Introduction

The aim of the file transfer project is to create a client-server communication environment where multiple clients are served by a single server. The job of the server is to provide services such as downloading, uploading, file directory changes and directory listing. The code for the project must utilize the various socket C programming libraries which can be found in the Fedora Linux operating system (among others). When UNIX programs perform any sort of I/O operation, they do so by using file descriptors. A file descriptor is simply an integer that may correspond to another terminal, a file, a pipe, and so on. A socket file descriptor is required when using the socket C programming libraries; for the purposes of the project, it provides a way of keeping track of other machines you are communicating with. There are two main types of sockets available for use: the stream socket (which operates under connection-oriented TCP protocols) and the datagram socket (which operates under connection-less UDP protocols). The project will make use of stream sockets to ensure reliable transmission of data.

Specifications

The overall goal of the server is to provide five main services:

- Handling download requests sent by the client
- Handling upload requests sent by the client
- Providing a change of directory when requested by the client
- Listing all of the files in the current working directory
- Providing error reporting functionality

Data is to be sent and received in the form of a protocol data unit (PDU). The PDU contains three main segments: the type of PDU being sent or received, the length of the data contained in the PDU, and the data itself (see figure 1).

Type	Length	Data

Figure 1: Structure of the PDU used in the project

The various PDU types cater to all of the required functionality of the server. The types are as follows:

- Type ‘D’ – Sent by the client to request a download operation
- Type ‘U’ – Sent by the client to request an upload operation
- Type ‘R’ – Sent by the server to notify the client that it is ready
- Type ‘F’ – Indicates that a PDU to and from the client or server contains file data
- Type ‘P’ – Sent by the client to request a change in directory
- Type ‘L’ – Sent by the client to request a list of files in the current directory
- Type ‘I’ – Contains the list of file names in the current directory

Implementation

The first design goal was to implement the PDUs using C. This was done by creating a struct with three components: type, length, and data[ ] (as per the specifications). The size of the data[ ] array could vary depending on the user's preference. Two PDU structs were created: one that contained data to be sent, and another which contained data to be received; the following code instantiates the PDUs:

```
struct PDU
{
        char type;
        int  length;
        char data[BUFLEN];
} rpdu,spdu;
```

In order to keep track of the file statistics (on both client and server), such as its size, a 'stat' struct named fstat was used; this struct requires the "system/stat.h" library:

```
struct stat fstat;
```

Client - Menu

After creating a stream socket successfully and connecting to the server, the client is presented with five choices:

1. Download a file
2. Upload a file
3. Change file directory
4. List files in current directory
5. Quit

And their choice is saved into an integer named choice. The following code performs the action:

```
int choice;

printf("What would you like to do?\n1. Download a file\n2. Upload a
file\n3. Change file directory\n4. List current directory\n5. Quit\n");
//Save client's choice
scanf("%d", &choice);
```

A switch statement is then performed on the choice the client has made, which each case corresponding to one of the above five options. Each case will be looked at separately, beginning with the download request operation.

Client – Download and Error Report handling operations

The code for the download operation on the client's side is as follows (see Appendix for the complete client.c and server.c codes):

```
switch(choice)
{
       case 1: //-- Download request

              printf("Name of the file you would like to download:\n");
              n = read(0, spdu.data, BUFLEN);     //Store filename inside the send PDU
              spdu.type = 'D';
              spdu.data[n-1] = '\0';              //Terminate filename string

              printf("You have requested the following file: %s \n", spdu.data);

              spdu.length = strlen(spdu.data);
              send(sd, (char *)&spdu, sizeof(spdu), 0);

              //Check if there was an error
              recv(sd, (char *)&rpdu, sizeof(rpdu), 0);
              if (rpdu.type == 'E')
              {

              printf("An error occurred. Message from the server: %s\n", rpdu.data);

              }
              else if(rpdu.type == 'R')  //Begin download
              {
                     FILE *fr = fopen(spdu.data, "w");


                     bytes_to_write = rpdu.length;      //Get file size from length field
                                                        //of the received PDU

                     while(bytes_to_write >  0)
                     {
                            recv(sd, (char *)&rpdu, sizeof(rpdu), 0); //r2
                            n = fwrite(rpdu.data, sizeof(char), bytes_to_write, fr);
                            bytes_to_write -= n;

                            if(bytes_to_write == 0)
                            {
                                   fclose(fr);
                                   printf("File downloaded successfully.\n");
                            }
                     }
              }

              break;        //return to choice selection menu
```

