Implementation of project "Part2Part" as scaled functionalities of the Peer-to-Peer architecture

Boboc Raul Madalin[1[2B4]]

"Alexandru Ioan Cuza" University
Faculty of Computer Science
raul05madalin@gmail.com

Abstract. A Peer-to-Peer (P2P) service is a decentralized platform whereby two individuals interact directly with each other, without intermediation by a third party. A generalization of P2P is extending the number of individuals to as many as there are connected to a network. The "Part2Part" project is therefore a generalization of P2P with the limited functionalities of requesting and sending files (through a graphic interface to communicate with users and using databases for file management) between individuals of the network.

1 Intoduction

1.1 Motivation

Mobility and adaptability are vital keys for the modern technology, therefore, the ability of having any files from a machine on another with ease and speed proves to be of huge importance.

This project has been chosen in order for the functionalities and difficulties of such implementations to be discovered and reasearched. The project involves knowledgement regarding graphic interfaces, databases, file management, networks and such complexity consists of a great challenge for a junior developer.

1.2 Implementation

Four notions had to be implemented: graphic interfaces, databases, file management and networks, the latter consisting of a client and a server functionalities, both available for each machine of connected to the network.

One machine's client is able to request files using the server (same machine's or other's). One machine's server is able to check if the requested file is shareable and to send it back or to tell the same machine's client to futher request the file until all the requests are met or are notified of failure.

2 Technologies

2.1 Graphic interface

A well built graphic interface has an important role in helping the user by performing repetitive instrucions in the backend and reducing human error in

inputs. Graphic interfaces are implemented in GTK version 3 and consists of a clean screen, buttons, and text entries. Even such a minimalist approach is more helpful for the user than in-line terminal inputs.

2.2 Database

A organized database can easily manage repetitive information, such as a file's name, format, size and availabilty. The project uses SQLite for database management and keeps track of those exact uniquely identifiable set of file properties. Text or JSON files could also been used but prove themselves to be slow and inefficient compared to databases.

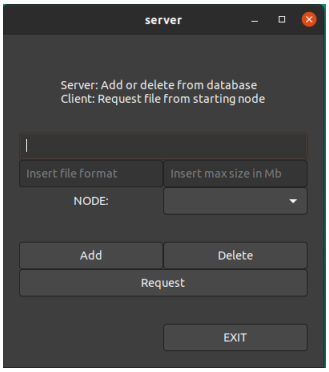
2.3 Network

The TCP technology is used for the network communication for it's guarantee of file's integrity, a vital point considering the objective of the project is file transfering between different connected machines. Therefore, UDP technology represents a bad approach as missing network packets resulting in incomplete files meaning failed transfers. Even more, the servers will be using threads so they can respond fast to as many requests (requests to not be denied).

3 Architecture

3.1 Graphic interface

Graphic interfaces are playing the role of guiding the user through the possible available options and also be notified of the state of the tasks running. The server and the client share the same graphic interface, combining all the functionalitites needed by both of them.



(a) Graphic interface

3.2 Network

Networking is the key element of the project. It has the role of allowing different machines to communicate with each other (request, send and receive files). The figure below shows how each node can communicate with each other.

Every machine (N) has a client functionality (C) and a server functionality (S), with the client it can request and receive files, with the server it can send files. If the requested node cannot deliver the file through his server (file missing or not shareable), the node will use its client to further request the file until it's found and delivered or is not found at all and the first node receives a corresponding message.

To facilitate the possibility of numerous amounts of requests on the server, it will be implemented using threads.

