```
/*                                  EEEN 20060 Communication Systems TCP
                                              Server code
                            Written by : Conor Burke and Fergal Lonergan
                            In conjunction with : Darren Coughlan and Sam Luby
                                 who wrote the client for this program.

                This program is a built to act like a server where it takes in information from
                the client and manipulates it in order to upload or download files over the
                internet or on an ethernet cable via a network.
                Our protocol was for us to first establish a secure connection with the client.
                Then we wait for a request from the client in the form of a character array
                and break it up into the relevant components we need for our report.
                If we are doing download we expected to first receive a D, then the file name,
                i.e.(filename.txt) and then an @ symbol which marked the end of the filename.
                We then extracted the mode, i.e. first letter, the file name i.e. filename.txt,
                and attributed them to a character mode and character array file_name respectively.
                We then declared a file pointer of type FILE called fpi for input file and attempt
                to open the file for binary reading and checking for errors to ensure the file
                is there. assuming it is we call the send_File function which takes the following
arguments:
                the socket which the data will be sent on, a file pointer and a character file_found
which
                will equal to "y" if the file has been found or "n" if it has not.
                This send_File function then proceeds to calculate the amount of bytes in the file
                by seeking to the end of it and attributing this position to nBytes. it then
                returns to the start of the file in order to read its contents correctly.
                It checks for errors whilst doing this ensuring that the connection has not been
closed,
                or that the acknowledgement that we are beginning to send and the file have been
sent
                correctly, as well as that there haven't been any errors to do with the file whilst
                sending. once the file has been sent correctly it prints this to the screen and then
                exits the function.
                If the client wishes to upload we expect to receive a request in the form
Ufilename.txt@1000000@
                where U denotes the mode to upload, filename.txt is the name of the file, and
1000000 is
                the size of the file. The @ symbols are again used as end markers for the different
character
                arrays in our request array. similarly to download we retrieve the mode, file name,
                and size of bytes and attribute them to a char mode, char array file_name and a char
array number,
                which is then converted to an int using the atoi function, respectively.
                meanwhile we have created a file pointer fpo denoting output file and opened it
checking for errors
                before sending an acknowledgement to the client saying that we are ready to receive
their upload.
                we then call the function receive file that will take care of the main file handling
                and receiving of bytes for the upload mode.
                receive_File takes in the the socket which the data will be received along, the file
                pointer fpo of type FILE where the received data will be stored, the character array
                containing the request as well as nBytes the number of bytes in the file.
                it then proceed to receive the bytes from the client and write them to our output
                file whilst checking for errors like a closed connection or a problem reading data
etc.
                once a successful transmission has occurred we exit the loop and return to main
where
                the program loops and asks if the client would like to download upload or exit
again.
                As well as the download or upload option there is an exit option that lets you exit
from the
                program safely.


*/


// including our libraries of functions
#include <stdio.h>
#include <winsock2.h>
#include <ws2tcpip.h>
#include <conio.h>

//*****************************************************************************************************
// These are our function prototypes that initialise our three functions printError of type void,
// and receive_File and send_File both of type int.
```

```
//***************************************************************************************************
// Our printError function is designed to print an informative error message to the screen in the
// case that an error has occurred. it takes in no value but uses our WSAGetLastError from our
// winsock2.h library which finds  the last error that occurred in our code and then prints
// it's position and details (error code and last error) to the screen. This function
// is used for troubleshooting throughout our program, especially in the initial stages when
// we are attempting to establish a secure connection with the client.
//***************************************************************************************************
// Our receive_file function is where most of our upload mode is handled. in main we first deal
// with the request in order to determine which mode is being called and what the file name and
length
// should be from the information sent from the client once a secure connection has been
established.
// Once we have determined that the client wishes to upload a file to our server, this is done by
// getting the first letter of the array of characters sent to us by the client, for upload this
// letter should be a "U", we then create a file pointer and then open a file called whatever
// name our client has sent to us after reading this from the array of bytes sent to us in the
// form of the request. We also find the size of the file and set this equal to nBytes using
// the same method we used to find the file name. After some quick error checking we then call the
// receive file function which takes the following argument, cSocket of type socket, which is the
// socket which we have opened for the client to send us the data, a file pointer fpo of type FILE,
// which is a pointer to the output file in which the data will be stored, a character array called
// request which is the request, which is the original information about the file the client wishes
// to upload, as well as an integer value nByte which is the number of bytes of data that the client
// to upload, i.e. the file size.
