```c
// Simple tftp client
// CPE 3300, Daniel Nimsgern
//
// Honestly I had a really hard time with this lab. Every time I tried to
// implement something there were several issues. The main hiccup I ran into
// was missing bytes and repeated packets which was fixed by accounting for the
// length of the header throughout the program and the ACK starts at 1 not 0.
//
// Build with gcc -o tftpclient tftpclient.c

/*=============================================================================
 |            Includes
 =============================================================================*/
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/time.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <bits/getopt_core.h>
#include <netinet/in.h>
#include <arpa/inet.h>

/*=============================================================================
 |            Global Variables
 =============================================================================*/

/* CLI ESC Codes */
#define ESC_BLACK_TXT        (char*)        "\033[1;30m"
#define ESC_RED_TXT          (char*)        "\033[1;31m"
#define ESC_GREEN_TXT        (char*)        "\033[1;32m"
#define ESC_YELLOW_TXT       (char*)        "\033[1;33m"
#define ESC_BLUE_TXT         (char*)        "\033[1;34m"
#define ESC_MAGENTA_TXT      (char*)        "\033[1;35m"
#define ESC_CYAN_TXT         (char*)        "\033[1;36m"
#define ESC_WHITE_TXT        (char*)        "\033[1;37m"
#define ESC_BR_GRAY_TXT      (char*)        "\033[1;90m"
#define ESC_BR_RED_TXT       (char*)        "\033[1;91m"
#define ESC_BR_GREEN_TXT     (char*)        "\033[1;92m"
#define ESC_BR_YELLOW_TXT    (char*)        "\033[1;93m"
#define ESC_BR_BLUE_TXT      (char*)        "\033[1;94m"
#define ESC_BR_MAGENTA_TXT   (char*)        "\033[1;95m"
#define ESC_BR_CYAN_TXT      (char*)        "\033[1;96m"
#define ESC_BR_WHITE_TXT     (char*)        "\033[1;97m"

/* Network Defines */
#define LAB_BROADCAST        (in_addr_t)       0xC0A818FF
#define HOME_BROADCAST       (in_addr_t)       0xC0A801FF
#define DEFAULT_TFTP_PORT    (unsigned short)  69
#define SOCK_TIMEOUT_US      (int)             5000000

/* TFTP Defines */
#define MAX_FILE_NAME        (int)          255
#define MAX_BLOCKS           (int)          65535
#define MAX_BLOCK_SIZE       (int)          512
#define TFTP_MODE            (char*)        "octet"

#define TFTP_RRQ             (uint8_t)      1
#define TFTP_WRQ             (uint8_t)      2
#define TFTP_DATA            (uint8_t)      3
```

```c
#define TFTP_ACK             (uint8_t)      4
#define TFTP_ERROR           (uint8_t)      5
#define TFTP_ERROR_NOTDIFF   (uint8_t)      0
#define TFTP_ERROR_FNF       (uint8_t)      1
#define TFTP_ERROR_ACCVIO    (uint8_t)      2
#define TFTP_ERROR_DSKFULL   (uint8_t)      3
#define TFTP_ERROR_ILLTFTP   (uint8_t)      4
#define TFTP_ERROR_UNID      (uint8_t)      5
#define TFTP_ERROR_FALEX     (uint8_t)      6
#define TFTP_ERROR_NOUSR     (uint8_t)      7

/*=============================================================================
 |            Function Definitions
 =============================================================================*/

/* server main routine */
int main(int argc, char** argv)
{
    // locals
    struct sockaddr_in server;
    server.sin_family = AF_INET;
    server.sin_addr.s_addr = htonl(LAB_BROADCAST);
    server.sin_port = htons(DEFAULT_TFTP_PORT);

    printf(ESC_WHITE_TXT);

    char* filename = calloc(MAX_FILE_NAME, sizeof(char));

        int sock; // socket descriptor

    char c;

    // User argument parsing
    while((c = getopt(argc,argv,"s:p:f:h"))!=-1)
    {
            switch(c)
            {
                    case 's':
                            if(!inet_pton(AF_INET,optarg,&(server.sin_addr)))