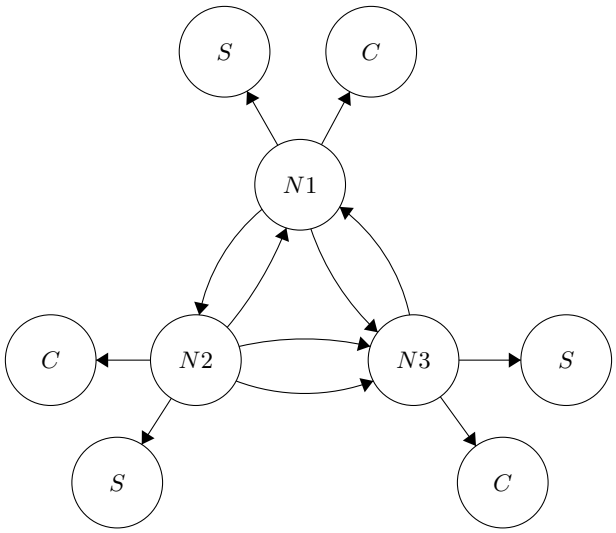


Fig. 2: Network

3.3 Server

Users interact with the server (S) through the graphic interface (GUI) only. They can introduce file names (NAME), formats (FMT) and sizes (SIZE) to control the database (DB). The network (NET) can either send statuses to the screen (SCRN) or request files from the database which will search for the requested file in machine's folder (FILES) and the process goes backward after the file is found and sent.

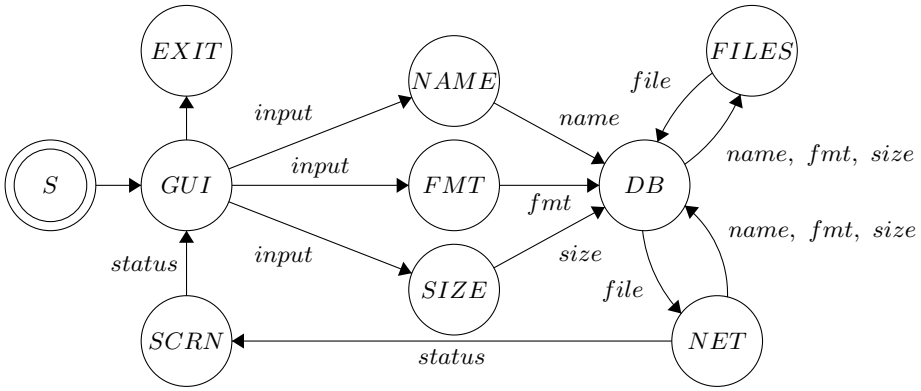


Fig. 3: Server

3.4 Client

Users interact with the client (C) through the graphic interface (GUI) only. They can introduce file names (NAME), formats (FMT) and sizes (SIZE) to reqeust files(RQST) from a certain node (NODE). The network (NET) can either send statuses to the screen (SCRN) or be requested of files which will be received in machine's folder (FILES).

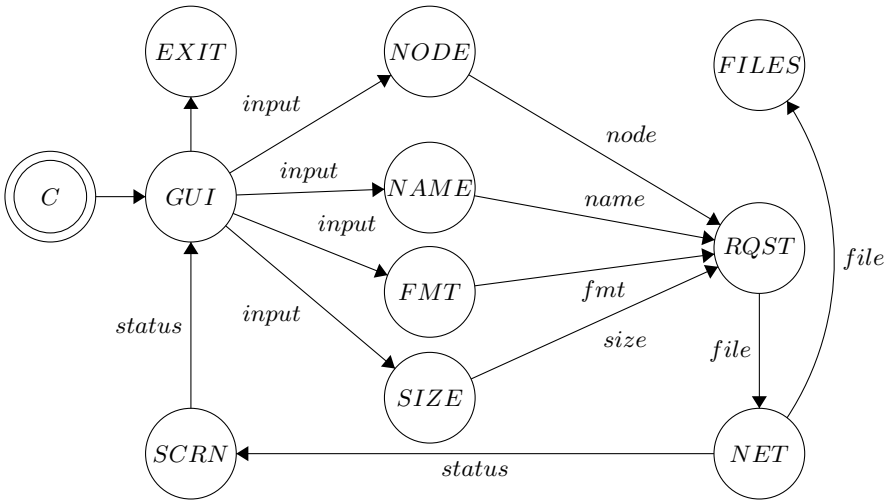


Fig. 4: Client

3.5 Database

Databases are keeping track of each of the machines' shareable and non-shareable files. Files have the following properties: id, name, format, size and availability. One user can add a file to the database (by mentioning the name, format and size, availability being by default shareable), mark it as shareable (using name) or mark it as non-shareable (using name).