// This function returns an integer value from 0 to 3 depending on the value of nRx. If the
// function returns 1 then our transmission failed because our function returns a negative value
// for bytes received and there has been a socket error. If it returns 2 then we have received no
// bytes as the connection has been closed by the server, and if it returns a 3 then we are
receiving
// the data correctly. Otherwise the data has been received from the client correctly and written to
the
// output file.
// Details on how the function works can be found at the function declaration.
//***************************************************************************************************
****
// Our final function is our send_File function which handles most of our download mode. In main it
// is called after ensuring a secure connection has been established and after dealing with the
request
// sent from the client. We then, after some error checking to make sure the request has been
received
// correctly, get the first letter from the request array and retrieve the file name from the array.
// we then check to make sure the client wishes to download by checking that the first character in
the
// request array was a "D". if this is true we create a file pointer fpi of type file, which will be
// to handle our input file for the remainder of the download operation. We then attempt to open
// the file of said file name and check to see that it is in our directory, if not we will print an
// error stating so, otherwise we tell the user that the file has been found and call our send_File
// function.
// Our send_File function takes in the arguments cSocket of type SOCKET, the socket we have created
in
// order to transmit the data, the file pointer fpi of type FILE, which points to the file from
which
// the client will be downloading, and the character array file_found, which contains a single
character
// which is either a "y" or an "n" depending on whether the file has been found or not. This will be
// as part of the reply, in order to tell the client that we have found the file and are now going
to
// begin sending the data.
// The function returns an integer between 0 and 4 depending on the outcome of the function. If the
// function returns a 1 then we had difficulty seeking to the end of the file. If it returns a 2
then
// we had difficulty seeking to the beginning of the file. If it returns a 3, then we had difficulty
// sending an acknowledgement to the client stating that we are going to send the file. Finally
// if it returns a 4 then an error occurred in sending the file to the client. Otherwise the file
has
// been read correctly from the input file and transmitted correctly to the client.
// Details on how the function works can be found at the function declaration.

void printError(void);  /// function to display error messages
int receive_File(SOCKET cSocket, FILE *fpo, char request[], int nByte);
int send_File(SOCKET cSocket, FILE *fpi, char file_found[]);

#define SERV_PORT 32980  // port to be used by server
#define BLK_SIZE 100  // maximum data block size in bytes

int main()
```

```
{
    WSADATA wsaData;  // create structure to hold Winsock data
    int retVal; // used to return values from functions to check for errors
    int nRx = 0; // used to calculate the amount of bytes received
    int endLine = 0, stop = 0;  // flags to control loops
    char request[100];  // array to hold received bytes (download)
    char response[100]; // array to hold our response   (upload)
    char file_found[10] = "y"; // char to know if the file has been found or not
    char data[BLK_SIZE];  // array of characters

    int end_of_Filename; // integer to know when to start reading nBytes from string

    printf("----- Server -----\n");
    // Initialise winsock, version 2.2, giving pointer to data structure
    retVal = WSAStartup(MAKEWORD(2,2), &wsaData);
    if (retVal != 0)  /// check for error
    {
        printf("*** WSAStartup failed: %d\n", retVal);
        printError();
        return 1;
    }
    printf("Initialising connection\n");
    printf("WSAStartup succeeded\n" );

    // Create a handle for a socket, to be used by the server for listening
    SOCKET serverSocket = INVALID_SOCKET;  // handle called serverSocket

    // Create the socket, and assign it to the handle
    // AF_INET means IP version 4,
    // SOCK_STREAM means socket works with streams of bytes,
    // IPPROTO_TCP means TCP transport protocol.
    serverSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    if (serverSocket == INVALID_SOCKET)  // check for error
    {
        printf("*** Failed to create socket\n");
        printError();
    }
    else printf("Socket created\n" );

    // Build a structure to identify the service offered
    struct sockaddr_in service;  // IP address and port structure
    service.sin_family = AF_INET;  // specify IP version 4 family
    service.sin_addr.s_addr = htonl(INADDR_ANY);  // set IP address
    // function htonl() converts 32-bit integer to network format
    // INADDR_ANY means we accept connection on any IP address
    service.sin_port = htons(SERV_PORT);  // set port number
    // function htons() converts 16-bit integer to network format

    // Bind the socket to the IP address and port just defined
    retVal = bind(serverSocket, (SOCKADDR *) &service, sizeof(service));
    if( retVal == SOCKET_ERROR)  // check for error
    {
        printf("*** Error binding to socket\n");
        printError();
    }
    else printf("Socket bound\n");

    // Listen for connection requests on this socket,
    // second argument is maximum number of requests to allow in queue
    retVal = listen(serverSocket, 2);
    if( retVal == SOCKET_ERROR)  // check for error
    {
        printf("*** Error trying to listen\n");
        printError();
    }
    else printf("Listening on port %d\n", SERV_PORT);

    // Create a new socket for the connection we expect
    // The serverSocket stays listening for more connection requests,
    // so we need another socket to connect with the client...
    SOCKET cSocket = INVALID_SOCKET;

    // Create a structure to identify the client (optional)
    struct sockaddr_in client;  // IP address and port structure
    int len = sizeof(client);  // initial length of structure

    // Wait until a connection is requested, then accept the connection.
    // If no need to know who is connecting, arguments 2 and 3 can be NULL
    cSocket = accept(serverSocket, (SOCKADDR *) &client, &len );
    if( cSocket == INVALID_SOCKET)  // check for error
    {
        printf("*** Failed to accept connection\n");
        printError();
    }
    else  // we have a connection, report who it is (if we care)
    {
        int clientPort = client.sin_port;  // get port number
        struct in_addr clientIP = client.sin_addr;  // get IP address
        // in_addr is a structure to hold an IP address
        printf("Accepted connection from %s using port %d\n",
               inet_ntoa(clientIP), ntohs(clientPort));
        // function inet_ntoa() converts IP address structure to string
        // function ntohs() converts 16-bit integer from network form to normal
    }
    printf("Connection succedded\n" );
    printf("--------------------\n" );

    // Main loop to receive requests and send responses
    // This example assumes that client sends first, so server receives first
    do
    {
        endLine = 0;
        // a character that will be either "D" or "U" depending on whether they wish to download or
upload
        char mode;
        char request[100]; //Holds the received data from the client request (e.g. "dfilename@"
or"ufilename@fileseize@")
        // variables incremented in loops to retrieve smaller character arrays from the larger
client request
        int i =0,n = 0;
        int nByte = 0; // variable to hold number of bytes in file ie file size
        int start_of_data =0; // variable to locate position of the start of data
        int send_test; // variable to check response was sent from server to client
        char number[50]; //variable to hold the char array of the size of the uoloaded file
        char filename[100];// variable to hold filename

        printf("\n------------------------------------------------------\n");
        printf("\nReceiving request from client\n");
        nRx = recv(cSocket, request, 100, 0);
        // nRx will be number of bytes received, or error indicator

        if( nRx < 0)  // error
        {
            printf("\n------------------------------------------------------\n");
            printf("Problem receiving, connection closed by client\n");
            printError();
            stop = 1;  // exit the loop if problem
        }
        else if (nRx == 0)  // connection closing
        {
            stop = 1;  // exit the loop in that case
        }
        else // we got some data
        {
            printf("Received request from client.\n");
            printf("------------------------------------------------------\n");
            mode = request[0]; // variable to hold u or d (download or upload)


            //Finding filename
            for(i=1; i<nRx && request[i]!= '@' ; i++)
            {
                filename[i-1] = request[i];
                end_of_Filename = i; //variable to signal position of the letter before the @
            }
            filename[end_of_Filename] = 0;

            //Download
            if(mode=='D')
            {
                printf("\n------------------------------------------------------\n");
                printf("The client wants to download %s\n", filename);
                FILE *fpi;  // file handle for input file
                // Open the input file and check for failure
                printf("Opening %s\n", filename);
                fpi = fopen(filename, "rb");  // open for binary read

                // checking to see if the file opened correctly
                if (fpi == NULL)
                {
                    printf("********************************************************\n");
                    perror("Send: Error opening input file\n");
                    printf("********************************************************\n");
