ECEN 602

NETWORK SIMULATION ASSIGNMENT – 03

TEAM 17

Mohammad Faisal Khan

Amiya Ranjan Panda

****
****

README

****
****

TITLE : Implementation of a Trivial File Transfer Protocol (TFTP) server with RRQ and WRQ features.

INTRODUCTION :

This code is a part of the Network simulation Assignment for ECEN 602 at Texas A&M University.

It has been sucessfully compiled, executed and tested on gcc compiler (part of standard LINUX).

TFTP is a simple method of transferring files between two systems that uses the User Datagram

Protocol (UDP). TFTP is basically a Stop-and-Wait protocol with timeout

similar to what we discussed in class.

While TFTP can transfer files in both directions (i.e., server to client and client to server),

the server implements reading files using the Read Request function (RRQ) and writing files using the Write Request function (WRQ) .

The server allows retrieval of files in the local directory from where the server is executed.

The sender of a data packet sets a timer when a block is first sent, and it

retransmits the data block on timeout. When the server is responding to a Read

Request (RRQ) from the client, it will be the server that implements the timeout

since it is the one sending the DATA packets; whereas, the client will be sending,ACK’s. When the server is responding to a Write Request (Bonus Feature),

however, it will be the client that implements the timeout since it is sending DATA

packets; whereas, the server will be sending ACK’s.There are two forms of transfer supported by TFTP: netascii and octet. The

netascii format is used for transferring text files, and the octet format is used for

transferring binary files. The modes are accessed by writing "netascii" aor "binary" in the client and the entering the same.

The menu can be accessed by giving the input "?" from the client. The implemented server can send files greater than 32MB in size.

Common Errors and Catches:

-If data is not input correctly on the command line as per the ordering given below, it throws a segmentation error.

-This is NOT to be assumed as an error.

-If data is missing from the command line, it throws segmentation error too.

-If a client disconnects using keyword 'quit', a smooth disconnect happens and resources are deallocated. If not, the other clients enter into hang state.

-This is not an iterative server, use of fork() in this code.

-The well-known socket for the TFTP server is port number 69 but the server uses a different one(asked in argument).

Creating MAKEFILE:

-The makefile rules are already set.

-Please use following command in the folder where makefile is present:

```
make makefile
```

Native TFTF Client in Linux is used for testing the implemented server.

Then, EXECUTE using the following commands:

```
./team17_server 127.0.0.1 50001 12
```

Package content:

1. server.c

2. Makefile

3.2048_bin file

4.2047_bin file

5.random(34mb)

Usage:

1. 'make clean' to remove all previously created object files.

2. 'make server' to compile the Server source code.

3. ./server Server_IP Server_Port

4.Use tftp command and follow instructions for generating RRQ and WRQ requests.

Tests:

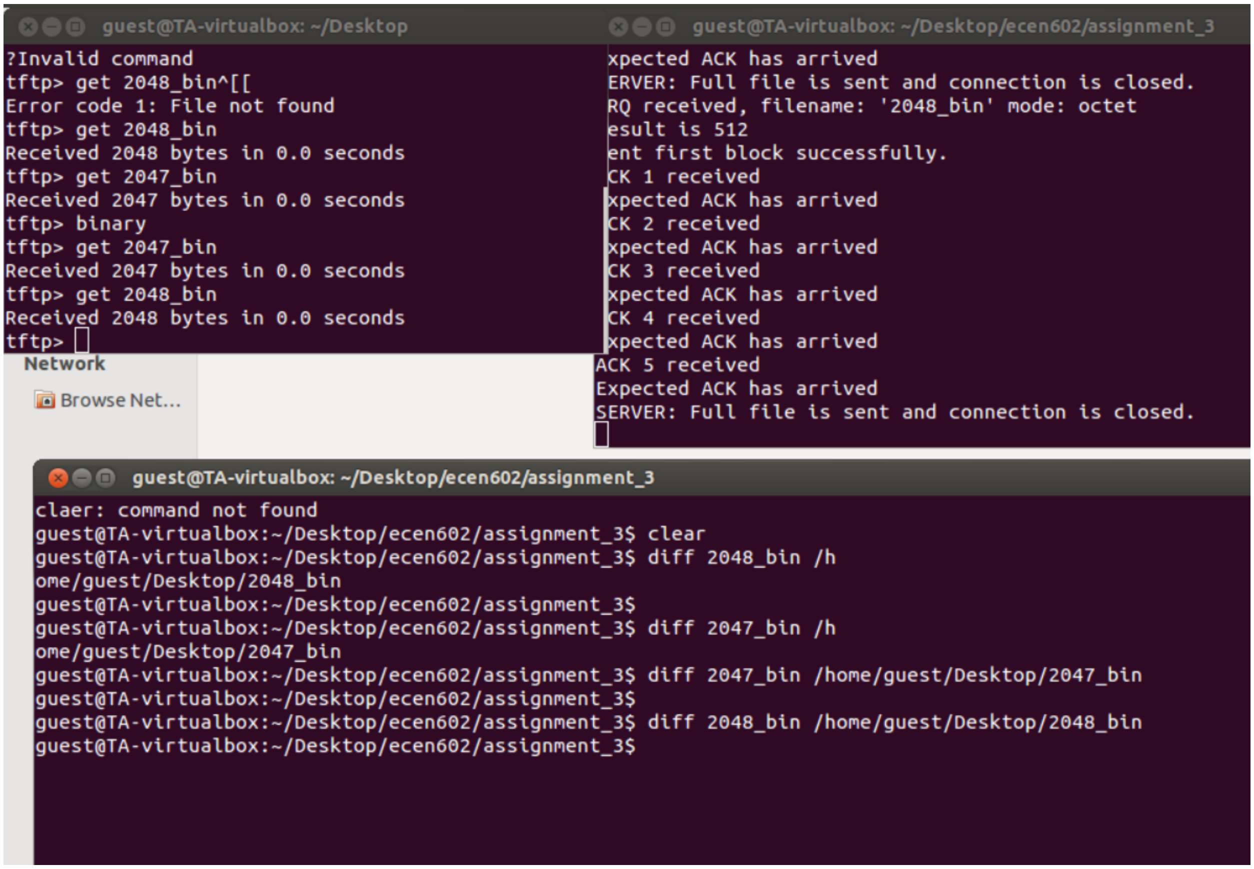
1. All RRQ tests run successfully. Block number wrap around feature tested OK.