4 Implementation

4.1 Graphic interface

The graphic interface is implemented using a grid of elements: labels, entries and buttons.

```
GtkApplication *app;

GtkWidget* window;
GtkWidget* grid;
GtkWidget* blank1;      // label
GtkWidget* screen;      // label
GtkWidget* blank2;      // label
GtkWidget* file;        // entry
GtkWidget* add;         // button
GtkWidget* delete;      // button
GtkWidget* blank3;      // label
GtkWidget* format;      // entry
GtkWidget* size;        // entry
GtkWidget* blank4;      // label
GtkWidget* exit_server; // button
GtkWidget* client_btn;
GtkWidget* nodes;
GtkWidget* node_label;
```

Entries are used for getting the file details from the user, while with the buttons it is decided what to be done next with the input.

```
window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_window_set_default_size(GTK_WINDOW(window), 400, 350)
    ;
    gtk_window_set_position(GTK_WINDOW (window),
    GTK_WIN_POS_CENTER_ALWAYS);
    g_signal_connect(window, "destroy", G_CALLBACK(destroy),
    NULL);
    gtk_container_set_border_width(GTK_CONTAINER(window), 20)
    ;

grid = gtk_grid_new ();
    gtk_container_add (GTK_CONTAINER (window), grid);
    gtk_grid_set_column_homogeneous(GTK_GRID (grid), TRUE);
```

```
10      gtk_grid_set_row_homogeneous(GTK_GRID (grid), TRUE);
11
12  blank1 = gtk_label_new ("");
13      gtk_grid_attach (GTK_GRID (grid), blank1, 0, 0, 2, 1);
14
15  screen = gtk_label_new ("SCREEN");
16      gtk_grid_attach (GTK_GRID (grid), screen, 0, 1, 2, 1);
17
18  blank2 = gtk_label_new ("");
19      gtk_grid_attach (GTK_GRID (grid), blank2, 0, 2, 2, 1);
20
21  file = gtk_entry_new();
22      gtk_grid_attach (GTK_GRID (grid), file, 0, 3, 2, 1);
23      gtk_entry_set_placeholder_text(GTK_ENTRY(file), "Insert
        file name");
24
25  format = gtk_entry_new();
26      gtk_grid_attach (GTK_GRID (grid), format, 0, 4, 1, 1);
27      gtk_entry_set_placeholder_text(GTK_ENTRY(format), "Insert
         file format");
28
29  size = gtk_entry_new();
30      gtk_grid_attach (GTK_GRID (grid), size, 1, 4, 1, 1);
31      gtk_entry_set_placeholder_text(GTK_ENTRY(size), "Insert
        max size in Mb");
32
33  blank3 = gtk_label_new ("");
34      gtk_grid_attach (GTK_GRID (grid), blank3, 0, 6, 2, 1);
35
36  add = gtk_button_new_with_label("Add");
37      gtk_grid_attach (GTK_GRID (grid), add, 0, 7, 1, 1);
38      g_signal_connect (add, "clicked", G_CALLBACK (
        get_file_name), file);
39      g_signal_connect (add, "clicked", G_CALLBACK (
        get_file_format), format);
40      g_signal_connect (add, "clicked", G_CALLBACK (
        get_file_size), size);
41      g_signal_connect (add, "clicked", G_CALLBACK (
        database_add), file);
42
43  delete = gtk_button_new_with_label("Delete");
44      gtk_grid_attach (GTK_GRID (grid), delete, 1, 7, 1, 1);
45      g_signal_connect (delete, "clicked", G_CALLBACK (
        get_file_name), file);
46      g_signal_connect (delete, "clicked", G_CALLBACK (
        get_file_format), format);
47      g_signal_connect (delete, "clicked", G_CALLBACK (
        database_delete), file);
48
49  blank4 = gtk_label_new ("");
```

```
50      gtk_grid_attach (GTK_GRID (grid), blank4, 0, 9, 2, 1);
51
52  exit_server = gtk_button_new_with_label("EXIT");
53      gtk_grid_attach (GTK_GRID (grid), exit_server, 1, 10, 1,
        1);
54      g_signal_connect (exit_server, "clicked", G_CALLBACK (
        destroy), file);
55
56  client_btn = gtk_button_new_with_label("Request");
57      gtk_grid_attach (GTK_GRID (grid), client_btn, 0, 8, 2, 1)
        ;
58      g_signal_connect (client_btn, "clicked", G_CALLBACK (
        get_file_name), file);
59      g_signal_connect (client_btn, "clicked", G_CALLBACK (
        get_file_format), format);
60      g_signal_connect (client_btn, "clicked", G_CALLBACK (
        get_file_size), size);
61      g_signal_connect (client_btn, "clicked", G_CALLBACK (
        get_network_node), nodes);
62      g_signal_connect (client_btn, "clicked", G_CALLBACK (
        client_service), file);
63
64  node_label = gtk_label_new ("NODE:");
65      gtk_grid_attach (GTK_GRID (grid), node_label, 0, 5, 1, 1)
        ;
66
67  nodes = gtk_combo_box_text_new();
68      gtk_grid_attach (GTK_GRID (grid), nodes, 1, 5, 1, 1);
69      gtk_combo_box_text_append_text(GTK_COMBO_BOX_TEXT(nodes),
         "1");
70      gtk_combo_box_text_append_text(GTK_COMBO_BOX_TEXT(nodes),
         "2");
71      gtk_combo_box_text_append_text(GTK_COMBO_BOX_TEXT(nodes),
         "3");
72      gtk_combo_box_text_append_text(GTK_COMBO_BOX_TEXT(nodes),
         "4");
73      gtk_combo_box_text_append_text(GTK_COMBO_BOX_TEXT(nodes),
         "5");
```

