FERHAT ELMAS
2006101102

CMPE 476 TERM PROJECT REPORT

The purpose of this project is to design a simple Application Layer Protocol (ALP) and implement it by Berkeley sockets programming. Moreover, ALP is operation on TCP.

General Properties of ALP

- ✓ ALP is a client – server protocol. There are multiple clients and there is a single server.
- ✓ ALP is specifically to be called Simple Mp3 Download Protocol (SMDP) and the server is Super Mp3 Server (SMS) and the client is Super Mp3 Client (SMC).
- ✓ SMS keeps two ascii files, namely master.txt and login.txt. Master file keeps information about Mp3 files. Typical record of this file has the unique ID (MID) of Mp3 file, the title of Mp3 file content (i.e. the name of the song) and the path to the actual Mp3 file. Actually, there is no path specified in master file becuase SMS saves Mp3 files into current working directory with their MIDs as names. Record structure is as follows:

MID (starts from 1 and increments for each MP3)	**Title (name of the song)**

For example, if MID of a song is 5, its actual Mp3 file will be in "5" file in the current working directory. Furthermore, there may be duplicates in titles because SMS require a title for each mp3 upload but doesn't check whether there is a mp3 with this title. Even if titles become same, they will be different records by differentiated by MIDs.

Login file keeps necessary information about registered users. Necessary information is composed of the user name and password. Moreover, this file as stated above is in ascii because this aim of this project is socket programming not security so there is no need to worry about keeping password in plain text. Record structure is as follows:

User name (Unique)	**Password (there may be duplicates)**

Moreover, Music files are saved in binary.

- ✓ Any client can anonymously browse the master file to see which Mp3 files exist and also retrieve a Mp3 file. Moreover, any client anonymously can inquire about the largest MID number.
- ✓ Any client can anonymously download Mp3 files. This can be accomplished by suppliying a MID, at random (SMS picks a MID for the user) or giving a MID range to download multiple Mp3 files in one time.
- ✓ Any registered client can add new Mp3 files to SMS.
- ✓ SMS accepts new registrations but these registrations are simple simulation because user enters a user name and password and if there are no user names that user enters and password meets certain requirements, user is registered then he can use these user name and password to upload Mp3 files.

FERHAT ELMAS
2006101102

General Assumptions

- ✓ 256 characters are sufficient for user name.
- ✓ 256 characters are sufficient for password.
- ✓ 256 characters are sufficient for Mp3 title.
- ✓ There is no path info for binary Mp3 files in master file since SMS saves Mp3 files into their MID files. A Mp3 that has MID 3 is saved into file 3.
- ✓ Registration isn't a real registration. Server requires only user name and password. If user provides a unique user name and strong password, he is registered and can upload new Mp3 files.
- ✓ I didn't go deep into bits and I used bytes so the smallest atomic unit of protocol is one byte.
- ✓ I tried to synchronize concurrent requests at the server but there may be some unforeseen bugs under heavy load.

Simple Mp3 Download Protocol (SMDP) Structure

- ✓ There are mainly three type operation can be done by this protocol
  - Registration
  - Browse (contains download and surfing master file)
  - Put

Registration – Client

Type	User name Length	User name	Password Length	Password

- ✓ Type is zero for registration (1 byte).
- ✓ User name length is actually an integer that is written in base 256 (4 bytes) . Actually, user name can 256 characters at most so we need only one byte but I write a generic function that is also used for MIDs. Therefore, it is 4 bytes.
- ✓ Username is a string that has at most 256 length.
- ✓ The same discussion is also valid for password length.
- ✓ Password is a string that has at most 256 length.

Registration – Server

Type

- ✓ Type is the response of the registration operation.

Type	Meaning
0	Registration is successful
1	Server internal error
2	Invalid characters in user name
3	Invalid characters in password
4	Weak password
5	User name is already exists
Other than above	Unspecified Response

Browse – Client

There are multiple options in this operation:

- ✓ Display master
- ✓ Get by MID
- ✓ Get by MID range
- ✓ Get by random
- ✓ Get the biggest MID

Display Master - Client

Type

- ✓ There is no need to send further information. (1 byte) Type is 1 for this option.

Display Master – Server

Type	**Response Length**	**MID**	**Title Length**	**Title**

- ✓ Type is a flag to check whether operation is successfully done, 0 for success and 1 for server internal error.
- ✓ MID, Title Length and Title make body of a record and response length specifies the total length of these records.
- ✓ Response length is 4 bytes.
- ✓ MID is 4 bytes.
- ✓ Title length is 4 bytes.
- ✓ Title can be at most 256 bytes.

Get by MID – client

Type	**MID**

- ✓ Type is needed for specifing operation (1 byte). Type is 2 for this option.
- ✓ MID specifies the requested Mp3 file (4 bytes).

Get by MID – server

Type	**Response Length**	**Title Length**	**Title**	**Mp3 File**

- ✓ Type is a flag (1 byte) to check whether operation is successfully done, 0 for success and 1 for server internal error, 2 for MID out of range.
- ✓ Response Length specifies the total length of the file plus title and 4 bytes for its length (4 bytes).
- ✓ Title Length is 4 bytes.
- ✓ Title is a string whose length is specified above integer.
- ✓ Mp3 File is the binary file that the user requested.

Get by MID range – client

- ✓ Firstly, requests the biggest MID and also knows the minimum since MIDs start from 1. Then, client adjusts range bounds and sequentially calls Get by MID function. Therefore, there is no protocol design for this operation because client can do all things by using other operations.

Get by MID range – server

- ✓ There is no protocol design for server, client calls other operations to simulate this operations.

Get by Random – client

Type

- ✓ Type specifies the operation (1 byte). Type is 3 for this option.

Get by Random – server

Type	Response Length	MID	Title Length	Title	Mp3 File

- ✓ Type is a flag (1 byte) to check whether operation is successfully done, 0 for success and 1 for server internal error, 2 for no music in server.
- ✓ Response length is the total length of the the rest of the response (4 bytes).
- ✓ MID is the id of the chosen Mp3 by server (4 bytes).
- ✓ Title length is the length of the title (4 bytes).
- ✓ Title is a string that is the name of the song.
- ✓ Mp3 file is the binary file that is chosen by the SMS.

Get the biggest MID – client

Type

- ✓ Type specifies the operation (1 byte). Type is 4 for this option.

Get the biggest MID – server

Type	MID (biggest)

- ✓ Type is a flag (1 byte) to check for whether operation is successfully done, 0 for success and 1 for server internal error.
- ✓ MID is the biggest id in master file (4 bytes).

Put – client

Type	Request Length	User name Length	User name	Password Length	Password	Title Length	Title	Mp3 File

- ✓ Type specifies the operation (1 byte). Type is 5 for this option.
- ✓ Request Length is the total length of the request, sum of the rest (4 bytes).
- ✓ User name length is an integer that specifies the length of the username (4 bytes).
- ✓ User name is the necessary user name to upload.
- ✓ Discussion of user name (length) is also valid for password length and password.
- ✓ Discussion of user name (length) is also valid for title length and title.
- ✓ Mp3 file is the binary file that user wants to upload to SMS.

Put – server

Type

- ✓ Type is a flag (1 byte) to check the operation success.
  - 0 for success
  - 1 for server internal error
  - 2 for authentication failure

Advantages and Disadvantages

Advantages

- ✓ This protocol is designed for just music upload and download so it is optimized, efficient.
- ✓ Protocol signaling is small so protocol overhead is negligible.

- ✓ There is only one byte traffic in
  - Registration server messages
  - Display master client messages
  - Get by random client messages
  - Get the biggest MID client messages
  - Put server messages
- ✓ There is only five bytes traffic in
  - Get by MID client messages
  - Get the biggest MID server messages
- ✓ Users can download multiple Mp3 files at one time by specifing a range.
- ✓ Users can download Mp3 files anonymously.
- ✓ Server forks a child process and gives the request to this child process and parent itself returns instatenously to accept new requests. Moreover, instantenous requests up to five can be served by the server.
- ✓ Server creates a shared memory to be used by itself and its child processes. In this shared memory, various global variables are kept to fasten the service of the request such as the biggest MID becuase there is a global that keeps the biggest MID and when a request has come to query biggest MID, there is no need to open master file and look for MIDs, instead of child process just sends the content of the global.

Disadvantages

- ✓ There is a lot of file and socket I/O operations, and memory allocations. There may be some errors in these operations. I tried to check for all of them. In my tests, I didn't see any unforeseen error point but some errors may not be produced.
- ✓ I tried to synchronize file IO operations and access for globals in the shared memory. It was successful, but some errors can be seen under heavy load.

