

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,

```
[baruah.dharmendra]@here3 ~ -/602/MP3> (19:53:05 11/04/21)
!! :s
-bash: syntax error near unexpected token `!'

[baruah.dharmendra]@here3 ~ -/602/MP3> (19:53:11 11/04/21)
!! :s
2047.txt 34MBin 3C831F.txt Makefile server WRQ_data.txt
2048Bin 34MB.txt 31F.txt README.md Server.c

[baruah.dharmendra]@here3 ~ -/602/MP3> (19:53:14 11/04/21)
!! :/server 127.0.0.1 5000
SERVER: Ready to establish TFTP connection...
RRQ received, filename: 2048Bin.md, netascii
SERVER: transmission started...
SERVER: data Sent <clock #1, #512 bytes>
SERVER: ACK 1 received
SERVER: data Sent <clock #2, #512 bytes>
SERVER: ACK 2 received
SERVER: data Sent <clock #3, #512 bytes>
SERVER: ACK 3 received
SERVER: data Sent <clock #4, #512 bytes>
SERVER: ACK 4 received
SERVER: data Sent <clock #5, #0 bytes>
SERVER: ACK 5 received
SERVER: Full file is sent and connection is closed.
```

```
[baruah.dharmendra]@shera3 ~/.602/ftftp_test_files> (19:55:15 11/04/21)
[!]: fftp 127.0.0.1 5800
ftftp verbose
Verbose mode on.
ftftp trace
Packet tracing on.
ftftp ascii
mode set to netascii
[ftftp] get 2848bin
getting from 127.0.0.1:2848bin to 2848bin [netascii]
Received 2848 bytes in 0.1 seconds [159980 bits/s]
ftftp: #
```

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

```
[baruah.dharma@red3]# ./602/MP3: (19:56:87 11/04/21)
... ./server 127.0.0.1 5000
SERVER: Ready to establish TFTP connection...
RRQ received, filename: 2047.txt mode: netascii
SERVER: transmission started
SERVER: data Sent <block #1, #512 bytes>
SERVER: ACK 1 received
SERVER: data Sent <block #2, #512 bytes>
SERVER: ACK 2 received
SERVER: data Sent <block #3, #512 bytes>
SERVER: ACK 3 received
SERVER: data Sent <block #4, #511 bytes>
SERVER: ACK 4 received
SERVER: Full file is sent and connection is closed.
```

```
[barush.dharmendra@heara3 ~]#082/(ftftp_test_files) (19:55:15 11/04/21)
[ : tftp 127.0.0.1 5000
tftp> verbose
Verbose mode on.
tftp> trace
Packet tracing on.
tftp> ascii
mode set to netascii
tftp> get 2048bin
getting from 127.0.0.1:2048bin to 2048bin [netascii]
Received 2048 bytes in 0.1 seconds [159988 bit/s]
tftp>
tftp>
tftp> get 2047.txt
getting from 127.0.0.1:2047.txt to 2047.txt [netascii]
Received 2047 bytes in 0.1 seconds [159408 bit/s]
tftp> █
```

