ECEN 602
Network Programming Assignment 3
TFTP Communication

TEAM : 1

SANKET AGARWAL : 329006490
DHIRAJ KUDVA : 829009538

Problem Description:
In this assignment, we have implemented a Trivial File Transfer Protocol (TFTP) server. TFTP is a simple method of transferring files between two systems that use the User Datagram Protocol (UDP). For this problem, only the server is implemented and the standard Client is used. The server is capable of handling multiple simultaneous clients. TFTP takes much less code to implement than the File Transfer Protocol (FTP), which has many more features and supports user authentication.

To start the tftp server we need to give the following command:
./server 127.0.0.1 24000 i.e. ./executable IP_add port_no

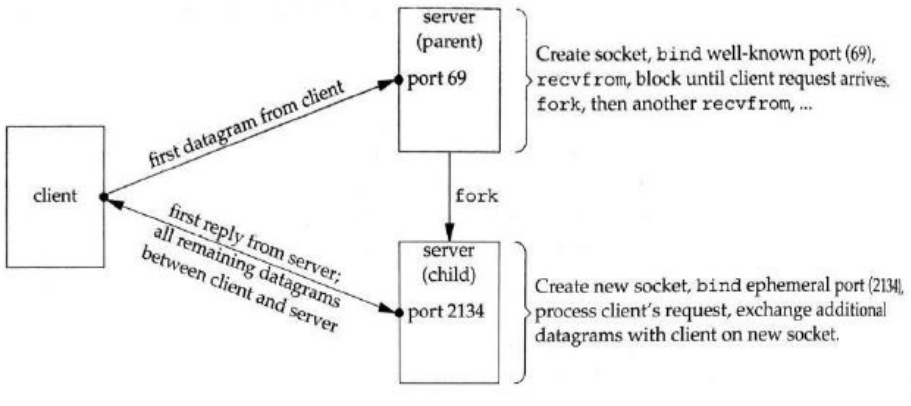
Two types of files were tested binary and netascii files. After getting back from the server, they were compared with the files in the server using the diff command. In case the diff command shows nothing, it means that the files were copied successfully. More test cases are mentioned below. Files were generated using an online file generator.

Team Contribution:
Due to the mid-semester exam, we didn't have time to divide work on an individual basis, thus the entire code was written over zoom calls by sharing the screen. In addition to developing the code, test cases were also developed over zoom calls. Thus both students have equal contribution to coding, debugging, and testing.

<u>BONUS FEATURE: WRQ is also implemented along with RRQ.</u>

Directory Structure:
The directory structure is as follows:

Server.c: Contains the code for the server implementation. It first listens to the client on the port requested by the client and then replied back to the client on an ephemeral port in order for the other clients to get connected.

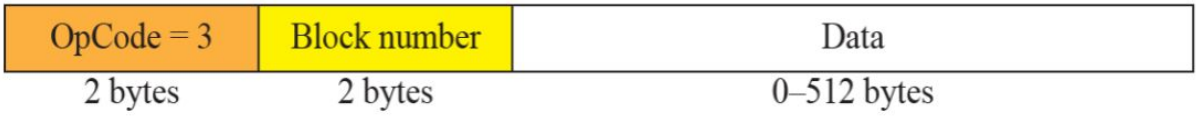


The server is capable of listening to multiple clients at the same time and respond to all the clients with files requested by each client. In case the file is not present with the server, it replies back to the client with an error message saying that the file is not present. The message format for different types of messages are as follows:

RRQ format

WRQ format

DATA format



ACK format

Makefile: It is used to compile the server code. The client is a default one so no client compilation is required.

Usage:
The directory shared has the following files:

1. Server.c
2. Makefile
3. Testfiles :
    a. 2047_team1
    b. 2048_team1
    c. team1_20MB
    d. team1_25MB
    e. team1_30MB
    f. team1_34MB
    g. team1_from_client
    h. team1netascii
    i. team2netascii

As the client is already installed on hera server, it just needs to be called using the “tftp” command on the hera server command prompt. But this has to be called from a different directory inside the current working directory. Hence to do this, a new folder called “client” is created and from this folder the tftp command is executed.
The server object is created by running the “make” command from the parent directory (not the client one) of this project.

To setup client and socket environment, open another terminal and point to the same working directory where all the collaterals (necessary files) are generated.

The next step would be to start the server. Run “./server 127.0.0.1 25000” where 25000 is the port number. Any port number can be selected within the valid range. “127.0.0.1” is the local IP address.
Once this is setup, server will show the below message

```
:: ./server 127.0.0.1 25000
Server is now read. Waiting for Clients to join...
```

This indicates that the server is waiting for the client.

In the second terminal, run “tftp“.Then enter the IP address and port number to connect with the server. This should be similar to be the address given by the server in the previous command.

```
[dhirajkudva]@hera3 ~/ECEN_602_git_new/socket_programming/Assignment_3_new/client> (20:32:
04 10/11/20)
:: tftp
(to) 127.0.0.1 25000
tftp> status
Connected to 127.0.0.1.
Mode: netascii Verbose: off Tracing: off Literal: off
Rexmt-interval: 5 seconds, Max-timeout: 25 seconds
```

The “status” command gives us detailed information about the tftp client and the mode of file that can be transferred.
“Netascii” -> transfers ascii files selected by using command “ascii”
“Octet” -> transfers binary files. Selected by using command “binary”

To get a file from the server we use “get <filename>” and ensure that the mode of the tftp is as per the type of file. To transfer a file from client to server, “put <filename> “ command is used.