4.2 Database

The database is used by the server so it can know rather if a file is or not shareable. A file's availability can be changed or a new file can be added to the database.

```
1  /* Add button function */
2  void add_file()
3  {
4      sqlite3 *db;
```

```
    char *err_msg = 0;
    sqlite3_stmt *res;

    /* Open database */
    int rc = sqlite3_open("test.db", &db);

    if (rc != SQLITE_OK) {

        fprintf(stderr, "Cannot open database: %s\n",
                sqlite3_errmsg(db));
        sqlite3_close(db);

        return 1;
    }

    /* Check if file exists */
    char sql[128];
    sprintf(sql, "select count(*) from test where name = '%s
    ';", file_name);

    rc = sqlite3_exec(db, sql, addfilecallback, 0, &err_msg);

    if (file_exists) {
        /* Update availability */
        sprintf(sql, "update test set available = 1 where
    name = '%s';", file_name);
        rc = sqlite3_exec(db, sql, 0, 0, &err_msg);
    }
    else {
        /* Add file to the database */
        sprintf(sql, "insert into test values(%d, '%s', '%s',
     %d, 1);", db_size + 1, file_name, file_format, file_size
    );
        rc = sqlite3_exec(db, sql, 0, 0, &err_msg);
    }

    if (rc != SQLITE_OK ) {

        fprintf(stderr, "Failed to select data\n");
        fprintf(stderr, "SQL error: %s\n", err_msg);

        sqlite3_free(err_msg);
        sqlite3_close(db);

        return 1;
    }

    sqlite3_close(db);
}
```

```
/* Checks if database entry exists */
int addfilecallback(void *NotUsed, int argc, char **argv,
                    char **azColName) {

    NotUsed = 0;

    for (int i = 0; i < argc; i++) {
        if (strcmp(argv[i], "1") == 0)
            file_exists = TRUE;
        else
            file_exists = FALSE;
    }

    printf("\n");

    return 0;}
```

