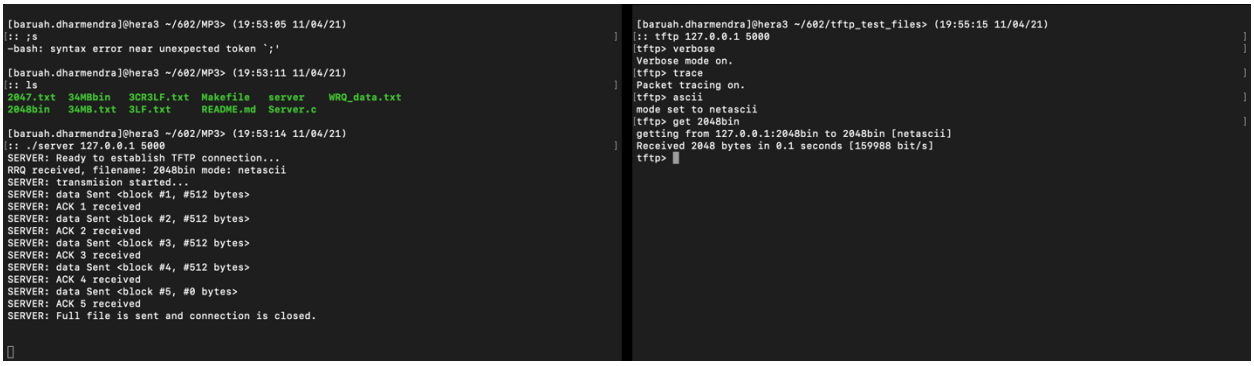
Machine Problem 3

Link to GitHub repository:
https://github.tamu.edu/baruah-dharmendra/ECEN602_Team04/tree/master/MP3

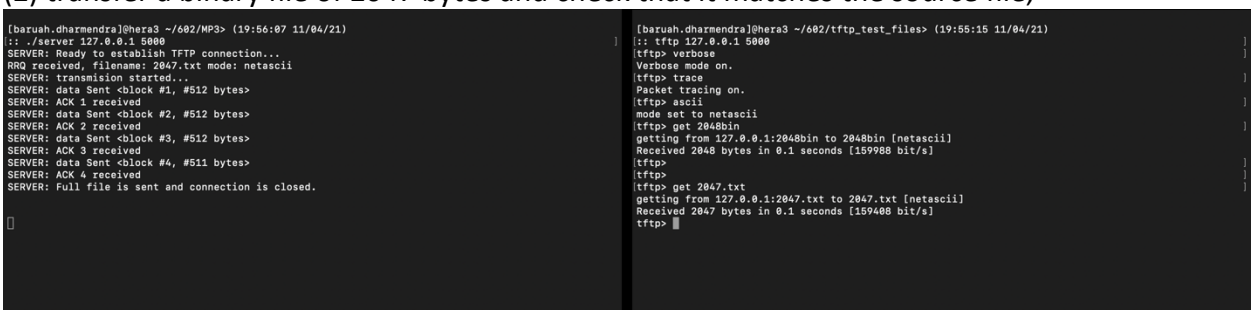
The readme file has all the implementation and test cases mentioned.