Execution

- ✓ I added a makefile so just run make is sufficient. Moreover, I tried to use POSIX standard functions so there is no need to take further steps to install sometihng.
- ✓ gcc -Wall client.c -o client
- ✓ gcc -Wall server.c -o server

Conclusion

- ✓ This project is my first network programming so I have learnt a lot from it.
  - Socket programming
  - Buffered usage, especially for C. If I would do the project in Java, I wouldn't learn this.
  - Synchronization
- ✓ Project specification is too abstract. I think, details should be specified. I hope that my assumptions don't harm the my overall point.

Appendix – Code

server.c

```
/* -*- Mode: C; indent-tabs-mode: t; c-basic-offset: 4; tab-width: 4 -*- */

/*

 * main.c

 * Copyright (C) Ferhat Elmas 2011 <elmas-ferhat@gmail.com>

 *

 * SMS is free software: you can redistribute it and/or modify it

 * under the terms of the GNU General Public License as published by the

 * Free Software Foundation, either version 3 of the License, or

 * (at your option) any later version.

 *

 * SMS is distributed in the hope that it will be useful, but

 * WITHOUT ANY WARRANTY; without even the implied warranty of

 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

 * See the GNU General Public License for more details.

 *

 * You should have received a copy of the GNU General Public License along

 * with this program.  If not, see <http://www.gnu.org/licenses/>.

 */


#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/socket.h>
```

```
#include <sys/mman.h>

#include <netinet/in.h>

#include <time.h>

#include <fcntl.h>

#include <unistd.h>


int *max_music_id;

int *music_name_total_length;


char *masterFilePath = NULL;

char *loginFilePath = NULL;

char *musicDirectory = NULL;


int isUsernameExits(char *, int);

int validate(char *, int, int);

void process(int);

void registration(int);

void browse(int);

void put(int);

int addProfile(char *, int, char*, int);

int isAuthenticated(char*, int, int);


void getMusicByRandom(int);

void getMusicByMID(int);

void getMasterFile(int);

void getBiggestMID(int);
```

```
void mystrcpy(char*, char*, int);

char* int2bytes(int);

int bytes2int(char*);

void getlock(int, int);


// writes the error message and exits

void error(char *msg) {

	perror(msg);

	exit(0);

}


// socket write error function

void printWriteError() {

	printf("ERROR, socket couldn't be written\n");

}


// socket read error function

void printReadError() {

	printf("ERROR, socket couldn't be read\n");

}


int main(int argc, char *argv[]){

	int sockfd,		//initial socket handle

		newsockfd,		//socket handle of the child

		portno,			//port number of the communication
```

```
        childpid,                       //process id of the child process

        temp_music_id; //temporary music id


    unsigned int  clilen;                       //length of the client address


    struct sockaddr_in serv_addr,  //server address

                                                cli_addr;     //client address


    char *temp_music_name;   //temporary music name

    FILE *masterFile, *loginFile;  //file descriptor of the master file and login file


    //masterFilePath = getenv("SMS_MASTER");

    //loginFilePath = getenv("SMS_LOGIN");

    //musicDirectory = getenv("SMS_MUSIC");


    masterFilePath = "master.txt";

    loginFilePath = "login.txt";

    musicDirectory = ".";


    max_music_id = mmap(NULL, sizeof(int), PROT_READ | PROT_WRITE, MAP_SHARED |
MAP_ANONYMOUS, -1, 0);

    *max_music_id = 0;

    music_name_total_length = mmap(NULL, sizeof(int), PROT_READ | PROT_WRITE, MAP_SHARED
| MAP_ANONYMOUS, -1, 0);

    *music_name_total_length = 0;
```

```
//check environment variables to start server
if (masterFilePath == NULL) {
	error("ERROR, SMS_MASTER isn't set\n");
}
if (loginFilePath == NULL) {
	error("ERROR, SMS_LOGIN isn't set\n");
}
if (musicDirectory == NULL) {
	error("ERROR, SMS_MUSIC isn't set\n");
}
if (argc < 2) {
	error("ERROR, No port is provided\n");
}

//allocate buffer and check whether it is successful
temp_music_name = malloc(sizeof(char) * 256);
if (temp_music_name == NULL) {
	error("ERROR, buffer couldn't be allocated\n");
}

//check the accessbility of the login file and unless exists, create new one
loginFile = fopen(loginFilePath, "a");
if (loginFile == NULL) {
	error("ERROR, login file couldn't be accessed\n");
}
fclose(loginFile);
```

```
	//check the accessbility of the master file and unless exists, create new one
	masterFile = fopen(masterFilePath, "a");
	if (masterFile == NULL) {
		error("ERROR, master file couldn't be accessed\n");
	}
	fclose(masterFile);

	//open master file and check whether it is successful
	masterFile = fopen(masterFilePath, "r");
	if (masterFile == NULL) {
		error("ERROR, master file couldn't be opened\n");
	}

	//read all entries in the master file, count the musics and calculate the total length names of the musics
	while(fscanf(masterFile, "%d\t%256[^\n]", &temp_music_id, temp_music_name) == 2) {
		(*music_name_total_length) += strlen(temp_music_name);
		(*max_music_id)++;
	}

	//close master file and deallocate the memory
	free(temp_music_name);
	fclose(masterFile);

	//open the socket
```

```
sockfd = socket(AF_INET, SOCK_STREAM, 0);
if (sockfd < 0) {
        error("ERROR, opening socket\n");
}

//reset the memory and set the address and port number
memset((char *) &serv_addr, sizeof(serv_addr), 0);
portno = atoi(argv[1]);
serv_addr.sin_family = AF_INET;
serv_addr.sin_addr.s_addr = INADDR_ANY;
serv_addr.sin_port = htons(portno);

//bind the socket to the address
if (bind(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0)    {
        error("ERROR, on binding\n");
}

//listen for the socket while instantaneous 5 request is being queued
listen(sockfd, 5);

printf("INFO, server is started and waiting for requests :)\n");

//accept and fork
for( ; ; ) {

        clilen = sizeof(cli_addr);
```

```
            newsockfd = accept(sockfd, (struct sockaddr *) &cli_addr, &clilen);


            if (newsockfd < 0) {
                printf("ERROR, on accept\n");
            } else {


                //accept is successful so fork and pass the request to the child process
                if ( (childpid = fork()) < 0) {
                    printf("ERROR, on fork\n");
                } else if (childpid == 0) {
                    close(sockfd);     //close since child process doesn't need
                    process(newsockfd);
                    exit(0);
                }


                close(newsockfd); //close since parent process doesn't need


            }


        }


        //close the socket
        close(sockfd);
        return 0;
}
```

```
void process(int newsockfd) {
	int n;	//temporary read - write length
	char req; //request type

	n = recv(newsockfd, &req, 1, 0);
	if (n < 0) {
		printReadError();
		req = '1';
		n = send(newsockfd, &req, 1, 0);
		if (n < 0) {
			printWriteError();
		}
		return;
	}

	if (req == '0') {
		registration(newsockfd);
	} else if (req == '1') {
		getMasterFile(newsockfd);
	} else if (req == '2') {
		getMusicByMID(newsockfd);
	} else if (req == '3') {
		getMusicByRandom(newsockfd);
	} else if (req == '4') {
		getBiggestMID(newsockfd);
	} else if (req == '5') {
```

```
                put(newsockfd);
        } else {
                printf("ERROR, Unspecified request type\n");
                req = '1';
                n = send(newsockfd, &req, 1, 0);
                if (n < 0) {
                        printWriteError();
                }
        }
}

void doRegistrationError(int newsockfd) {
        int n;
        char res;
        res = '1';
        n = send(newsockfd, &res, 1, 0);
        if (n < 0) printWriteError();
}

void registration(int newsockfd) {
        int n,  //temporary read-write
              username_length,  //user name length
              password_length;   //password length

        char res,      //response type
                      *username, //username
```

```
                *password,  //password
                *length = malloc(sizeof(char) * 4);


printf("\nINFO, registration operation is called\n");


//buffer for request length parameters
if (length == NULL) {
        doRegistrationError(newsockfd);
        return;
}


//get user name length
n = recv(newsockfd, length, 4, 0);
if (n < 0) {
        printReadError();
        doRegistrationError(newsockfd);
        free(length);
        return;
}