                    file_found[0] = "n"; // change file_found to "n" so the client wont expect a
download
                    return 1;
                }
                else
                {
                    printf("File to be sent to client has been found\n");
                    printf("\n------------------------------------------------------\n");
                    //download function
                    send_File(cSocket, fpi, file_found); // function that handles file and byte
sending

                }
            }

            //Upload
            else if(mode=='U')
            {

                FILE *fpo; // file pointer to output file of type FILE

                printf("\n------------------------------------------------------\n");
                printf("The client wants to upload %s\n", filename);
                printf("Creating file to hold upload file\n");
                fpo = fopen(filename, "wb"); // opening the file for binary write

                // error checking to see if file is opened correctly
                if (fpo == NULL)
                {
                    printf("\n********************************************************\n");
                    perror("Send: Error creating output file");
                    printf("\n********************************************************\n");
                    return 1;
                }
                else
                {
                    printf("File created\n");
                    printf("\n------------------------------------------------------\n");
                }

                // Finding filesize
                // for loop which starts reading at end_of_file+2 where the file size starts after
first @ (only for upload)
                for(i=end_of_Filename+2; i<nRx || request[i] != '@'; i++)
                {
                    number[n] = request[i];
                    n++;
                    start_of_data = n+2;
                }
                // convert char array to integer giving the number of bytes in the file
                nByte = atoi(number);

                // error check to ensure file is opened correctly and the size of file is greater
                // than 0
                if(fpo !=NULL && nByte>0)
                {
                    printf("\n------------------------------------------------------\n");
                    printf("Sending acknowledgement that file has been created and ready to
upload");
                    printf("\n------------------------------------------------------\n");
                    send_test = send(cSocket, file_found, 1, 0); // telling the client that the file
is ready to be received
                    receive_File(cSocket, fpo, request,nByte); // receiving file


                }
            }
            // mode to exit the program safely
            else if(mode=='E')
            {
                printf("User requested to exit\n");
                stop=1;
            }
            else
            {
                printf("\n------------------------------------------------------\n");
                printf("User entered incorrect command");
                printf("\n------------------------------------------------------\n");
                break;
            }
        }
    }
    while (stop == 0);    // repeat until told to stop
    // When this loop exits, it is time to close the connection and tidy up


    printf("\n------------------------------------------------------\n");
    printf("Connection closing...");
    printf("\n------------------------------------------------------\n");

    // Shut down the sending side of the TCP connection first
    retVal = shutdown(cSocket, SD_SEND);
    if( retVal != 0)  // check for error
    {
        printf("\n********************************************************\n");
        printf("*** Error shutting down sending\n");
        printError();
        printf("\n********************************************************\n");
    }

    // Then close the client socket
    retVal = closesocket(cSocket);
    if( retVal != 0)  // check for error
    {
        printf("\n********************************************************\n");
        printf("*** Error closing client socket\n");
        printError();
        printf("\n********************************************************\n");
    }
    else
    {
        printf("\n********************************************************\n");
        printf("Client socket closed\n");
        printf("\n********************************************************\n");
    }

    // Then close the server socket
    retVal = closesocket(serverSocket);
    if( retVal != 0)  // check for error
    {
        printf("\n********************************************************\n");
        printf("*** Error closing server socket\n");
        printError();
        printf("\n********************************************************\n");
    }
    else
    {
        printf("\n------------------------------------------------------\n");
        printf("Server socket closed");
        printf("\n------------------------------------------------------\n");
    }
    // Finally clean up the winsock system
    retVal = WSACleanup();
    printf("WSACleanup returned %d\n",retVal);
    printf("Transfer complete!\n" );
    // Prompt for user input, so window stays open when run outside CodeBlocks
    printf("\nPress return to exit:");
    gets(response);
    return 0;

}

/*
the receive_File function takes in the arguments cSocket of type socket, the file
```

```
pointer fpo, which denotes the output file, of type FILE, a character array called
request which received_data which holds the received data from the client and
an integer value nByte which is the length of the file in bytes.