Test cases

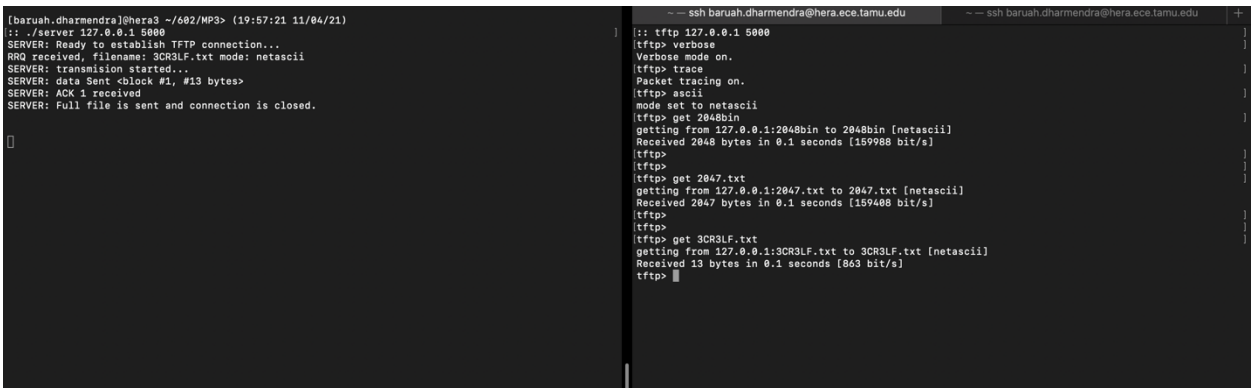
(1) transfer a binary file of 2048 bytes and check that it matches the source file,



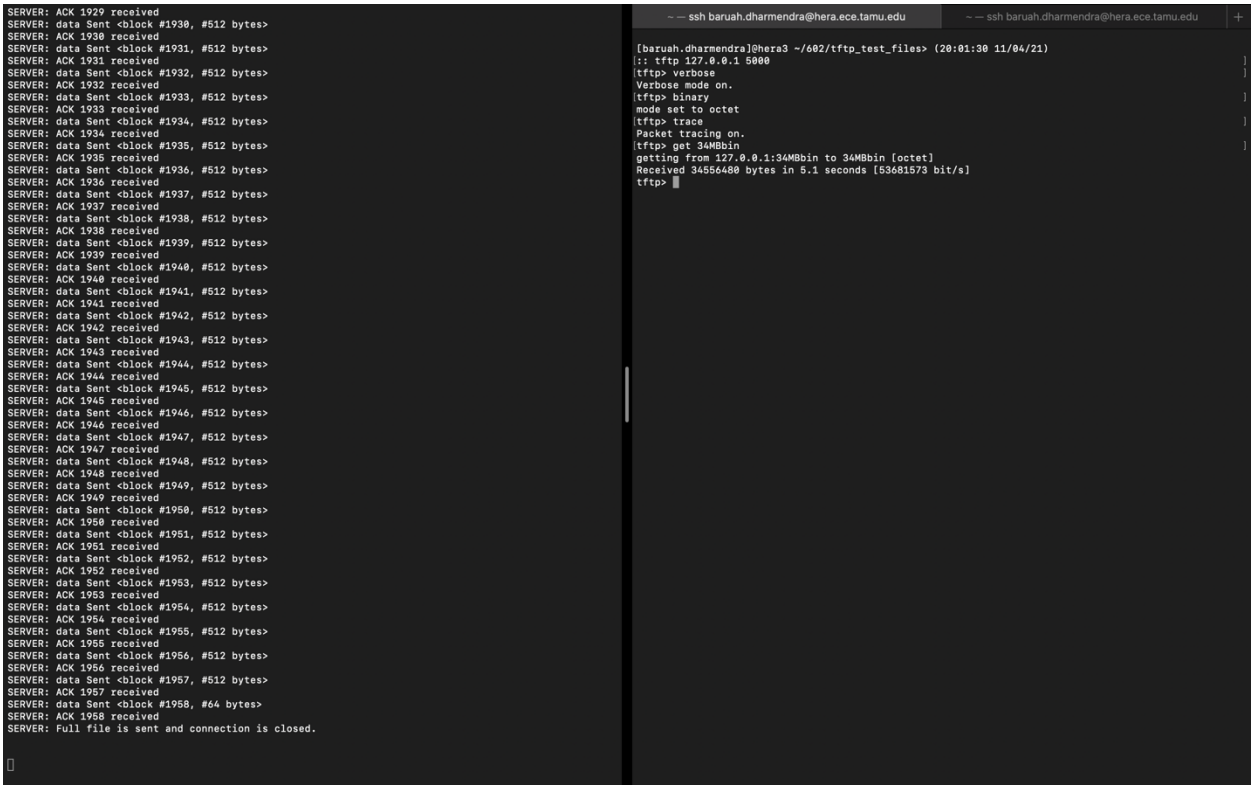
(2) transfer a binary file of 2047 bytes and check that it matches the source file,