TEST CASES:

1. transfer a binary file of 2048 bytes and check that it matches the source file,

In this program, a 2048 file is transferred from server to client.
The transfer is initiated by get command followed by the file needed from the server. As the file is binary, the mode is set to binary and checked using status command
The successful transmission is indicated on the client by
“Received 2048 bytes in X seconds”
Also while transmitting 2048 bytes, it is sent in 4 packets each of 512 bytes and then a final block indicating end of packet.
The server indicates end of file by message: “Full file sent completely”
Diff method checks and confirms that the same file is received by the client.
Client side output:

```
[dhirajkudva]@hera3 ~/ECEN_602_git_new/socket_programming/Assignment_3_new/client> (00:46:3
:: tftp
(to) 127.0.0.1 25000
tftp> status
Connected to 127.0.0.1.
Mode: netascii Verbose: off Tracing: off Literal: off
Rexmt-interval: 5 seconds, Max-timeout: 25 seconds
tftp> binary
tftp> verbose
Verbose mode on.
tftp> trace
Packet tracing on.
tftp> get 2048_team1
getting from 127.0.0.1:2048_team1 to 2048_team1 [octet]
Received 2048 bytes in 0.1 seconds [148575 bit/s]
tftp> quit

[dhirajkudva]@hera3 ~/ECEN_602_git_new/socket_programming/Assignment_3_new/client> (00:51:
41 10/13/20)
:: diff 2048_team1 ../2048_team1
```

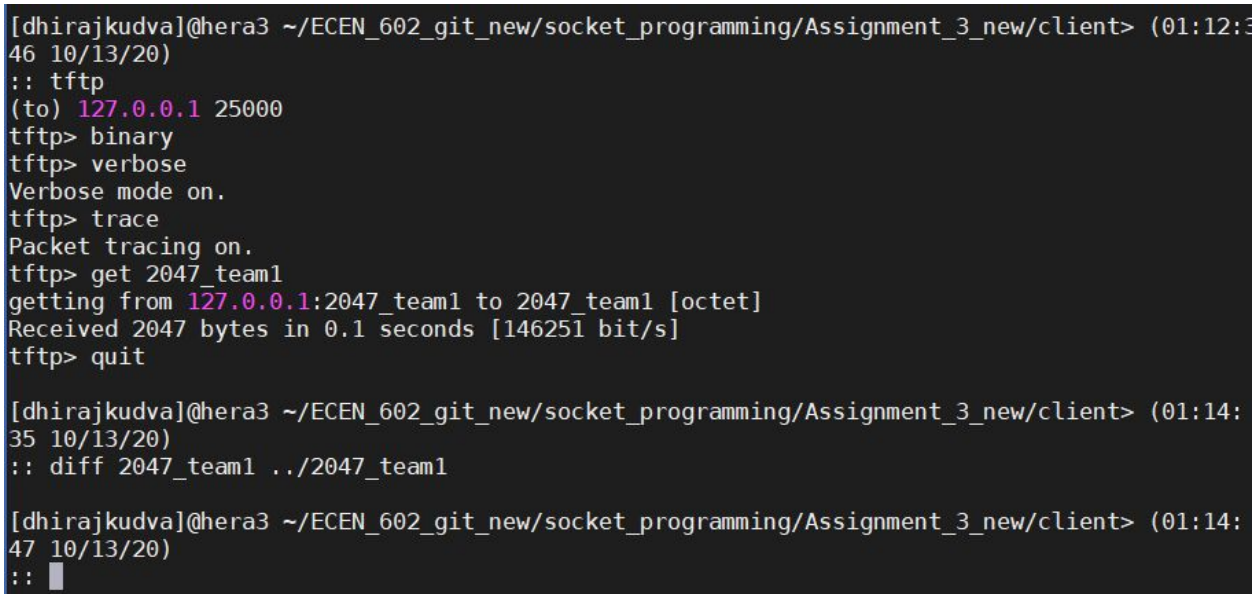
Server side output:

```
[dhirajkudva]@hera3 ~/ECEN_602_git_new/socket_programming/Assignment_3_new> (00:50:42 10/1
3/20)
:: ./server 127.0.0.1 25000
Server is now read. Waiting for Clients to join...
RRQ: filename: 2048_team1 mode: octet
read result is 512
First block successfully sent .
RECEIVED DATAAcknowledgement 1 received
Expected Acknowledgement has arrived
actual bytes read from file 512RECEIVED DATAAcknowledgement 2 received
Expected Acknowledgement has arrived
actual bytes read from file 512RECEIVED DATAAcknowledgement 3 received
Expected Acknowledgement has arrived
actual bytes read from file 512RECEIVED DATAAcknowledgement 4 received
Expected Acknowledgement has arrived
actual bytes read from file 0RECEIVED DATAAcknowledgement 5 received
Expected Acknowledgement has arrived
SERVER: Full file sent completely
```

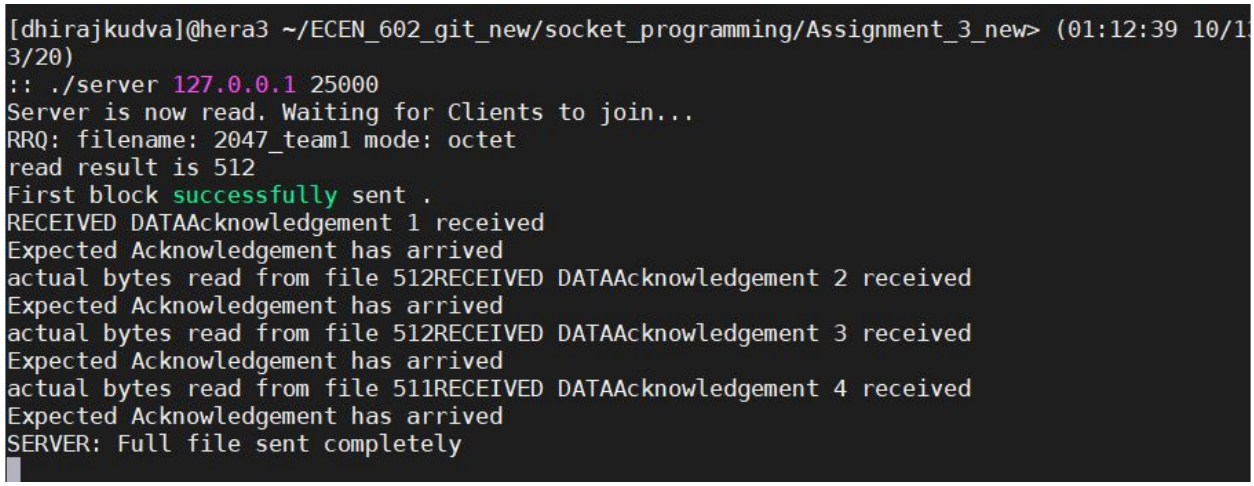
2. transfer a binary file of 2047 bytes and check that it matches the source file

It is similar to the earlier testcase, but the only difference is that instead of 2048, 2047 bytes are sent. So in this case, instead of five packets, 4 packets are sent: 3 packets of 512 bytes and last packet of 511 bytes. The last packet also indicates end of file.
Diff command again checks that both the packets have same content.

Client side output:



Server side output:



3. transfer a netascii file that includes two CR's and check that the resulting file matches the input file

In this case, instead of binary the mode is set to netascii. Rest of the steps are same.

