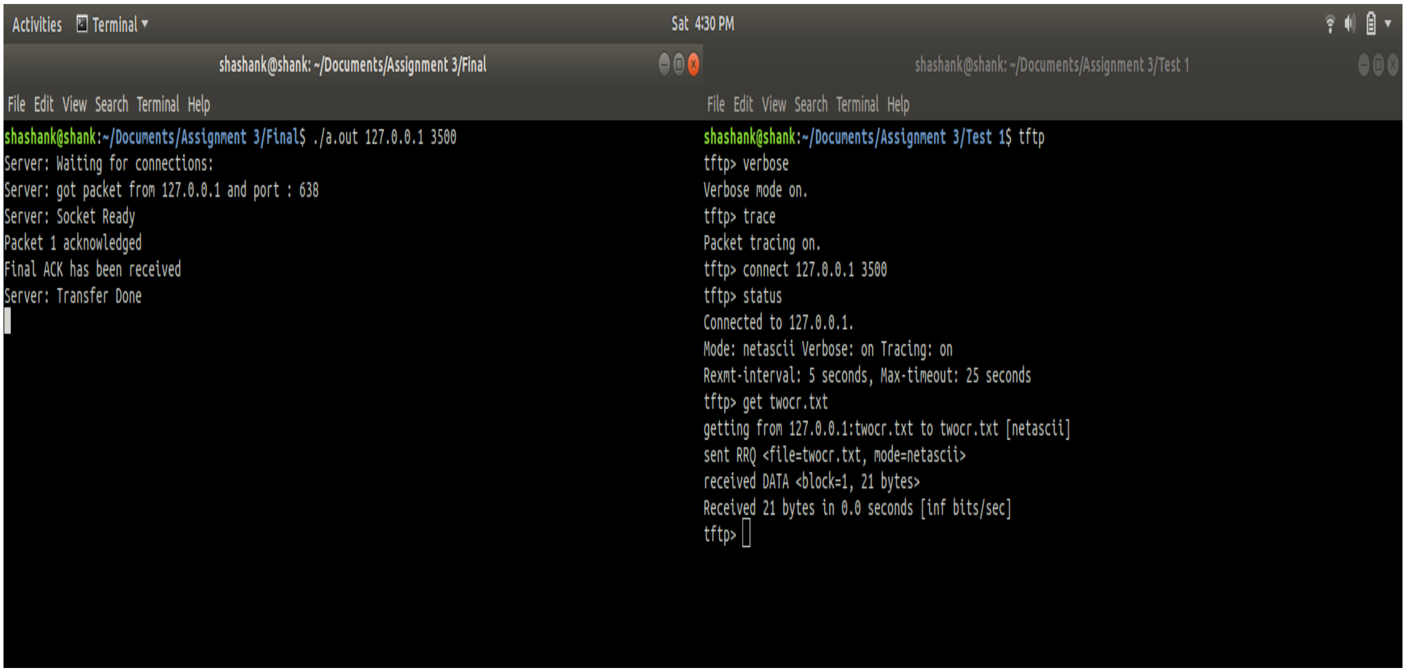
3. Transferring a netascii file that includes two CR’s and checking that it matches the source file


shashank@shank: ~/Documents/Assignment 3/Test 1
File Edit View Search Terminal Help

```
shashank@shank:~/Documents/Assignment 3/Test 1$ tftp
tftp> verbose
Verbose mode on.
tftp> trace
Packet tracing on.
tftp> connect 127.0.0.1 3500
tftp> status
Connected to 127.0.0.1.
Mode: netascii Verbose: on Tracing: on
Rexmt-interval: 5 seconds, Max-timeout: 25 seconds
tftp> get twocr.txt
getting from 127.0.0.1:twocr.txt to twocr.txt [netascii]
sent RRQ <file=twocr.txt, mode=netascii>
received DATA <block=1, 21 bytes>
Received 21 bytes in 0.0 seconds [inf bits/sec]
tftp> 
```

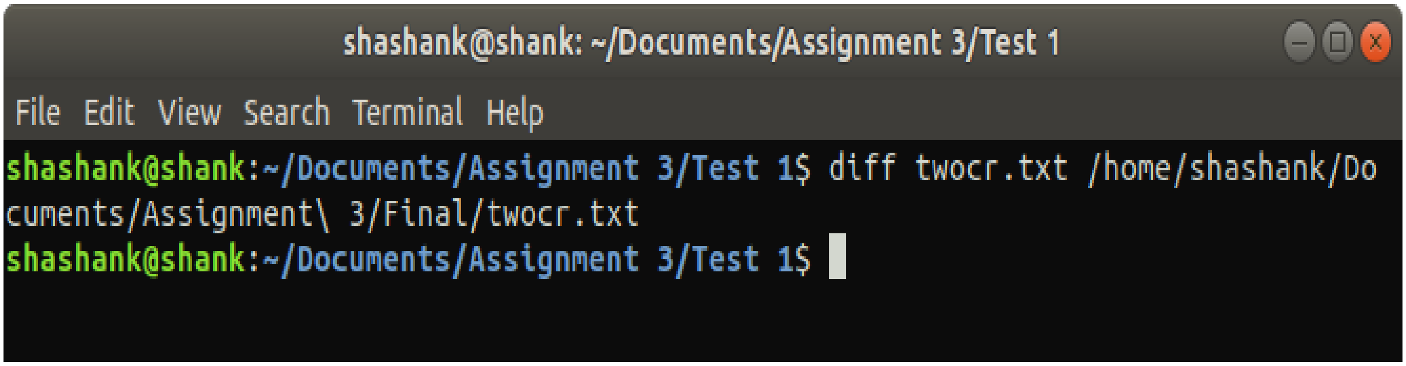

shashank@shank: ~/Documents/Assignment 3/Final
File Edit View Search Terminal Help

```
shashank@shank:~/Documents/Assignment 3/Final$ ./a.out 127.0.0.1 3500
Server: Waiting for connections:
Server: got packet from 127.0.0.1 and port : 638
Server: Socket Ready
Packet 1 acknowledged
Final ACK has been received
Server: Transfer Done
```


Activities
Terminal ▾
Sat 4:30 PM
shashank@shank: ~/Documents/Assignment 3/Final
File Edit View Search Terminal Help
shashank@shank: ~/Documents/Assignment 3/Test 1
File Edit View Search Terminal Help

```
shashank@shank:~/Documents/Assignment 3/Final$ ./a.out 127.0.0.1 3500
Server: Waiting for connections:
Server: got packet from 127.0.0.1 and port : 638
Server: Socket Ready
Packet 1 acknowledged
Final ACK has been received
Server: Transfer Done
```

```
shashank@shank:~/Documents/Assignment 3/Test 1$ tftp
tftp> verbose
Verbose mode on.
tftp> trace
Packet tracing on.
tftp> connect 127.0.0.1 3500
tftp> status
Connected to 127.0.0.1.
Mode: netascii Verbose: on Tracing: on
Rexmt-interval: 5 seconds, Max-timeout: 25 seconds
tftp> get twocr.txt
getting from 127.0.0.1:twocr.txt to twocr.txt [netascii]
sent RRQ <file=twocr.txt, mode=netascii>
received DATA <block=1, 21 bytes>
Received 21 bytes in 0.0 seconds [inf bits/sec]
tftp> 
```