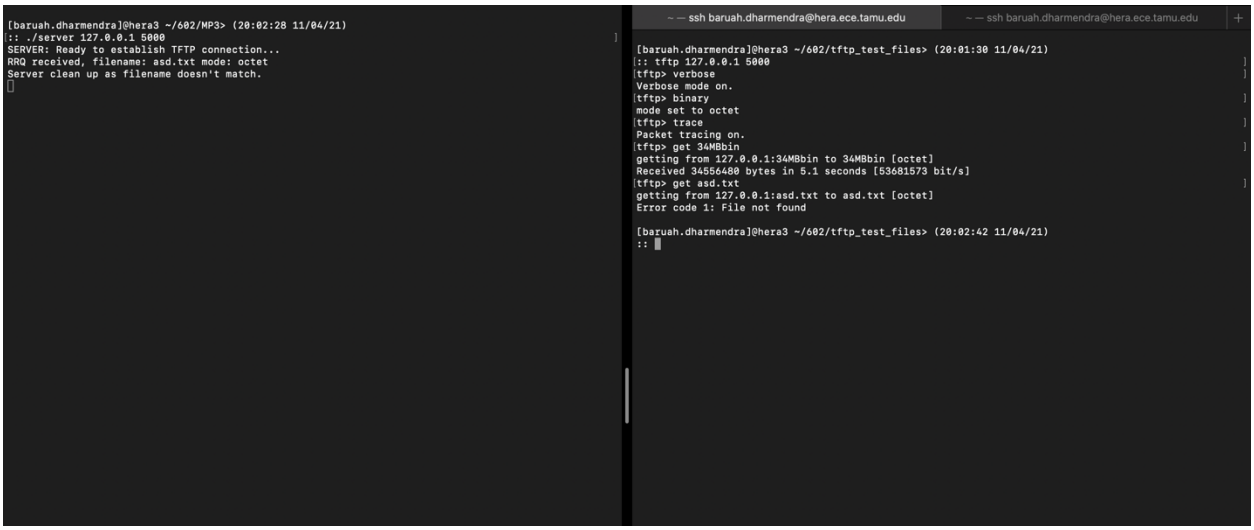
(3) transfer a netascii file that includes two CR,Äôs and check that the resulting file matches the input file,