username_length = bytes2int(length);


//allocate user name buffer and read the user name
username = malloc(sizeof(char) * username_length);
if (username == NULL) {
        printf("ERROR, memory couldn't be allocated for username\n");
        doRegistrationError(newsockfd);
```

```
            free(length);
            return;
}
n = recv(newsockfd, username, username_length, 0);
if (n < 0) {
            printReadError();
            doRegistrationError(newsockfd);
            free(length);
            free(username);
            return;
}

//get password length
n = recv(newsockfd, length, 4, 0);
if (n < 0) {
            printReadError();
            doRegistrationError(newsockfd);
            free(length);
            free(username);
            return;
}
password_length = bytes2int(length);

//allocate buffer for password and read the password
password = malloc(sizeof(char) * password_length);
if (password == NULL) {
```

```
            printf("ERROR, memory couldn't be allocated for password\n");
            doRegistrationError(newsockfd);
            free(length);
            free(username);
            return;
}
n = recv(newsockfd, password, password_length, 0);
if (n < 0) {
            printReadError();
            doRegistrationError(newsockfd);
            free(length);
            free(username);
            free(password);
            return;
}

if (!validate(username, 0, username_length)) {

            res = '2';
            n = send(newsockfd, &res, 1, 0);
            if (n < 0) printWriteError();
            printf("INFO, registration is successfully answered\n");

} else if (!validate(password, 0, password_length)) {

            res = '3';
```

```
            n = send(newsockfd, &res, 1, 0);
            if (n < 0) printWriteError();
            printf("INFO, registration is successfully answered\n");

} else if (password_length < 8) {

            res = '4';
            n = send(newsockfd, &res, 1, 0);
            if (n < 0) printWriteError();
            printf("INFO, registration is successfully answered\n");

} else if (isUsernameExits(username, username_length)) {

            res = '5';
            n = send(newsockfd, &res, 1, 0);
            if (n < 0) printWriteError();
            printf("INFO, registration is successfully answered\n");

} else {
            n = addProfile(username, username_length, password, password_length);
            if (n < 0) {
                        doRegistrationError(newsockfd);
                        printf("INFO, registration is successfully answered\n");
            } else {
                        res = '0';
                        n = send(newsockfd, &res, 1, 0);
```

```
                        if (n < 0) printWriteError();
                        else printf("INFO, registration operation is successfully completed\n");
                }
        }

        free(username);
        free(password);
        free(length);
}

void getMasterFile(int newsockfd) {
        int n,
                music_id;

        char res,
                        *temp,
                        *musicName = malloc(sizeof(char) * 257);

        FILE *fd;

        printf("\nINFO, Browse master file operation is called\n");

        //check whether buffer is allocated
        if (musicName == NULL) {
                res = '1';
                n = send(newsockfd, &res, 1, 0);
```

```
		if (n < 0) {
			printf("ERROR, socket couldn't be written");
		}
		return;
}

//open master file and check whether it is successful
fd = fopen(masterFilePath, "r");
if (fd == NULL) {

		res = '1';
		n = send(newsockfd, &res, 1, 0);
		if (n < 0) {
			printf("ERROR, socket couldn't be written\n");
		}
		free(musicName);
		return;
}
getlock(fileno(fd), F_RDLCK);

//send success
res = '0';
n = send(newsockfd, &res, 1, 0);
if (n < 0) {
		printf("ERROR, socket couldn't be written\n");
		free(musicName);
```

```
		getlock(fileno(fd), F_UNLCK);
		fclose(fd);
		return;
}

//send buffer_length
temp = int2bytes((*music_name_total_length) + 8 * (*max_music_id));
n = send(newsockfd, temp, 4, 0);
free(temp);
if (n < 0) {
		printf("ERROR, socket couldn't be written\n");
		free(musicName);
		getlock(fileno(fd), F_UNLCK);
		fclose(fd);
		return;
}

//read all music data
while(fscanf(fd, "%d\t%256[^\n]", &music_id, musicName) == 2) {

			//send MID
			temp = int2bytes(music_id);
			n = send(newsockfd, temp, 4, 0);
			free(temp);
			if (n < 0) {
				printf("ERROR, socket couldn't be written\n");
```

```
		free(musicName);
		getlock(fileno(fd), F_UNLCK);
		fclose(fd);
		return;
}

//send music name length
temp = int2bytes(strlen(musicName));
n = send(newsockfd, temp, 4, 0);
free(temp);
if (n < 0) {
		printf("ERROR, socket couldn't be written\n");
		free(musicName);
		getlock(fileno(fd), F_UNLCK);
		fclose(fd);
		return;
}

//send the music name
n = send(newsockfd, musicName, strlen(musicName), 0);
if (n < 0) {
		printf("ERROR, socket couldn't be written\n");
		free(musicName);
		getlock(fileno(fd), F_UNLCK);
		fclose(fd);
		return;
```

```
                        }
        }

        free(musicName);
        getlock(fileno(fd), F_UNLCK);
        fclose(fd);

        printf("INFO, Browse master file is successfully completed\n");
}

void getMusicByMID(int newsockfd) {
        int i,
                n,
                music_id,
                id,
                music_name_length,
                buffer_length,
                file_length;

        char res,
                        *temp,
                        *buffer;

        FILE *fd, *fmusic;

        printf("\nINFO, Browse by MID is called\n");
```

```
//allocate buffer and check whether it is successful
buffer = malloc(sizeof(char) * 1024);
if (buffer == NULL) {
	printf("ERROR, memory couldn't be allocated\n");
	res = '1';
	n = send(newsockfd, &res, 1, 0);
	if (n < 0) {
		printf("ERROR, socket couldn't be written\n");
	}
	return;
}

//get MID
n = recv(newsockfd, buffer, 4, 0);
if (n < 0) {
	printf("ERROR, socket couldn't be read\n");
	res = '1';
	n = send(newsockfd, &res, 1, 0);
	if (n < 0) {
		printf("ERROR, socket couldn't be written\n");
	}
	free(buffer);
	return;
}
music_id = bytes2int(buffer);
```

```
//check MID range
if (music_id < 1 || music_id > (*max_music_id)) {
        res = '2';
        n = send(newsockfd, &res, 1, 0);
        if (n < 0) {
                printf("ERROR, socket couldn't be written\n");
        }
        free(buffer);
        return;
}

//open relevant music file and check whether it is successful
sprintf(buffer, "%d", music_id);
fmusic = fopen(buffer, "rb");
if (fmusic == NULL) {
        printf("ERROR, music file couldn't be found\n");
        res = '1';
        n = send(newsockfd, &res, 1, 0);
        if (n < 0) {
                printf("ERROR, socket couldn't be written\n");
        }
        free(buffer);
        return;
}
getlock(fileno(fmusic), F_RDLCK);
```

```
//open master file for the music name and check whether it is successful
fd = fopen(masterFilePath, "r");
if (fd == NULL) {
        res = '1';
        n = send(newsockfd, &res, 1, 0);
        if (n < 0) {
                printf("ERROR, socket couldn't be written\n");
        }
        free(buffer);
        getlock(fileno(fmusic), F_UNLCK);
        fclose(fmusic);
        return;
}
getlock(fileno(fd), F_RDLCK);

//find music name from the master file
while(fscanf(fd, "%d\t%256[^\n]", &id, buffer) == 2) {

                //searched music
                if(id == music_id) {

                        //start to prepare the response
                        res = '0';
                        n = send(newsockfd, &res, 1, 0);
                        if (n < 0) {
```

```
            printf("ERROR, socket couldn't be written\n");
            getlock(fileno(fmusic), F_UNLCK);
            fclose(fmusic);
            getlock(fileno(fd), F_UNLCK);
            fclose(fd);
            free(buffer);
            return;
}
//calculate the response length
music_name_length = strlen(buffer);

buffer_length = 4 + music_name_length;
fseek(fmusic, 0, SEEK_END);
file_length = ftell(fmusic);
fseek(fmusic, 0, SEEK_SET);

buffer_length += file_length;

//send the response length
temp = int2bytes(buffer_length);
n = send(newsockfd, temp, 4, 0);
free(temp);
if (n < 0) {
            printf("ERROR, socket couldn't be written\n");
            getlock(fileno(fmusic), F_UNLCK);
            fclose(fmusic);
```

```
            getlock(fileno(fd), F_UNLCK);
            fclose(fd);
            free(buffer);
            return;