2. Idle client identification and subsequent termination of file trasfer tested OK.

3. Feature with multiple clients connecting and downloading the same file from the server tested OK.

4. All WRQ tests run successfully.

Test Case 01 – Transferring of 2047 and 2048 binary files.

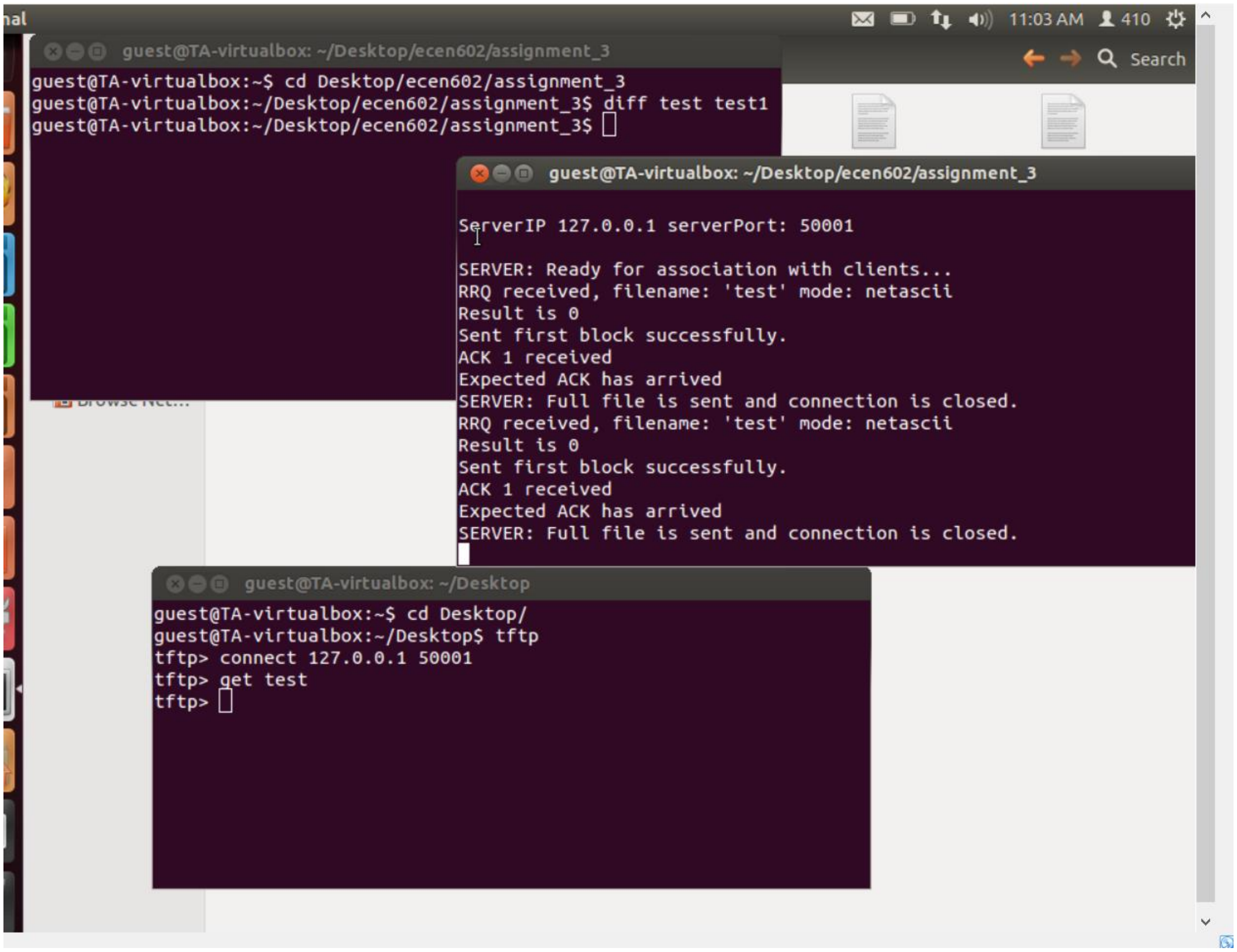

guest@TA-virtualbox: ~/Desktop
guest@TA-virtualbox: ~/Desktop/ecen602/assignment_3
Network
Browse Net...
guest@TA-virtualbox: ~/Desktop/ecen602/assignment_3

```
?Invalid command
tftp> get 2048_bin^[[
Error code 1: File not found
tftp> get 2048_bin
Received 2048 bytes in 0.0 seconds
tftp> get 2047_bin
Received 2047 bytes in 0.0 seconds
tftp> binary
tftp> get 2047_bin
Received 2047 bytes in 0.0 seconds
tftp> get 2048_bin
Received 2048 bytes in 0.0 seconds
tftp> 
```

```
xpected ACK has arrived
ERVER: Full file is sent and connection is closed.
RQ received, filename: '2048_bin' mode: octet
esult is 512
ent first block successfully.
CK 1 received
xpected ACK has arrived
CK 2 received
xpected ACK has arrived
CK 3 received
xpected ACK has arrived
CK 4 received
xpected ACK has arrived
ACK 5 received
Expected ACK has arrived
SERVER: Full file is sent and connection is closed.
```

```
claer: command not found
guest@TA-virtualbox:~/Desktop/ecen602/assignment_3$ clear
guest@TA-virtualbox:~/Desktop/ecen602/assignment_3$ diff 2048_bin /h
ome/guest/Desktop/2048_bin