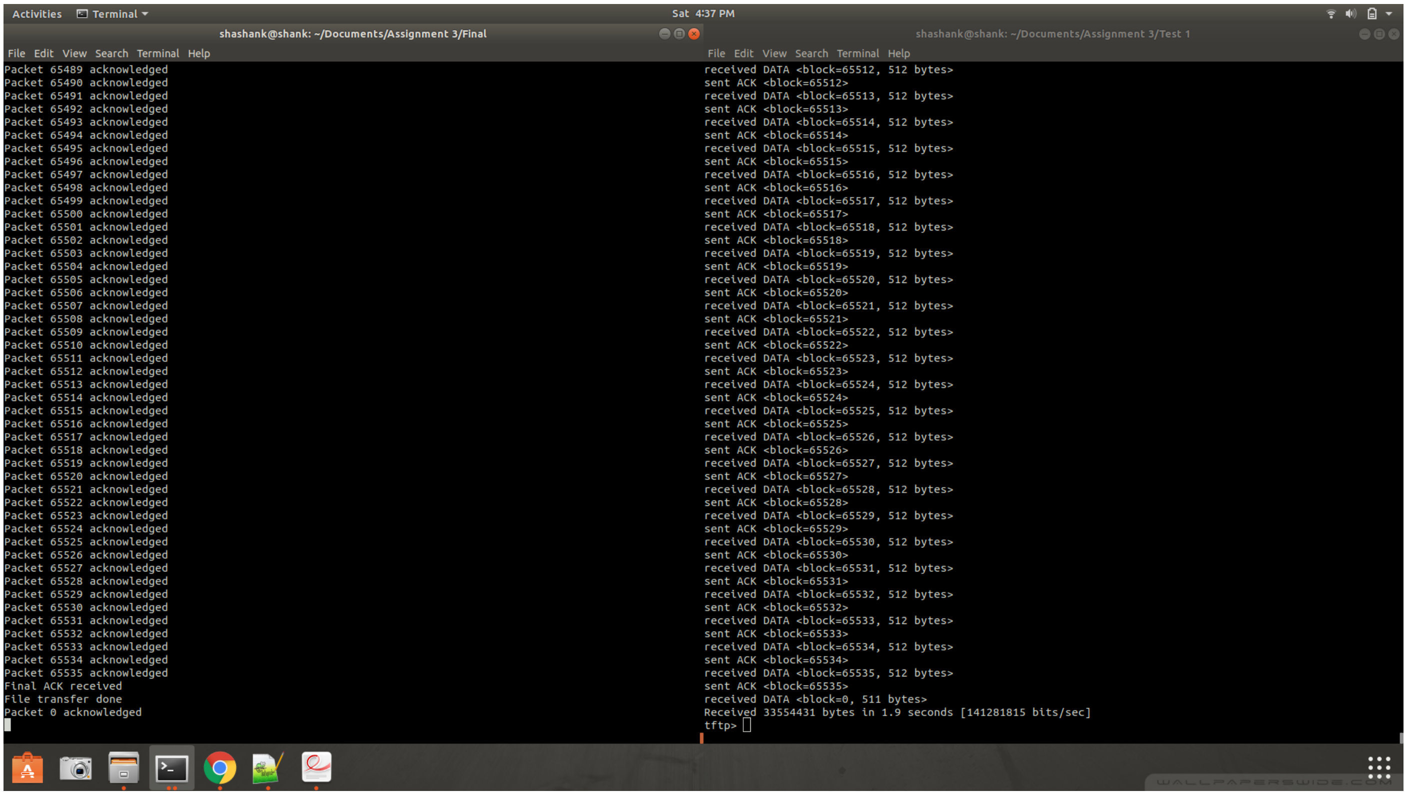
shashank@shank: ~/Documents/Assignment 3/Test 1
File Edit View Search Terminal Help

```
shashank@shank:~/Documents/Assignment 3/Test 1$ diff twocr.txt /home/shashank/Documents/Assignment\ 3/Final/twocr.txt
shashank@shank:~/Documents/Assignment 3/Test 1$ 
```

This test case demonstrates the ability of the server to handle a client request to send netascii file which has 2 CR’s (server to client transfer). The first screenshot shows the interaction from the client side, prior to this step the server is launched using the steps mentioned in the first section. The server is running on ip address 127.0.0.1 and port number 3500. The client connects to the server and for more clarity the verbose and trace modes are enabled. The client then requests the netascii file from the server in this case it “twocr.txt”. The server services this request and starts ending the file, the third screenshot represents this situation, for every packet the server sends it needs a ack before sending the next packet and after receiving every ack the server then prints the acknowledgement for more clarity.

Later the file is then checked for difference to verify the correctness of the tftp transfer, which is shown in the last screenshot.

4. Transfer a binary file of 34 MB and checking for block number wrap-around



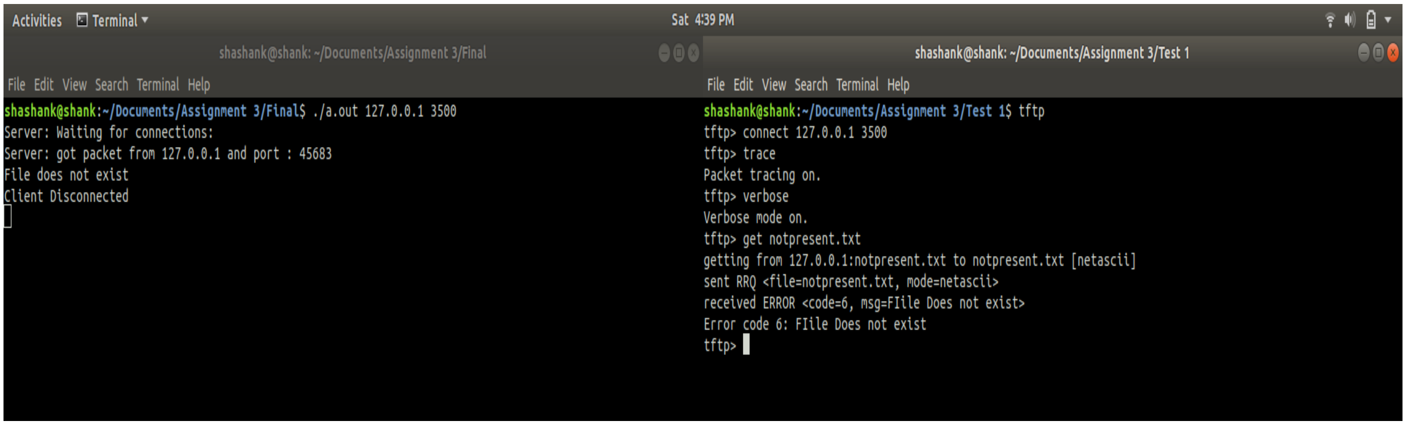
Activities
Terminal
Sat 4:37 PM
shashank@shank: ~/Documents/Assignment 3/Final
File Edit View Search Terminal Help
shashank@shank: ~/Documents/Assignment 3/Test 1
File Edit View Search Terminal Help

```
Packet 65489 acknowledged
Packet 65490 acknowledged
Packet 65491 acknowledged
Packet 65492 acknowledged
Packet 65493 acknowledged
Packet 65494 acknowledged
Packet 65495 acknowledged
Packet 65496 acknowledged
Packet 65497 acknowledged
Packet 65498 acknowledged
Packet 65499 acknowledged
Packet 65500 acknowledged
Packet 65501 acknowledged
Packet 65502 acknowledged
Packet 65503 acknowledged
Packet 65504 acknowledged
Packet 65505 acknowledged
Packet 65506 acknowledged
Packet 65507 acknowledged
Packet 65508 acknowledged
Packet 65509 acknowledged
Packet 65510 acknowledged
Packet 65511 acknowledged
Packet 65512 acknowledged
Packet 65513 acknowledged
Packet 65514 acknowledged
Packet 65515 acknowledged
Packet 65516 acknowledged
Packet 65517 acknowledged
Packet 65518 acknowledged
Packet 65519 acknowledged
Packet 65520 acknowledged
Packet 65521 acknowledged
Packet 65522 acknowledged
Packet 65523 acknowledged
Packet 65524 acknowledged
Packet 65525 acknowledged
Packet 65526 acknowledged
Packet 65527 acknowledged
Packet 65528 acknowledged
Packet 65529 acknowledged
Packet 65530 acknowledged
Packet 65531 acknowledged
Packet 65532 acknowledged
Packet 65533 acknowledged
Packet 65534 acknowledged
Packet 65535 acknowledged
Final ACK received
File transfer done
Packet 0 acknowledged
```

```
received DATA <block=65512, 512 bytes>
sent ACK <block=65512>
received DATA <block=65513, 512 bytes>
sent ACK <block=65513>
received DATA <block=65514, 512 bytes>
sent ACK <block=65514>
received DATA <block=65515, 512 bytes>
sent ACK <block=65515>
received DATA <block=65516, 512 bytes>
sent ACK <block=65516>
received DATA <block=65517, 512 bytes>
sent ACK <block=65517>
received DATA <block=65518, 512 bytes>
sent ACK <block=65518>
received DATA <block=65519, 512 bytes>
sent ACK <block=65519>
received DATA <block=65520, 512 bytes>
sent ACK <block=65520>
received DATA <block=65521, 512 bytes>
sent ACK <block=65521>
received DATA <block=65522, 512 bytes>
sent ACK <block=65522>
received DATA <block=65523, 512 bytes>
sent ACK <block=65523>
received DATA <block=65524, 512 bytes>
sent ACK <block=65524>
received DATA <block=65525, 512 bytes>
sent ACK <block=65525>
received DATA <block=65526, 512 bytes>
sent ACK <block=65526>
received DATA <block=65527, 512 bytes>
sent ACK <block=65527>
received DATA <block=65528, 512 bytes>
sent ACK <block=65528>
received DATA <block=65529, 512 bytes>
sent ACK <block=65529>
received DATA <block=65530, 512 bytes>
sent ACK <block=65530>
received DATA <block=65531, 512 bytes>
sent ACK <block=65531>
received DATA <block=65532, 512 bytes>
sent ACK <block=65532>
received DATA <block=65533, 512 bytes>
sent ACK <block=65533>
received DATA <block=65534, 512 bytes>
sent ACK <block=65534>
received DATA <block=65535, 512 bytes>
sent ACK <block=65535>
received DATA <block=0, 511 bytes>
Received 33554431 bytes in 1.9 seconds [141281815 bits/sec]
tftp> 
```