The operation begins by sending a 'D' type PDU with the filename (entered by the user) and its length to the server. A PDU is then received which either contains an error report or an acknowledgement of the file's existence on the server (in the form of an 'R' type PDU). If the PDU received is an error report, the error message from the server is printed to stdout and the user is returned to the menu to make a new selection. If an acknowledgement is received, the file to be received is created and written to, then a message confirming the successful download operation is sent to the user. Upon completion of the download, the user is once again returned to the menu to either perform another action or exit.

Client – Upload and Error Report handling operations

The code for the upload operation on the client's side is as follows:

```
case 2: //-- Upload request

       printf("Name of the file you would like to upload:\n");
       n = read(0, spdu.data, BUFLEN);          //Store filename inside the send PDU
       spdu.type = 'U';
       spdu.data[n-1] = '\0';                   //Terminate filename string

       printf("You requested to upload: %s\n", spdu.data);
       spdu.length = strlen(spdu.data);      //Set the length field to the filename length
       send(sd, (char *)&spdu, sizeof(spdu), 0);

       recv(sd, (char *)&rpdu, sizeof(rpdu), 0);
       if (rpdu.type == 'E')                          //Check if there was an error
       {
              printf("An error occurred. Message from the server: %s\n", rpdu.data);

       else if(rpdu.type == 'R')  //Begin upload
       {
              printf("Server is ready. Beginning upload...\n");
              FILE *fs = fopen(spdu.data, "r");
              lstat(spdu.data,&fstat);
              bytes_to_read = fstat.st_size;
              spdu.length = fstat.st_size;

              if (fs == NULL)
              {
                     printf("Cannot upload because the file does not exist.\n");
                     break;
              }
              while(bytes_to_read > 0)
              {
                     n = fread(spdu.data, sizeof(char), bytes_to_read, fs);
                     spdu.type = 'F';
                     spdu.length = bytes_to_read;
                     send(sd, (char *)&spdu, sizeof(spdu), 0);
                     bytes_to_read -= n;

                     if(bytes_to_read == 0)
                     {
                     fclose(fs);
                     printf("File uploaded successfully.\n");
                     }
               }
          }

       break;        //return to choice selection menu
```

The user is asked to enter the name of the file they would like to upload (the file must be in the current working directory of the client application). A 'U' type PDU is sent to the server, containing the name of the file and its length. A PDU is then received, which will either contain an error report or an acknowledgement in the form of an 'R' type PDU. The error report is handled in the same fashion as the download request operation. If an 'R' type PDU is received, the uploading begins; the file specified by the user is opened, read, and its statistics are found using the lstat() function (only its size is of interest). The size of the file is sent inside F-type PDUs to the server so it can keep track of how many bytes need to be written. Upon completion, a message is printed to stdout to inform the user that the upload operation has completed. The user is then returned to the menu.

Client – Change directory request

The code for the file directory change request on the client's side is as follows:

```
case 3: //-- Change file directory request

       printf("Directory to change to:\n");
       n = read(0, spdu.data, BUFLEN);          //Store directory name inside the send
                                                //PDU
       spdu.type = 'P';
       spdu.data[n-1] = '\0';                          //Terminate directory name string
       send(sd, (char *)&spdu, sizeof(spdu), 0);

       recv(sd, (char *)&rpdu, sizeof(rpdu), 0);

       if (rpdu.type == 'E')
       {
              printf("An error occurred. Message from the server: %s", rpdu.data);
       }

       else if(rpdu.type == 'R')
       {
              printf("Directory changed. Current directory: %s\n", spdu.data);
       }

       break;        //return to choice selection menu
```

The user is first prompted to enter the directory they would like to change to. A 'P' type PDU is sent to the server, containing the directory name. A PDU is received, which either contains an error report (if the directory does not exist on the server) or an acknowledgement letting the user know that the directory has indeed been changed, as well as re-iterating what the current directory is. Upon completion, the user is returned to the operation selection menu.