guest@TA-virtualbox:~/Desktop/ecen602/assignment_3$
guest@TA-virtualbox:~/Desktop/ecen602/assignment_3$ diff 2047_bin /h
ome/guest/Desktop/2047_bin
guest@TA-virtualbox:~/Desktop/ecen602/assignment_3$ diff 2047_bin /home/guest/Desktop/2047_bin
guest@TA-virtualbox:~/Desktop/ecen602/assignment_3$
guest@TA-virtualbox:~/Desktop/ecen602/assignment_3$ diff 2048_bin /home/guest/Desktop/2048_bin
guest@TA-virtualbox:~/Desktop/ecen602/assignment_3$
```

Test Case 01 – Transferring of NETASCII files.



11:03 AM
410
Search
guest@TA-virtualbox: ~/Desktop/ecen602/assignment_3

```
guest@TA-virtualbox:~$ cd Desktop/ecen602/assignment_3
guest@TA-virtualbox:~/Desktop/ecen602/assignment_3$ diff test test1
guest@TA-virtualbox:~/Desktop/ecen602/assignment_3$ 
```

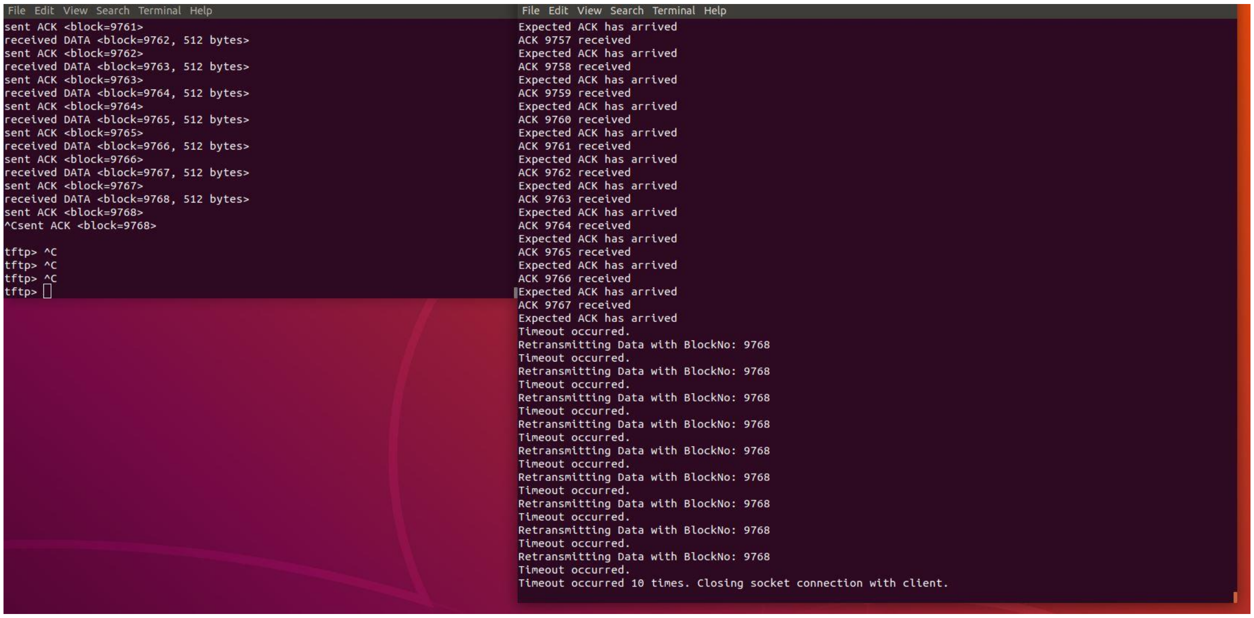
guest@TA-virtualbox: ~/Desktop/ecen602/assignment_3

```
ServerIP 127.0.0.1 serverPort: 50001

SERVER: Ready for association with clients...
RRQ received, filename: 'test' mode: netascii
Result is 0
Sent first block successfully.
ACK 1 received
Expected ACK has arrived
SERVER: Full file is sent and connection is closed.
RRQ received, filename: 'test' mode: netascii
Result is 0
Sent first block successfully.
ACK 1 received
Expected ACK has arrived
SERVER: Full file is sent and connection is closed.
```

```
guest@TA-virtualbox:~$ cd Desktop/
guest@TA-virtualbox:~/Desktop$ tftp
tftp> connect 127.0.0.1 50001
tftp> get test
tftp> 
```