}

//send music name length
temp = int2bytes(music_name_length);
n = send(newsockfd, temp, 4, 0);
free(temp);
if (n < 0) {
            printf("ERROR, socket couldn't be written\n");
            getlock(fileno(fmusic), F_UNLCK);
            fclose(fmusic);
            getlock(fileno(fd), F_UNLCK);
            fclose(fd);
            free(buffer);
            return;
}

//send music name
n = send(newsockfd, buffer, music_name_length, 0);
if (n < 0) {
            printf("ERROR, socket couldn't be written\n");
            getlock(fileno(fmusic), F_UNLCK);
            fclose(fmusic);
```

```
            getlock(fileno(fd), F_UNLCK);
            fclose(fd);
            free(buffer);
            return;
}

//send the music file
while(file_length > 1024) {
            for(i=0; i<1024; i++) {
                        buffer[i] = fgetc(fmusic);
            }
            n = send(newsockfd, buffer, 1024, 0);
            if (n < 0) {
                        printf("ERROR, socket couldn't be written\n");
                        getlock(fileno(fmusic), F_UNLCK);
                        fclose(fmusic);
                        getlock(fileno(fd), F_UNLCK);
                        fclose(fd);
                        free(buffer);
                        return;
            }
            file_length -= 1024;
}
for(i=0; i<file_length; i++) {
            buffer[i] = fgetc(fmusic);
}
```

```
                        n = send(newsockfd, buffer, file_length, 0);
                        if (n < 0) {
                                printf("ERROR, socket couldn't be written\n");
                                getlock(fileno(fmusic), F_UNLCK);
                                fclose(fmusic);
                                getlock(fileno(fd), F_UNLCK);
                                fclose(fd);
                                free(buffer);
                                return;
                        }
                        getlock(fileno(fmusic), F_UNLCK);
                        fclose(fmusic);
                        break;
                }
        }

        getlock(fileno(fd), F_UNLCK);
        fclose(fd);
        free(buffer);

        printf("INFO, Browse by MID is successfully completed\n");
}

void getMusicByRandom(int newsockfd) {
        int i,
                n,
```

```
            music_id,
            id,
            music_name_length,
            buffer_length,
            file_length;

char res,
                    *temp,
                    *buffer;

FILE *fd, *fmusic;

printf("\nINFO, Browse by random is called\n");

//check whether there is a music file in the server
if ((*max_music_id) == 0) {
        res = '2';
        n = send(newsockfd, &res, 1, 0);
        if (n < 0) {
                printf("ERROR, socket couldn't be written\n");
        }
        return;
}

//allocate the buffer and check whether it is successful
buffer = malloc(sizeof(char) * 1024);
```

```
if (buffer == NULL) {
        printf("ERROR, memort couldn't be allocated\n");
        res = '1';
        n = send(newsockfd, &res, 1, 0);
        if (n < 0) {
                printf("ERROR, socket couldn't be written\n");
        }
        return;
}

//randomly generate a MID
srand(time(NULL));
music_id = (rand() % (*max_music_id)) + 1;

//open the relevant music file and check whether it is successful
sprintf(buffer, "%d", music_id);
fmusic = fopen(buffer, "rb");
if (fmusic == NULL) {
        printf("ERROR, music file couldn't be found\n");
        res = '1';
        n = send(newsockfd, &res, 1, 0);
        if (n < 0) {
                printf("ERROR, socket couldn't be written\n");
        }
        free(buffer);
        return;
```

```
}
getlock(fileno(fmusic), F_RDLCK);

//open the master file and check whether it is successful
fd = fopen(masterFilePath, "r");
if (fd == NULL) {
        res = '1';
        n = send(newsockfd, &res, 1, 0);
        if (n < 0) {
                printf("ERROR, socket couldn't be written\n");
        }
        free(buffer);
        getlock(fileno(fmusic), F_UNLCK);
        fclose(fmusic);
        return;
}
getlock(fileno(fd), F_RDLCK);

//find the relevant file info in the master file
while(fscanf(fd, "%d\t%256[^\n]", &id, buffer) == 2) {

                if(id == music_id) {

                        //start to prepare the response
                        res = '0';
                        n = send(newsockfd, &res, 1, 0);
```

```
if (n < 0) {
        printf("ERROR, socket couldn't be written\n");
        getlock(fileno(fmusic), F_UNLCK);
        fclose(fmusic);
        getlock(fileno(fd), F_UNLCK);
        fclose(fd);
        free(buffer);
        return;
}

//calculate response length
music_name_length = strlen(buffer);

buffer_length = 4 + 4 + music_name_length;
fseek(fmusic, 0, SEEK_END);
file_length = ftell(fmusic);
fseek(fmusic, 0, SEEK_SET);

buffer_length += file_length;

//send response length
temp = int2bytes(buffer_length);
n = send(newsockfd, temp, 4, 0);
free(temp);
if (n < 0) {
        printf("ERROR, socket couldn't be written\n");
```

```
            getlock(fileno(fmusic), F_UNLCK);
            fclose(fmusic);
            getlock(fileno(fd), F_UNLCK);
            fclose(fd);
            free(buffer);
            return;
}

//send MID of the chosen music file
temp = int2bytes(music_id);
n = send(newsockfd, temp, 4, 0);
free(temp);
if (n < 0) {
            printf("ERROR, socket couldn't be written\n");
            getlock(fileno(fmusic), F_UNLCK);
            fclose(fmusic);
            getlock(fileno(fd), F_UNLCK);
            fclose(fd);
            free(buffer);
            return;
}

//send the length of the music name
temp = int2bytes(music_name_length);
n = send(newsockfd, temp, 4, 0);
free(temp);
```

```
if (n < 0) {
        printf("ERROR, socket couldn't be written\n");
        getlock(fileno(fmusic), F_UNLCK);
        fclose(fmusic);
        getlock(fileno(fd), F_UNLCK);
        fclose(fd);
        free(buffer);
        return;
}

//send the music name
n = send(newsockfd, buffer, music_name_length, 0);
if (n < 0) {
        printf("ERROR, socket couldn't be written\n");
        getlock(fileno(fmusic), F_UNLCK);
        fclose(fmusic);
        getlock(fileno(fd), F_UNLCK);
        fclose(fd);
        free(buffer);
        return;
}

//send the music file content
while(file_length > 1024) {
        for(i=0; i<1024; i++) {
                buffer[i] = fgetc(fmusic);
```

```
		}
		n = send(newsockfd, buffer, 1024, 0);
		if (n < 0) {
			printf("ERROR, socket couldn't be written\n");
			getlock(fileno(fmusic), F_UNLCK);
			fclose(fmusic);
			getlock(fileno(fd), F_UNLCK);
			fclose(fd);
			free(buffer);
			return;
		}
		file_length -= 1024;
	}
	for(i=0; i<file_length; i++) {
		buffer[i] = fgetc(fmusic);
	}
	n = send(newsockfd, buffer, file_length, 0);
	if (n < 0) {
		printf("ERROR, socket couldn't be written\n");
		getlock(fileno(fmusic), F_UNLCK);
		fclose(fmusic);
		getlock(fileno(fd), F_UNLCK);
		fclose(fd);
		free(buffer);
		return;
	}
```

```
                                getlock(fileno(fmusic), F_UNLCK);
                                fclose(fmusic);
                                break;
                        }
                }

                getlock(fileno(fd), F_UNLCK);
                fclose(fd);
                free(buffer);

                printf("INFO, Browse by random is successfully completed\n");
}

void getBiggestMID(int newsockfd) {
                int n;

                char res,
                                *temp;

                printf("\nINFO, Browse the biggest MID is called\n");

                //start to prepare the response
                res = '0';
                n = send(newsockfd, &res, 1, 0);
                if (n < 0) {
                        printf("ERROR, socket couldn't be written\n");
```

```
		res = '1';
		n = send(newsockfd, &res, 1, 0);
		if (n < 0) {
			printf("ERROR, socket couldn't be written\n");
		}
		return;
	}

	//send the biggest music id
	temp = int2bytes((*max_music_id));
	n = send(newsockfd, temp, 4, 0);
	free(temp);
	if (n < 0) {
		printf("ERROR, socket couldn't be written\n");
	} else {
		printf("INFO, Browse the biggest MID is successfully completed\n");
	}
}

//send error to the client
void printPutError(int newsockfd) {
	char res;
	int n;

	res = '1';
	n = send(newsockfd, &res, 1, 0);
```

```
	if (n < 0) {
		printf("ERROR, socket couldn't be written\n");
	}
}

void put(int newsockfd) {
	int i,
		n,
		buffer_length,
		username_length,