It prints to let the user know where in the program it is and also what it is
intending to do. it then receives the data from the client. it expects these in
blocks of 100 (BLK_SIZE)so the recv function orders the received bytes accordingly.
it then checks to see that there have been no errors with the connection.
once it has received the bytes and there was no error
it then prints the received bytes into the desired file location before getting
more bytes. it also prints a progress bar so that the user knows how far along the
download is if the upload ha worked correctly it prints to the screen to let the user
know it has succeeded and then exits the function after closing the file.
*/
int receive_File(SOCKET cSocket, FILE *fpo, char received_data[], int nByte)
{
    int bytes_rec = 0; // keeping track of number of bytes to date retrieved from client
    int stop=0; // end loop condition
    int nRx = 0; // number of bytes received per transmission
    int data_written=0; // data written to file
    int retVal = 0, ret = 0; // return values from functions for error detection
    int threshold = 10; // used in progress bar to show another 10% has been received
    long percentage = 0; // used to calculate percentage of file received

    printf("\n------------------------------------------------------\n");
    printf("In Receiving File Funciton\n");
    printf("Receiving data and writing to file");
    printf("\n------------------------------------------------------\n");
    printf("---------------------------------------------------------------");
    printf("\n\t\t       PROGRESS BAR \n");
    printf("[");
    // while the bytes received is less than the total bytes expected and no errors keep receiving
    while(bytes_rec<nByte&&stop==0)
    {
        nRx = recv(cSocket, received_data, BLK_SIZE,0); // setting nRx to the number of total bytes
received

        if( nRx == SOCKET_ERROR)  // check for error
        {
            printf("\n********************************************************\n");
            printf("Problem receiving\n");
            printError();
            printf("\n********************************************************\n");
            stop = 1;  // exit the loop if problem
        }
        else if  (nRx == 0)  // connection closed
        {
            printf("\n********************************************************\n");
            printf("nRx has reached 0\n"); //test
            printf("Connection closed by server");
            printf("\n********************************************************\n");
            stop = 1;
        }
        else
        {
            // keeping track of amount of bytes written to output file
            data_written = fwrite(received_data, 1, nRx, fpo);
            bytes_rec += data_written; // increasing total bytes received by the bytes written

            percentage = (bytes_rec/(double)nByte)*100; // calculating percentage
            if(percentage>threshold) // if precentage goes up by another 10% progress bar shows this
            {
                printf("===%d%%",threshold);
                threshold+=10;
            }


        }// to exit the while loop when the entire file has been read/sent


        while(bytes_rec<nRx)   //while the number of bytes sent is less than the total number of
bytes in the file
        {
            //fread attempts to read in up to 100 bytes from the file . retSend is how many bytes
are read.
            ret = (int) fread(received_data, 1, 100,fpo);

            if (ferror(fpo))  // check for error
            {
                printError();
                fclose(fpo);   // close input file
                return 3;
            }

        }
    }

    printf("===100%%]\n"); // end of progress bar
    printf("\n------------------------------------------------------\n");
    printf("Total number of bytes received: %d\n", bytes_rec);
    printf("Upload succeeded");
    printf("\n------------------------------------------------------\n");

    fclose(fpo); // close output file

}
/*
the send_File function takes in three arguments, cSocket of type socket which
is the socket we will send the data along to the client, a file pointer fpi,
denoting the input file, of type FILE and the character array file_found, which
contains one letter a y or an n depending on whether the file has been found in
our directory or not.
it prints to tell the user where it is in the program and then calculates the
size of the file to be send using the fseek and ftell functions and attributing
the value to nBytes, the number of bytes in the file. whilst it does this it
checks for errors and prints them to the screen should one occur. It then sends
a small acknowledgement to the client telling them that we have found the file,
that it is nBytes long and that we are about to send the data. it then sends
the data checking to make sure an error hasn't occurred whilst printing a progress
report to the screen so the user can see how long there is left in the download
and if the file transmits correctly it closes the file and then prints that we
have had a successful transmission and the file is sent and exits the function.