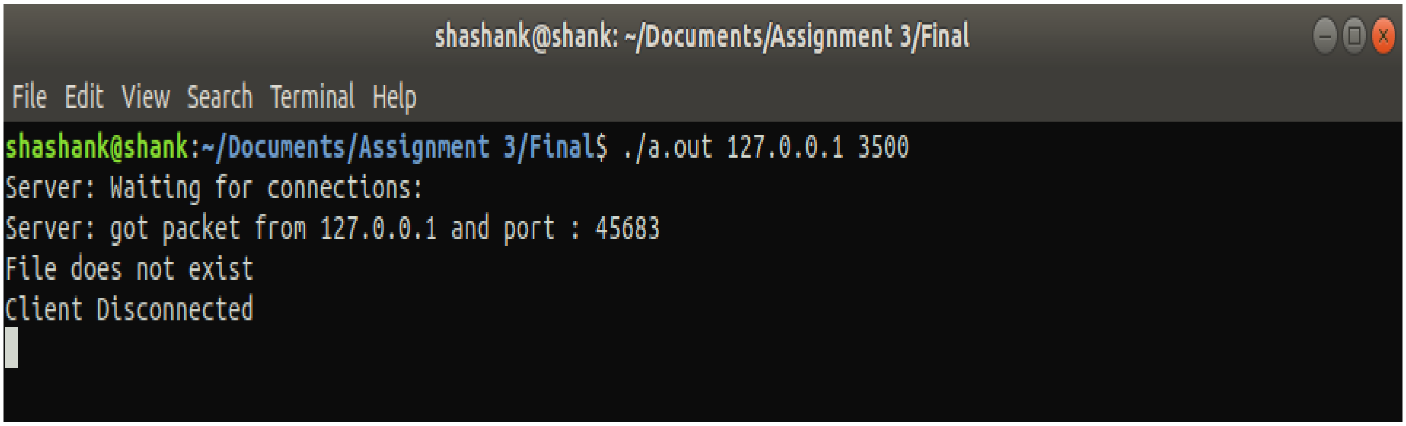
This test case demonstrates the ability of the tftp server to handle a request to transfer a 34MB binary file to the client on request. In this case the file named “34bin.txt” is transferred from the server to the client using tftp. The procedure for this test case is similar to the one mentioned in the previous cases. The above screenshot represents the logs of the 34MB file transfer from server to the client.

5. Checking for error message when transferring a file that does not exist.

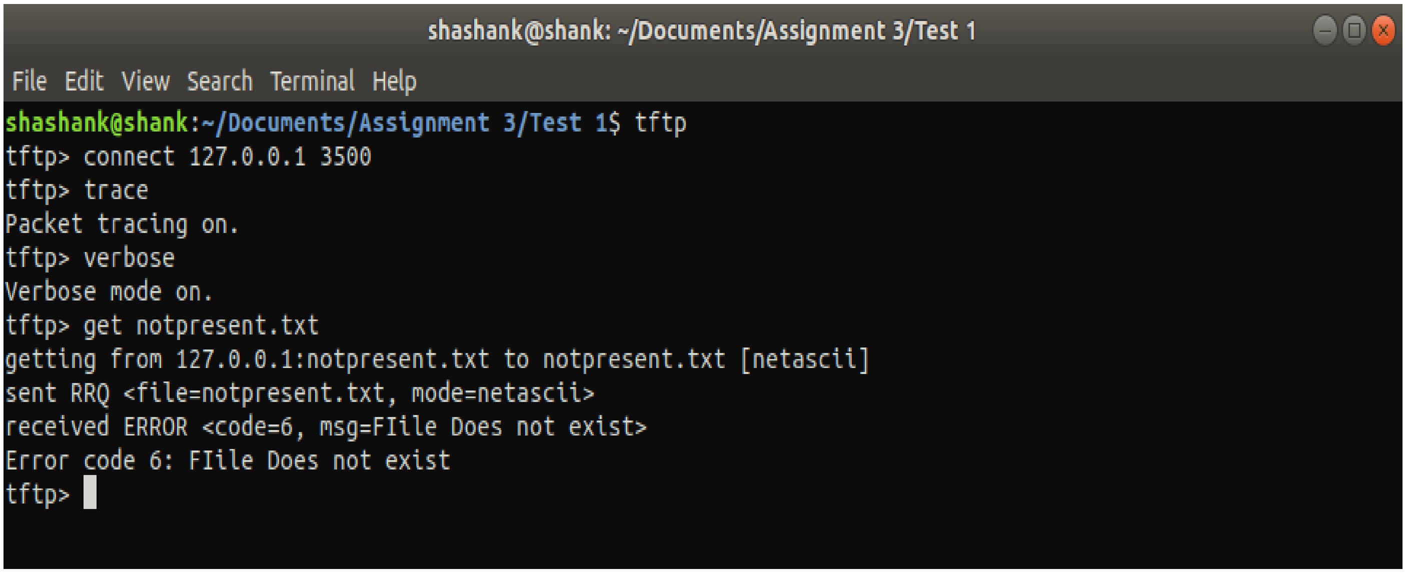

Activities
Terminal
Sat 4:39 PM
shashank@shank: ~/Documents/Assignment 3/Final
File Edit View Search Terminal Help
shashank@shank: ~/Documents/Assignment 3/Test 1
File Edit View Search Terminal Help

```
shashank@shank:~/Documents/Assignment 3/Final$ ./a.out 127.0.0.1 3500
Server: Waiting for connections:
Server: got packet from 127.0.0.1 and port : 45683
File does not exist
Client Disconnected
```

```
shashank@shank:~/Documents/Assignment 3/Test 1$ tftp
tftp> connect 127.0.0.1 3500
tftp> trace
Packet tracing on.
tftp> verbose
Verbose mode on.
tftp> get notpresent.txt
getting from 127.0.0.1:notpresent.txt to notpresent.txt [netascii]
sent RRQ <file=notpresent.txt, mode=netascii>
received ERROR <code=6, msg=FIile Does not exist>
Error code 6: FIile Does not exist
tftp>
```


shashank@shank: ~/Documents/Assignment 3/Final
File Edit View Search Terminal Help

```
shashank@shank:~/Documents/Assignment 3/Final$ ./a.out 127.0.0.1 3500
Server: Waiting for connections:
Server: got packet from 127.0.0.1 and port : 45683
File does not exist
Client Disconnected
```

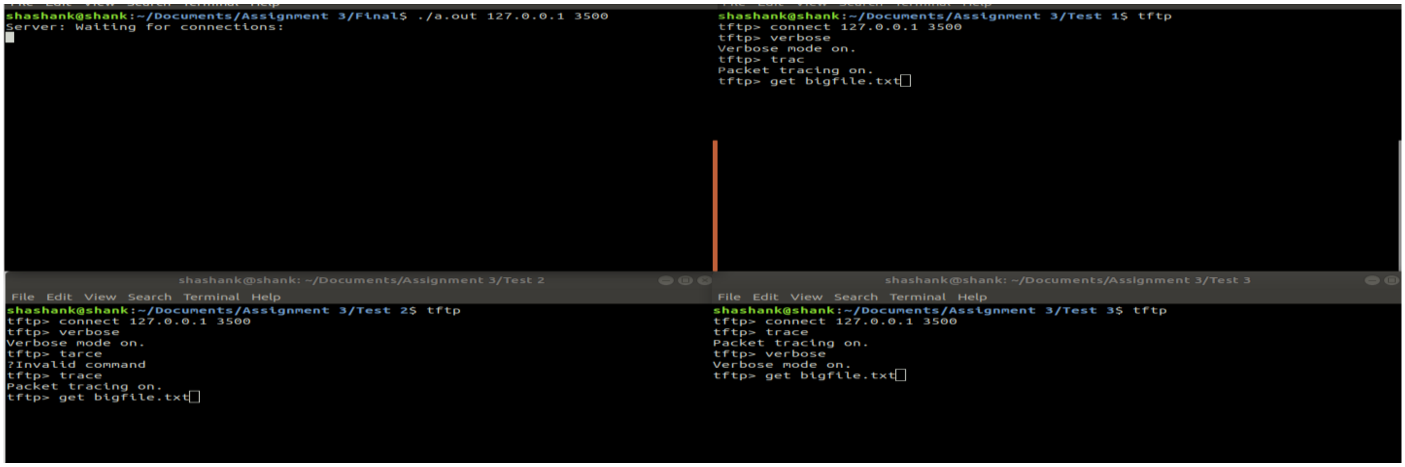

shashank@shank: ~/Documents/Assignment 3/Test 1
File Edit View Search Terminal Help

```
shashank@shank:~/Documents/Assignment 3/Test 1$ tftp
tftp> connect 127.0.0.1 3500
tftp> trace
Packet tracing on.
tftp> verbose
Verbose mode on.
tftp> get notpresent.txt
getting from 127.0.0.1:notpresent.txt to notpresent.txt [netascii]
sent RRQ <file=notpresent.txt, mode=netascii>
received ERROR <code=6, msg=FIile Does not exist>
Error code 6: FIile Does not exist
tftp>
```

In this test case the client request for a file which does not exist, in other words the client requests for a invalid file. The test cases demonstrate the server response to such a case. After the initial steps the client requests for a invalid file, in this case “notpresent.txt”. The server responds back with a error message stating that the file doesn’t exist and the error message is printed as shown in the second screenshot. Furthermore, after receiving the error message the client prints the error code 6 as shown in the third screenshot.