		password_length,
		musicname_length,
		file_length;

	char res,
			*buffer;

	FILE *fd;

	printf("\nINFO, Put is called\n");

	//allocate the buffer and check whether it is successful
	buffer = malloc(sizeof(char) * 1024);
	if (buffer == NULL) {
		printf("ERROR, buffer couldn't be allocated\n");
		printPutError(newsockfd);
```

```
            return;
}
memset(buffer, 1024, 0);

//get request length
n = recv(newsockfd, buffer, 4, 0);
if (n < 0) {
            printf("ERROR, socket couldn't be read\n");
            free(buffer);
            printPutError(newsockfd);
            return;
}
buffer_length = bytes2int(buffer);

//get user name length
n = recv(newsockfd, buffer, 4, 0);
if (n < 0) {
            printf("ERROR, socket couldn't be read\n");
            free(buffer);
            printPutError(newsockfd);
            return;
}
username_length = bytes2int(buffer);

//get user name
n = recv(newsockfd, buffer, username_length, 0);
```

```
if (n < 0) {
        printf("ERROR, socket couldn't be read\n");
        free(buffer);
        printPutError(newsockfd);
        return;
}
memset(buffer, 4, 0);

//get password length
n = recv(newsockfd, &buffer[username_length], 4, 0);
if (n < 0) {
        printf("ERROR, socket couldn't be read\n");
        free(buffer);
        printPutError(newsockfd);
        return;
}
password_length = bytes2int(&buffer[username_length]);
memset(&buffer[username_length], 4, 0);

//get password
n = recv(newsockfd, &buffer[username_length], password_length, 0);
if (n < 0) {
        printf("ERROR, socket couldn't be read\n");
        free(buffer);
        printPutError(newsockfd);
        return;
```

```
}

if(!isAuthenticated(buffer, username_length, password_length)) {
        buffer_length = buffer_length - 4 -username_length - 4 -password_length;
        while(buffer_length > 0) {
                n = recv(newsockfd, buffer, 1, 0);
                buffer_length--;
        }
        res = '2';
        n = send(newsockfd, &res, 1, 0);
        if (n < 0) {
                printf("ERROR, socket couldn't be written\n");
        }
        free(buffer);
        printf("INFO, User isn't authenticated\n");
        printf("INFO, Put is successfully answered\n");
        return;
}

memset(buffer, 1024, 0);

//get music name length
n = recv(newsockfd, buffer, 4, 0);
if (n < 0) {
        printf("ERROR, socket couldn't be read\n");
        free(buffer);
```

```
            printPutError(newsockfd);
            return;
}
musicname_length = bytes2int(buffer);
memset(buffer, 4, 0);

//get music name
n = recv(newsockfd, buffer, musicname_length, 0);
if (n < 0) {
            printf("ERROR, socket couldn't be read\n");
            free(buffer);
            printPutError(newsockfd);
            return;
}

//open master file to add new music and check whether it is successful
fd = fopen(masterFilePath, "a");
if (fd == NULL) {
            printf("ERROR, master file couldn't be read\n");
            free(buffer);
            printPutError(newsockfd);
            return;
}
getlock(fileno(fd), F_WRLCK);

//increment the number of musics and increase music name length counter accordingly
```

```
(*max_music_id)++;
(*music_name_total_length) += musicname_length;

//add MID and music name to the master file
fprintf(fd, "%d\t", *max_music_id);
for(i=0; i<musicname_length; i++) {
        fputc(buffer[i], fd);
}
fputc('\n', fd);
getlock(fileno(fd), F_UNLCK);
fclose(fd);
fd = NULL;

//open the relevant music file to save uploaded data and check whether it is successful
sprintf(buffer, "%d", (*max_music_id));
fd = fopen(buffer, "wb");
if (fd == NULL) {
        printf("ERROR, music file couldn't be created\n");
        free(buffer);
        printPutError(newsockfd);
        return;
}
getlock(fileno(fd), F_WRLCK);

//save the uploaded music data
file_length = buffer_length - 4 -username_length - 4 -password_length - 4 - musicname_length;
```

```
while(file_length > 1024) {
        n = recv(newsockfd, buffer, 1024, 0);
        if (n < 0) {
                printf("ERROR, socket couldn't be read\n");
                free(buffer);
                printPutError(newsockfd);
                getlock(fileno(fd), F_UNLCK);
                fclose(fd);
                return;
        }
        for(i=0; i<1024; i++) {
                fputc(buffer[i], fd);
        }
        file_length -= 1024;
}
n = recv(newsockfd, buffer, file_length, 0);
if (n < 0) {
        printf("ERROR, socket couldn't be read\n");
        free(buffer);
        printPutError(newsockfd);
        getlock(fileno(fd), F_UNLCK);
        fclose(fd);
        return;
}
for(i=0; i<file_length; i++) {
        fputc(buffer[i], fd);
```

```
		}
		getlock(fileno(fd), F_UNLCK);
		fclose(fd);

		free(buffer);

		res = '0';
		n = send(newsockfd, &res, 1, 0);
		i = 0;
		while(n < 0 && i < 2) {
			n = send(newsockfd, &res, 1, 0);
		}
		if (n < 0) {
			printf("ERROR, socket couldn't be written\n");
		} else {
			printf("INFO, Put is successfully completed\n");
		}
}

int isUsernameExits(char *username, int username_length) {
		int diff;
		char local_buffer[257];

		FILE *fd = fopen(loginFilePath, "r");
		if (fd == NULL) return 0;
		getlock(fileno(fd), F_RDLCK);
```

```
	while(fscanf(fd, "%256s", local_buffer) == 1) {
		diff = strncmp(username, local_buffer, username_length);
		if (diff == 0) {
			getlock(fileno(fd), F_UNLCK);
			fclose(fd);
			return 1;
		} else fscanf(fd, "%256s", local_buffer);
	}
	getlock(fileno(fd), F_UNLCK);
	fclose(fd);
	return 0;
}

int isAuthenticated(char *buffer, int ulen, int plen) {
	int udiff, pdiff;
	char local_buffer[257];

	FILE *fd = fopen(loginFilePath, "r");
	if (fd == NULL) return 0;
	getlock(fileno(fd), F_RDLCK);
	while(fscanf(fd, "%256s", local_buffer) == 1) {
		udiff = strncmp(buffer, local_buffer, ulen);
		if (udiff == 0) {
			fscanf(fd, "%256s", local_buffer);
			pdiff = strncmp(&buffer[ulen], local_buffer, plen);
			if (pdiff == 0) {
```

```
                        getlock(fileno(fd), F_UNLCK);
                        fclose(fd);
                        return 1;
                }
        } else {
                fscanf(fd, "%256s", local_buffer);
        }
    }
    getlock(fileno(fd), F_UNLCK);
    fclose(fd);
    return 0;
}

int addProfile(char * username, int username_length, char *password, int password_length) {
    int i;
    FILE *fd = fopen(loginFilePath, "a");
    if (fd == NULL) return -1;
    getlock(fileno(fd), F_WRLCK);
    for(i=0; i<username_length; i++) {
        fputc(username[i], fd);
    }
    fputc(' ', fd);

    for(i=0; i<password_length; i++) {
        fputc(password[i], fd);
    }
```

```
        fputc('\n', fd);
        getlock(fileno(fd), F_UNLCK);
        fclose(fd);
        return 0;
}

int validate(char *str, int start, int len) {
        int i, val;
        for(i=start; i<start+len; i++) {
                val = (int)str[i];
                if((val < 48 && val != 46) ||
                   (val > 57 && val < 65) ||
                   (val > 90 && val < 97 && val != 95) ||
                   val > 122) {
                        return 0;
                }
        }
        return 1;
}

char* int2bytes(int integer) {
        char *bytes = malloc(sizeof(char) * 4);
        int base2 = 256 * 256;
        int base3 = base2 * 256;

        bytes[0] = (integer / base3);
```

```
	integer -= base3 * bytes[0];
	bytes[1] = (integer / base2);
	integer -= base2 * bytes[1];
	bytes[2] = (integer / 256);
	integer -= 256 * bytes[2];
	bytes[3] = integer;
	return bytes;
}

int bytes2int(char *bytes) {
	int integer = 0;
	if (bytes[3] < 0) integer += 256 + bytes[3];
	else integer += bytes[3];

	if (bytes[2] < 0) integer += (256 + bytes[2]) * 256;
	else integer += bytes[2] * 256;

	if (bytes[1] < 0) integer += (256 + bytes[1]) * 256 * 256;
	else integer += bytes[1] * 256 * 256;

	if (bytes[0] < 0) integer += (256 + bytes[0]) * 256 * 256 * 256 ;
	else integer += bytes[0] * 256 * 256 * 256;

	return integer;
}
```

```
void mystrcpy(char *dest, char *src, int len) {
        int i;
        for(i=0; i<len; i++) {
                dest[i] = src[i];
        }
}


void getlock(int fd, int type) {
   struct flock lockinfo;


   /* we'll lock the entire file */
   lockinfo.l_whence = SEEK_SET;
   lockinfo.l_start = 0;
   lockinfo.l_len = 0;


   /* keep trying until we succeed */
   while (1) {
      lockinfo.l_type = type;
         /* if we get the lock, return immediately */
      if (!fcntl(fd, F_SETLK, &lockinfo)) return;
   }
}
```