Client output:

```
[dhirajkudva]@hera3 ~/ECEN_602_git_new/socket_programming/Assignment_3_new/client> (01:19:3
01 10/13/20)
:: tftp
(to) 127.0.0.1 25000
tftp> ascii
tftp> verbose
Verbose mode on.
tftp> trace
Packet tracing on.
tftp> status
Connected to 127.0.0.1.
Mode: netascii Verbose: on Tracing: on Literal: off
Rexmt-interval: 5 seconds, Max-timeout: 25 seconds
tftp> get team1netascii
getting from 127.0.0.1:team1netascii to team1netascii [netascii]
Received 40 bytes in 0.1 seconds [3092 bit/s]
tftp> quit

[dhirajkudva]@hera3 ~/ECEN_602_git_new/socket_programming/Assignment_3_new/client> (01:20:
57 10/13/20)
:: diff team1netascii ../team1netascii

[dhirajkudva]@hera3 ~/ECEN_602_git_new/socket_programming/Assignment_3_new/client> (01:21:
```

Server output:

```
[dhirajkudva]@hera3 ~/ECEN_602_git_new/socket_programming/Assignment_3_new> (01:18:57 10/1
3/20)
:: ./server 127.0.0.1 25000
Server is now read. Waiting for Clients to join...
RRQ: filename: team1netascii mode: netascii
read result is 40
First block successfully sent .
RECEIVED DATAAcknowledgement 1 received
Expected Acknowledgement has arrived
SERVER: Full file sent completely
```

4. transfer a binary file of 34 MB and see if block number wrap-around works,

In this case the mode is binary. As the file size is greater than 65535 * 512 bytes, the acknowledgement number gets wrapped around and again starts from 0 after reaching 65535.

Client side output :

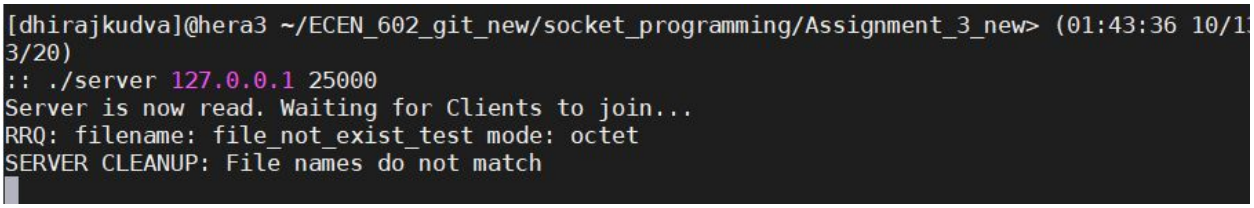
```
[dhirajkudva]@hera3 ~/ECEN_602_git_new/socket_programming/Assignment_3_new/client> (01:29:3
24 10/13/20)
:: tftp
(to) 127.0.0.1 25000
tftp> binary
tftp> verbose
Verbose mode on.
tftp> trace
Packet tracing on.
tftp> status
Connected to 127.0.0.1.
Mode: octet Verbose: on Tracing: on Literal: off
Rexmt-interval: 5 seconds, Max-timeout: 25 seconds
tftp> get team1_34MB
getting from 127.0.0.1:team1_34MB to team1_34MB [octet]
Received 35651584 bytes in 4.1 seconds [69606885 bit/s]
tftp> quit

[dhirajkudva]@hera3 ~/ECEN_602_git_new/socket_programming/Assignment_3_new/client> (01:36:
15 10/13/20)
:: diff team1_34MB ../team1_34MB

[dhirajkudva]@hera3 ~/ECEN_602_git_new/socket_programming/Assignment_3_new/client> (01:36:
```

Server output - (workaround highlighted)

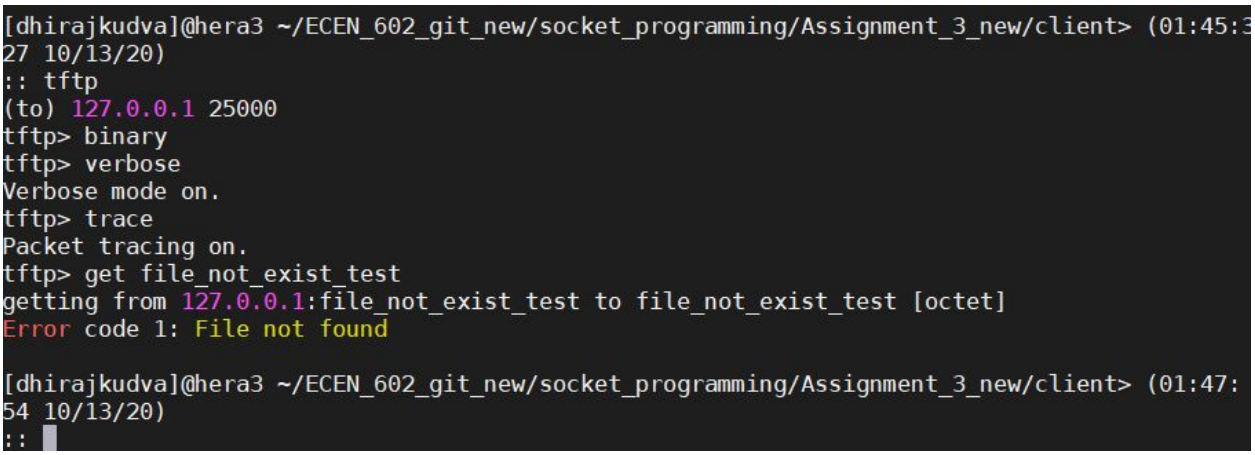
```
actual bytes read from file 512RECEIVED DATAAcknowledgement 65530 received
Expected Acknowledgement has arrived
actual bytes read from file 512RECEIVED DATAAcknowledgement 65531 received
Expected Acknowledgement has arrived
actual bytes read from file 512RECEIVED DATAAcknowledgement 65532 received
Expected Acknowledgement has arrived
actual bytes read from file 512RECEIVED DATAAcknowledgement 65533 received
Expected Acknowledgement has arrived
actual bytes read from file 512RECEIVED DATAAcknowledgement 65534 received
Expected Acknowledgement has arrived
actual bytes read from file 512RECEIVED DATAAcknowledgement 65535 received
Expected Acknowledgement has arrived
actual bytes read from file 512RECEIVED DATAAcknowledgement 0 received
Expected Acknowledgement has arrived
actual bytes read from file 512RECEIVED DATAAcknowledgement 1 received
Expected Acknowledgement has arrived
actual bytes read from file 512RECEIVED DATAAcknowledgement 2 received
Expected Acknowledgement has arrived
actual bytes read from file 512RECEIVED DATAAcknowledgement 3 received
Expected Acknowledgement has arrived
actual bytes read from file 512RECEIVED DATAAcknowledgement 4 received
Expected Acknowledgement has arrived
actual bytes read from file 512RECEIVED DATAAcknowledgement 5 received
Expected Acknowledgement has arrived
actual bytes read from file 512RECEIVED DATAAcknowledgement 6 received
Expected Acknowledgement has arrived
actual bytes read from file 512RECEIVED DATAAcknowledgement 7 received
Expected Acknowledgement has arrived
actual bytes read from file 512RECEIVED DATAAcknowledgement 8 received
Expected Acknowledgement has arrived
actual bytes read from file 512RECEIVED DATAAcknowledgement 9 received
Expected Acknowledgement has arrived
actual bytes read from file 512RECEIVED DATAAcknowledgement 10 received
```