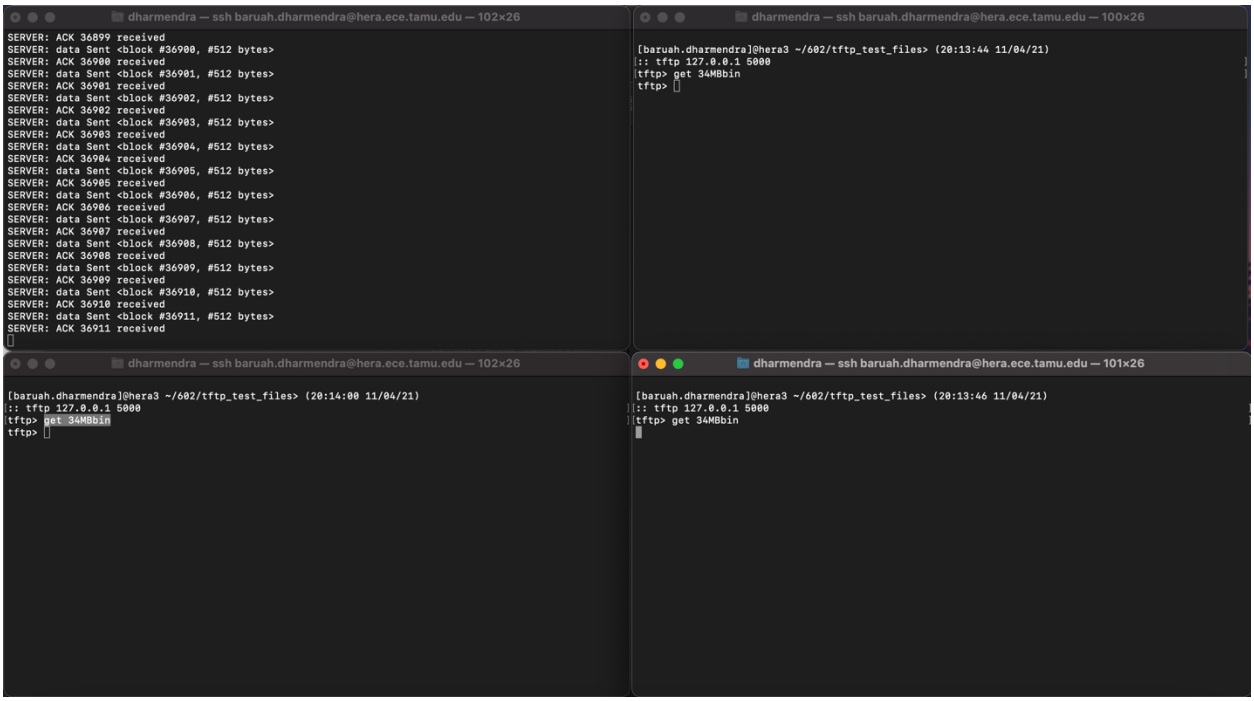
(4) transfer a binary file of 34 MB and see if block number wrap-around works,



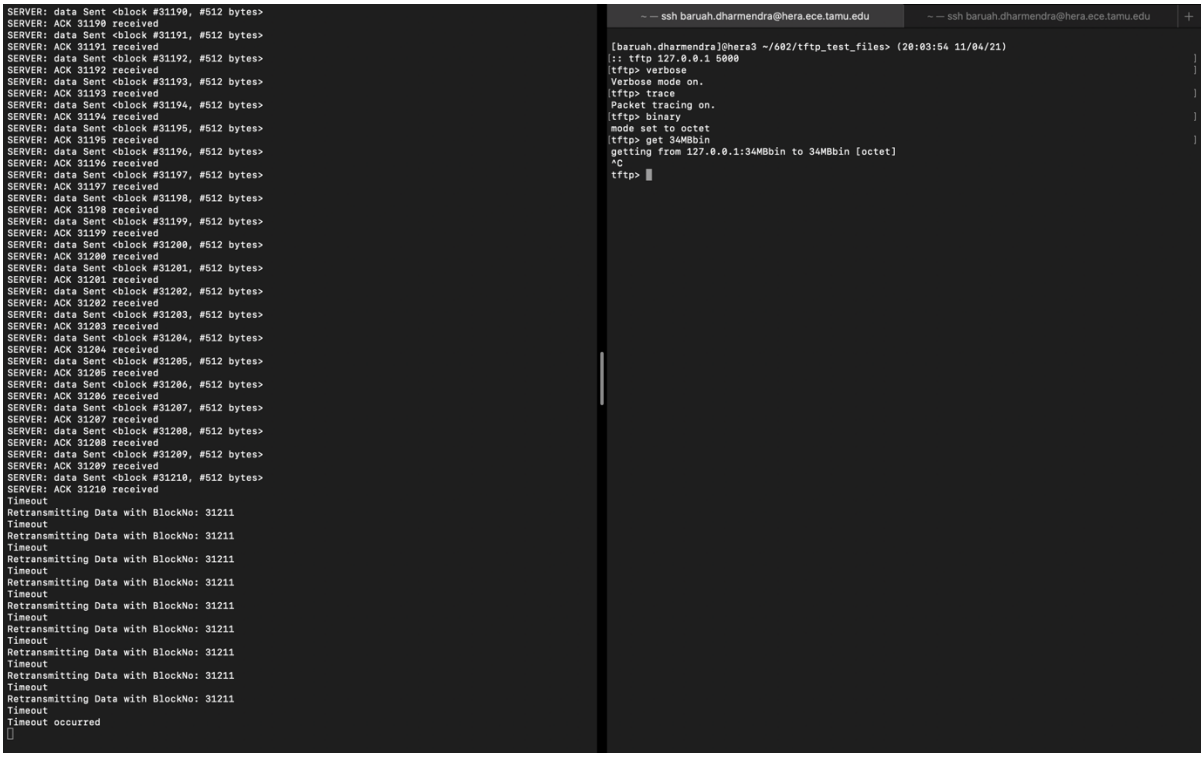
(5) check that you receive an error message if you try to transfer a file that does not exist and that your server cleans up and the child process exits,