Client – Change directory request

The code for the listing of the current file directory request is as follows:

```
case 4: //-- List directory request

        spdu.type = 'L';
        send(sd, (char *)&spdu, sizeof(spdu), 0); //Send L-type PDU

        recv(sd, (char *)&rpdu, sizeof(rpdu), 0);

        if(rpdu.type == 'l')
        {
                printf("\nThe files in the current directory are:\n%s\n\n", rpdu.data);
        }

        break;          //return to choice selection menu
```

An ‘L’ type PDU is sent to the server to let it know that a listing of the current file directory is needed. The server is expected to (based on the way this operation is coded on its end) return the list of files as a string inside the data portion of the received ‘l’ type PDU.

Server – Download and Error Reporting operations

The code for the file download operation on the server’s side is as follows:

```
while(1)
{
    rpdu.type = 'X'; //Set PDU type to unknown. This avoids looping errors caused by
                         //the switch statement

    //Receive the client's choice of action
    recv(sd, (char*)&rpdu, sizeof(rpdu), 0);
    printf("Received a %c type PDU.\n", rpdu.type);

    //Perform the requested operation based on their choice
    switch (rpdu.type)
    {
     case 'D': //-- Client download request;
     printf("Client has requested to download a file.\n");
     FILE *fs = fopen(rpdu.data,"r");

     if (fs == NULL) //-- Error Reporting
     {
         fprintf(stderr, "ERROR: File %s not found on Server.\n", rpdu.data);
         spdu.type = 'E';
         sprintf(spdu.data, "Unfortunately, %s was not found on the server.", rpdu.data);
         send(sd,(char *)&spdu, sizeof(spdu),0);
     }
    else
    {
        lstat(rpdu.data,&fstat);
        int bytes_to_read = fstat.st_size;
        spdu.length = fstat.st_size;
        spdu.type = 'R';
        send(sd, (char *)&spdu, sizeof(spdu), 0);
        printf("%s was found in the server.\nBeginning file Transfer.....\n", rpdu.data);

        printf ("Size of the file  = %d bytes \n", bytes_to_read);

        while(bytes_to_read > 0)
        {
              n = fread(spdu.data, sizeof(char), bytes_to_read, fs);
              bytes_to_read -= n;
              spdu.type = 'F';
              spdu.length = fstat.st_size;
              send(sd, (char *)&spdu, sizeof(spdu) ,0);

              if(bytes_to_read == 0)
              {
                     fclose(fs);
                     printf("File sent successfully.\n");
              }
         }
     }

     break;
```

Before the switch() operation is made based off of the PDU type received, the very first recv() call must be made. The receive PDU type is initially set to ‘X’ (arbitrarily chosen) to prevent looping errors that are due to certain switch cases being performed based on previously requested operations. If the received PDU type is ‘D’, then the file with the received filename from the client is opened. If the FILE pointer points to NULL (meaning that the file doesn’t exist in the current working directory), an error report is sent to the client in the form of an ‘E’ type PDU. If the file does exist, an acknowledgement is sent with the (as an ‘R’ type PDU). The opened file’s statistics are retrieved using lstat() and ‘F’ type PDUs containing its size are sent to the client. Once there are no more bytes to be read from the file, a message confirming that the file has been sent is printed to stdout.

Server – Upload and Error Reporting operations

The code for the file upload operation on the server's side is as follows:

```
case 'U': //-- Client upload request
       printf("Client had requested to upload a file named : %s.\n", rpdu.data);
       FILE *fr = fopen(rpdu.data, "w");

       if (fr == NULL) //-- Error Reporting
       {
              fprintf(stderr, "Failed to create file.\n");
              spdu.type = 'E';
              sprintf(spdu.data,"Unfortunately, %s could not be created on the
              server.\n",rpdu.data);
              send(sd,(char *)&spdu, sizeof(spdu) ,0);
              break;
       }
       else
       {
              spdu.type = 'R';
              send(sd, (char *)&spdu, sizeof(spdu), 0);   //Notify the client that the
                                                         //server is ready

              recv(sd, (char *)&rpdu, sizeof(rpdu), 0);  //Receive an F-type PDU with
                                                        //the file length
              bytes_to_write = rpdu.length;

              while  (bytes_to_write > 0)
              {
                     n = fwrite(rpdu.data, sizeof(char), bytes_to_write, fr);
                     bytes_to_write -= n;

                     if(bytes_to_write == 0)
                     {
                            fclose(fr);
                            printf("File uploaded successfully.\n");
                     }
              }
       }
       break;
```