                    {
                        printf("%sImproper IP address%s\n",ESC_RED_TXT,
                            ESC_BR_GRAY_TXT);
                        exit(1);
                    }
                            break;
                    case 'p':
                            server.sin_port = atof(optarg);
                            break;
            case 'f':
                strcpy(filename,optarg);
                break;
                    case 'h':
                            printf("\n");
                printf("-h prints this help statement\n\n");
                printf("-s is the IPv4 address of the TFTP server\n\n");
                printf("-p override the TFTP server port (default: 69)\n\n");
                printf("-f file name to download from the TFTP server\n\n");
                            exit(1);
                            break;
            }
    }
```

```c
    // ready to go
    printf("Connecting to TFTP server on port: %d\n", ntohs(server.sin_port));

    // for UDP, we want IP protocol domain (AF_INET)
    // and UDP transport type (SOCK_DGRAM)
    // no alternate protocol - 0, since we have already specified IP

    struct timeval sockTimeout;
    sockTimeout.tv_sec = 0;
    sockTimeout.tv_usec = SOCK_TIMEOUT_US;

    if ((sock = socket(AF_INET, SOCK_DGRAM, 0)) < 0 ||
       (setsockopt(sock, SOL_SOCKET, SO_RCVTIMEO, &sockTimeout,
                sizeof(sockTimeout)) < 0))
    {
            perror("Error on socket creation and configuration\n");
            exit(1);
    }

    uint8_t* sendBuffer = calloc(MAX_BLOCK_SIZE+4,sizeof(uint8_t));
    int sendLength = 2+strlen(filename)+1+strlen(TFTP_MODE)+1;
    uint8_t* receiveBuffer = calloc(MAX_BLOCK_SIZE+4,sizeof(uint8_t));
    FILE* receiveFile = NULL;
    uint16_t currentBlock = 1;
    struct sockaddr_in from;
    socklen_t server_len = sizeof(server);
    int sent = 0;
    int received = 0;
    int retransmitAttempts = 0;

    sendBuffer[1] = TFTP_RRQ;
    strcpy(sendBuffer+2,filename);
    strcpy(sendBuffer+strlen(filename)+3,TFTP_MODE);

    sent = sendto(sock, sendBuffer, sendLength, 0,
                    (struct sockaddr *)&server, server_len);

    printf("Sent %d bytes to %s\n",sent,inet_ntoa(server.sin_addr));
    printf("packet contained: %d%d %s %s\n", sendBuffer[0], sendBuffer[1],
        sendBuffer+2, sendBuffer+strlen(filename)+3);

    do
    {
        if((received = recvfrom(sock, receiveBuffer, MAX_BLOCK_SIZE+4, 0,
                        (struct sockaddr *)&from, &server_len)) < 0)
        {
            if (retransmitAttempts <= 5)
            {
                printf("Respose Timeout: retransmitting\n");
                if (retransmitAttempts != 0)
                {
                    sendBuffer[0] = 0;
                    sendBuffer[1] = TFTP_ACK;
                    sendBuffer[2] = currentBlock >> 8;
                    sendBuffer[3] = currentBlock & 0x00FF;
                    sendLength = 4;
                }
                sent = sendto(sock, sendBuffer, sendLength, 0,
                            (struct sockaddr *)&server, server_len);

                retransmitAttempts++;
```

```c
        }
        else
        {
            printf("Response Timeout: too many failed attempts\n");
            fclose(receiveFile);
            free(filename);
            free(sendBuffer);
            free(receiveBuffer);

            exit(1);
        }

    }

    // print info to console
        // printf("\033[1A\n\rReceived message from %s port %d\n\033[1B"
,
        //          inet_ntoa(from.sin_addr), ntohs(from.sin_port));

    server.sin_port = from.sin_port;

    if (received < 0)
        {
            perror("Error receiving data");
        }
        else