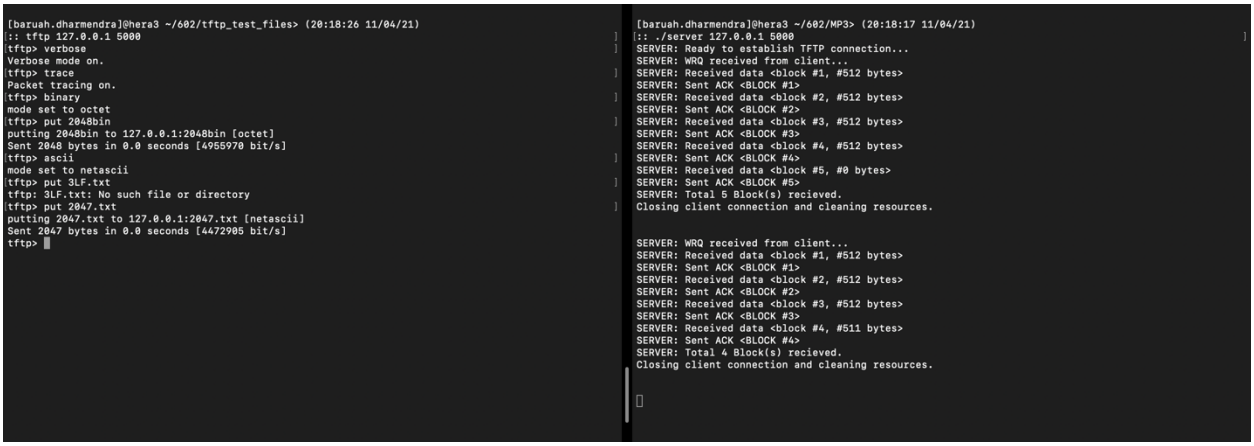
(6) Connect to the TFTP server with three clients simultaneously and test that the transfers work correctly (you will probably need a big file to have them all running at the same time),



(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 (you will need a big file),



(8) separate test cases for the bonus feature (see the Bonus Feature section).



Readme.md

```
# Trivial File Transfer Protocol (TFTP) server

## Purpose:

This Project is developed as a part of Machine Problem 3 of Computer
Networks and Communication course. It is performed as a team of two where
we are supposed to implement Trivial File Transfer Protocol (TFTP) server.

---

## Implementation:

 The Trivial File Transfer Protocol (TFTP) server performs the following
implementation:
1.  Start the server first with the command line: server IPAdr Port.
2.  Start the client second with a command line: tftp, than IPAdr Port.
3.  A client use get to receives a file using UDP server.
4.  Server and Clients  may  exit  unceremoniously  at  any time during
the transmition.
5.  The  server receive an error message if tring to transfer a file that
does not exist and that the server cleans up and the child process exits.

---
```

Running

Installation:

Clone this repository
```
git@github.tamu.edu:baruah-dharmendra/ECEN602_Team04.git
```

Building:

For this we will need standard C++ compiler installed in the machine in which the program is run. To build it can be directly done from the make file or individually by compiling the server client.