3. Timeout feature after 10 seconds.

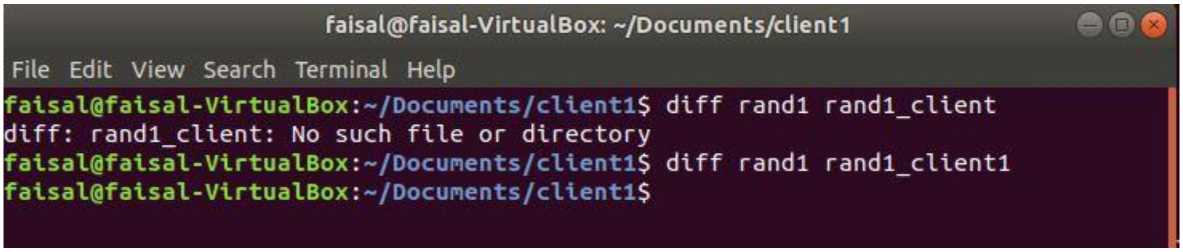

File Edit View Search Terminal Help
File Edit View Search Terminal Help

```
sent ACK <block=9761>
received DATA <block=9762, 512 bytes>
sent ACK <block=9762>
received DATA <block=9763, 512 bytes>
sent ACK <block=9763>
received DATA <block=9764, 512 bytes>
sent ACK <block=9764>
received DATA <block=9765, 512 bytes>
sent ACK <block=9765>
received DATA <block=9766, 512 bytes>
sent ACK <block=9766>
received DATA <block=9767, 512 bytes>
sent ACK <block=9767>
received DATA <block=9768, 512 bytes>
sent ACK <block=9768>
^Csent ACK <block=9768>

tftp> ^C
tftp> ^C
tftp> ^C
tftp> 
```

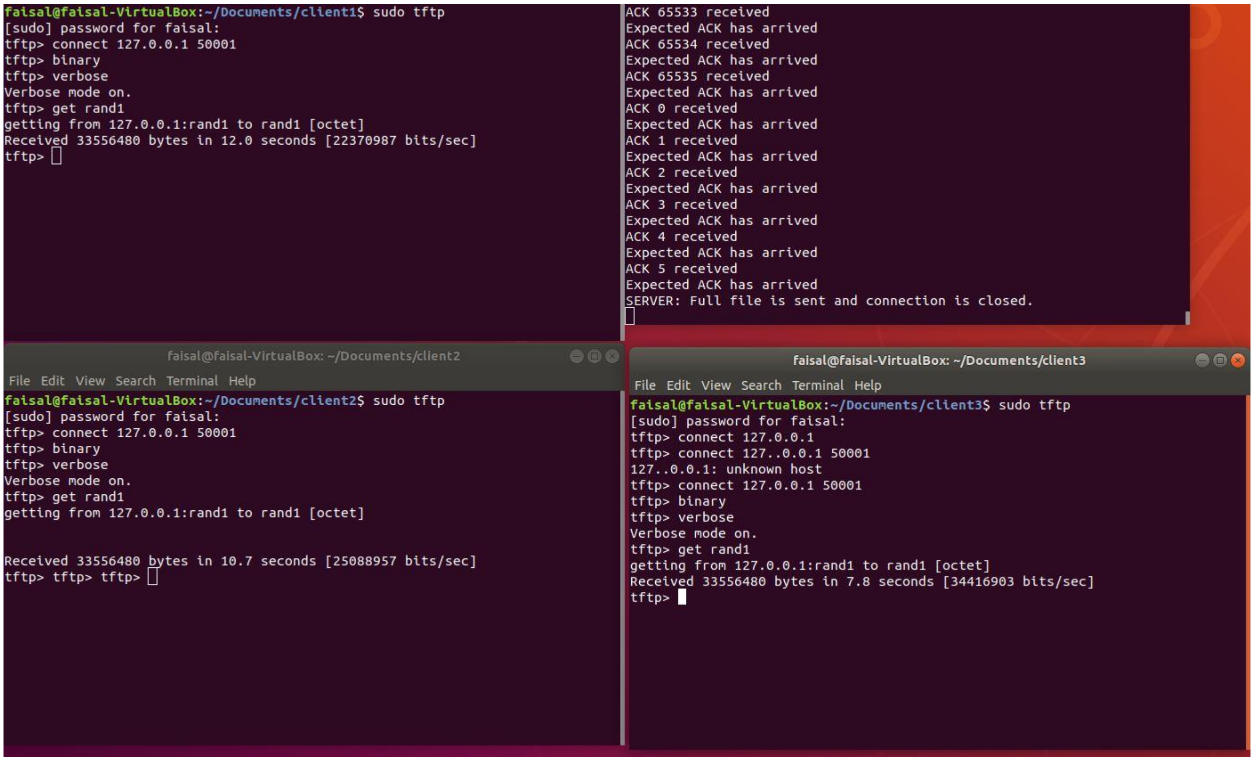
```
Expected ACK has arrived
ACK 9757 received
Expected ACK has arrived
ACK 9758 received
Expected ACK has arrived
ACK 9759 received
Expected ACK has arrived
ACK 9760 received
Expected ACK has arrived
ACK 9761 received
Expected ACK has arrived
ACK 9762 received
Expected ACK has arrived
ACK 9763 received
Expected ACK has arrived
ACK 9764 received
Expected ACK has arrived
ACK 9765 received
Expected ACK has arrived
ACK 9766 received
Expected ACK has arrived
ACK 9767 received
Expected ACK has arrived
Timeout occurred.
Retransmitting Data with BlockNo: 9768
Timeout occurred.
Retransmitting Data with BlockNo: 9768
Timeout occurred.
Retransmitting Data with BlockNo: 9768
Timeout occurred.
Retransmitting Data with BlockNo: 9768
Timeout occurred.
Retransmitting Data with BlockNo: 9768
Timeout occurred.
Retransmitting Data with BlockNo: 9768
Timeout occurred.
Retransmitting Data with BlockNo: 9768
Timeout occurred.
Retransmitting Data with BlockNo: 9768
Timeout occurred.
Retransmitting Data with BlockNo: 9768
Timeout occurred.
Timeout occurred 10 times. Closing socket connection with client.
```