4.3 File transfer

The file transfering is set between a server (sender) and a client (receiver). It starts with the file size for checkups and then pieces of the file. The function does not give importance to the file format as long as the sent and received file are of the same format (pdf(S) - pdf(C) is valid; pdf(S) - jpg(C)) is not valid).

```
/* Sending file size */
    len = send(peer_socket, transfer_file_size, sizeof(
    transfer_file_size), 0);
    if (len < 0)
    {
        sprintf(screen_text, "Error on sending greetings -->
    %s", strerror(errno));
        gtk_label_set_text(screen, screen_text);
        exit(EXIT_FAILURE);
    }

    sprintf(screen_text, "Server sent %d bytes for the size\n
    ", len);
    gtk_label_set_text(screen, screen_text);

    offset = 0;
    remain_data = file_stat.st_size;

    /* Sending file data */
    while (((sent_bytes = sendfile(peer_socket, fd, &offset,
    BUFSIZ)) > 0) && (remain_data > 0))
    {
        sprintf(screen_text, "1. Server sent %d bytes from
    file's data, offset is now : %d and remaining data = %d\n
    ", sent_bytes, offset, remain_data);
```

```
        gtk_label_set_text(screen, screen_text);
        remain_data -= sent_bytes;
        sprintf(screen_text, "2. Server sent %d bytes from
    file's data, offset is now : %d and remaining data = %d\n
    ", sent_bytes, offset, remain_data);
        gtk_label_set_text(screen, screen_text);
    }
    sprintf(screen_text, "Done");
    gtk_label_set_text(screen, screen_text);
```