        {
            switch ((receiveBuffer[0]<<8)|receiveBuffer[1])
    {
    case TFTP_RRQ:
        printf("%sRECEIVING RRQ NOT IMPLEMENTED IN THIS PROGRAM%s\n",
            ESC_YELLOW_TXT, ESC_WHITE_TXT);
        break;

    case TFTP_WRQ:
        printf("%sRECEIVING WRQ NOT IMPLEMENTED IN THIS PROGRAM%s\n",
            ESC_YELLOW_TXT, ESC_WHITE_TXT);
        break;

    case TFTP_DATA:
        if (receiveFile == NULL)
        {
            receiveFile = fopen(filename, "a+");
        }
        if(fwrite(receiveBuffer+4, received-4, 1, receiveFile))
        {
            sendBuffer[0] = 0;
            sendBuffer[1] = TFTP_ACK;
            sendBuffer[2] = currentBlock >> 8;
            sendBuffer[3] = currentBlock & 0x00FF;
            sendLength = 4;
            sent = sendto(sock, sendBuffer, sendLength, 0,
                        (struct sockaddr *)&server, server_len);

            // printf("\033[1A\n\rACK contained: %d%d %d%d\n\033[1B",
            //      sendBuffer[0], sendBuffer[1], sendBuffer[2],
            //      sendBuffer[3]);

            currentBlock++;

            printf("\r%s[", ESC_WHITE_TXT);
            for (int i = 0; i < MAX_BLOCKS/819; i++)
```

```c
                {
                    if (currentBlock*11/819 >= i)
                    {
                        printf("%s=%s", ESC_GREEN_TXT, ESC_WHITE_TXT);
                    }
                    else
                    {
                        printf("%s-%s", ESC_WHITE_TXT, ESC_WHITE_TXT);
                    }
                }
                printf("%s]%s", ESC_WHITE_TXT, ESC_WHITE_TXT);

            }
            break;

        case TFTP_ACK:
            printf("%sRECEIVING ACK NOT IMPLEMENTED IN THIS PROGRAM%s\n",
                    ESC_YELLOW_TXT, ESC_WHITE_TXT);
            break;

        case TFTP_ERROR:
            printf("%sTFTP ERROR ", ESC_RED_TXT);
            switch ((receiveBuffer[2]<<8)|receiveBuffer[3])
            {
            case TFTP_ERROR_NOTDIFF:
                printf("See Error Message: ");
                break;

            case TFTP_ERROR_FNF:
                printf("File Not Found: ");
                break;

            case TFTP_ERROR_ACCVIO:
                printf("Access Violation: ");
                break;

            case TFTP_ERROR_DSKFULL:
                printf("Disk Full or Allocation Exceeded: ");
                break;

            case TFTP_ERROR_ILLTFTP:
                printf("Illegal TFTP Operation: ");
                break;

            case TFTP_ERROR_UNID:
                printf("Unkown Transfer ID: ");
                break;

            case TFTP_ERROR_FALEX:
                printf("File Already Exists: ");
                break;

            case TFTP_ERROR_NOUSR:
                printf("No Such User: ");
                break;

            default:
                printf("UNKOWN ERROR CODE\n");
                break;
            }
            printf("%s%s\n", (receiveBuffer+2), ESC_WHITE_TXT);
```

```c
                fclose(receiveFile);
                free(filename);
                free(sendBuffer);
                free(receiveBuffer);

                exit(1);
                break;

            default:
                printf("%sUNKNOWN OPCODE RECEIVED%s\n", ESC_RED_TXT,
                    ESC_WHITE_TXT);
                break;
        }
    }

    } while (received-4 >= MAX_BLOCK_SIZE);

    printf("\r%s[", ESC_WHITE_TXT);
    for (int i = 0; i < MAX_BLOCKS/819; i++)
    {
        printf("%s=%s", ESC_GREEN_TXT, ESC_WHITE_TXT);
    }
    printf("%s]%s", ESC_WHITE_TXT, ESC_WHITE_TXT);

    fclose(receiveFile);
    free(filename);
    free(sendBuffer);
    free(receiveBuffer);

    // close socket
    close(sock);
    // done
    return(0);
}
```