5. check that you receive an error message if you try to transfer a file that does not exist and that your server cleans up and the child process exits

Here when we try to access a file that is not present in the server, it indicates an error message and child process is cleaned up.
Server output:

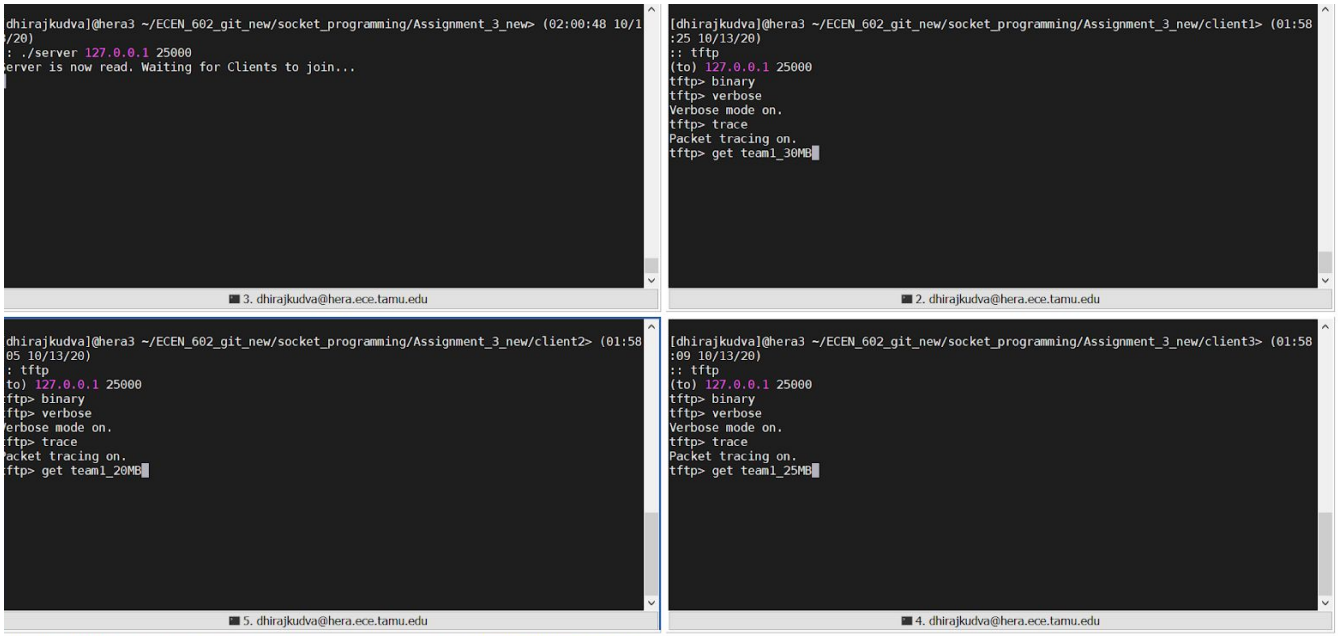


Client output:



6. Connect to the TFTP server with three clients simultaneously and test that the transfers work correctly (you will probably need a big file to have them all running at the same time),

In this case, three binary files are shared with three different clients simultaneously. And we ensure that the file is correct using diff command.



Server side output:

```
actual bytes read from file 512RECEIVED DATAAcknowledgement 45614 received
Expected Acknowledgement has arrived
actual bytes read from file 512RECEIVED DATAAcknowledgement 15563 received
Expected Acknowledgement has arrived
actual bytes read from file 512RECEIVED DATAAcknowledgement 45615 received
Expected Acknowledgement has arrived
actual bytes read from file 512RECEIVED DATAAcknowledgement 15564 received
Expected Acknowledgement has arrived
actual bytes read from file 512RECEIVED DATAAcknowledgement 45616 received
actual bytes read from file 512RECEIVED DATAAcknowledgement 15565 received
Expected Acknowledgement has arrived
Expected Acknowledgement has arrived
actual bytes read from file 512RECEIVED DATAAcknowledgement 15566 received
Expected Acknowledgement has arrived
actual bytes read from file 512RECEIVED DATAAcknowledgement 15567 received
Expected Acknowledgement has arrived
actual bytes read from file 512RECEIVED DATAAcknowledgement 45617 received
Expected Acknowledgement has arrived
actual bytes read from file 512RECEIVED DATAAcknowledgement 15568 received
```

The different acknowledgement number indicates that the server is communicating with different servers.

Client side output:
Checked that the requested file is same as the one on server using diff command on client.