For building without the make file:

Build the server: ``` gcc -o Server server.c ```

For building with the makefile we can just use the command
```
make all
```
Execution:

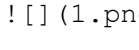
Open the terminal window and run: ``` ./Server 127.0.0.1 5000 ```

If the server response following should be visible in the terminal.
```
SERVER: Ready for association with clients...

```

Test cases

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

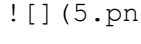


(2) transfer a binary file of 2047 bytes and check that it matches the source file,

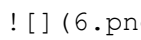
(3) transfer a netascii file that includes two CR,Äôs and check that the resulting file matches the input file,

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

(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,

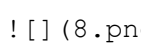


(6) Connect to the TFTP server with three clients simultaneously and test that the transfers work correctly (you will probably need a big file to have them all running at the same time),



(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 (you will need a big file), and

(8) separate test cases for the bonus feature (see the Bonus Feature section).



Team

@dharmendrabaruah
@yehtungchi

Effort

The entire project was completed with equal efforts from either member of the team maintaining synergy. It was carried out in the university library, where both of the members were responsible for the analysis, coding, debugging, testing and documentation of the server client application.

Server Code

```
#include <stdlib.h>
#include <string.h>
#include <netdb.h>
#include <errno.h>
#include <unistd.h>
```

```
#include <time.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/time.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <arpa/inet.h>

#define MAX_FN_SIZE 512


// display error of the source code with system error function
int err_sys(const char* x)
{
  perror(x);
  exit(1);
}

int timeout_indicator(int FD, int time_sec) {

  fd_set rset;
  struct timeval tv;
  FD_ZERO(&rset);
  FD_SET (FD, &rset);
  tv.tv_sec = time_sec;
  tv.tv_usec = 0;

  return (select (FD + 1, &rset, NULL, NULL, &tv));
}
// RRQ formatting of frame with opcode, filename, mode, etc
void RRQFormat(char buff[1024], char filename[512], char Mode[512]){
    int i, j;
    int modeIndex=0;
    for (i = 0; buff[2+i] != '\0'; i++) {        // Iterate till EOFn is
reached
        filename[i]=buff[2+i];
    }
    filename[i]='\0';
    bzero(Mode, 512);
    for (j = i+3; buff[j] != '\0'; j++) {    // Iterate till RRQ Mode is
read
        Mode[modeIndex]=buff[j];
        modeIndex++;
    }
    Mode[modeIndex]='\0';
    printf("RRQ received, filename: %s mode: %s\n", filename, Mode);
}

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

  int sockfd, newsock_fd;
  char data[512] = {0};
```

```
struct sockaddr_in client;
socklen_t clientlen = sizeof(struct sockaddr_in);
char *p0, *p1;
int i, LF_count, ret;
int send_res;
int last_block;
int NEACK_count = 0;

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

int ret_val, recbyte;
char buff [1024] = {0};
char ack_packet[32] = {0};
char file_payload[516] = {0};
char file_payload_copy[516] = {0};
char filename[MAX_FN_SIZE];
char Mode[512];
unsigned short int OC1, OC2, BlockNo;
int b, j;
FILE *fp;
struct addrinfo dynamic_addr, *ai, *clinfo, *p;
int yes;
int pid;
int read_ret;
int blocknum = 0;
char ips[INET6_ADDRSTRLEN];
int timeout_count = 0;
int count = 0;
char c;
char nextchar = -1;
int NEACK = 0;
char *ephemeral_port;

ephemeral_port = malloc (sizeof ephemeral_port);

yes = 1;
socklen_t addrlen;
memset(&dynamic_addr, 0, sizeof dynamic_addr);
dynamic_addr.ai_family   = AF_INET;
dynamic_addr.ai_socktype = SOCK_DGRAM;
dynamic_addr.ai_flags    = AI_PASSIVE;

  // setting up the server connection to the socket
if ((ret = getaddrinfo(NULL, argv[2], &dynamic_addr, &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;
  }

  freeaddrinfo(ai);
  printf("SERVER: Ready to establish TFTP connection...\n");

  while(1) {