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

```
[baruah.dharmendra]@hara3 ~/602/MP3> (19:57:21 11/04/21)
[::] ./server 127.0.0.1 5000
SERVER: Ready to establish TFTP connection...
RRQ received, filename: 3CR3LF.txt mode: netascii
SERVER: transmission started...
SERVER: data Sent <block #1, #13 bytes>
SERVER: ACK 1 received
SERVER: Full file is sent and connection is closed.
```

```
% ssh baruah.dharmendra@hera.ecs.tamu.edu  
[~]$  
[~]: tftp 127.0.0.1 5000  
tftp> verbose  
Verbose mode on.  
tftp> trace  
Packet tracing on.  
tftp> ascii  
mode set to netascii  
tftp> get 2048bin  
getting from 127.0.0.1:2048bin to 2048bin [netascii]  
Received 2048 bytes in 0.1 seconds (169988 bit/s)  
tftp>  
tftp>  
tftp> get 2047.txt  
getting from 127.0.0.1:2047.txt to 2047.txt [netascii]  
Received 2047 bytes in 0.1 seconds (169948 bit/s)  
tftp>  
tftp>  
tftp> get 3C3R3F.txt  
getting from 127.0.0.1:3C3R3F.txt to 3C3R3F.txt [netascii]  
Received 13 bytes in 0.1 seconds (863 bit/s)  
tftp>
```

```

SERVER: ACK 1929 received
SERVER: data Sent <block #1930, #512 bytes>
SERVER: ACK 1930 received
SERVER: data Sent <block #1931, #512 bytes>
SERVER: ACK 1931 received
SERVER: data Sent <block #1932, #512 bytes>
SERVER: ACK 1932 received
SERVER: data Sent <block #1933, #512 bytes>
SERVER: ACK 1933 received
SERVER: data Sent <block #1934, #512 bytes>
SERVER: ACK 1934 received
SERVER: data Sent <block #1935, #512 bytes>
SERVER: ACK 1935 received
SERVER: data Sent <block #1936, #512 bytes>
SERVER: ACK 1936 received
SERVER: data Sent <block #1937, #512 bytes>
SERVER: ACK 1937 received
SERVER: data Sent <block #1938, #512 bytes>
SERVER: ACK 1938 received
SERVER: data Sent <block #1939, #512 bytes>
SERVER: ACK 1939 received
SERVER: data Sent <block #1940, #512 bytes>
SERVER: ACK 1940 received
SERVER: data Sent <block #1941, #512 bytes>
SERVER: ACK 1941 received
SERVER: data Sent <block #1942, #512 bytes>
SERVER: ACK 1942 received
SERVER: data Sent <block #1943, #512 bytes>
SERVER: ACK 1943 received
SERVER: data Sent <block #1944, #512 bytes>
SERVER: ACK 1944 received
SERVER: data Sent <block #1945, #512 bytes>
SERVER: ACK 1945 received
SERVER: data Sent <block #1946, #512 bytes>
SERVER: ACK 1946 received
SERVER: data Sent <block #1947, #512 bytes>
SERVER: ACK 1947 received
SERVER: data Sent <block #1948, #512 bytes>
SERVER: ACK 1948 received
SERVER: data Sent <block #1949, #512 bytes>
SERVER: ACK 1949 received
SERVER: data Sent <block #1950, #512 bytes>
SERVER: ACK 1950 received
SERVER: data Sent <block #1951, #512 bytes>
SERVER: ACK 1951 received
SERVER: data Sent <block #1952, #512 bytes>
SERVER: ACK 1952 received
SERVER: data Sent <block #1953, #512 bytes>
SERVER: ACK 1953 received
SERVER: data Sent <block #1954, #512 bytes>
SERVER: ACK 1954 received
SERVER: data Sent <block #1955, #512 bytes>
SERVER: ACK 1955 received
SERVER: data Sent <block #1956, #512 bytes>
SERVER: ACK 1956 received
SERVER: data Sent <block #1957, #512 bytes>
SERVER: ACK 1957 received
SERVER: data Sent <block #1958, #64 bytes>
SERVER: ACK 1958 received
SERVER: Full file is sent and connection is closed.

```

```
[baruah.dharmendra]@hera3 ~:/602/MP3> (20:02:28 11/04/21)
[:: ./server 127.0.0.1 5000
SERVER: Ready to establish TFTP connection...
R00 received, filename: asd.txt mode: octet
Server clean up as filename doesn't match.
]
```

```
-- ssh baruah.dharmendra@hera.ece.tamu.edu
[baruah.dharmendra]@hera3 ~:/602/tftp_test_files> (20:01:30 11/04/21)
[:: tftp 127.0.0.1 5000
tftp> verbose
Verbose mode on.
tftp> binary
mode set to octet
tftp> trace
Packet tracing on.
tftp> get 24MBbin
getting from 127.0.0.1:34MBbin to 34MBbin [octet]
Received 34556480 bytes in 5.1 seconds [53681573 bit/s]
tftp> get asd.txt
getting from 127.0.0.1:asd.txt to asd.txt [octet]
Error code 1: File not found

[baruah.dharmendra]@hera3 ~:/602/tftp_test_files> (20:02:42 11/04/21)
[:: ]
```

(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),

The image displays four terminal windows arranged in a 2x2 grid, illustrating the process of downloading a file using the TFTP protocol. Each window has a title bar indicating the user 'dharmendra' and the remote host 'ssh baruah.dharmendra@hera.ece.tamu.edu'.