Apart from printing the error message the server and client also get disconnected as shown in screenshot 2. If the client wants to send or receive a packet it has to establish a new connection with the server by using the “connect” command and the process will start from the beginning.

6. Connecting 3 clients simultaneously and transferring the files.



```
shashank@shank:~/Documents/Assignment 3/Final$ ./a.out 127.0.0.1 3500
Server: Waiting for connections:
```

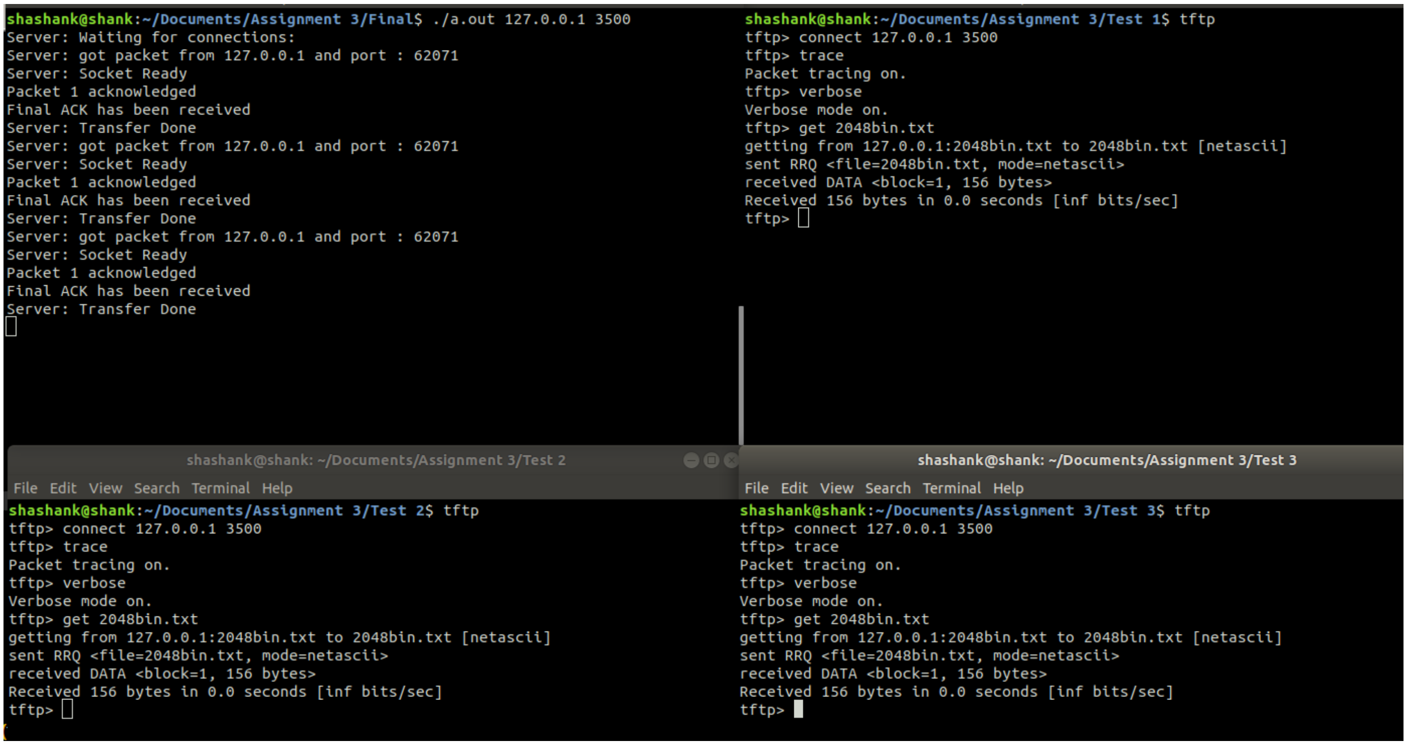
```
shashank@shank:~/Documents/Assignment 3/Test 1$ tftp
tftp> connect 127.0.0.1 3500
tftp> verbose
Verbose mode on.
tftp> trac
Packet tracing on.
tftp> get bigfile.txt
```

shashank@shank: ~/Documents/Assignment 3/Test 2
File Edit View Search Terminal Help

```
shashank@shank:~/Documents/Assignment 3/Test 2$ tftp
tftp> connect 127.0.0.1 3500
tftp> verbose
Verbose mode on.
tftp> tarce
?Invalid command
tftp> trace
Packet tracing on.
tftp> get bigfile.txt
```

shashank@shank: ~/Documents/Assignment 3/Test 3
File Edit View Search Terminal Help

```
shashank@shank:~/Documents/Assignment 3/Test 3$ tftp
tftp> connect 127.0.0.1 3500
tftp> trace
Packet tracing on.
tftp> verbose
Verbose mode on.
tftp> get bigfile.txt
```



```
shashank@shank:~/Documents/Assignment 3/Final$ ./a.out 127.0.0.1 3500
Server: Waiting for connections:
Server: got packet from 127.0.0.1 and port : 62071
Server: Socket Ready
Packet 1 acknowledged
Final ACK has been received
Server: Transfer Done
Server: got packet from 127.0.0.1 and port : 62071
Server: Socket Ready
Packet 1 acknowledged
Final ACK has been received
Server: Transfer Done
Server: got packet from 127.0.0.1 and port : 62071
Server: Socket Ready
Packet 1 acknowledged
Final ACK has been received
Server: Transfer Done
```

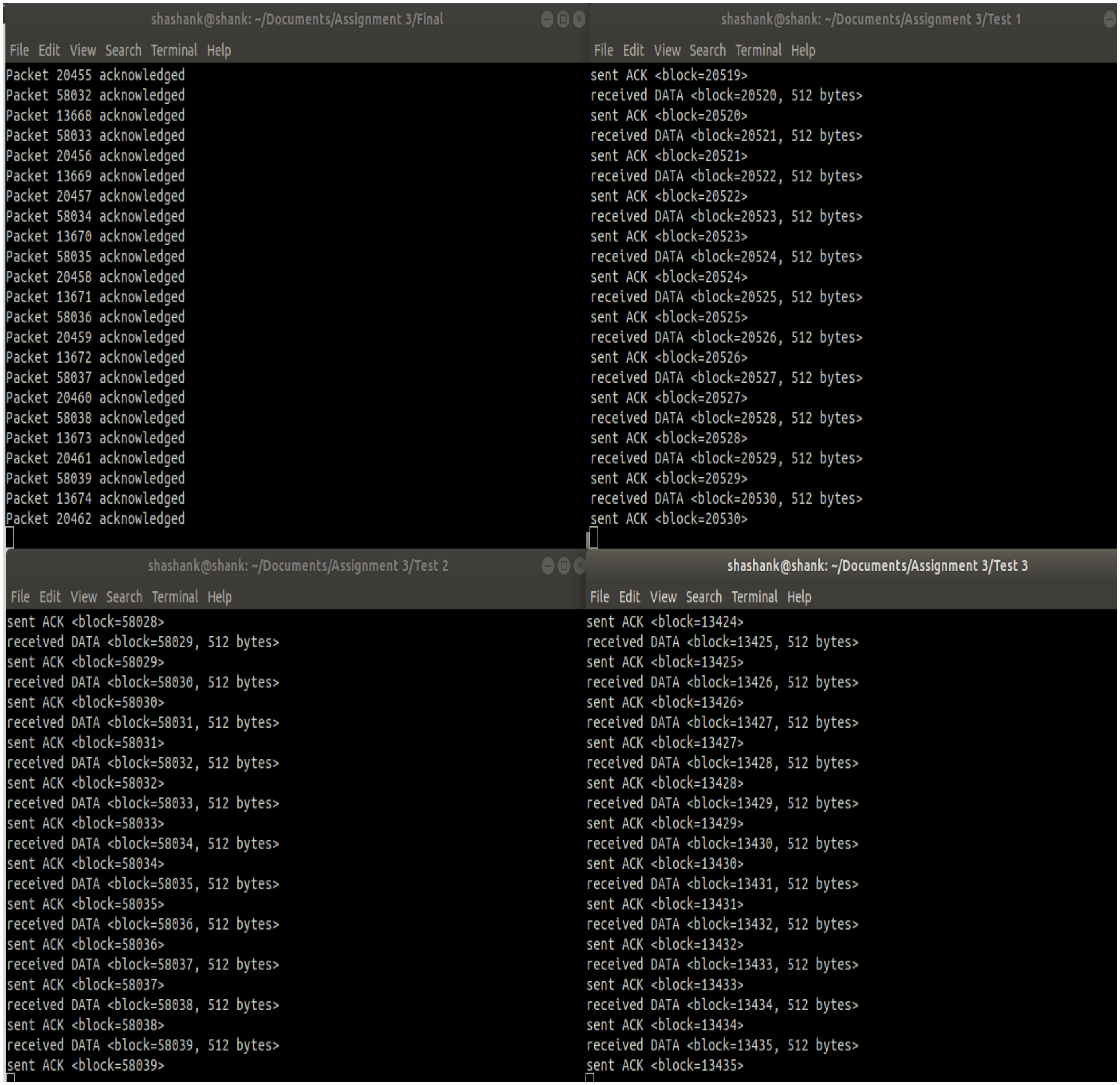
```
shashank@shank:~/Documents/Assignment 3/Test 1$ tftp
tftp> connect 127.0.0.1 3500
tftp> trace
Packet tracing on.
tftp> verbose
Verbose mode on.
tftp> get 2048bin.txt
getting from 127.0.0.1:2048bin.txt to 2048bin.txt [netascii]
sent RRQ <file=2048bin.txt, mode=netascii>
received DATA <block=1, 156 bytes>
Received 156 bytes in 0.0 seconds [inf bits/sec]
tftp>
```