      //recieve bytes from the client in buffer and return the size
    recbyte = recvfrom(sockfd, buff, sizeof(buff), 0, (struct
sockaddr*)&client, &clientlen);
    if (recbyte < 0) {
      err_sys("ERR: Couldn't receive data");
      return 7;
    }
    else {
    }
    memcpy(&OC1,&buff,2);
    OC1 = ntohs(OC1);
    pid = fork();
    if (pid == 0) {                              // Child process
      if (OC1 == 1){                             // RRQ processing
        bzero(filename, MAX_FN_SIZE);
          RRQFormat(buff, filename, Mode);
        // FIXME: Implement mode check (netascii (text files) | octet
(binary files))
        fp = fopen (filename, "r");
        if (fp != NULL) {                          // Valid Filename
          close(sockfd);
          // Create ephemeral socket for further datagram exchange
          *ephemeral_port = htons(0);
          if ((ret = getaddrinfo(NULL, ephemeral_port, &dynamic_addr,
&clinfo)) != 0) {
            fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(ret));
            return 10;
          }
          for(p = clinfo; p != NULL; p = p->ai_next) {
            if ((newsock_fd = socket(p->ai_family, p->ai_socktype,p-
>ai_protocol)) == -1) {
              err_sys("ERR: 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);
              err_sys("ERR: SERVER (newsock_fd): bind");
              continue;
```

```
            }
            break;
          }
          freeaddrinfo(clinfo);

          // Create data packet
          bzero(file_payload, sizeof(file_payload));
          bzero(data, sizeof(data));
               // Retrieve data from file pointer corresponding to
filename received in RRQ
          read_ret = fread (&data,1,512,fp);
          if(read_ret>=0) {
            data[read_ret]='\0';
            printf("",read_ret);
          }
          if(read_ret < 512)
            last_block = 1;
          BlockNo = htons(1);                              // 1st 512 Byte
block
          OC2 = htons(3);                                  // Opcode = 3:
Data
          memcpy(&file_payload[0], &OC2, 2);               // 2 Bytes
          memcpy(&file_payload[2], &BlockNo, 2);           // 4 Bytes
// 516 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, clientlen);
          NEACK = 1;
          if (send_res < 0)
            err_sys("couldn't send first packet: ");
          else
            printf("SERVER: transmision started...\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, &clientlen);
              timeout_count = 0;
              if (recbyte < 0) {
                err_sys("Couldn't receive data\n");
                return 6;
              }
              else {
              }
              memcpy(&OC1, &buff[0], 2);
```

```
              if (ntohs(OC1) == 4) {                                  //
Opcode = 4: ACK
                bzero(&blocknum, sizeof(blocknum));
                memcpy(&blocknum, &buff[2], 2);
                blocknum = ntohs(blocknum);
                printf("SERVER: data Sent <block #%d, #%d bytes>\n",
blocknum, read_ret);
                //check reach end of file
                if(blocknum == NEACK) {                          // Expected
ACK has arrived
                  printf("SERVER: ACK %i received\n",blocknum);
                  NEACK = (NEACK + 1)%65536;
              if(last_block == 1){
                    close(newsock_fd);
                    fclose(fp);
                printf("SERVER: Full file is sent and connection is
closed.\n\n\n");
                exit(5);
                last_block = 0;
              }
              else {
                    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) {
                      if(read_ret < 512)
                        last_block = 1;
                      data[read_ret]='\0';
                      BlockNo = htons(((blocknum+1)%65536));
                      OC2 = htons(3);
                      memcpy(&file_payload[0], &OC2, 2);
                      memcpy(&file_payload[2], &BlockNo, 2);
                      //memcpy(&file_payload[4], &data, 512);
                      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, clientlen);