4. 34 mb file transfer.


faisal@faisal-VirtualBox: ~/Documents/client1
File Edit View Search Terminal Help

```
faisal@faisal-VirtualBox:~/Documents/client1$ diff rand1 rand1_client
diff: rand1_client: No such file or directory
faisal@faisal-VirtualBox:~/Documents/client1$ diff rand1 rand1_client1
faisal@faisal-VirtualBox:~/Documents/client1$
```

5. Multiple clients, heavy file transfer.


faisal@faisal-VirtualBox: ~/Documents/client2
File Edit View Search Terminal Help
faisal@faisal-VirtualBox: ~/Documents/client3
File Edit View Search Terminal Help

```
faisal@faisal-VirtualBox:~/Documents/client1$ sudo tftp
[sudo] password for faisal:
tftp> connect 127.0.0.1 50001
tftp> binary
tftp> verbose
Verbose mode on.
tftp> get rand1
getting from 127.0.0.1:rand1 to rand1 [octet]
Received 33556480 bytes in 12.0 seconds [22370987 bits/sec]
tftp>
```

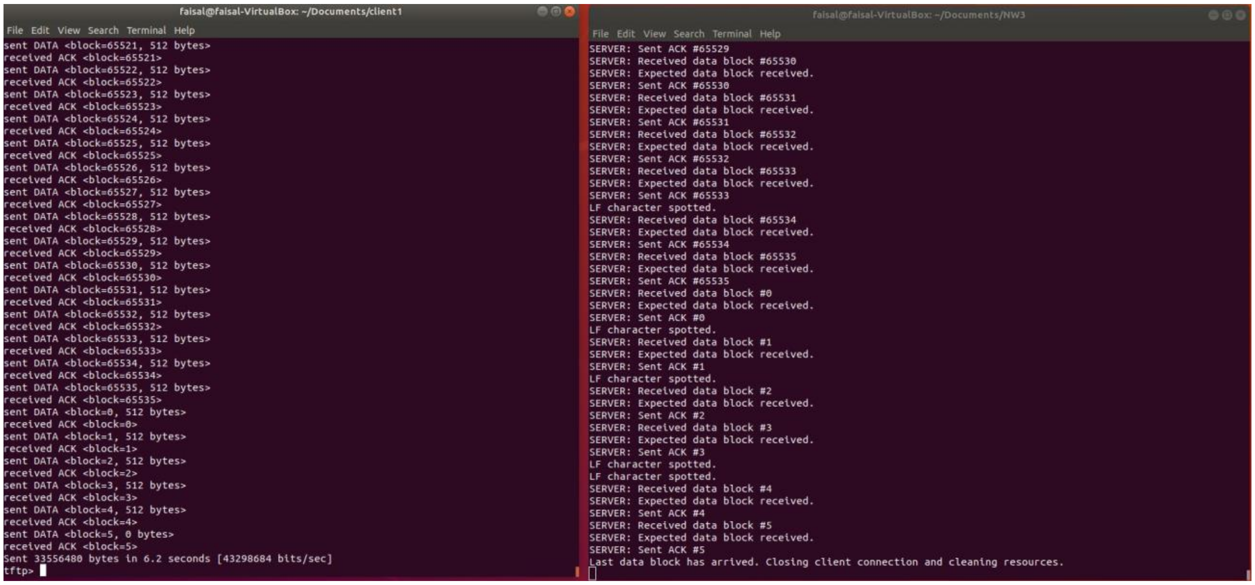
```
ACK 65533 received
Expected ACK has arrived
ACK 65534 received
Expected ACK has arrived
ACK 65535 received
Expected ACK has arrived
ACK 0 received
Expected ACK has arrived
ACK 1 received
Expected ACK has arrived
ACK 2 received
Expected ACK has arrived
ACK 3 received
Expected ACK has arrived
ACK 4 received
Expected ACK has arrived
ACK 5 received
Expected ACK has arrived
SERVER: Full file is sent and connection is closed.
```

```
faisal@faisal-VirtualBox:~/Documents/client2$ sudo tftp
[sudo] password for faisal:
tftp> connect 127.0.0.1 50001
tftp> binary
tftp> verbose
Verbose mode on.
tftp> get rand1
getting from 127.0.0.1:rand1 to rand1 [octet]


Received 33556480 bytes in 10.7 seconds [25088957 bits/sec]
tftp> tftp> tftp>
```

```
faisal@faisal-VirtualBox:~/Documents/client3$ sudo tftp
[sudo] password for faisal:
tftp> connect 127.0.0.1
tftp> connect 127..0.0.1 50001
127..0.0.1: unknown host
tftp> connect 127.0.0.1 50001
tftp> binary
tftp> verbose
Verbose mode on.
tftp> get rand1
getting from 127.0.0.1:rand1 to rand1 [octet]
Received 33556480 bytes in 7.8 seconds [34416903 bits/sec]
tftp>
```