shashank@shank: ~/Documents/Assignment 3/Test 2
File Edit View Search Terminal Help

```
shashank@shank:~/Documents/Assignment 3/Test 2$ tftp
tftp> connect 127.0.0.1 3500
tftp> trace
Packet tracing on.
tftp> verbose
Verbose mode on.
tftp> get 2048bin.txt
getting from 127.0.0.1:2048bin.txt to 2048bin.txt [netascii]
sent RRQ <file=2048bin.txt, mode=netascii>
received DATA <block=1, 156 bytes>
Received 156 bytes in 0.0 seconds [inf bits/sec]
tftp>
```

shashank@shank: ~/Documents/Assignment 3/Test 3
File Edit View Search Terminal Help

```
shashank@shank:~/Documents/Assignment 3/Test 3$ tftp
tftp> connect 127.0.0.1 3500
tftp> trace
Packet tracing on.
tftp> verbose
Verbose mode on.
tftp> get 2048bin.txt
getting from 127.0.0.1:2048bin.txt to 2048bin.txt [netascii]
sent RRQ <file=2048bin.txt, mode=netascii>
received DATA <block=1, 156 bytes>
Received 156 bytes in 0.0 seconds [inf bits/sec]
tftp>
```


shashank@shank: ~/Documents/Assignment 3/Final
File Edit View Search Terminal Help
shashank@shank: ~/Documents/Assignment 3/Test 1
File Edit View Search Terminal Help
shashank@shank: ~/Documents/Assignment 3/Test 2
File Edit View Search Terminal Help
shashank@shank: ~/Documents/Assignment 3/Test 3
File Edit View Search Terminal Help

```
Packet 20455 acknowledged
Packet 58032 acknowledged
Packet 13668 acknowledged
Packet 58033 acknowledged
Packet 20456 acknowledged
Packet 13669 acknowledged
Packet 20457 acknowledged
Packet 58034 acknowledged
Packet 13670 acknowledged
Packet 58035 acknowledged
Packet 20458 acknowledged
Packet 13671 acknowledged
Packet 58036 acknowledged
Packet 20459 acknowledged
Packet 13672 acknowledged
Packet 58037 acknowledged
Packet 20460 acknowledged
Packet 58038 acknowledged
Packet 13673 acknowledged
Packet 20461 acknowledged
Packet 58039 acknowledged
Packet 13674 acknowledged
Packet 20462 acknowledged
```

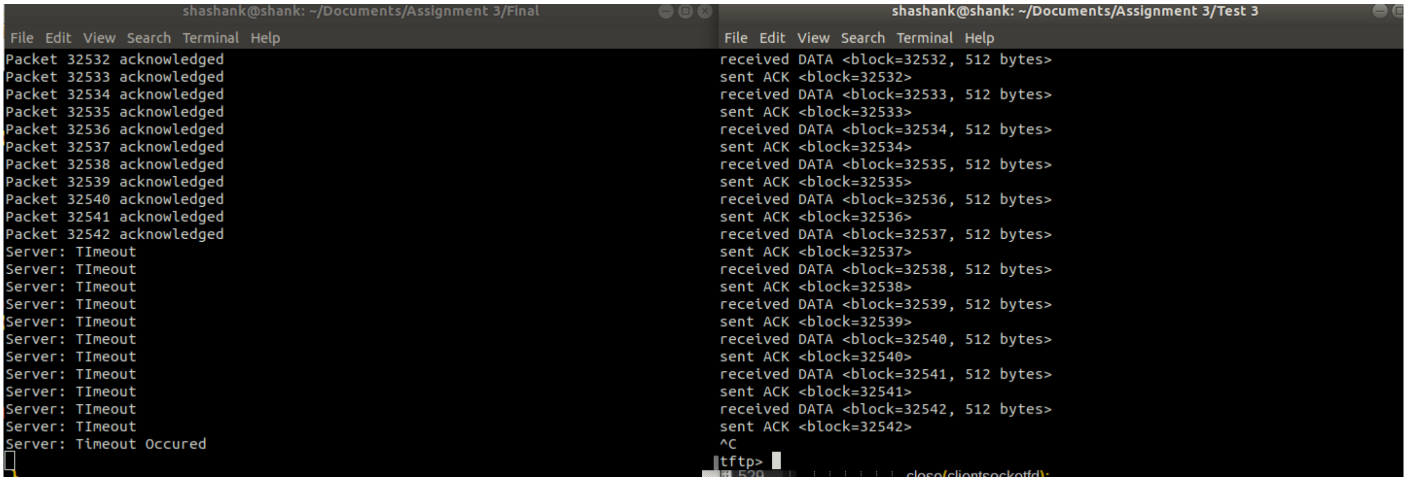
```
sent ACK <block=20519>
received DATA <block=20520, 512 bytes>
sent ACK <block=20520>
received DATA <block=20521, 512 bytes>
sent ACK <block=20521>
received DATA <block=20522, 512 bytes>
sent ACK <block=20522>
received DATA <block=20523, 512 bytes>
sent ACK <block=20523>
received DATA <block=20524, 512 bytes>
sent ACK <block=20524>
received DATA <block=20525, 512 bytes>
sent ACK <block=20525>
received DATA <block=20526, 512 bytes>
sent ACK <block=20526>
received DATA <block=20527, 512 bytes>
sent ACK <block=20527>
received DATA <block=20528, 512 bytes>
sent ACK <block=20528>
received DATA <block=20529, 512 bytes>
sent ACK <block=20529>
received DATA <block=20530, 512 bytes>
sent ACK <block=20530>
```

```
sent ACK <block=58028>
received DATA <block=58029, 512 bytes>
sent ACK <block=58029>
received DATA <block=58030, 512 bytes>
sent ACK <block=58030>
received DATA <block=58031, 512 bytes>
sent ACK <block=58031>
received DATA <block=58032, 512 bytes>
sent ACK <block=58032>
received DATA <block=58033, 512 bytes>
sent ACK <block=58033>
received DATA <block=58034, 512 bytes>
sent ACK <block=58034>
received DATA <block=58035, 512 bytes>
sent ACK <block=58035>
received DATA <block=58036, 512 bytes>
sent ACK <block=58036>
received DATA <block=58037, 512 bytes>
sent ACK <block=58037>
received DATA <block=58038, 512 bytes>
sent ACK <block=58038>
received DATA <block=58039, 512 bytes>
sent ACK <block=58039>
```

```
sent ACK <block=13424>
received DATA <block=13425, 512 bytes>
sent ACK <block=13425>
received DATA <block=13426, 512 bytes>
sent ACK <block=13426>
received DATA <block=13427, 512 bytes>
sent ACK <block=13427>
received DATA <block=13428, 512 bytes>
sent ACK <block=13428>
received DATA <block=13429, 512 bytes>
sent ACK <block=13429>
received DATA <block=13430, 512 bytes>
sent ACK <block=13430>
received DATA <block=13431, 512 bytes>
sent ACK <block=13431>
received DATA <block=13432, 512 bytes>
sent ACK <block=13432>
received DATA <block=13433, 512 bytes>
sent ACK <block=13433>
received DATA <block=13434, 512 bytes>
sent ACK <block=13434>
received DATA <block=13435, 512 bytes>
sent ACK <block=13435>
```

In this test there are multiple clients (3 clients) connected to the server and the server simultaneously handles the requests of all the clients. In the first screenshot represents the case where the server is running, and the clients are trying to connect to the server. The second screenshot shows that all the 3 clients could successfully connect to the server. Also, all the three client requests for file 2048bin.txt in this case. The server handles all the three requests and services all the three clients which is shown in the third screenshot. This test case demonstrates that the server is able to handle multiple client request simultaneously.