When a 'U' type PDU is received, the above case statements are executed. The received PDU contains the file name to be uploaded to the server; this filename is used to create a file pointer on the server. An error report is sent if the file pointer with the received filename points to NULL. If the file pointer is created successfully, an 'R' type PDU is sent to the client, indicating that the server is ready to begin the upload. An 'F' type PDU is received, containing the data and length of the file to be written. The bytes are written to the file and a confirmation that the uploaded was successful is sent to stdout.

Server – File directory change and Error Reporting operations

The code for the file directory change operation on the server's side is as follows:

```
case 'P': //-- Client change directory request

        printf("Client has requested a change of directory to: %s\n", rpdu.data);
        char *directory = rpdu.data;       //Directory string was received in the first
                                           //recv() call

        if(chdir(directory) == -1)
        {
                spdu.type = 'E';
                sprintf(spdu.data,"The directory: '%s' does not exist\n",rpdu.data);
                send(sd, (char *)&spdu, sizeof(spdu) ,0);
                break;
        }

        else
        {
                spdu.type = 'R';
                send(sd, (char *)&spdu, sizeof(spdu) ,0);
        }

        break;
```

A received PDU of 'P' type indicates that the client would like to change the file directory. The received 'P' type PDU contains the name of the directory in its data field, which is assigned to a character pointer (string) named directory. A call is made to the chdir() function, which changes the working directory of the process (and therefore enters the specified file directory). The chdir() functions returns a -1 if the directory change is unsuccessful; this will cause the server to send an error report to the client. Upon successful completion, an acknowledgement in the form of an 'R' type PDU is sent to the client to confirm that the directory was changed.

Server – Current file directory listing operation

The code for the file directory listing operation on the server's side is as follows:

```
case 'L': //-- Client list directory request

        if((dir = opendir(".")) != NULL)          //Open the current working directory
        {
                spdu.data[0] = '\0';
                while ((entry = readdir(dir)) != NULL)    //Add each file name to the list
                {
                        strcat(spdu.data, entry->d_name);
                        strcat(spdu.data, " ");
                }

                closedir(dir);
        }

        spdu.type = 'l';
        send(sd, (char *)&spdu, sizeof(spdu) ,0);
        printf("List of items in current directory sent.\n");

        break;
```

This operation performs after an 'L' type PDU is received from the client. It requires the use of functions and data-types found in the "dirent.h" library. A directory (DIR) type named dir is created, and assigned a value based on the opendir() function. Upon failure, the opendir() function returns NULL (meaning that the directory doesn't exist); however, since "." is specified as the path of opendir(), the current directory will be assigned to 'dir'. A directory entry (dirent) datatype was also created, and named 'entry' (see Appendix for all of the code). A call to the readdir() function is made, which reads a single item of the specified directory and returns it as a directory entry type. The d_name (item name) of each entry is concatenated with the data[ ] character array (of the PDU to be sent) to form a string that contains the names of all the items in the current working directory. Finally, the PDU is sent to the client as an 'l' type, to indicate that it contains said list.

Conclusion

The file transfer project proved to be a success in many aspects. The implementation of all required functionality of both server and client were completed successfully. The project provided several challenges to overcome, but the result was that a successful implementation yielded an increase in knowledge about programming, especially with regards to socket program design using the C language. Such knowledge and skills developed during the project will serve to be useful in assignments and projects that may be encountered in the future.