6.New feature WRQ.


faisal@faisal-VirtualBox: ~/Documents/client1
File Edit View Search Terminal Help
faisal@faisal-VirtualBox: ~/Documents/NW3
File Edit View Search Terminal Help

```
sent DATA <block=65521, 512 bytes>
received ACK <block=65521>
sent DATA <block=65522, 512 bytes>
received ACK <block=65522>
sent DATA <block=65523, 512 bytes>
received ACK <block=65523>
sent DATA <block=65524, 512 bytes>
received ACK <block=65524>
sent DATA <block=65525, 512 bytes>
received ACK <block=65525>
sent DATA <block=65526, 512 bytes>
received ACK <block=65526>
sent DATA <block=65527, 512 bytes>
received ACK <block=65527>
sent DATA <block=65528, 512 bytes>
received ACK <block=65528>
sent DATA <block=65529, 512 bytes>
received ACK <block=65529>
sent DATA <block=65530, 512 bytes>
received ACK <block=65530>
sent DATA <block=65531, 512 bytes>
received ACK <block=65531>
sent DATA <block=65532, 512 bytes>
received ACK <block=65532>
sent DATA <block=65533, 512 bytes>
received ACK <block=65533>
sent DATA <block=65534, 512 bytes>
received ACK <block=65534>
sent DATA <block=65535, 512 bytes>
received ACK <block=65535>
sent DATA <block=0, 512 bytes>
received ACK <block=0>
sent DATA <block=1, 512 bytes>
received ACK <block=1>
sent DATA <block=2, 512 bytes>
received ACK <block=2>
sent DATA <block=3, 512 bytes>
received ACK <block=3>
sent DATA <block=4, 512 bytes>
received ACK <block=4>
sent DATA <block=5, 0 bytes>
received ACK <block=5>
Sent 33556480 bytes in 6.2 seconds [43298684 bits/sec]
tftp>
```

```
SERVER: Sent ACK #65529
SERVER: Received data block #65530
SERVER: Expected data block received.
SERVER: Sent ACK #65530
SERVER: Received data block #65531
SERVER: Expected data block received.
SERVER: Sent ACK #65531
SERVER: Received data block #65532
SERVER: Expected data block received.
SERVER: Sent ACK #65532
SERVER: Received data block #65533
SERVER: Expected data block received.
SERVER: Sent ACK #65533
LF character spotted.
SERVER: Received data block #65534
SERVER: Expected data block received.
SERVER: Sent ACK #65534
SERVER: Received data block #65535
SERVER: Expected data block received.
SERVER: Sent ACK #65535
SERVER: Received data block #0
SERVER: Expected data block received.
SERVER: Sent ACK #0
LF character spotted.
SERVER: Received data block #1
SERVER: Expected data block received.
SERVER: Sent ACK #1
LF character spotted.
SERVER: Received data block #2
SERVER: Expected data block received.
SERVER: Sent ACK #2
SERVER: Received data block #3
SERVER: Expected data block received.
SERVER: Sent ACK #3
LF character spotted.
LF character spotted.
SERVER: Received data block #4
SERVER: Expected data block received.
SERVER: Sent ACK #4
SERVER: Received data block #5
SERVER: Expected data block received.
SERVER: Sent ACK #5
Last data block has arrived. Closing client connection and cleaning resources.
```

CODE :

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

#define MAX_FILE_NAME 512


int timeout_indicator(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));
}

int error_sys(const char* x)
{
    perror(x);
    exit(1);
}

int main(int argc, char *argv[])
{
    char data[512] = {0};
    int sockfd, newsock_fd;
    struct sockaddr_in client;
    socklen_t clientlength = sizeof(struct sockaddr_in);
    int g, ret;
    int send_res;
    int last_block;

    if (argc != 3){
        error_sys ("USAGE: ./server <Server_IP> <Port_Number>");
        return 0;
    }

    char *serverIP = argv[1];
    unsigned short int serverPort = atoi(argv[2]);

    printf("\nServerIP %s serverPort: %d\n\n", serverIP, serverPort );

    int recbyte;
    char buff [1024] = {0};
    char ack_packet[32] = {0};
    char file_payload[516] = {0};
    char file_payload_copy[516] = {0};
    char filename[MAX_FILE_NAME];
    char Mode[512];
    unsigned short int opcode1, opcode2, BlockNo;
    int b, j;
    FILE *fp;
    struct addrinfo hints, *ai, *clientinfo, *p;
    int yes = 1;
    int pid, read_ret;
    int blocknum = 0, timeout_count = 0, NACK = 0;
    char *ephemeral_port;

    ephemeral_port = (char *)malloc (sizeof ephemeral_port);

    socklen_t addrlen;