client.c

```
/* -*- Mode: C; indent-tabs-mode: t; c-basic-offset: 4; tab-width: 4 -*- */
/*
```

```
* main.c
* Copyright (C) Ferhat Elmas 2011 <elmas.ferhat@gmail.com>
*
* SMSC is free software: you can redistribute it and/or modify it
* under the terms of the GNU General Public License as published by the
* Free Software Foundation, either version 3 of the License, or
* (at your option) any later version.
*
* SMSC is distributed in the hope that it will be useful, but
* WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
* See the GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License along
* with this program.  If not, see <http://www.gnu.org/licenses/>.
*/


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

```
//server, buffer and port number
struct hostent *server;
char *buffer;
int portno;

//utility functions
char* int2bytes(int);
int bytes2int(char*);
void mystrcpy(char *, char *, int);
int cli_connect();

//functions that takes input from user
char* getUsername(int*);
char* getPassword(int*);
char* getMusicName(int*);
char* getMusicFileName(int*);

void doOperation();
void printGeneralOperations();
void doRegistrationOperation();
void doBrowseOperation();
void printPutOperations();
void displayMaster();
void getbyMID(int, int);
void getbyrange();
void getbyrandom();
```

```
int getbiggestMID(int);

void printMusicName(int);



int main(int argc, char *argv[]){


        if (argc < 3) {

                printf("ERROR,  hostname and port aren't given\n");

                exit(0);

        }


        buffer = malloc(sizeof(char) * 1024);

        if (buffer == NULL) {

                printf("ERROR, buffer couldn't be allocated\n");

                exit(0);

        }


        portno = atoi(argv[2]);

        if (portno < 1024) {

                printf("ERROR, portno must be bigger than 1023");

                exit(0);

        }


        server = gethostbyname(argv[1]);

        if(server == NULL) {

                printf("ERROR, there is no host with specified hostname: %s", argv[1]);
```

```
                exit(0);
        }
        printf("Welcome to the SMS Client\n");
        doOperation();

        return 0;
}

void doOperation() {
        int c = 0;
        while(1) {
                while(1) {
                        printGeneralOperations();
                        scanf("%d", &c);
                        if (c == 1 || c == 2 || c == 3 || c == 4) break;
                }

                if (c == 1) {
                        doRegistrationOperation();
                } else if (c == 2) {
                        doBrowseOperation();
                } else if (c == 3) {
                        printPutOperations();
                } else {
                        exit(0);
                }
```

```
        }
}

void printGeneralOperations() {
        printf("\n");
        printf("\tPress 1 : Registration\n");
        printf("\tPress 2 : Browse\n");
        printf("\tPress 3 : Put\n");
        printf("\tPress 4 : Exit\n");
}

void doRegistrationOperation() {

        int i,
            n,
            username_length,
            password_length,
            sockfd;

        char req,
                        *temp,
                        *username,
                        *password;
        //user name is taken from the user
        username = getUsername(&username_length);
        if (username == NULL) {
```

```
        printf("ERROR, username couldn't be read\n");
        return;
}

//password is taken from the user
password = getPassword(&password_length);
if(password == NULL) {
        printf("ERROR, password couldn't be read\n");
        return;
}

i=0;
sockfd = cli_connect();
while(sockfd < 0 && i<2){
        i++;
        sockfd = cli_connect();
}
if (sockfd < 0) {
        printf("ERROR, Connection is tried 3 times but couldn't be established\n");
        free(username);
        free(password);
        return;
}
printf("INFO, Connection is established\n");

//set request operation type
```

```
req = '0';
n = send(sockfd, &req, 1, 0);
if (n < 0) {
        printf("ERROR, request-operation type couldn't be written to the socket\n");
        free(username);
        free(password);
        close(sockfd);
        return;
}

//set username length
temp = int2bytes(username_length);
n = send(sockfd, temp, 4, 0);
free(temp);
if (n < 0) {
        printf("ERROR, request-user name length couldn't be written to the socket\n");
        free(username);
        free(password);
        close(sockfd);
        return;
}

//set username
n = send(sockfd, username, username_length, 0);
if (n < 0) {
        printf("ERROR, request-user name couldn't be written to the socket\n");
```

```
            free(username);
            free(password);
            close(sockfd);
            return;
}

//set password length
temp = int2bytes(password_length);
n = send(sockfd, temp, 4, 0);
free(temp);
if (n < 0) {
            printf("ERROR, request-password length couldn't be written to the socket\n");
            free(username);
            free(password);
            close(sockfd);
            return;
}

//set password
n = send(sockfd, password, password_length, 0);
if (n < 0) {
            printf("ERROR, request-password couldn't be written to the socket\n");
            free(username);
            free(password);
            close(sockfd);
            return;
```

```
}

printf("INFO, request is successfully sent\n");

n = recv(sockfd, &req, 1, 0);
if (n < 0) {
        printf("ERROR, response-type couldn't be read from the socket\n");
        free(username);
        free(password);
        close(sockfd);
        return;
}

switch(req) {
        case '0' : printf("Registration is successfully completed\n"); break;
        case '1' : printf("Server returned error\n"); break;
        case '2' : printf("Username contains invalid characters: %s\n", username); break;
        case '3' : printf("Password contains invalid characters: %s\n", password); break;
        case '4' : printf("Weak Password\n"); break;
        case '5' : printf("Username already exits, change username\n"); break;
        default : printf("Unspecified Response\n"); break;
}

free(username);
free(password);
close(sockfd);
```

```
}

void printBrowseSpecificOperations() {
	printf("\n");
	printf("\tPress 1 : Display the content of the master file\n");
	printf("\tPress 2 : Get by MID\n");
	printf("\tPress 3 : Get by MID range\n");
	printf("\tPress 4 : Get by random\n");
	printf("\tPress 5 : Get the biggest MID\n");
	printf("\tPress 6 : Go top menu\n");
}

void doBrowseOperation() {
	int c = 0;
	while(1) {
		while(1) {
			printBrowseSpecificOperations();
			scanf("%d", &c);
			if (c == 1 || c == 2 || c == 3 || c == 4 || c == 5 || c == 6) break;
		}

		if (c == 1) {
			displayMaster();
		} else if (c == 2) {
			getbyMID(-1, 1);
		} else if (c == 3) {
```

```
                getbyrange();
            } else if (c == 4) {
                getbyrandom();
            } else if (c == 5) {
                getbiggestMID(1);
            } else {
                doOperation();
            }
        }

}

void displayMaster() {
        int i,
            n,
            buffer_length,
            musicname_length,
            music_id,
            sockfd;

        char    req;

        i = 0;
        sockfd = cli_connect();
        while (sockfd < 0 && i < 2) {
            sockfd = cli_connect();
```

```
}
if (sockfd < 0) {
        printf("ERROR, Connection is tried 3 times but couldn't be set up\n");
        return;
}
printf("INFO, Connection is established\n");

//set request
req = '1';
n = send(sockfd, &req, 1, 0);
if (n < 0) {
        printf("ERROR, request-operation type couldn't be written to the socket\n");
        close(sockfd);
        return;
}
printf("INFO, Request is successfully sent\n");

//get response
n = recv(sockfd, buffer, 1, 0);
if (n < 0) {
        printf("ERROR, response-type couldn't be read from the socket\n");
        close(sockfd);
        return;
}

//interpret the response
```

```
        if(buffer[0] == '1') {
                printf("ERROR, response-type is error, something unexpected may have happened in the
server\n");
        } else {
                n = recv(sockfd, buffer, 4, 0);
                if (n < 0) {
                        printf("ERROR, response-length couldn't be read from the socket\n");
                        close(sockfd);
                        return;
                }
                buffer_length = bytes2int(buffer);

                if(buffer_length == 0) {
                        printf("There are music files at the server\n");
                        close(sockfd);
                        return;
                }

                while (buffer_length > 0) {
                        n = recv(sockfd, buffer, 4, 0);
                        if (n < 0) {
                                printf("ERROR, response-music id couldn't be read from the socket\n");
                                close(sockfd);
                                return;
                        }
                        music_id = bytes2int(buffer);
```

```
                        n = recv(sockfd, buffer, 4, 0);
                        if (n < 0) {
                                printf("ERROR, response-music name length couldn't be read from the
socket\n");
                                close(sockfd);
                                return;
                        }
                        musicname_length = bytes2int(buffer);

                        buffer_length -= (8 + musicname_length);