Appendix

Server.c:

```
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/signal.h>
#include <netinet/in.h>
#include <stdlib.h>
#include <string.h>
#include <netdb.h>
#include <stdio.h>
#include <time.h>
#include <sys/stat.h>
#include <dirent.h>

#define BUFLEN 256                  /* buffer length */
#define SERVER_TCP_PORT 3000        /* well-known port */

int serve(int);
void reaper(int);

int main(int argc, char *argv[]) {
       int       sd, new_sd, port, client_len;
       struct sockaddr_in client, server; // client addr

       switch(argc){
              case 1:
                     port = SERVER_TCP_PORT;
                     break;
              case 2:
                     port = atoi(argv[1]);
                     break;
              default:
                     fprintf(stderr, "Usage: %d [port]\n", argv[0]);
                     exit(1);
              }

       /* Create a stream socket   */
       if ((sd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
              fprintf(stderr, "Can't creat a socket\n");
              exit(1);
       }

       /* Bind an address to the socket   */
       bzero((char *)&server, sizeof(struct sockaddr_in));
       server.sin_family = AF_INET;
       server.sin_port = htons(port);
       server.sin_addr.s_addr = htonl(INADDR_ANY);
       if (bind(sd, (struct sockaddr *)&server, sizeof(server)) == -1){
              fprintf(stderr, "Can't bind name to socket\n");
              exit(1);
       }

       /* queue up to 5 connect requests  */
```

```
        listen(sd, 5);

        (void) signal(SIGCHLD, reaper);

        while(1)
        {
          client_len = sizeof(client);
          new_sd = accept(sd, (struct sockaddr *)&client, &client_len);
          printf("Server created.\n\n");

          if(new_sd < 0)
          {
            fprintf(stderr, "Can't accept client \n");
            exit(1);
          }
          switch (fork())
          {
          case 0:             /* child */
               (void) close(sd);
               exit(serve(new_sd));
          default:            /* parent */
               (void) close(new_sd);
               break;
          case -1:
               fprintf(stderr, "fork: error\n");
          }
    }
}

int serve(int sd)
{
        struct stat fstat;          //Used for file statistics
        int n, bytes_to_write, i, j;
        DIR *dir;                   //Directory
        struct dirent *entry;              //Directory entry

        struct PDU
        {
               char type;
               int  length;
               char data[BUFLEN];

        } rpdu, spdu;


 while(1)
{
    rpdu.type = 'X'; //Set PDU type to unknown. This avoids looping errors caused by
                       //the switch statement

    //Receive the client's choice of action
    recv(sd, (char*)&rpdu, sizeof(rpdu), 0);
    printf("Received a %c type PDU.\n", rpdu.type);
 //Perform the requested operation based on their choice
 switch (rpdu.type)
 {
  case 'D': //-- Client download request;
  printf("Client has requested to download a file.\n");
  FILE *fs = fopen(rpdu.data,"r");

  if (fs == NULL) //-- Error Reporting
  {
      fprintf(stderr, "ERROR: File %s not found on Server.\n", rpdu.data);
      spdu.type = 'E';
      sprintf(spdu.data, "Unfortunately, %s was not found on the server.", rpdu.data);
      send(sd,(char *)&spdu, sizeof(spdu),0);
  }


 else
 {
     lstat(rpdu.data,&fstat);
     int bytes_to_read = fstat.st_size;
     spdu.length = fstat.st_size;
     spdu.type = 'R';