CLIENT1:

```
:: tftp
(to) 127.0.0.1 25000
tftp> binary
tftp> verbose
Verbose mode on.
tftp> trace
Packet tracing on.
tftp> get team1_30MB
getting from 127.0.0.1:team1_30MB to team1_30MB [octet]
Received 31457280 bytes in 3.6 seconds [70626200 bit/s]
tftp> quit

[dhirajkudva]@hera3 ~/ECEN_602_git_new/socket_programming/Assignment_3_new/client1> (02:16
:04 10/13/20)
:: diff team1_30MB ../team1_30MB

[dhirajkudva]@hera3 ~/ECEN_602_git_new/socket_programming/Assignment_3_new/client1> (02:16
:17 10/13/20)
::
```

CLIENT2:

```
tftp> quit

[dhirajkudva]@hera3 ~/ECEN_602_git_new/socket_programming/Assignment_3_new/client2> (02:16
:56 10/13/20)
:: diff team1_20MB
diff: missing operand after 'team1_20MB'
diff: Try 'diff --help' for more information.

[dhirajkudva]@hera3 ~/ECEN_602_git_new/socket_programming/Assignment_3_new/client2> (02:17
:02 10/13/20)
:: diff team1_20MB ../team1_2
team1_20MB  team1_25MB

[dhirajkudva]@hera3 ~/ECEN_602_git_new/socket_programming/Assignment_3_new/client2> (02:17
:02 10/13/20)
:: diff team1_20MB ../team1_20MB

[dhirajkudva]@hera3 ~/ECEN_602_git_new/socket_programming/Assignment_3_new/client2> (02:17
:17 10/13/20)
::
```

CLIENT3:

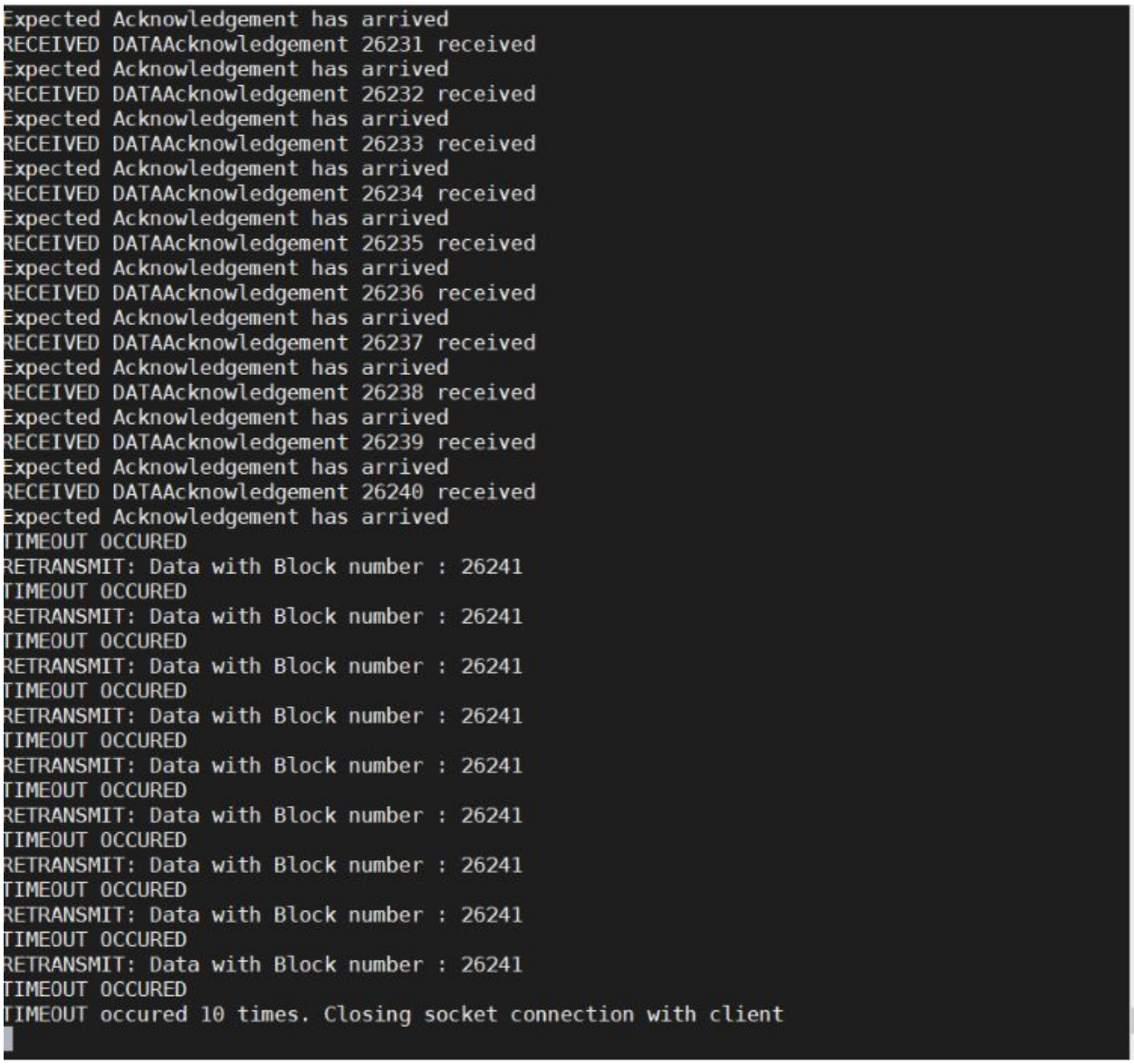
```
[dhirajkudva]@hera3 ~/ECEN_602_git_new/socket_programming/Assignment_3_new/client2> (02:17
:02 10/13/20)
:: diff team1_20MB ../team1_20MB
```

7. terminate the TFTP client in the middle of a transfer and see if your TFTP server recognizes after 10 timeouts that the client is no longer there

Client output:

```
[dhirajkudva]@hera3 ~/ECEN_602_git_new/socket_programming/Assignment_3_new/client3> (20:57
:03 10/11/20)
:: tftp
(to) 127.0.0.1 25000
tftp> status
Connected to 127.0.0.1.
Mode: netascii Verbose: off Tracing: off Literal: off
Rexmt-interval: 5 seconds, Max-timeout: 25 seconds
tftp> binary
tftp> verbose
Verbose mode on.
tftp> trace
Packet tracing on.
tftp> get team1_34MB
getting from 127.0.0.1:team1_34MB to team1_34MB [octet]

tftp>
```

Server output:



8. BONUS FEATURE

In this part WRQ is implemented.
Client sends binary and netascii file to the server and server again sends back the file to the client. To ensure that the client receives the same file, it is checked again on the client side.