- Top-Left Window (Title: 102x26):** Shows a series of server acknowledgments and data sent messages. The messages are: 'SERVER: ACK 36899 received', 'SERVER: data Sent <block #36900, #512 bytes>', 'SERVER: ACK 36900 received', 'SERVER: data Sent <block #36901, #512 bytes>', 'SERVER: ACK 36901 received', 'SERVER: data Sent <block #36902, #512 bytes>', 'SERVER: ACK 36902 received', 'SERVER: data Sent <block #36903, #512 bytes>', 'SERVER: ACK 36903 received', 'SERVER: data Sent <block #36904, #512 bytes>', 'SERVER: ACK 36904 received', 'SERVER: data Sent <block #36905, #512 bytes>', 'SERVER: ACK 36905 received', 'SERVER: data Sent <block #36906, #512 bytes>', 'SERVER: ACK 36906 received', 'SERVER: data Sent <block #36907, #512 bytes>', 'SERVER: ACK 36907 received', 'SERVER: data Sent <block #36908, #512 bytes>', 'SERVER: ACK 36908 received', 'SERVER: data Sent <block #36909, #512 bytes>', 'SERVER: ACK 36909 received', 'SERVER: data Sent <block #36910, #512 bytes>', 'SERVER: ACK 36910 received', 'SERVER: data Sent <block #36911, #512 bytes>', and 'SERVER: ACK 36911 received'. The prompt is at the end of the last line.
- Top-Right Window (Title: 100x26):** Shows the user's command sequence: '[baruah.dharmendra]@hera3 ~/602/tftp_test_files> (20:13:44 11/04/21)', ';; tftp 127.0.0.1 5000', 'tftp> get 34MBbin', and 'tftp>'. The prompt is at the end of the last line.
- Bottom-Left Window (Title: 102x26):** Shows the user's command sequence: '[baruah.dharmendra]@hera3 ~/602/tftp_test_files> (20:14:00 11/04/21)', ';; tftp 127.0.0.1 5000', 'tftp> get 34MBbin', and 'tftp>'. The prompt is at the end of the last line.
- Bottom-Right Window (Title: 101x26):** Shows the user's command sequence: '[baruah.dharmendra]@hera3 ~/602/tftp_test_files> (20:13:46 11/04/21)', ';; tftp 127.0.0.1 5000', 'tftp> get 34MBbin', and an empty line. The prompt is at the end of the last line.

(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),

```
SERVER: data Sent <block #31198, #512 bytes>  
SERVER: ACK 31198 received  
SERVER: data Sent <block #31197, #512 bytes>  
SERVER: ACK 31197 received  
SERVER: data Sent <block #31192, #512 bytes>  
SERVER: ACK 31192 received  
SERVER: data Sent <block #31193, #512 bytes>  
SERVER: ACK 31193 received  
SERVER: data Sent <block #31194, #512 bytes>  
SERVER: ACK 31194 received  
SERVER: data Sent <block #31195, #512 bytes>  
SERVER: ACK 31195 received  
SERVER: data Sent <block #31196, #512 bytes>  
SERVER: ACK 31196 received  
SERVER: data Sent <block #31197, #512 bytes>  
SERVER: ACK 31197 received  
SERVER: data Sent <block #31198, #512 bytes>  
SERVER: ACK 31198 received  
SERVER: data Sent <block #31199, #512 bytes>  
SERVER: ACK 31199 received  
SERVER: data Sent <block #31200, #512 bytes>  
SERVER: ACK 31200 received  
SERVER: data Sent <block #31201, #512 bytes>  
SERVER: ACK 31201 received  
SERVER: data Sent <block #31202, #512 bytes>  
SERVER: ACK 31202 received  
SERVER: data Sent <block #31203, #512 bytes>  
SERVER: ACK 31203 received  
SERVER: data Sent <block #31204, #512 bytes>  
SERVER: ACK 31204 received  
SERVER: data Sent <block #31205, #512 bytes>  
SERVER: ACK 31205 received  
SERVER: data Sent <block #31206, #512 bytes>  
SERVER: ACK 31206 received  
SERVER: data Sent <block #31207, #512 bytes>  
SERVER: ACK 31207 received  
SERVER: data Sent <block #31208, #512 bytes>  
SERVER: ACK 31208 received  
SERVER: data Sent <block #31209, #512 bytes>  
SERVER: ACK 31209 received  
SERVER: data Sent <block #31210, #512 bytes>  
SERVER: ACK 31210 received  
  
Timeout  
Retransmitting Data with BlockNo: 31211  
Timeout  
Retransmitting Data with BlockNo: 31211  
Timeout  
Retransmitting Data with BlockNo: 31211  
Timeout  
Retransmitting Data with BlockNo: 31211  
Timeout  
Retransmitting Data with BlockNo: 31211  
Timeout  
Retransmitting Data with BlockNo: 31211  
Timeout  
Retransmitting Data with BlockNo: 31211  
Timeout  
Retransmitting Data with BlockNo: 31211  
Timeout occurred
```

```
-- ssh baruah.dharmendra@hera.ece.tamu.edu +  
[baruah.dharmendra@hera3 ~/#02/tftp_test_files] (20:03:54 11/04/21)  
[[:: tftp 127.0.0.1 5000  
tftp> verbose  
Verbose mode on.  
tftp> trace  
Packet tracing on.  
tftp> binary  
mode set to octet  
tftp> get 34MBbin  
getting from 127.0.0.1:34MBbin to 34MBbin [octet]  
^C  
tftp>
```