     send(sd, (char *)&spdu, sizeof(spdu), 0);
     printf("%s was found in the server.\nBeginning file Transfer.....\n", rpdu.data);

     printf ("Size of the file  = %d bytes \n", bytes_to_read);

     while(bytes_to_read > 0)
     {
           n = fread(spdu.data, sizeof(char), bytes_to_read, fs);
           bytes_to_read -= n;
           spdu.type = 'F';
           spdu.length = fstat.st_size;
           send(sd, (char *)&spdu, sizeof(spdu) ,0);

           if(bytes_to_read == 0)
           {
                  fclose(fs);
                  printf("File sent successfully.\n");
           }
      }
  }

  break;

case 'U': //-- Client upload request
     printf("Client had requested to upload a file named : %s.\n", rpdu.data);
     FILE *fr = fopen(rpdu.data, "w");

     if (fr == NULL) //-- Error Reporting
     {
            fprintf(stderr, "Failed to create file.\n");
            spdu.type = 'E';
            sprintf(spdu.data,"Unfortunately, %s could not be created on the
            server.\n",rpdu.data);
            send(sd,(char *)&spdu, sizeof(spdu) ,0);
            break;
     }
```

```
      else
      {
              spdu.type = 'R';
              send(sd, (char *)&spdu, sizeof(spdu), 0);   //Notify the client that the
                                                          //server is ready

              recv(sd, (char *)&rpdu, sizeof(rpdu), 0);  //Receive an F-type PDU with
                                                         //the file length
              bytes_to_write = rpdu.length;

              while  (bytes_to_write > 0)
              {
                      n = fwrite(rpdu.data, sizeof(char), bytes_to_write, fr);
                      bytes_to_write -= n;

                      if(bytes_to_write == 0)
                      {
                              fclose(fr);
                              printf("File uploaded successfully.\n");
                      }
              }
      }
      break;


case 'P': //-- Client change directory request

      printf("Client has requested a change of directory to: %s\n", rpdu.data);
      char *directory = rpdu.data;      //Directory string was received in the first
                                        //recv() call

      if(chdir(directory) == -1)
      {
              spdu.type = 'E';
              sprintf(spdu.data,"The directory: '%s' does not exist\n",rpdu.data);
              send(sd, (char *)&spdu, sizeof(spdu) ,0);
              break;
      }

      else
      {
              spdu.type = 'R';
              send(sd, (char *)&spdu, sizeof(spdu) ,0);
      }

      break;
```

```
case 'L': //-- Client list directory request

       if((dir = opendir(".")) != NULL)          //Open the current working directory
       {
              spdu.data[0] = '\0';
              while ((entry = readdir(dir)) != NULL)    //Add each file name to the list
              {
                     strcat(spdu.data, entry->d_name);
                     strcat(spdu.data, " ");
              }

              closedir(dir);
       }

       spdu.type = 'l';
       send(sd, (char *)&spdu, sizeof(spdu) ,0);
       printf("List of items in current directory sent.\n");

       break;

  default: //-- Default case

       fprintf(stderr, "Client made an invalid choice or chose to quit.");
       close(sd);
       exit(0);

             }
       }

       return(0);
}


/*     reaper        */
void   reaper(int sig)
{
       int    status;
       while(wait3(&status, WNOHANG, (struct rusage *)0) >= 0);
}
```