Server output:

```
actual bytes read from file 512RECEIVED DATAAcknowledgement 2030 received
Expected Acknowledgement has arrived
actual bytes read from file 512RECEIVED DATAAcknowledgement 2031 received
Expected Acknowledgement has arrived
actual bytes read from file 512RECEIVED DATAAcknowledgement 2032 received
Expected Acknowledgement has arrived
actual bytes read from file 512RECEIVED DATAAcknowledgement 2033 received
Expected Acknowledgement has arrived
actual bytes read from file 512RECEIVED DATAAcknowledgement 2034 received
Expected Acknowledgement has arrived
actual bytes read from file 512RECEIVED DATAAcknowledgement 2035 received
Expected Acknowledgement has arrived
actual bytes read from file 512RECEIVED DATAAcknowledgement 2036 received
Expected Acknowledgement has arrived
actual bytes read from file 512RECEIVED DATAAcknowledgement 2037 received
Expected Acknowledgement has arrived
actual bytes read from file 512RECEIVED DATAAcknowledgement 2038 received
Expected Acknowledgement has arrived
actual bytes read from file 512RECEIVED DATAAcknowledgement 2039 received
Expected Acknowledgement has arrived
actual bytes read from file 512RECEIVED DATAAcknowledgement 2040 received
Expected Acknowledgement has arrived
actual bytes read from file 512RECEIVED DATAAcknowledgement 2041 received
Expected Acknowledgement has arrived
actual bytes read from file 512RECEIVED DATAAcknowledgement 2042 received
Expected Acknowledgement has arrived
actual bytes read from file 512RECEIVED DATAAcknowledgement 2043 received
Expected Acknowledgement has arrived
actual bytes read from file 512RECEIVED DATAAcknowledgement 2044 received
Expected Acknowledgement has arrived
actual bytes read from file 512RECEIVED DATAAcknowledgement 2045 received
Expected Acknowledgement has arrived
actual bytes read from file 512RECEIVED DATAAcknowledgement 2046 received
Expected Acknowledgement has arrived
actual bytes read from file 512RECEIVED DATAAcknowledgement 2047 received
Expected Acknowledgement has arrived
actual bytes read from file 512RECEIVED DATAAcknowledgement 2048 received
Expected Acknowledgement has arrived
actual bytes read from file 0RECEIVED DATAAcknowledgement 2049 received
Expected Acknowledgement has arrived
SERVER: Full file sent completely
```

Client output:

```
[dhirajkudva]@hera3 ~/ECEN_602_git_new/socket_programming/Assignment_3_new/client> (02:41:
01 10/13/20)
:: tftp
(to) 127.0.0.1 25000
tftp> binary
tftp> verbose
Verbose mode on.
tftp> trace
Packet tracing on.
tftp> put exchange client_to_server
putting exchange to 127.0.0.1:client_to_server [octet]
Sent 1048576 bytes in 0.2 seconds [54942787 bit/s]
tftp> get client_to_server server_to_client
getting from 127.0.0.1:client_to_server to server_to_client [octet]
Received 1048576 bytes in 0.2 seconds [39357618 bit/s]
```

```
tftp> quit

[dhirajkudva]@hera3 ~/ECEN_602_git_new/socket_programming/Assignment_3_new/client> (02:43:
28 10/13/20)
:: diff exchange server_to_client
```

CODE:

Server code:

```
/*********************************TFTP server implementation*************/
//Author: Sanket Agarwal and Dhiraj Kudva
//Organisation: Texas A&M university
//Usage: Used to handle the TFTP requests from the client end.

#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_SIZE 512

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

int timeout(int file_descriptor, int time){

       fd_set receive_set;
       struct timeval time_value;
       FD_ZERO(&receive_set);
       FD_SET (file_descriptor, &receive_set);
       time_value.tv_sec = time;
       time_value.tv_usec = 0;

       return (select (file_descriptor + 1, &receive_set, NULL, NULL, &time_value));
//timeout will be called with 1 sec as described in the pdf.


}

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

       char data_read[512] = {0}; //initialising it to zero.
       int socket_file_descriptor, new_socket_file_descriptor;
       struct sockaddr_in client;
       socklen_t length_client = sizeof(struct sockaddr_in);

       int g, return_get_addr;
       int send_response;
       int last;
```

```
       int return_value, receive_byte;
       char buffer[1024] = {0};
       char ack[32] = {0};
       char payload[516] = {0};
       char payload_copy[516] = {0};
       char file_name[MAX_SIZE];
       char mode[512];

       unsigned short int op_code1, op_code2, block_number;
       int b,j;//general variables used inside for loops.

       FILE *file_pointer;

       struct addrinfo address_dynamic, *addi, *client_info, *p;
       int true_yes;
       int pid;
       int read_return;
       int block_num=0;
       char ip[INET6_ADDRSTRLEN];
       int timeout_c = 0;
       int count =0;
       char c;
       char next_character = -1;
       int neg_ack = 0;
       char *ephemeral_pt;

       ephemeral_pt = malloc (sizeof ephemeral_pt);

       if (argc!=3)
       {
              system_error ("Incorrect USage: ./server <ip> <portnumber>");
              return 0;
       }

       true_yes = 1;
  //socklen_t addrlen;
  //configuring the address of the socket.
       memset(&address_dynamic, 0, sizeof address_dynamic);
  address_dynamic.ai_family   = AF_INET;
  address_dynamic.ai_socktype = SOCK_DGRAM;
  address_dynamic.ai_flags    = AI_PASSIVE;

  // get the address information and report error in case of issues with the acutal
issue.
   if ((return_get_addr = getaddrinfo(NULL, argv[2], &address_dynamic, &addi)) != 0) {
    fprintf(stderr, "error server: %s\n", gai_strerror(return_get_addr));
    exit(1);
  }

    for(p = addi; p != NULL; p = p->ai_next) {
    socket_file_descriptor = socket(p->ai_family, p->ai_socktype, p->ai_protocol);
    if (socket_file_descriptor < 0) {
      continue;//do not go to setup the socket.
    }
    setsockopt(socket_file_descriptor, SOL_SOCKET, SO_REUSEADDR, &true_yes,
sizeof(int));
    if (bind(socket_file_descriptor, p->ai_addr, p->ai_addrlen) < 0) {
      close(socket_file_descriptor); // close because of bind failure.
      continue;
    }
    break;// exit with the socket being binded.
  }
```

```
  freeaddrinfo(addi);// release the address.
  printf("Server is now read. Waiting for Clients to join...\n");

  while(1){
        //get data from the respective socket. and store it in a buffer.
        receive_byte = recvfrom(socket_file_descriptor, buffer, sizeof(buffer), 0,
(struct sockaddr*)&client, &length_client);

        //check if the number of recived bytes are zero or not.
      if(receive_byte<0){
            system_error("Error: receive bytes unsucessful");
            return 0;
        }
        //else bytes are received correctly proceed ahead.

      memcpy (&op_code1, &buffer, 2);
      op_code1 = ntohs(op_code1);

      pid = fork();//create the child.

      if(pid==0){//child process...


            if (op_code1 ==1 ){// RRQ process, do according.
                  //initialize to zero first.

                  bzero(file_name, MAX_SIZE);

                  // check for the EOF character using a for loop.

                  for (b=0; buffer[2+b]!='\0'; b++)
                  {
                        file_name[b] = buffer[2+b];
                  }
                  //once you get the file name add a eof to the file name. b is
already incremented to the position before exiting so can be added at the same
location

                  file_name[b] = '\0';
                  bzero(mode, 512);
                  g=0; // will be used to keep track of the mode counter.

                  for (j = b+3; buffer[j] != '\0'; j++) {    // reading until '\0'
of mode.
                        mode[g]=buffer[j];
                        g++;
                  }
                  mode[g]='\0';// same logic as the file_name mentioned above on
line 147
                  printf("RRQ: filename: %s mode: %s\n", file_name, mode);

                  // open the file now.

                  file_pointer = fopen(file_name, "r");

                  if (file_pointer!=NULL){//i.e. file is opened now.

                        close(socket_file_descriptor);// because we dont want to
resend on the same port. we are sending back on an ephemeral_pt
                        *ephemeral_pt = htons(0);
```

```
                        if ((return_get_addr = getaddrinfo(NULL, ephemeral_pt,
&address_dynamic, &client_info)) != 0) {
                                fprintf(stderr, "return_get_addr: %s\n",
gai_strerror(return_get_addr));
                                return 1;
                        }

                        for(p = client_info; p != NULL; p = p->ai_next) {
                                if ((new_socket_file_descriptor =
socket(p->ai_family, p->ai_socktype,p->ai_protocol)) == -1) {// same as the above one.
                                system_error("Error: SERVER (child): new socket not
created");
                                continue;
                                }


setsockopt(new_socket_file_descriptor,SOL_SOCKET,SO_REUSEADDR,&true_yes,sizeof(int));
                                if (bind(new_socket_file_descriptor, p->ai_addr,
p->ai_addrlen) == -1) {
                                        close(new_socket_file_descriptor);
                                        system_error("Error: SERVER
(new_socket_file_descriptor): bind issues.");
                                        continue;
                                }
                                break;
                        }
                        freeaddrinfo(client_info);// same as above.
                        //debug
                        //printf("Checkpoint1010: reached new socket creation.");

                        // now we need to create the datapacket to send back on the
newly created socket.

                        bzero(payload, sizeof(payload));//initialize to zero
initially.
                        bzero(data_read, sizeof(data_read));

                        //read from the file now.
                        read_return = fread(&data_read,1,512,file_pointer);

                        if (read_return>=0){
                                data_read[read_return] = '\0'; //same logic as above.
                                printf("read result is %d\n",read_return);
                        }

                        if(read_return < 512)
                                last = 1;

                        block_number = htons(1);                                //1st
block
                        op_code2 = htons(3);                                    // 3 is
data
                        memcpy(&payload[0], &op_code2, 2);              // 2 Bytes
                        memcpy(&payload[2], &block_number, 2);          // 4th Byte

                        //for (b = 0; data_read[b] != '\0'; b++) {
                        //      payload[b+4] = data_read[b];// concating the data.
                        //}

                        //bzero(payload_copy, sizeof(payload_copy)); // reset it to
zero.
                        memcpy(&payload[4], data_read, read_return); // copy all
the bytes.
```

```
                        send_response = sendto(new_socket_file_descriptor, payload,
(read_return + 4), 0, (struct sockaddr*)&client, length_client);

                        neg_ack = 1;
                        if (send_response < 0)
                               system_error("ERROR to send first packet: \n");
                        else
                               printf("First block successfully sent .\n");

                        while(1){
                               if(timeout(new_socket_file_descriptor,1)!= 0){
                                      bzero(buffer,sizeof(buffer));
                                      bzero(payload,sizeof(payload));
                                      receive_byte =
recvfrom(new_socket_file_descriptor, buffer, sizeof(buffer), 0, (struct
sockaddr*)&client,&length_client);
                                      timeout_c = 0;
                                      if(receive_byte < 0){
                                             system_error("ERROR: Data not
received");
                                             return 6;
                                      }
                                      else{
                                             printf("RECEIVED DATA");
                                      }
                                      memcpy(&op_code1, &buffer[0],2);
                                      if(ntohs(op_code1)==4){
       //OPCODE = 4; ACK
                                             bzero(&block_num, sizeof(block_num));
                                             memcpy(&block_num,&buffer[2],2);
                                             block_num = ntohs(block_num);
                                             printf("Acknowledgement %i
received\n",block_num);
                                             //EOF check
                                             if(block_num == neg_ack){
                                                    printf("Expected Acknowledgement
has arrived\n");
                                                    neg_ack = (neg_ack + 1)%65536;
                                                    if(last == 1){

close(new_socket_file_descriptor);
                                                           fclose(file_pointer);
                                                           printf("SERVER: Full file
sent completely \n");
                                                           exit(5);
                                                           last = 0;