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

```
[baruah.dharmendra]@hera3 ~/602/tftp_test_files> (20:18:26 11/04/21)
[:: tftp 127.0.0.1 6000
tftp> verbose
Verbose mode on.
tftp> trace
Packet tracing on.
tftp> binary
mode set to octet
tftp> put 2048bin
putting 2048bin to 127.0.0.1:2048bin [octet]
Sent 2048 bytes in 0.0 seconds [4955970 bit/s]
tftp> ascii
mode set to netascii
tftp> put 3LF.txt
tftp: 3LF.txt: No such file or directory
tftp> put 2047.txt
putting 2047.txt to 127.0.0.1:2047.txt [netascii]
Sent 2047 bytes in 0.0 seconds [4472905 bit/s]
tftp> ]

[baruah.dharmendra]@hera3 ~/602/MP3> (20:18:17 11/04/21)
[:: ./server 127.0.0.1 6000
SERVER: Ready to establish TFTP connection...
SERVER: WRQ received from client...
SERVER: Received data <block #1, #512 bytes>
SERVER: Sent ACK <BLOCK #1>
SERVER: Received data <block #2, #512 bytes>
SERVER: Sent ACK <BLOCK #2>
SERVER: Received data <block #3, #512 bytes>
SERVER: Sent ACK <BLOCK #3>
SERVER: Received data <block #4, #512 bytes>
SERVER: Sent ACK <BLOCK #4>
SERVER: Received data <block #5, #0 bytes>
SERVER: Sent ACK <BLOCK #5>
SERVER: Total 5 Block(s) recieved.
Closing client connection and cleaning resources.

SERVER: WRQ received from client...
SERVER: Received data <block #1, #512 bytes>
SERVER: Sent ACK <BLOCK #1>
SERVER: Received data <block #2, #512 bytes>
SERVER: Sent ACK <BLOCK #2>
SERVER: Received data <block #3, #512 bytes>
SERVER: Sent ACK <BLOCK #3>
SERVER: Received data <block #4, #511 bytes>
SERVER: Sent ACK <BLOCK #4>
SERVER: Total 4 Block(s) recieved.
Closing client connection and cleaning resources.

]
```

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 uncereemoniously at any time during the transmtion.
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,Ãs 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,

![] (4.png)

(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,

![] (5.png)

(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),

![] (6.png)

(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

![] (7.png)

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

![] (8.png)

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);
block
    OC2 = htons(3);
Data
    memcpy(&file_payload[0], &OC2, 2);
    memcpy(&file_payload[2], &BlockNo, 2);
// 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: transmission 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) {
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. \nCclosing 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;
    }
}
}
}
}
}
}
}

```