                        printf("Music %d : ", music_id);
                        n = recv(sockfd, buffer, musicname_length, 0);
                        if (n < 0) {
                                printf("ERROR, response-music name couldn't be read from the
socket\n");
                                return;
                                close(sockfd);
                        }
                        printMusicName(musicname_length);
                        printf("\n");
                }
        }
        close(sockfd);
}
```

```
void printMusicName(int musicname_length) {
	int i;
	for(i=0; i<musicname_length; i++) {
		putchar(buffer[i]);
	}
}

void getbyrange() {
	int i,
		low,
		high,
		biggest;

	printf("Please enter the lower bound MID : ");
	scanf("%d", &low);

	printf("Please enter the higher bound MID : ");
	scanf("%d", &high);

	if(low > high) {
		i = low;
		low = high;
		high = i;
	}
```

```
	biggest = getbiggestMID(0);
	if (high > biggest) high = biggest;

	if (high > 0 && low < 1) low = 1;

	if(low > 0) {
		for(i=low; i<=high; i++) {
			getbyMID(i, 0);
		}
	} else {
		printf("INFO, There are no files in the specified range\n");
	}
}

void getbyMID(int mid, int choice) {
	int	i,
			n,
			sockfd,
			music_id,
			buffer_length,
			musicname_length,
			file_length;

	char *temp;
	FILE *fd;
```

```
if (choice) {
        printf("Please enter the MID : ");
        scanf("%d", &music_id);
} else {
        music_id = mid;
}

if (music_id < 0) {
        printf("MID starts from 1\n");
        return;
}

i = 0;
sockfd = cli_connect();
while (sockfd < 0 && i < 2) {
        sockfd = cli_connect();
}
if (sockfd < 0) {
        printf("ERROR, Connection is tried 3 times but couldn't be set up\n");
        return;
}
printf("INFO, Connection is established\n");

memset(buffer, 1024, 0);
//set operation type
buffer[0] = '2';
```

```
	//set MID
	temp = int2bytes(music_id);
	mystrcpy(&buffer[1], temp, 4);
	free(temp);

	n = send(sockfd, buffer, 5, 0);
	if (n < 0) {
		printf("ERROR, request-operation type and music id couldn't be written to the socket\n");
	} else {
		printf("INFO, Request is successfully sent\n");
		n = recv(sockfd, buffer, 1, 0);
		if (n < 0) {
			printf("ERROR, response-type couldn't be read from the socket\n");
			close(sockfd);
			return;
		}

		//interpret the result
		if (buffer[0] == '1') {
			printf("ERROR, response-type is error, something unexpected may have happened in the server\n");
		} else if (buffer[0] == '2') {
			printf("INFO, there is no music file with specified MID : %d\n", music_id);
		} else {
```

```
n = recv(sockfd, buffer, 4, 0);
if (n < 0) {
        printf("ERROR, response-length couldn't be read from the socket\n");
        close(sockfd);
        return;
}
buffer_length = bytes2int(buffer);

n = recv(sockfd, buffer, 4, 0);
if (n < 0) {
        printf("ERROR, response-music name length couldn't be read from the
socket\n");
        close(sockfd);
        return;
}
musicname_length = bytes2int(buffer);

file_length = buffer_length - 4 - musicname_length;



n = recv(sockfd, buffer, musicname_length, 0);
if (n < 0) {
        printf("ERROR, response-music name couldn't be read from the
socket\n");
        close(sockfd);
        return;
```

```
}
printf("INFO, MID : %d, ", music_id);
printMusicName(musicname_length);
printf(" download is starting\n");

sprintf(buffer, "%d", music_id);
fd = fopen(buffer, "wb");
if (fd == NULL) {
        printf("ERROR, file to download the music couldn't be opened\n");
        close(sockfd);
        return;
}
while(file_length > 1024) {
        file_length -= 1024;
        n = recv(sockfd, buffer, 1024, 0);
        if (n < 0) {
                printf("ERROR, response-file part couldn't be read from the
socket\n");
                fclose(fd);
                close(sockfd);
                return;
        }
        for(i=0; i<1024; i++) {
                fputc(buffer[i], fd);
        }
}
```

```
                        n = recv(sockfd, buffer, file_length, 0);
                        if (n < 0) {
                                printf("ERROR, response-file part couldn't be read from the socket\n");
                                fclose(fd);
                                close(sockfd);
                                return;
                        }
                        for(i=0; i<file_length; i++) {
                                fputc(buffer[i], fd);
                        }
                        fclose(fd);
                        printf("INFO, MID : %d download is successfully completed\n", music_id);

                }
        }
        close(sockfd);
}

void getbyrandom() {
        int     i,
                        n,
                        sockfd,
                        music_id,
                        buffer_length,
                        musicname_length,
                        file_length;
```

```
FILE *fd;


i = 0;
sockfd = cli_connect();
while (sockfd < 0 && i < 2) {
        sockfd = cli_connect();
}
if (sockfd < 0) {
        printf("ERROR, Connection is tried 3 times but couldn't be set up\n");
        return;
}
printf("INFO, Connection is successfully established\n");


memset(buffer, 1024, 0);
//set operation type
buffer[0] = '3';
n = send(sockfd, buffer, 1, 0);
if (n < 0) {
        printf("ERROR, request-type couldn't be written to the socket\n");
        close(sockfd);
        return;
}
printf("INFO, Request is successfully sent\n");


n = recv(sockfd, buffer, 1, 0);
```

```
	if (n < 0) {
		printf("ERROR, response-type couldn't be read from the socket\n");
		close(sockfd);
		return;
	}

	if (buffer[0] == '1') {
		printf("ERROR, response-type is error, something unexpected may have happened in the server\n");
	} else if (buffer[0] == '2') {
		printf("INFO, there are no music files at the server\n");
	} else {
		n = recv(sockfd, buffer, 4, 0);
		if (n < 0) {
			printf("ERROR, response-length couldn't be read from the socket\n");
			close(sockfd);
			return;
		}
		buffer_length = bytes2int(buffer);

		n = recv(sockfd, buffer, 4, 0);
		if (n < 0) {
			printf("ERROR, response-music id couldn't be read from the socket\n");
			close(sockfd);
			return;
		}
```

```
		music_id = bytes2int(buffer);


		n = recv(sockfd, buffer, 4, 0);
		if (n < 0) {
			printf("ERROR, response-music name length couldn't be read from the
socket\n");
			close(sockfd);
			return;
		}
		musicname_length = bytes2int(buffer);

		file_length = buffer_length - 4 - 4 - musicname_length;



		n = recv(sockfd, buffer, musicname_length, 0);
		if (n < 0) {
			printf("ERROR, response-music name couldn't be read from the socket\n");
			close(sockfd);
			return;
		}
		printf("INFO, MID : %d, ", music_id);
		printMusicName(musicname_length);
		printf(" download is starting\n");

		sprintf(buffer, "%d", music_id);
		fd = fopen(buffer, "wb");
```

```
if (fd == NULL) {
        printf("ERROR, file to download the music couldn't be opened\n");
        close(sockfd);
        return;
}
while(file_length > 1024) {
        file_length -= 1024;
        n = recv(sockfd, buffer, 1024, 0);
        if (n < 0) {
                printf("ERROR, response-file part couldn't be read from the socket\n");
                fclose(fd);
                close(sockfd);
                return;
        }
        for(i=0; i<1024; i++) {
                fputc(buffer[i], fd);
        }
}
n = recv(sockfd, buffer, file_length, 0);
if (n < 0) {
        printf("ERROR, response-file part couldn't be read from the socket\n");
        fclose(fd);
        close(sockfd);
        return;
}
for(i=0; i<file_length; i++) {
```

```
                fputc(buffer[i], fd);
            }
            fclose(fd);
            printf("INFO, MID : %d download is successfully completed\n", music_id);

        }
        close(sockfd);
}

int getbiggestMID(int choice) {
        int i,
            n,
            sockfd;

        memset(buffer, 1024, 0);
        //set operation type
        buffer[0] = '4';

        i = 0;
        sockfd = cli_connect();
        while (sockfd < 0 && i < 2) {
            sockfd = cli_connect();
        }
        if (sockfd < 0) {
            printf("ERROR, Connection is tried 3 times but couldn't be set up\n");
            return 0;
```