7. Terminating the client in the middle of transfer.

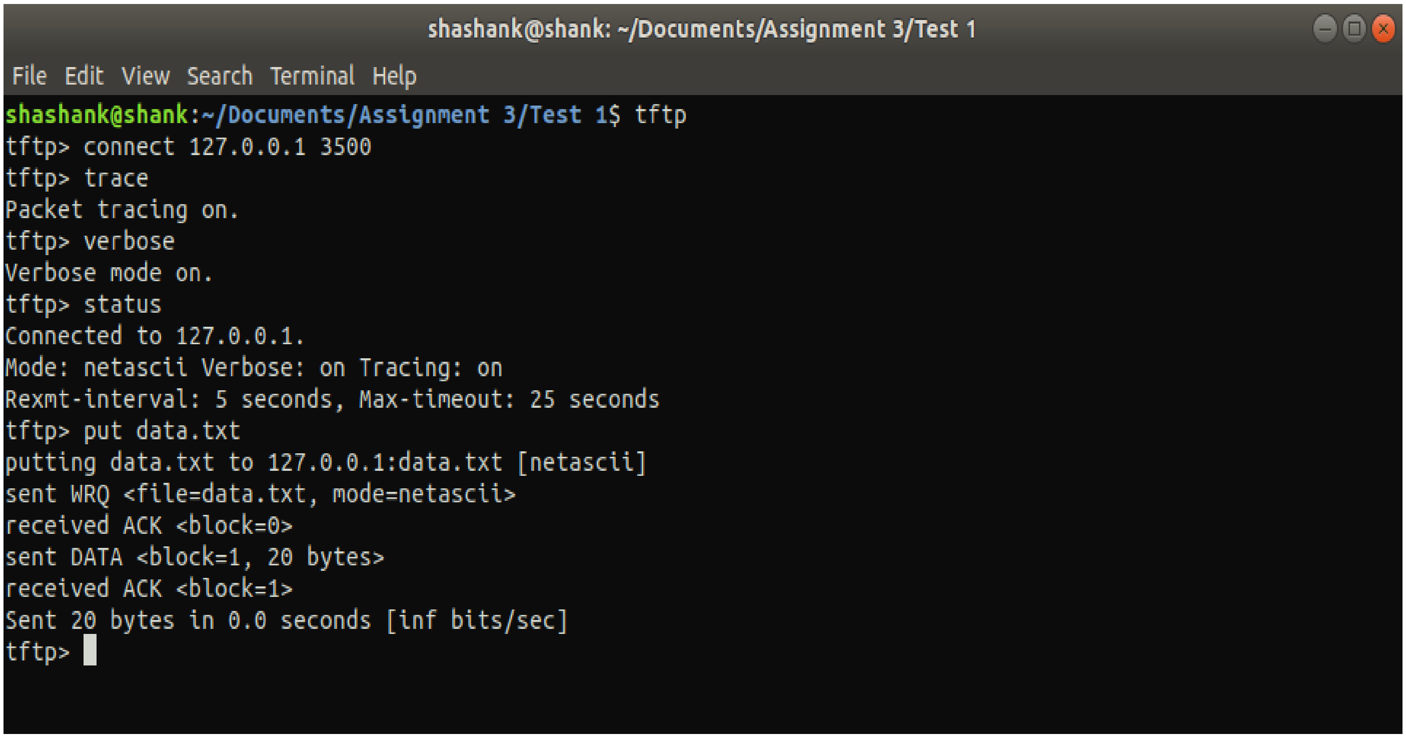

shashank@shank: ~/Documents/Assignment 3/Final
File Edit View Search Terminal Help
shashank@shank: ~/Documents/Assignment 3/Test 3
File Edit View Search Terminal Help

```
Packet 32532 acknowledged
Packet 32533 acknowledged
Packet 32534 acknowledged
Packet 32535 acknowledged
Packet 32536 acknowledged
Packet 32537 acknowledged
Packet 32538 acknowledged
Packet 32539 acknowledged
Packet 32540 acknowledged
Packet 32541 acknowledged
Packet 32542 acknowledged
Server: TImeout
Server: TImeout
Server: TImeout
Server: TImeout
Server: TImeout
Server: TImeout
Server: TImeout
Server: TImeout
Server: TImeout
Server: TImeout
Server: TImeout
Server: Timeout Occured
```

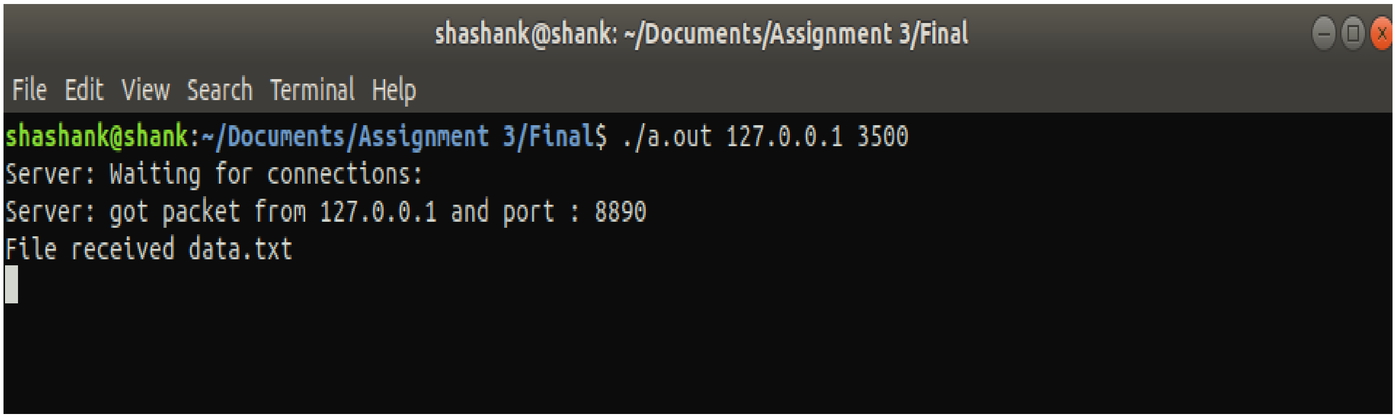
```
received DATA <block=32532, 512 bytes>
sent ACK <block=32532>
received DATA <block=32533, 512 bytes>
sent ACK <block=32533>
received DATA <block=32534, 512 bytes>
sent ACK <block=32534>
received DATA <block=32535, 512 bytes>
sent ACK <block=32535>
received DATA <block=32536, 512 bytes>
sent ACK <block=32536>
received DATA <block=32537, 512 bytes>
sent ACK <block=32537>
received DATA <block=32538, 512 bytes>
sent ACK <block=32538>
received DATA <block=32539, 512 bytes>
sent ACK <block=32539>
received DATA <block=32540, 512 bytes>
sent ACK <block=32540>
received DATA <block=32541, 512 bytes>
sent ACK <block=32541>
received DATA <block=32542, 512 bytes>
sent ACK <block=32542>
^C
tftp>
```

In this test case the client terminates during the transfer, the test case demonstrates the server behaviour in such a situation. As shown in the screenshot, the server is sending the requested file but during the transmission the client terminates. In this situation the server continues to send and terminates after 10 retries which is set for a timeout condition.