4.4 Network

The network is split up into two parts, the server functionality and the client functionality. The server will be implemented using threads because of the possibilty of a high amount of requests.

```

/* Server - concurrent using threads*/
int server()
{
    struct sockaddr_in server;  // structura folosita de
    server
  struct sockaddr_in from;
  int nr;        //mesajul primit de trimis la client
  int sd;        //descriptorul de socket
  int pid;
  pthread_t th[100];    //Identificatorii thread-urilor care
    se vor crea
    int i=0;

  /* crearea unui socket */
  if ((sd = socket (AF_INET, SOCK_STREAM, 0)) == -1)
    {
      perror ("[server]Eroare la socket().\n");
      return errno;
    }
  /* utilizarea optiunii SO_REUSEADDR */
  int on=1;
  setsockopt(sd,SOL_SOCKET,SO_REUSEADDR,&on,sizeof(on));

  /* pregatirea structurilor de date */
  bzero (&server, sizeof (server));
  bzero (&from, sizeof (from));

  /* umplem structura folosita de server */
  /* stabilirea familiei de socket-uri */
    server.sin_family = AF_INET;
  /* acceptam orice adresa */
    server.sin_addr.s_addr = htonl (INADDR_ANY);
```

```
32  /* utilizam un port utilizator */
33
34    int new_port = 5000;
35    server.sin_port = htons (new_port);
36
37  /* atasam socketul */
38  while (bind (sd, (struct sockaddr *) &server, sizeof (
      struct sockaddr)) == -1)
39    {
40        if (errno != EADDRINUSE)
41            return errno;
42        else {
43            new_port++;
44            server.sin_port = htons (new_port);
45        }
46    }
47  /* punem serverul sa asculte daca vin clienti sa se
      conecteze */
48  if (listen (sd, 2) == -1)
49    {
50      perror ("[server]Eroare la listen().\n");
51      return errno;
52    }
53
54    sprintf(screen_text, "%s%d", "Port: ", new_port);
55    update_screen();
56
57    sprintf(screen_text, "%s\n%s",  "Server: Add or delete
      from database",
58                                    "Client: Request file
      from starting node");
59    update_screen();
60
61  /* servim in mod concurent clientii...folosind thread-uri
      */
62  while (1)
63    {
64      int client;
65      thData * td; //parametru functia executata de thread
66      int length = sizeof (from);
67
68      printf ("[server]Asteptam la portul %d...\n",new_port);
69      fflush (stdout);
70
71      /* acceptam un client (stare blocanta pina la
      realizarea conexiunii) */
72      if ( (client = accept (sd, (struct sockaddr *) &from, &
      length)) < 0)
73    {
74      perror ("[server]Eroare la accept().\n");
```

```
75        continue;
76      }
77      GUI = 0;
78
79          /* s-a realizat conexiunea, se astepta mesajul */
80
81      // int idThread; //id-ul threadului
82      // int cl; //descriptorul intors de accept
83
84      td=(struct thData*)malloc(sizeof(struct thData));
85      td->idThread=i++;
86      td->cl=client;
87
88      pthread_create(&th[i], NULL, &treat, td);
89
90      }//while
91  }
92
93  static void *treat(void * arg) {
94      struct thData tdL;
95      tdL= *((struct thData*)arg);
96      printf ("\n[thread]- %d - Asteptam mesajul...\n", tdL.
        idThread);
97      fflush (stdout);
98      pthread_detach(pthread_self());
99      raspunde((struct thData*)arg);
100     GUI = 1;
101     /* am terminat cu acest client, inchidem conexiunea */
102     close ((intptr_t)arg);
103     return(NULL);
104
105 }
```

```
1   /* Client */
2   int client()
3   {
4     int sd;     // descriptorul de socket
5     struct sockaddr_in server;  // structura folosita pentru
        conectare
6         // mesajul trimis
7     int nr=0;
8     char buf[10];
9     int client_socket;
10          int len;
11          struct sockaddr_in remote_addr;
12          char buffer[BUFSIZ];
13          int file_size_c;
14          FILE *received_file;
15          int remain_data = 0;
16          char operation[10];
```

```
17          char received_name[100];
18
19  printf("[CLIENT]: thread %d\n", clients);
20
21  /* cream socketul */
22  if ((sd = socket (AF_INET, SOCK_STREAM, 0)) == -1)
23    {
24      perror ("Eroare la socket().\n");
25      return errno;
26    }
27
28    int on=1;
29  setsockopt(sd,SOL_SOCKET,SO_REUSEADDR,&on,sizeof(on));
30
31  /* umplem structura folosita pentru realizarea conexiunii
      cu serverul */
32  /* familia socket-ului */
33  server.sin_family = AF_INET;
34
35  /* portul de conectare */
36
37  int new_port;
38  if (new_request == 1) {
39    printf("[CLIENT]: new_request %d\n", new_request);
40    new_port = 4999 + network_node;
41    server.sin_port = htons (new_port);
42    if (connect (sd, (struct sockaddr *) &server,sizeof (
      struct sockaddr)) == -1) {
43          sprintf(screen_text, "%s%d", "Connecting to port: ",
      new_port);
44          update_screen();
45          if (errno != ECONNREFUSED) {
46              printf("[CLIENT]: %d\n", errno);
47              return errno;
48          }