Client.c:

```
#include <sys/types.h>
#include <sys/stat.h>
#include <errno.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>

#define SERVER_TCP_PORT 3000      /* well-known port */
#define BUFLEN          256       /* buffer length */

int main(int argc, char **argv)
{
       int    n, i, bytes_to_read, bytes_to_write;
       int    sd, port, choice;
       struct hostent              *hp;
       struct sockaddr_in server;
       char   *host, *bp;
       struct stat fstat;
       struct PDU
       {
              char type;
              int  length;
              char data[BUFLEN];

       } rpdu, spdu;

       switch(argc){
       case 2:
              host = argv[1];
              port = SERVER_TCP_PORT;
              break;
       case 3:
              host = argv[1];
              port = atoi(argv[2]);
              break;
       default:
              fprintf(stderr, "Usage: %s host [port]\n", argv[0]);
              exit(1);
       }

       /* Create a stream socket   */
       if ((sd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
              fprintf(stderr, "Can't creat a socket\n");
              exit(1);
       }

       bzero((char *)&server, sizeof(struct sockaddr_in));
       server.sin_family = AF_INET;
       server.sin_port = htons(port);
       if (hp = gethostbyname(host))
         bcopy(hp->h_addr, (char *)&server.sin_addr, hp->h_length);
       else if ( inet_aton(host, (struct in_addr *) &server.sin_addr) ){
         fprintf(stderr, "Can't get server's address\n");
         exit(1);
       }

       /* Connecting to the server */
       if (connect(sd, (struct sockaddr *)&server, sizeof(server)) == -1){
         fprintf(stderr, "Can't connect \n");
         exit(1);
       }


while(1)
       {
              printf("\nWhat would you like to do?\n");
              printf("1. Download a file\n2. Upload a file\n3. Change file directory\n4.
              List current directory\n5. Quit\n");
              //Save client's choice
              scanf("%d", &choice);

  switch(choice)
  {
       case 1: //-- Download request

              printf("Name of the file you would like to download:\n");
              n = read(0, spdu.data, BUFLEN);     //Store filename inside the send PDU
              spdu.type = 'D';
              spdu.data[n-1] = '\0';              //Terminate filename string

              printf("You have requested the following file: %s \n", spdu.data);

              spdu.length = strlen(spdu.data);
              send(sd, (char *)&spdu, sizeof(spdu), 0);

              //Check if there was an error
              recv(sd, (char *)&rpdu, sizeof(rpdu), 0);
              if (rpdu.type == 'E')
              {

              printf("An error occurred. Message from the server: %s\n", rpdu.data);

              }
              else if(rpdu.type == 'R')  //Begin download
              {
                     FILE *fr = fopen(spdu.data, "w");


                     bytes_to_write = rpdu.length;     //Get file size from length field
                                                       //of the received PDU

                     while(bytes_to_write >  0)
                     {
                            recv(sd, (char *)&rpdu, sizeof(rpdu), 0); //r2
                            n = fwrite(rpdu.data, sizeof(char), bytes_to_write, fr);
                            bytes_to_write -= n;
```

```
                        if(bytes_to_write == 0)
                        {
                               fclose(fr);
                               printf("File downloaded successfully.\n");
                        }
                 }
          }

          break;         //return to choice selection menu

    case 2: //-- Upload request

      printf("Name of the file you would like to upload:\n");
      n = read(0, spdu.data, BUFLEN);          //Store filename inside the send PDU
      spdu.type = 'U';
      spdu.data[n-1] = '\0';                   //Terminate filename string

      printf("You requested to upload: %s\n", spdu.data);
      spdu.length = strlen(spdu.data);      //Set the length field to the filename length
      send(sd, (char *)&spdu, sizeof(spdu), 0);

      recv(sd, (char *)&rpdu, sizeof(rpdu), 0);
      if (rpdu.type == 'E')                           //Check if there was an error
      {
             printf("An error occurred. Message from the server: %s\n", rpdu.data);
      }
      else if(rpdu.type == 'R')  //Begin upload
      {
             printf("Server is ready. Beginning upload...\n");
             FILE *fs = fopen(spdu.data, "r");
             lstat(spdu.data,&fstat);
             bytes_to_read = fstat.st_size;
             spdu.length = fstat.st_size;

             if (fs == NULL)
             {
                    printf("Cannot upload because the file does not exist.\n");
                    break;
             }
             while(bytes_to_read > 0)
             {
                    n = fread(spdu.data, sizeof(char), bytes_to_read, fs);
                    spdu.type = 'F';
                    spdu.length = bytes_to_read;
                    send(sd, (char *)&spdu, sizeof(spdu), 0);
                    bytes_to_read -= n;

                    if(bytes_to_read == 0)
                    {
                    fclose(fs);
                    printf("File uploaded successfully.\n");
                    }
              }
         }

      break;         //return to choice selection menu
    case 3: //-- Change file directory request

         printf("Directory to change to:\n");
         n = read(0, spdu.data, BUFLEN);           //Store directory name inside the send
                                                    //PDU
         spdu.type = 'P';
         spdu.data[n-1] = '\0';                            //Terminate directory name string
         send(sd, (char *)&spdu, sizeof(spdu), 0);

         recv(sd, (char *)&rpdu, sizeof(rpdu), 0);

         if (rpdu.type == 'E')
         {
                printf("An error occurred. Message from the server: %s", rpdu.data);
         }

         else if(rpdu.type == 'R')
         {
                printf("Directory changed. Current directory: %s\n", spdu.data);
         }

         break;        //return to choice selection menu


    case 4: //-- List directory request

         spdu.type = 'L';
         send(sd, (char *)&spdu, sizeof(spdu), 0); //Send L-type PDU

         recv(sd, (char *)&rpdu, sizeof(rpdu), 0);

         if(rpdu.type == 'l')
         {
                printf("\nThe files in the current directory are:\n%s\n\n", rpdu.data);
         }

         break;        //return to choice selection menu

    case 5: //-- Quit
         printf("You've chosen to quit. Goodbye\n");
         close(sd);
         exit(0);

         default:
         printf("Invalid choice.\n");
         break;

         }
    }
}
```