8. Writing a file in netascii mode

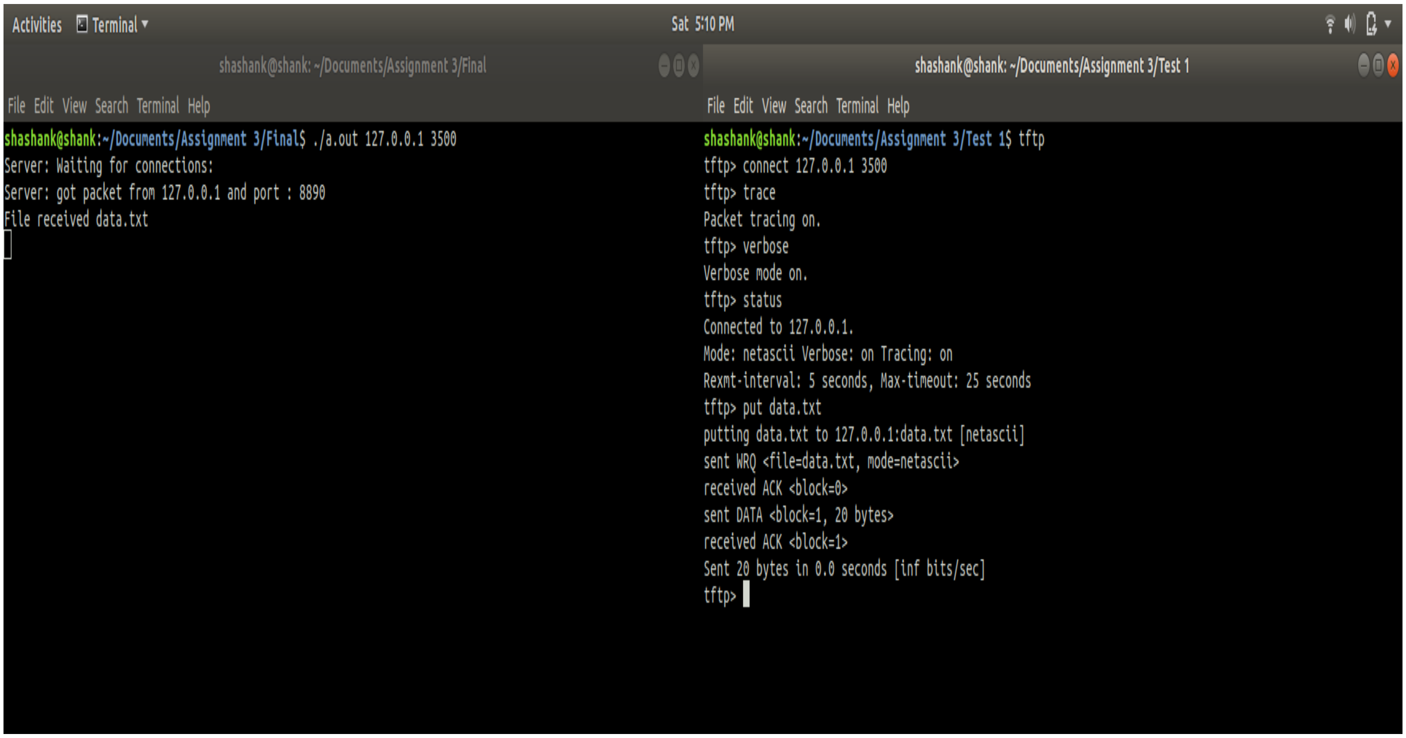

shashank@shank: ~/Documents/Assignment 3/Test 1
File Edit View Search Terminal Help

```
shashank@shank:~/Documents/Assignment 3/Test 1$ tftp
tftp> connect 127.0.0.1 3500
tftp> trace
Packet tracing on.
tftp> verbose
Verbose mode on.
tftp> status
Connected to 127.0.0.1.
Mode: netascii Verbose: on Tracing: on
Rexmt-interval: 5 seconds, Max-timeout: 25 seconds
tftp> put data.txt
putting data.txt to 127.0.0.1:data.txt [netascii]
sent WRQ <file=data.txt, mode=netascii>
received ACK <block=0>
sent DATA <block=1, 20 bytes>
received ACK <block=1>
Sent 20 bytes in 0.0 seconds [inf bits/sec]
tftp>
```



shashank@shank: ~/Documents/Assignment 3/Final
File Edit View Search Terminal Help

```
shashank@shank:~/Documents/Assignment 3/Final$ ./a.out 127.0.0.1 3500
Server: Waiting for connections:
Server: got packet from 127.0.0.1 and port : 8890
File received data.txt
```



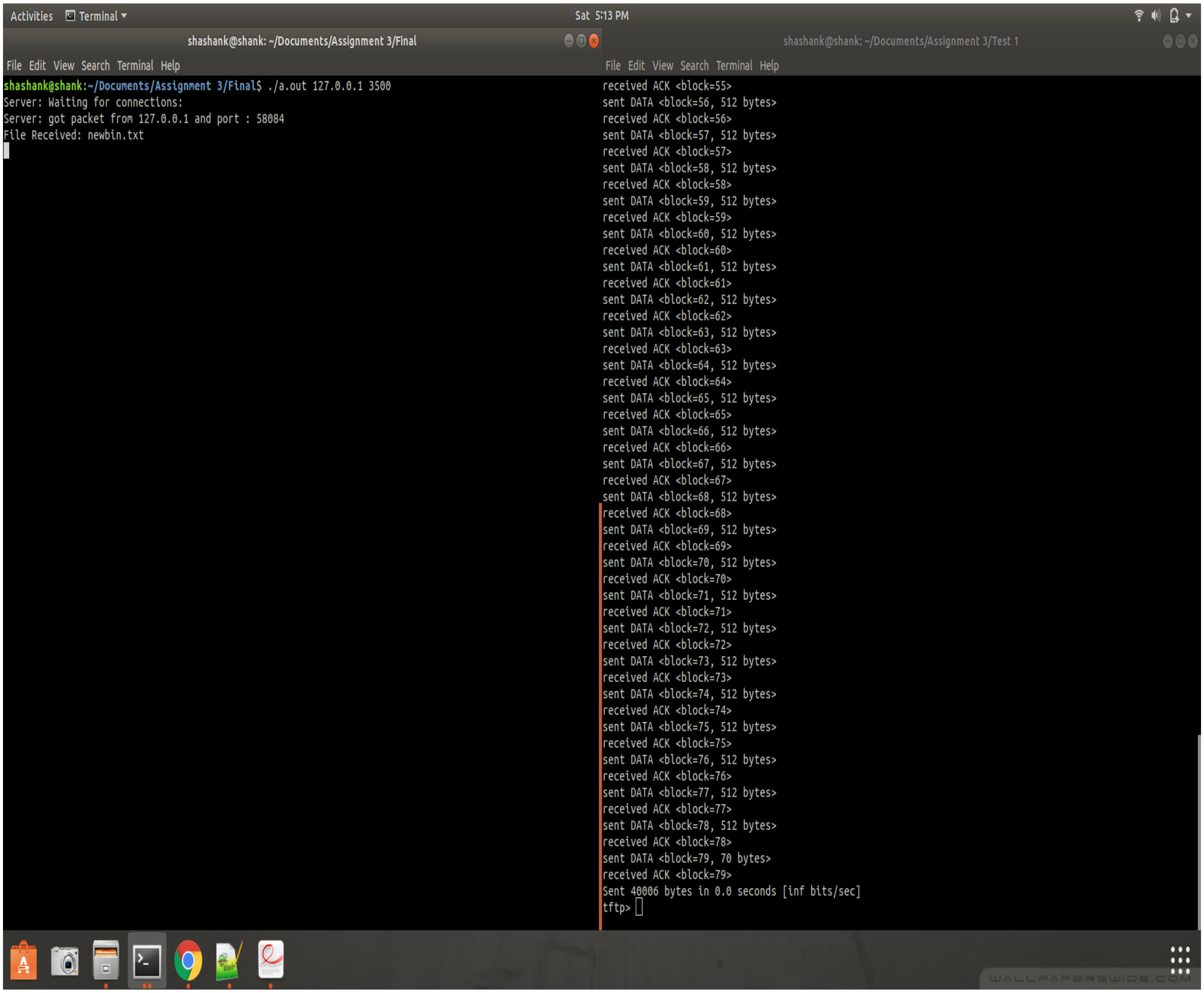
Activities
Terminal ▾
Sat 5:10 PM
shashank@shank: ~/Documents/Assignment 3/Final
File Edit View Search Terminal Help
shashank@shank: ~/Documents/Assignment 3/Test 1
File Edit View Search Terminal Help

```
shashank@shank:~/Documents/Assignment 3/Final$ ./a.out 127.0.0.1 3500
Server: Waiting for connections:
Server: got packet from 127.0.0.1 and port : 8890
File received data.txt
```

```
shashank@shank:~/Documents/Assignment 3/Test 1$ tftp
tftp> connect 127.0.0.1 3500
tftp> trace
Packet tracing on.
tftp> verbose
Verbose mode on.
tftp> status
Connected to 127.0.0.1.
Mode: netascii Verbose: on Tracing: on
Rexmt-interval: 5 seconds, Max-timeout: 25 seconds
tftp> put data.txt
putting data.txt to 127.0.0.1:data.txt [netascii]
sent WRQ <file=data.txt, mode=netascii>
received ACK <block=0>
sent DATA <block=1, 20 bytes>
received ACK <block=1>
Sent 20 bytes in 0.0 seconds [inf bits/sec]
tftp>
```

All the above test cases covered situation where client was downloading the file, in this case the client will upload the data.txt. All the initial conditions are similar to the one mentioned in section 1, the client sends the data in netascii mode and as shown in the first screenshot. The server which is running on the ip 127.0.0.1 will receive the file and print the “File received” as shown in the second screenshot. The third screenshot shows all the logs on both the server and client side by side.