                      if (send_res < 0)
                        err_sys("ERR: Sendto ");
                    }
                  }
                }
                else {
                  printf("Expected ACK hasn't arrived: NEACK: %d,
blocknum: %d\n", NEACK, blocknum);
                }
              }
            }
            else {
                timeout_count++;
```

```
                                printf("Timeout\n");
                                if (timeout_count == 10){
                                  printf("Timeout occurred\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, clientlen);
                  bzero(file_payload_copy, sizeof(file_payload_copy));
                  memcpy(&file_payload_copy[0], &file_payload[0], 516);
                  if (send_res < 0)
                    err_sys("ERR: Sendto ");
                }
              }
            }
          }
          else {                                                // Generate ERR
message and send to client if file is not found
            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,
clientlen);       // FIXME: Confirm port number
            printf("Server clean up as filename doesn't match.\n");
            close(sockfd);
            fclose(fp);
            exit(4);
          }
        }
        else if (OC1 == 2) {                                  // WRQ
processing
          *ephemeral_port = htons(0);
          if ((ret = getaddrinfo(NULL, ephemeral_port, &dynamic_addr,
&clinfo)) != 0) {
            fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(ret));
            return 10;
          }
          for(p = clinfo; p != NULL; p = p->ai_next) {
            if ((newsock_fd = socket(p->ai_family, p->ai_socktype,p-
>ai_protocol)) == -1) {
              err_sys("ERR: 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);
            err_sys("ERR: SERVER (newsock_fd): bind");
            continue;
          }
          break;
        }
        freeaddrinfo(clinfo);
    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");
        OC2 = htons(4);
    BlockNo = htons (0);
        bzero(ack_packet, sizeof(ack_packet));
        memcpy(&ack_packet[0], &OC2, 2);
        memcpy(&ack_packet[2], &BlockNo, 2);
        send_res = sendto(newsock_fd, ack_packet, 4, 0, (struct
sockaddr*)&client, clientlen);
    NEACK = 1;
        if (send_res < 0)
          err_sys("WRQ ACK ERR: Sendto ");
    while(1){
          bzero(buff, sizeof(buff));
          recbyte = recvfrom(newsock_fd, buff, sizeof(buff), 0, (struct
sockaddr*)&client, &clientlen);
          if (recbyte < 0) {
            err_sys("WRQ: Couldn't receive data\n");
            return 9;
          }
          bzero(data, sizeof(data));
          memcpy(&BlockNo, &buff[2], 2);
      LF_count = 0;
          for (b = 0; buff[b+4] != '\0'; b++) {
         if (buff[b+4] == '\n'){
          //printf("The data contains LF character.\n");
          LF_count++;
          if (b-LF_count<0)
            printf("ERR: buffer overflow");
          data[b-LF_count] = '\n';
            }
        else
              data[b - LF_count] = buff[b+4];
          }
      fwrite(data, 1, (recbyte - 4 - LF_count), fp_wr);
      BlockNo = ntohs(BlockNo);
      if (NEACK == BlockNo){
          NEACK_count++;
        printf("SERVER: Received data <block #%d, #%d bytes>\n",
NEACK,recbyte-4);
            OC2 = htons(4);
```

```
        BlockNo = ntohs(NEACK);
            bzero(ack_packet, sizeof(ack_packet));
            memcpy(&ack_packet[0], &OC2, 2);
            memcpy(&ack_packet[2], &BlockNo, 2);
        printf("SERVER: Sent ACK <BLOCK #%d>\n", htons(BlockNo));
            send_res = sendto(newsock_fd, ack_packet, 4, 0, (struct
sockaddr*)&client, clientlen);
        if (recbyte < 516) {
              printf("SERVER: Total %d Block(s) recieved. \nClosing client
connection and cleaning resources. \n\n\n",NEACK_count);
              NEACK_count = 0;
              close(newsock_fd);
              fclose(fp_wr);
              exit(9);
        }
            NEACK = (NEACK + 1)%65536;
      }
    }
      }
    }
  }
}
```