```
	}
	n = send(sockfd, buffer, 1, 0);
	if (n < 0) {
		printf("ERROR, request-type couldn't be written to the socket\n");
	} else {
		if (choice) {
			printf("INFO, Request is successfully sent\n");
		}
		n = recv(sockfd, buffer, 1, 0);
		if (n < 0) {
			printf("ERROR, response-type couldn't be read from the socket\n");
			close(sockfd);
			return 0;
		}

		if (buffer[0] == '1') {
			printf("ERROR, response-type is error, something unexpected may have
happened in the server\n");
		} else {
			n = recv(sockfd, buffer, 4, 0);
			if (n < 0) {
				printf("ERROR, response-biggest music id couldn't be read from the
socket\n");
				close(sockfd);
				return 0;
			}
```

```
                        if (choice) {
                                printf("The biggest MID : %d\n", bytes2int(buffer));
                        } else {
                                close(sockfd);
                                return bytes2int(buffer);
                        }
                }
        }
        close(sockfd);
        return 0;
}

void printBrowseOperations() {

        printBrowseSpecificOperations();

}

void printPutOperations() {
        int i,
                n,
                sockfd,
                buffer_length,
                username_length,
                password_length,
                musicname_length,
```

```
            musicFileName_length,
            musicFile_length;

char *temp,
                    *username = NULL,
                    *password = NULL,
                    *musicname = NULL,
                    *musicFileName = NULL;

FILE *fd;

username = getUsername(&username_length);
if (username == NULL) {
        printf("ERROR, username couldn't be read\n");
        return;
}
password = getPassword(&password_length);
if (password == NULL) {
        printf("ERROR, password couldn't be read\n");
        return;
}

musicname = getMusicName(&musicname_length);
if (musicname == NULL) {
        printf("ERROR, music name couldn't be read\n");
        return;
```

```
}

musicFileName = getMusicFileName(&musicFileName_length);
if (musicFileName == NULL) {
        printf("ERROR, music file name couldn't be read\n");
        return;
}

fd = fopen(musicFileName, "rb");
if (fd == NULL) {
        printf("ERROR, music file couldn't be opened to read\n");
        free(username);
        free(password);
        free(musicname);
        return;
}
free(musicFileName);

//get file size
fseek(fd, 0, SEEK_END);
musicFile_length = ftell(fd);
fseek(fd, 0, SEEK_SET);

i = 0;
sockfd = cli_connect();
while (sockfd < 0 && i < 2) {
```

```
            sockfd = cli_connect();
        }
        if (sockfd < 0) {
            printf("ERROR, Connection is tried 3 times but couldn't be set up\n");
            free(username);
            free(password);
            free(musicname);
            free(musicFileName);
            fclose(fd);
            return;
        }
        printf("INFO, Connection is successfully established\n");

        buffer_length = 4 + username_length + 4 + password_length + 4 + musicname_length +
musicFile_length;

        memset(buffer, 1024, 0);
        //set operation type
        buffer[0] = '5';

        //set buffer length
        temp = int2bytes(buffer_length);
        mystrcpy(&buffer[1], temp, 4);
        free(temp);

        //set username length
```

```
temp = int2bytes(username_length);
mystrcpy(&buffer[5], temp, 4);
free(temp);

//set username
mystrcpy(&buffer[9], username, username_length);
free(username);

//set password_length
temp = int2bytes(password_length);
mystrcpy(&buffer[9 + username_length], temp, 4);
free(temp);

//set password
mystrcpy(&buffer[13 + username_length], password, password_length);
free(password);

//set music name length
temp = int2bytes(musicname_length);
mystrcpy(&buffer[13 + username_length + password_length], temp, 4);
free(temp);

//set music name
mystrcpy(&buffer[17 + username_length + password_length], musicname, musicname_length);
free(musicname);
```

```
	n = send(sockfd, buffer, 5 + buffer_length - musicFile_length, 0);
	if (n < 0) {
		printf("ERROR, request-type, user name, password and music name couldn't be written to the socket\n");
		fclose(fd);
		close(sockfd);
		return;
	}
	printf("INFO, meta data of file and user is successfully sent\n");
	printf("INFO, file is being uploaded\n");
	while(musicFile_length > 1024) {
		for(i=0; i<1024; i++) {
			buffer[i] = fgetc(fd);
		}
		n = send(sockfd, buffer, 1024, 0);
		if (n < 0) {
			printf("ERROR, request-part file couldn't be written to the socket\n");
			fclose(fd);
			close(sockfd);
			return;
		}
		musicFile_length -= 1024;
	}
	for(i=0; i<musicFile_length; i++) {
		buffer[i] = fgetc(fd);
	}
```

```
	fclose(fd);

	n = send(sockfd, buffer, musicFile_length, 0);

	if (n < 0) {

		printf("ERROR, request-part file couldn't be written to the socket\n");

		close(sockfd);

		return;

	}

	printf("INFO, file data is sent, waiting for response\n");


	n = recv(sockfd, buffer, 1, 0);

	if (n < 0) {

		printf("ERROR, response-type couldn't be read from the socket\n");

		close(sockfd);

		return;

	}

	if (buffer[0] == '0') {

		printf("INFO, Music file is successfully uploaded to the server\n");

	} else if (buffer[0] == '1') {

		printf("ERROR, response-type is error, something unexpected may have happened in the
server\n");

	} else {

		printf("ERROR, response-type is error, server returned authentication failure\n");

	}


	close(sockfd);

}
```

```
void mystrcpy(char *dest, char *src, int len) {
	int i;
	for(i=0; i<len; i++) {
		dest[i] = src[i];
	}
}

char* getMusicFileName(int *musicFileName_length) {
	char *musicFileName;

	//use the type-ahead input
	while(fgetc(stdin) != '\n');

	musicFileName = malloc(sizeof(char) * 257);
	if (musicFileName == NULL) return musicFileName;
	printf("Please enter music file name (max-256) : ");
	scanf("%256[^\n]", musicFileName);
	*musicFileName_length = strlen(musicFileName);
	return musicFileName;
}

char* getMusicName(int *musicname_length) {
	char *musicname;

	//use the type-ahead input
```

```
	while(fgetc(stdin) != '\n');


	musicname = malloc(sizeof(char) * 257);
	if(musicname == NULL) return musicname;
	printf("Please enter music name (max-256) : ");
	scanf("%256[^\n]", musicname);
	*musicname_length = strlen(musicname);
	return musicname;
}

char* getUsername(int *username_length) {
	char *username;

	//use the type-ahead input
	while(fgetc(stdin) != '\n');


	username = malloc(sizeof(char) * 257);
	if (username == NULL) return username;
	printf("Please enter username (max-256) : ");
	scanf("%256[^\n]", username);
	*username_length = strlen(username);
	return username;
}

char* getPassword(int *password_length) {
	char *password;
```

```
        //use the type-ahead input
        while(fgetc(stdin) != '\n');

        password = malloc(sizeof(char) * 257);
        if (password == NULL) return password;
        printf("Please enter password (max-256) : ");
        scanf("%256[^\n]", password);
        *password_length = strlen(password);
        return password;
}

int cli_connect() {
        int sockfd;
        struct sockaddr_in serv_addr;

        sockfd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
        if (sockfd < 0)  {
                printf("ERROR, socket couldn't be opened\n");
                return -1;
        }

        memset((char *) &serv_addr, sizeof(serv_addr), 0);
        serv_addr.sin_family = AF_INET;
        bcopy((char *)server->h_addr, (char *)&serv_addr.sin_addr.s_addr, server->h_length);
```

```
	serv_addr.sin_port = htons(portno);


	if (connect(sockfd, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0) {
		printf("ERROR, connection couldn't be set up\n");
		return -1;
	}
	return sockfd;
}


char* int2bytes(int integer) {
	char *bytes = malloc(sizeof(char) * 4);
	int base2 = 256 * 256;
	int base3 = base2 * 256;

	bytes[0] = (integer / base3);
	integer -= base3 * bytes[0];
	bytes[1] = (integer / base2);
	integer -= base2 * bytes[1];
	bytes[2] = (integer / 256);
	integer -= 256 * bytes[2];
	bytes[3] = integer;
	return bytes;
}

int bytes2int(char *bytes) {
```

```
        int integer = 0;

        if (bytes[3] < 0) integer += 256 + bytes[3];

        else integer += bytes[3];


        if (bytes[2] < 0) integer += (256 + bytes[2]) * 256;

        else integer += bytes[2] * 256;


        if (bytes[1] < 0) integer += (256 + bytes[1]) * 256 * 256;

        else integer += bytes[1] * 256 * 256;


        if (bytes[0] < 0) integer += (256 + bytes[0]) * 256 * 256 * 256 ;

        else integer += bytes[0] * 256 * 256 * 256;


        return integer;

}
```

Makefile

```
program: client server

client:

        gcc -Wall client.c -o client

server:

        gcc -Wall server.c -o server
```