49          else {
50              printf("[CLIENT]: ECONNREFUSED\n");
51              sprintf(screen_text, "%s", "Port unavailable...")
      ;
52              update_screen();
53              sprintf(screen_text, "%s\n%s",  "Server: Add or
      delete from database",
54                                              "Client: Request
      file from starting node");
55              update_screen();
56
57              new_request = 0;
58              network_node = 1;
59              first_port = new_port;
60              close(sd);
```

```
61              client();
62              pthread_exit(1);
63          }
64      }
65  }
66  else {
67      printf("[CLIENT]: linked_request %d\n", new_request);
68      new_port = first_port + network_node;
69      if (new_port >= 5005) new_port -= 5;
70      if (new_port == first_port) {
71          printf("[CLIENT]: not found port%d\n", new_port);
72          sprintf(screen_text, "%s\n%s",  "Server: Add or
    delete from database",
73                                          "Client: Request file
     from starting node");
74          update_screen();
75          new_request = 1;
76          close(sd);
77          pthread_exit(1);
78      }
79      server.sin_port = htons (new_port);
80      while (connect (sd, (struct sockaddr *) &server,sizeof (
    struct sockaddr)) == -1 && new_port < 5011 /* add
    avoidance first port*/) {
81          sprintf(screen_text, "%s%d", "Connecting to port: ",
    new_port);
82          update_screen();
83        if (errno != ECONNREFUSED) {
84          return errno;
85      }
86        else {
87          sprintf(screen_text, "%s", "Port unavailable...");
88          update_screen();
89              network_node++;
90              close(sd);
91              client();
92              pthread_exit(1);
93        }
94      }
95  }
96  if (new_port == first_port) {
97      printf("[CLIENT]: not found port\n");
98      sprintf(screen_text, "%s\n%s",  "Server: Add or delete
    from database",
99                                      "Client: Request file
    from starting node");
100     update_screen();
101     new_request = 1;
102     close(sd);
103     pthread_exit(1);
```

```
104   }
105   printf("[CLIENT]: found port\n");
106
107     sprintf(screen_text, "%s%d", "Connected to port: ",
      new_port);
108     update_screen();
109
110     sprintf(screen_text, "%s", "Sending request...");
111     update_screen();
112
113     sprintf(screen_text, "%s", "Waiting for response...");
114     update_screen();
115
116     if (write (sd, global_file_name, sizeof(global_file_name)
      ) <= 0)
117     {
118       perror ("[Client]Eroare la write() spre server.\n");
119       return errno;
120     }
121     sleep(1);
122     if (write (sd, global_file_format, sizeof(
      global_file_format)) <= 0)
123     {
124       perror ("[Client]Eroare la write() spre server.\n");
125       return errno;
126     }
127     sleep(1);
128     if (write (sd, global_file_size, sizeof(global_file_size)
      ) <= 0)
129     {
130       perror ("[Client]Eroare la write() spre server.\n");
131       return errno;
132     }
133     sleep(1);
134     pthread_mutex_unlock(&locked);
135
136     //printf("[Client] %s\n", file_name);
137
138     char client_response[20];
139     if (read (sd, client_response, sizeof(client_response))
      <= 0)
140             {
141               printf("[Thread %d]\n",sd);
142               perror ("Eroare la read() de la client.\n");
143
144             }
145             else printf("\n[SERVER] : %s\n", client_response)
      ;
146
147     sprintf(screen_text, "%s", client_response);
```

```
148     update_screen();
149
150     if (strcmp(client_response, "Found file") != 0) {
151         printf("\n[FUCK] : %s\n", client_response);
152         if (new_request == 1) first_port = new_port;
153         new_request = 0;
154         network_node++;
155         close(sd);
156         client(new_port);
157         pthread_exit(1);
158     }
159
160     sprintf(received_name, "%s.%s", global_file_name,
    global_file_format);
161     printf ("Mesajul a fost receptionat...%s\n",
    received_name);
162
163     /* Receiving file size */
164 }
```

5 Conclusions

In all its complexity, the project does never handle security regarding network connectivity. This aspect could be tackled by creating log files for servers so they can check if the client who sent the request is allowed to connect or will be denied.

References

1. Peer-to-Peer (P2P) Service, https://www.investopedia.com/terms/p/peertopeer-p2p-service.asp .
2. Seminar page, https://profs.info.uaic.ro/~computernetworks/files/NetEx/S12/ServerConcThread/servTcpConcTh2.c .
3. Seminar page, https://profs.info.uaic.ro/~computernetworks/files/NetEx/S12/ServerPreThread/cliTcpNr.c .
4. SQLite C, https://zetcode.com/db/sqlitec/ .
5. Send and receive file, https://stackoverflow.com/questions/11952898/c-send-and-receive-file .