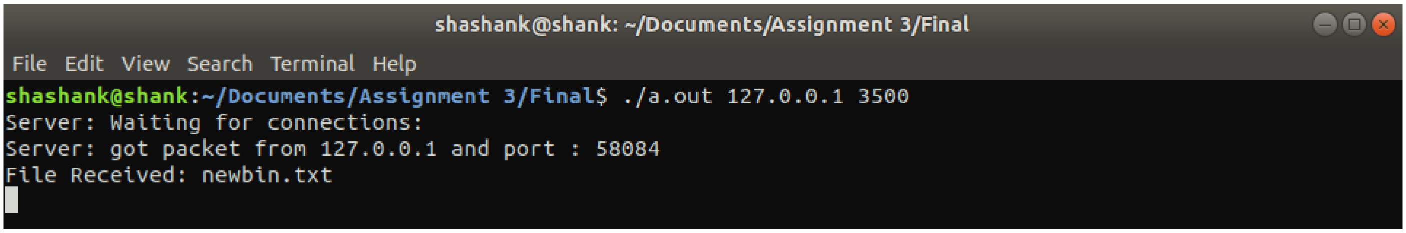
9. Writing a file in octet mode



Activities
Terminal
Sat 5:13 PM
shashank@shank: ~/Documents/Assignment 3/Final
File Edit View Search Terminal Help
shashank@shank: ~/Documents/Assignment 3/Test 1
File Edit View Search Terminal Help
WALLPAPERSWIDE.COM

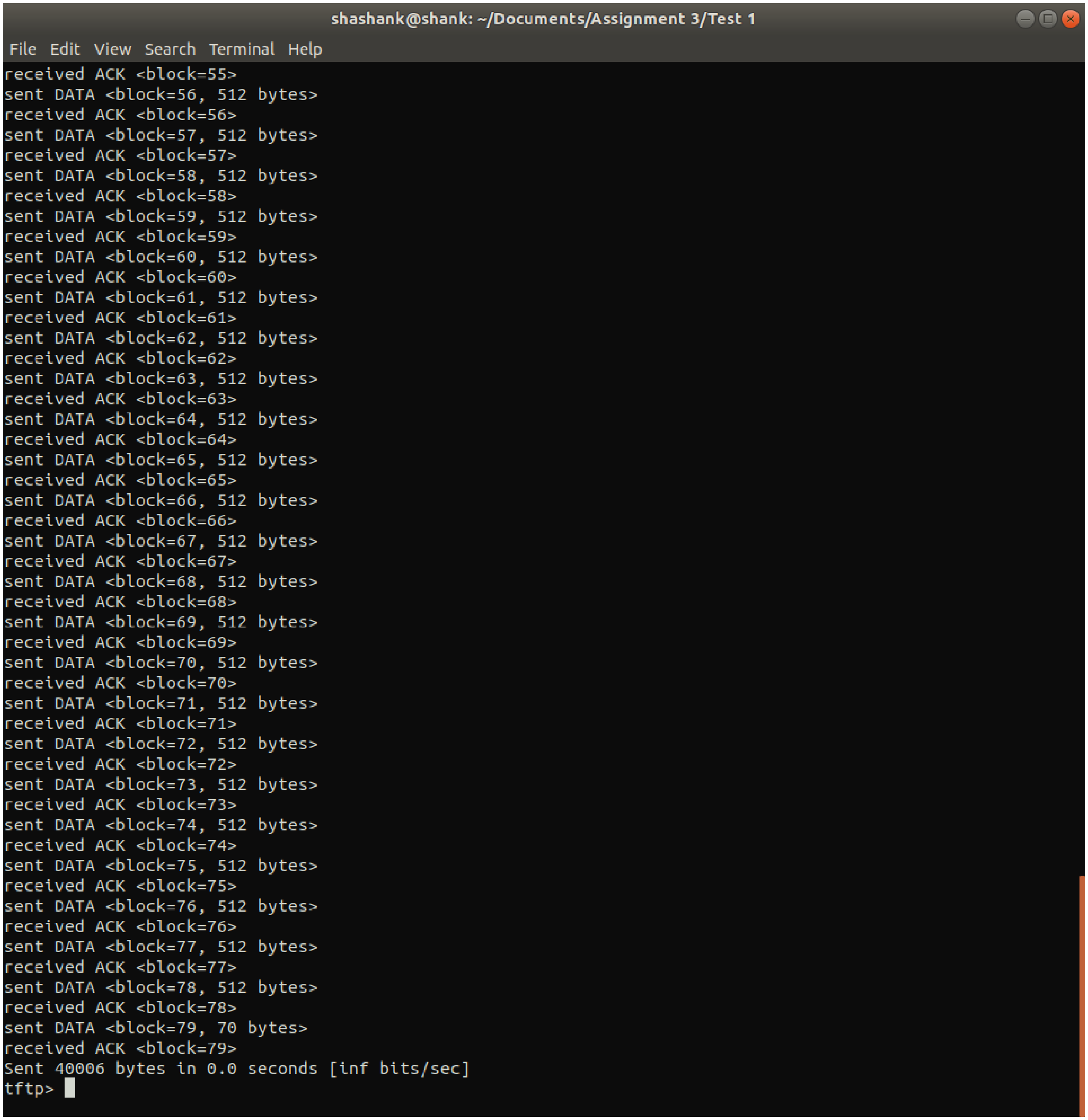
```
shashank@shank:~/Documents/Assignment 3/Final$ ./a.out 127.0.0.1 3500
Server: Waiting for connections:
Server: got packet from 127.0.0.1 and port : 58084
File Received: newbin.txt
```

```
received ACK <block=55>
sent DATA <block=56, 512 bytes>
received ACK <block=56>
sent DATA <block=57, 512 bytes>
received ACK <block=57>
sent DATA <block=58, 512 bytes>
received ACK <block=58>
sent DATA <block=59, 512 bytes>
received ACK <block=59>
sent DATA <block=60, 512 bytes>
received ACK <block=60>
sent DATA <block=61, 512 bytes>
received ACK <block=61>
sent DATA <block=62, 512 bytes>
received ACK <block=62>
sent DATA <block=63, 512 bytes>
received ACK <block=63>
sent DATA <block=64, 512 bytes>
received ACK <block=64>
sent DATA <block=65, 512 bytes>
received ACK <block=65>
sent DATA <block=66, 512 bytes>
received ACK <block=66>
sent DATA <block=67, 512 bytes>
received ACK <block=67>
sent DATA <block=68, 512 bytes>
received ACK <block=68>
sent DATA <block=69, 512 bytes>
received ACK <block=69>
sent DATA <block=70, 512 bytes>
received ACK <block=70>
sent DATA <block=71, 512 bytes>
received ACK <block=71>
sent DATA <block=72, 512 bytes>
received ACK <block=72>
sent DATA <block=73, 512 bytes>
received ACK <block=73>
sent DATA <block=74, 512 bytes>
received ACK <block=74>
sent DATA <block=75, 512 bytes>
received ACK <block=75>
sent DATA <block=76, 512 bytes>
received ACK <block=76>
sent DATA <block=77, 512 bytes>
received ACK <block=77>
sent DATA <block=78, 512 bytes>
received ACK <block=78>
sent DATA <block=79, 70 bytes>
received ACK <block=79>
Sent 40006 bytes in 0.0 seconds [inf bits/sec]
tftp> 
```



shashank@shank: ~/Documents/Assignment 3/Final
File Edit View Search Terminal Help

```
shashank@shank:~/Documents/Assignment 3/Final$ ./a.out 127.0.0.1 3500
Server: Waiting for connections:
Server: got packet from 127.0.0.1 and port : 58084
File Received: newbin.txt
```



shashank@shank: ~/Documents/Assignment 3/Test 1
File Edit View Search Terminal Help

```
received ACK <block=55>
sent DATA <block=56, 512 bytes>
received ACK <block=56>
sent DATA <block=57, 512 bytes>
received ACK <block=57>
sent DATA <block=58, 512 bytes>
received ACK <block=58>
sent DATA <block=59, 512 bytes>
received ACK <block=59>
sent DATA <block=60, 512 bytes>
received ACK <block=60>
sent DATA <block=61, 512 bytes>
received ACK <block=61>
sent DATA <block=62, 512 bytes>
received ACK <block=62>
sent DATA <block=63, 512 bytes>
received ACK <block=63>
sent DATA <block=64, 512 bytes>
received ACK <block=64>
sent DATA <block=65, 512 bytes>
received ACK <block=65>
sent DATA <block=66, 512 bytes>
received ACK <block=66>
sent DATA <block=67, 512 bytes>
received ACK <block=67>
sent DATA <block=68, 512 bytes>
received ACK <block=68>
sent DATA <block=69, 512 bytes>
received ACK <block=69>
sent DATA <block=70, 512 bytes>
received ACK <block=70>
sent DATA <block=71, 512 bytes>
received ACK <block=71>
sent DATA <block=72, 512 bytes>
received ACK <block=72>
sent DATA <block=73, 512 bytes>
received ACK <block=73>
sent DATA <block=74, 512 bytes>
received ACK <block=74>
sent DATA <block=75, 512 bytes>
received ACK <block=75>
sent DATA <block=76, 512 bytes>
received ACK <block=76>
sent DATA <block=77, 512 bytes>
received ACK <block=77>
sent DATA <block=78, 512 bytes>
received ACK <block=78>
sent DATA <block=79, 70 bytes>
received ACK <block=79>
Sent 40006 bytes in 0.0 seconds [inf bits/sec]
tftp>
```

In this case the client tries to upload a file in octet mode, the file in this case is “newbin.txt”. The initial setup is similar to the one mentioned in the section 1, after the initial setup client requests to send the file in octet mode as shown in the third screenshot. The server after receiving all the packets of newbin.txt prints the “File received” message as shown in the second screenshot. The first screenshot shows the log of both server and client side by side.