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

    if ((ret = getaddrinfo(NULL, argv[2], &hints, &ai)) != 0) {
        fprintf(stderr, "SERVER: %s\n", gai_strerror(ret));
        exit(1);
    }

      for(p = ai; p != NULL; p = p->ai_next) {
        sockfd = socket(p->ai_family, p->ai_socktype, p->ai_protocol);
        if (sockfd < 0) {
            continue;
        }
        setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &yes,
sizeof(int));
        if (bind(sockfd, p->ai_addr, p->ai_addrlen) < 0) {
            close(sockfd);
            continue;
        }
        break;
    }
    if (p == NULL) {
        fprintf(stderr, "Failed to bind socket\n");
        return 1;
    }

    freeaddrinfo(ai);
    printf("SERVER: Ready for association with clients...\n");

    while(1) {
        recbyte = recvfrom(sockfd, buff, sizeof(buff), 0, (struct
sockaddr*)&client, &clientlength);
        if (recbyte < 0) {
            error_sys("ERROR: Couldn't receive data");
            return 1;
        }
        else {
      }
        memcpy(&opcode1,&buff,2);
        opcode1 = ntohs(opcode1);
        pid = fork();
        if (pid == 0) {
            if (opcode1 == 1){
                bzero(filename, MAX_FILE_NAME);
                for (b = 0; buff[2+b] != '\0'; b++) {
                filename[b]=buff[2+b];
            }
            filename[b]='\0';
            bzero(Mode, 512);
            g = 0;
            for (j = b+3; buff[j] != '\0'; j++) {
                Mode[g]=buff[j];
                g++;
            }
            Mode[g]='\0';
            printf("RRQ received, filename: '%s' mode: %s\n",
filename, Mode);
            fp = fopen (filename, "r");
            if (fp != NULL) {
                close(sockfd);
            *ephemeral_port = htons(0);
            if ((ret = getaddrinfo(NULL, ephemeral_port, &hints,
&clientinfo)) != 0) {
              fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(ret));
              return 10;
            }
            for(p = clientinfo; p != NULL; p = p->ai_next) {
              if ((newsock_fd = socket(p->ai_family, p->ai_socktype,p-
>ai_protocol)) == -1) {
                error_sys("ERROR: SERVER (child): socket");
                continue;
              }

setsockopt(newsock_fd,SOL_SOCKET,SO_REUSEADDR,&yes,sizeof(int));
              if (bind(newsock_fd, p->ai_addr, p->ai_addrlen) == -1) {
                    close(newsock_fd);
                error_sys("ERROR: SERVER (newsock_fd): bind");
                continue;
              }
              break;
            }
            freeaddrinfo(clientinfo);

            // Create data packet
            bzero(file_payload, sizeof(file_payload));
            bzero(data, sizeof(data));
            read_ret = fread (&data,1,512,fp);// Retrieve data from
file pointer corresponding to filename received in RRQ
            if(read_ret>=0) {
              data[read_ret]='\0';
              printf("Result is %d\n",read_ret);
            }
            if(read_ret < 512)
              last_block = 1;
            BlockNo = htons(1);                             // 1st 512
Byte block
            opcode2 = htons(3);                                 //
Opcode = 3: Data
            memcpy(&file_payload[0], &opcode2, 2);              // 2
Bytes
            memcpy(&file_payload[2], &BlockNo, 2);          // 4 Bytes
            for (b = 0; data[b] != '\0'; b++) {
              file_payload[b+4] = data[b];
            }
          int p = 0;
            bzero(file_payload_copy, sizeof(file_payload_copy));
            memcpy(&file_payload_copy[0], &file_payload[0], 516);
            send_res=sendto(newsock_fd, file_payload, (read_ret + 4),
0, (struct sockaddr*)&client, clientlength);
            NACK = 1;
            if (send_res < 0)
              error_sys("Couldn't send first packet: ");
            else
              printf("Sent first block successfully.\n");