*/
int send_File(SOCKET cSocket, FILE *fpi, char file_found[])
{

    printf("\n------------------------------------------------------\n");
    printf("In Sending File Function");
    char data[100];  // array of characters
    int retVal, ret;  // return code from functions used in error detection
    int nBytes; // number of bytes in file
    int BytesSending=0; // number of bytes being sent presently
    int SentBytes = 0; // amount of bytes sent to date to client
    int threshold = 10; // used in progress bar to show another 10% has been received
    long percentage = 0; // used to calculate percentage of file received

    //Find size of file to be sent to client to download.

    retVal = fseek(fpi, 0, SEEK_END);  // set current position to end of file
    if (retVal != 0)  // there was an error print it and close file
    {
        printf("\n********************************************************\n");
        perror("Error in fseek");
        printf("errno = %d\n", errno);
        fclose (fpi);
        printf("\n********************************************************\n");
        return 2;
    }
    nBytes = ftell(fpi);  // find out what current position is which is size of file and set it to
nBytes
    printf("File size is %ld bytes", nBytes);  // print it

    retVal = fseek(fpi, 0, SEEK_SET);   // set current position to start of file
    if (retVal != 0)  // if there was an error print it and close input file
    {
        printf("\n------------------------------------------------------\n");
        perror("Error in fseek");
        printf("errno = %d\n", errno);
        fclose (fpi);
        printf("\n------------------------------------------------------\n");
        return 3;
    }

    ret = sprintf( data, "%c%d@",file_found[0], nBytes); //adding our value of nByte to string
    retVal = send(cSocket, data, ret, 0);  // send bytes to TCP
    // retVal is greater 0 if succeeded, zero or less if failed
    if (retVal < 0)  // if there was an error print it and close file
    {
        printf("\n------------------------------------------------------\n");
        perror("Error in sending acknowledgement");
        printf("errno = %d\n", errno);
        fclose (fpi);
        printf("\n------------------------------------------------------\n");
        return 3;
    }

    printf("Sending file to client\n");
    // while we haven't reached the end of file and the bytes sent is less than size of file send
bytes
    while(!feof(fpi)&&SentBytes<nBytes)
    {
        BytesSending = (int) fread(data, 1, BLK_SIZE, fpi); // bytes left to be sent

        retVal = send(cSocket, data, BytesSending, 0);  // send bytes to TCP
        SentBytes+=BytesSending; // increase bytes sent to date
        printf("---------------------------------------------------------------");
        printf("\n\t\t       PROGRESS BAR \n");
        printf("[");

        while(SentBytes<nBytes)   //while the number of bytes sent is less than the total number of
bytes in the file
        {
            //fread attempts to read in up to 100 bytes from the file . retSend is how many bytes
are read.
            ret = (int) fread(data, 1, BLK_SIZE, fpi);
            if (ferror(fpi))  // check for error
            {
                printError();
                fclose(fpi);   // close input file
                return 3;
            }
            else
            {
                retVal = send(cSocket, data, ret, 0); //sends retVal amount of bytes to the client
each time.
                SentBytes+=retVal;    //incrementing the amount of bytes sent to client to date

                percentage = (SentBytes/(double)nBytes)*100; // calculating the percentage
                if(percentage>threshold) // if percentage increase by 10% another section of
progress bar printed
                {
                    printf("===%d%%",threshold);
                    threshold+=10;
                }
            }
        }
        printf("===100%%]\n"); // end of progress bar
        printf("Download has worked correctly\n");
    }
    fclose(fpi);  // close the file
    printf("Succesful Transmission of data.\n");
    printf("File Sent\n\n");
    printf("\n------------------------------------------------------\n");

}// end of sendFile



/* Function to print informative error messages
   when something goes wrong...  */
void printError(void)
{
    char lastError[1024];
    int errCode;

    errCode = WSAGetLastError();  /// get the error code for the last error
    FormatMessage(
        FORMAT_MESSAGE_FROM_SYSTEM | FORMAT_MESSAGE_IGNORE_INSERTS,
        NULL,
        errCode,
        MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT),
        lastError,
        1024,
        NULL);  /// convert error code to error message
    printf("WSA Error Code %d = %s\n", errCode, lastError);
}
```