                                                    }
                                                    else{

bzero(data_read,sizeof(data_read));
                                                           read_return = fread
(&data_read,1,512,file_pointer);
                                                           if(read_return >= 0){
                                                                  if(read_return <
512)
                                                                         last = 1;

data_read[read_return]='\0';
                                                           block_number =
htons(((block_num+1)%65536));
                                                           op_code2 =
htons(3);
```

```
memcpy(&payload[0],&op_code2,2);

memcpy(&payload[2],&block_number,2);


//for(b=0;data_read[b] != '\0';b++){
                                                                        //     payload[b+4]
= data_read[b];
                                                                        //}


//bzero(payload_copy,sizeof(payload_copy));
                                                                        printf("actual
bytes read from file %d",read_return);

memcpy(&payload[4],data_read, read_return);
                                                                        int send_response =
sendto(new_socket_file_descriptor,payload,(read_return+4),0,(struct sockaddr*)&client,
length_client);

                                                                        if(send_response <
0)

system_error("ERROR: send to");
                                                                }
                                                        }
                                                }
                                                else {
                                                        printf("ERROR: Expected ACK has
not arrived :NEG ACK = %d, Block number =%d\n", neg_ack,block_number);
                                                }
                                        }
                                }
                                else{
                                        timeout_c ++ ;
                                        printf("TIMEOUT OCCURED\n");
                                        if(timeout_c == 10){
                                                printf("TIMEOUT occured 10 times.
Closing socket connection with client\n");
                                                close(new_socket_file_descriptor);
                                                fclose(file_pointer);
                                                exit(6);
                                        }
                                        else{
                                                bzero(payload,sizeof(payload));

memcpy(&payload[0],&payload_copy[0],516);
                                                memcpy(&block_number,&payload[2],2);
                                                block_number = htons(block_number);
                                                printf("RETRANSMIT: Data with Block
number : %d\n",block_number );
                                                send_response =
sendto(new_socket_file_descriptor,payload_copy,(read_return+4),0,(struct
sockaddr*)&client,length_client);

bzero(payload_copy,sizeof(payload_copy));

memcpy(&payload_copy[0],&payload[0],516);

                                                if(send_response < 0)
                                                        system_error("ERROR: In sendto
");
```

```
                                        }
                                }
                        }
                }
                else {
                        unsigned short int ERROR_CODE = htons(1);
                        unsigned short int ERROR_OPCODE = htons(5);
                        char ERROR_MESSAGE[512] = "File not found";
                        char ERROR_BUFFER [516] = {0};
                        memcpy(&ERROR_BUFFER[0],&ERROR_OPCODE,2);
                        memcpy(&ERROR_BUFFER[2],&ERROR_CODE,2);
                        memcpy(&ERROR_BUFFER[4], &ERROR_MESSAGE, 512);
                        sendto(socket_file_descriptor,ERROR_BUFFER, 516, 0
,(struct sockaddr*)&client, length_client);
                        printf("SERVER CLEANUP: File names do not match \n");
                        close(socket_file_descriptor);
                        fclose(file_pointer);
                        exit(4);
                }
        }
        else if(op_code1 == 2){
                *ephemeral_pt = htons(0);
                if((return_get_addr = getaddrinfo(NULL, ephemeral_pt,
&address_dynamic, &client_info)) != 0){
                        fprintf(stderr, "getaddrinfo:
%s\n",gai_strerror(return_get_addr) );
                        return 10;
                }
                for (p = client_info; p != NULL; p = p->ai_next){
                        if((new_socket_file_descriptor = socket(p->ai_family,
p->ai_socktype, p->ai_protocol))== -1){
                                system_error("ERROR: SERVER: child socket");
                                continue;
                        }

setsockopt(new_socket_file_descriptor,SOL_SOCKET,SO_REUSEADDR,&true_yes, sizeof(int));
                        if(bind(new_socket_file_descriptor, p->ai_addr,
p->ai_addrlen) == -1){
                                close(new_socket_file_descriptor);
                                system_error("ERROR: SERVER: bind new
socket");
                                continue;
                        }
                        break;
                }
                freeaddrinfo(client_info);
                printf("WRQ: Request received to SERVER from client\n");
                                bzero(file_name, MAX_SIZE);

        // check for the EOF character using a for loop.

        for (b=0; buffer[2+b]!='\0'; b++)
        {
                file_name[b] = buffer[2+b];
        }
        //once you get the file name add a eof to the file name. b is
already incremented to the position before exiting so can be added at the same
location

        file_name[b] = '\0';
                printf("FILENAME : %s\n",file_name );
                FILE *file_pointer_write = fopen(file_name, "w+"); //check
this here
```

```
                        if(file_pointer_write == NULL)
                        {
                               printf("SERVER:WRQ: File cannot be opened\n");
                        }
                        op_code2 = htons(4);
                        block_number =  htons(0);
                        bzero(ack, sizeof(ack));
                        memcpy(&ack[0],&op_code2,2);
                        memcpy(&ack[2],&block_number, 2);
                        send_response = sendto(new_socket_file_descriptor, ack, 4,
0, (struct sockaddr*)&client, length_client);
                        neg_ack = 1;
                        if(send_response < 0)
                               system_error("ERROR: WRQ: ACK: sendto");
                        while(1){
                               bzero(buffer, sizeof(buffer));
                               receive_byte = recvfrom(new_socket_file_descriptor,
buffer, sizeof(buffer), 0, (struct sockaddr*)&client, &length_client);
                               if(receive_byte < 0){
                                      system_error("ERROR:WRQ: Data not received");
                                      return 9;
                               }
                               bzero(data_read, sizeof(data_read));
                               memcpy(&block_number,&buffer[2],2);
                               g=receive_byte;
                               for(b =0; g>0;g--){
                                      //if(buffer[b+4] == '\n'){
                                      //     printf("LF CHARACTER PRESENT\n");;
                                      //     g++;
                                      //     if(b-g < 0)
                                      //            printf("ERROR: b - g less than
zero\n");
                                      //     data_read[b-g] = '\n';
                                      //}
                                      //else{
                                             data_read[b] = buffer[b+4];
                                             b++;
                                      //}
                               }
                               fwrite(data_read,1, (receive_byte-
4),file_pointer_write);
                               block_number = ntohs(block_number);

                               if(neg_ack == block_number){
                                      printf("SERVER: DATA RECEiVED: BLOCK NUMBER
:%d\n",neg_ack );
                                      printf("SERVER: DATA EXPECTED received\n");
                                      op_code2 = htons(4);
                                      block_number = ntohs(neg_ack);
                                      bzero(ack,sizeof(ack));
                                      memcpy(&ack[0],&op_code2,2);
                                      memcpy(&ack[2],&block_number,2);
                                      printf("SERVER ACK SENT for block number
%d\n",htons(block_number) );
                                      send_response =
sendto(new_socket_file_descriptor,ack,4,0,(struct sockaddr*)&client, length_client);
                                      if(receive_byte < 516){
                                             printf("Last Data block has
arrived.Closing connection\n");
                                             close(new_socket_file_descriptor);
                                             fclose(file_pointer_write);
                                             exit(9);
```

```
                                        }
                                        neg_ack = (neg_ack + 1)%65536;
                                    }
                                }
                            }

                        }
                }

        }
```

MAKEFILE:

```
# ************************************* Make file code ********************
# Author: Sanket Agarwal & Dhiraj Kudva (agarwal.220196, dhirajkudva)@tamu.edu
#Organisation: Texas A&M University
#Description: Machine problem 3, Compiles server source code.



#for compiling server.c
sample : echos.o


echos.o: server.c
	gcc -I. server.c -o server
#clean to discard previous .o files
clean:
	rm -f sample *.o core
```