            while(1){
              if (timeout_indicator (newsock_fd, 1) != 0) {
                bzero(buff, sizeof(buff));
                bzero(file_payload, sizeof(file_payload));
                recbyte = recvfrom(newsock_fd, buff, sizeof(buff), 0,
(struct sockaddr*)&client, &clientlength);
                timeout_count = 0;
                if (recbyte < 0) {
                  error_sys("Couldn't receive data\n");
                  return 6;
                }
                else {}
                memcpy(&opcode1, &buff[0], 2);
                if (ntohs(opcode1) == 4) {
// Opcode = 4: ACK
                  bzero(&blocknum, sizeof(blocknum));
                  memcpy(&blocknum, &buff[2], 2);
                  blocknum = ntohs(blocknum);
                  printf("ACK %i received\n", blocknum);
                  //check reach end of file
                  if(blocknum == NACK) {
                    printf("Expected ACK has arrived\n");
                    NACK = (NACK + 1)%65536;
                if(last_block == 1){
                      close(newsock_fd);
                      fclose(fp);
                  printf("SERVER: Full file is sent and connection is
closed.\n");
                  exit(5);
                  last_block = 0;
                }
                else {
                      bzero(data, sizeof(data));
                      read_ret = fread (&data,1,512,fp);
                      if(read_ret>=0) {
                        if(read_ret < 512)
                          last_block = 1;
                        data[read_ret]='\0';
                        BlockNo = htons(((blocknum+1)%65536));
                        opcode2 = htons(3);
                        memcpy(&file_payload[0], &opcode2, 2);
                        memcpy(&file_payload[2], &BlockNo, 2);
                        for (b = 0; data[b] != '\0'; b++) {
                          file_payload[b+4] = data[b];
                        }
                        bzero(file_payload_copy,
sizeof(file_payload_copy));
                        memcpy(&file_payload_copy[0],
&file_payload[0], 516);
                        int send_res = sendto(newsock_fd,
file_payload, (read_ret + 4), 0, (struct sockaddr*)&client,
clientlength);
                        if (send_res < 0)
                          error_sys("ERROR: Sendto ");
                      }
                    }
                  }
                  else {
                    printf("Expected ACK hasn't arrived: NACK: %d,
blocknum: %d\n", NACK, blocknum);
                  }
                }
              }
              else {
                timeout_count++;
                printf("Timeout occurred.\n");
                if (timeout_count == 10){
                  printf("Timeout occurred 10 times. Closing socket
connection with client.\n");
                  close(newsock_fd);
                  fclose(fp);
                  exit(6);
                }
                else {
                  bzero(file_payload, sizeof(file_payload));
                  memcpy(&file_payload[0], &file_payload_copy[0],
516);
                  memcpy(&BlockNo, &file_payload[2], 2);
                  BlockNo = htons(BlockNo);
                  printf ("Retransmitting Data with BlockNo: %d\n",
BlockNo);
                  send_res = sendto(newsock_fd, file_payload_copy,
(read_ret + 4), 0, (struct sockaddr*)&client, clientlength);
                  bzero(file_payload_copy, sizeof(file_payload_copy));
                  memcpy(&file_payload_copy[0], &file_payload[0],
516);
                  if (send_res < 0)
                    error_sys("ERROR: Sendto ");
                }
              }
            }
          }
          else {
            unsigned short int ERRCode = htons(1);
            unsigned short int ERRoc = htons(5);             // Opcode
= 5: Error
            char ERRMsg[512] = "File not found";
            char ERRBuff[516] = {0};
            memcpy(&ERRBuff[0], &ERRoc, 2);
            memcpy(&ERRBuff[2], &ERRCode, 2);
            memcpy(&ERRBuff[4], &ERRMsg, 512);
            sendto(sockfd, ERRBuff, 516, 0, (struct sockaddr*)&client,
clientlength);
            printf("Server clean up as filename doesn't match.\n");
            close(sockfd);
            fclose(fp);
            exit(4);
          }
        }
        else if (opcode1 == 2) {                                 //
WRQ processing
          *ephemeral_port = htons(0);
          if ((ret = getaddrinfo(NULL, ephemeral_port, &hints,
&clientinfo)) != 0) {
            fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(ret));
            return 10;
          }
          for(p = clientinfo; p != NULL; p = p->ai_next) {
            if ((newsock_fd = socket(p->ai_family, p->ai_socktype,p-
>ai_protocol)) == -1) {
              error_sys("ERROR: SERVER (child): socket");
              continue;
            }

setsockopt(newsock_fd,SOL_SOCKET,SO_REUSEADDR,&yes,sizeof(int));
            if (bind(newsock_fd, p->ai_addr, p->ai_addrlen) == -1) {
                    close(newsock_fd);
              error_sys("ERROR: SERVER (newsock_fd): bind");
              continue;
            }
            break;
          }
          freeaddrinfo(clientinfo);
       printf("SERVER: WRQ received from client.\n");
       FILE *fp_wr = fopen("WRQ_data.txt", "w+");
       if (fp_wr == NULL)
         printf("SERVER: WRQ: Problem in opening file");
          opcode2 = htons(4);
       BlockNo = htons (0);
          bzero(ack_packet, sizeof(ack_packet));
          memcpy(&ack_packet[0], &opcode2, 2);
          memcpy(&ack_packet[2], &BlockNo, 2);
          send_res = sendto(newsock_fd, ack_packet, 4, 0, (struct
sockaddr*)&client, clientlength);
       NACK = 1;
          if (send_res < 0)
            error_sys("WRQ ACK ERROR: Sendto ");
       while(1){
            bzero(buff, sizeof(buff));
            recbyte = recvfrom(newsock_fd, buff, sizeof(buff), 0,
(struct sockaddr*)&client, &clientlength);
            if (recbyte < 0) {
              error_sys("WRQ: Couldn't receive data\n");
              return 9;
            }

            bzero(data, sizeof(data));
            memcpy(&BlockNo, &buff[2], 2);
         g = 0;
            for (b = 0; buff[b+4] != '\0'; b++) {
           if (buff[b+4] == '\n'){
             printf("LF character spotted.\n");
             g++;
             if (b-g<0)
               printf("ERROR: b-g is less than 0");
             data[b-g] = '\n';
              }
           else
```

```
                data[b - g] = buff[b+4];
            }
         fwrite(data, 1, (recbyte - 4 - g), fp_wr);
         BlockNo = ntohs(BlockNo);
         if (NACK == BlockNo){
           printf("SERVER: Received data block #%d\n", NACK);
           printf("SERVER: Expected data block received.\n");
              opcode2 = htons(4);
           BlockNo = ntohs(NACK);
              bzero(ack_packet, sizeof(ack_packet));
              memcpy(&ack_packet[0], &opcode2, 2);
              memcpy(&ack_packet[2], &BlockNo, 2);
           printf("SERVER: Sent ACK #%d\n", htons(BlockNo));
              send_res = sendto(newsock_fd, ack_packet, 4, 0, (struct
sockaddr*)&client, clientlength);
           if (recbyte < 516) {
                printf("Last data block has arrived. Closing client
connection and cleaning resources. \n");
                close(newsock_fd);
                fclose(fp_wr);
                exit(9);
           }
              NACK = (NACK + 1)%65536;
          }
        }
        }
      }
  }
}
```