Design Document - Reads Profiler

Loghin Vlad-Andrei

Alexandru Ioan Cuza University, Faculty of Computer Science

Abstract. This document represents the design of the server/client online library application. It contains details regarding the technologies used, logical implementation, programming implementation examples, possible future additions in the form of updates, and references to the external documentation used.

1 Introduction

1.1 Client Application Description

ReadsProfiler is an application that gives an user access to an online library of books. Through the client application, the user can look for specific books by search or by filter - published year, rating, ISBN (International Standard Book Number) or Genre. Once a desired book is found, the user can download a copy to their local machine.

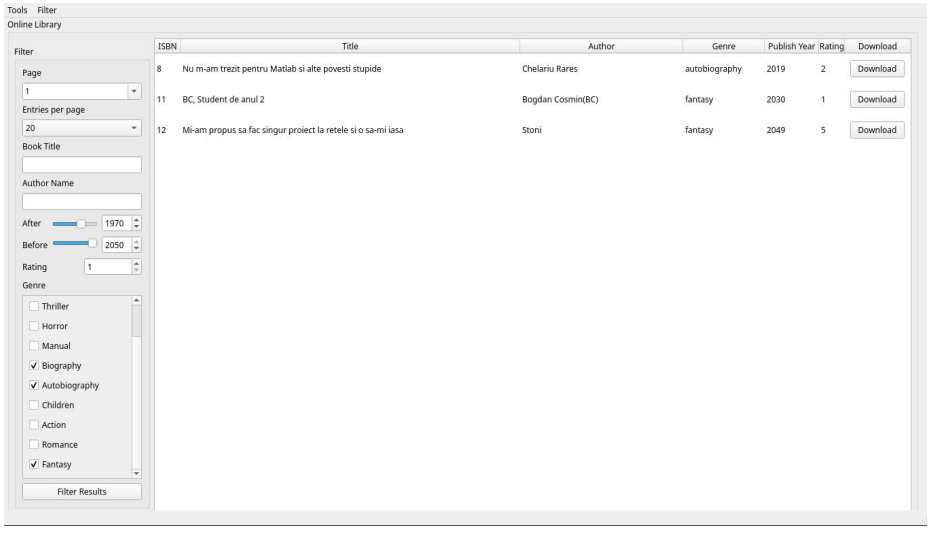

Fig. 1. Image above contains a prototype of the client application interface

1.2 Server Application Description

In order to access the library, the client application has to connect to a dedicated server which will process all clients' applications requests. The server is able to treat all connected clients concurrently without errors due to the technologies used in its' implementation.
For a user to be granted access to the library, he will have to log into the server using an account. Afterwards, any filter/download requests will be handled by the server through direct communication.

2 Utilized Technologies

2.1 List of Technologies Utilized in the Application's Implementation

1. **TCP/IP communication between server and client**
   The communication between any client and the server is done by TCP (Transmission Control Protocol). Almost all data sent between the application and the server needs to be accurate : login information, book. This meant that the communication had to be implemented with TCP in mind as it ensures that messages are transmitted correctly.
2. **Multithreading on the Server**
   In order to ensure concurrent communication (more than one client can be "served" at a given time), dynamic multithreading was implemented on the server. Once a client attempts to connect to the server, a thread is created and assigned to the communication with it. The thread will destroy itself once the client disconnects. Each thread will have its' own resources and an address to the server instance, in order to use the server's procedures.
3. **File Section Locking**
   Threads can access a server's function that will log specific events that happen during the run time. Due to the possible existence of multiple threads, the data racing problem arises when more threads write on a log file at the same time. In order to solve this, file sequence locks will be used.
4. **Non-Relational Databases - JSON Database format**
   Storing users' login information and the digital books is done with the use of JSON format non-relational databases, due to them not requiring usage of normal forms (each user has a single password, each book has a single author, publish date, etc.). This format has been chosen due to the ease of implementation in this case.
5. **Singleton Pattern (Emulation)**
   Due to multiple threads requiring the usage of server functions, the thread has to have access to the server. Since only one server has to be running at all times, we will assign an address to the existing server instance for each thread, which will be created only at the start-up of the server-side application. This is an emulation of the Singleton Pattern as it does not use a static address.

6. **Qt 5 - Client Graphical Interface**
   The client application's graphical interface was designed using Qt 5.12.6 due to its' flexible interface tools and its' c++ and c integration. Due to its' implementation of objects and widgets with heavy use of abstract classes, it is intuitive to use and understand, and by providing an event driven programming paradigm, integration with the server-side application was easily achievable.

3 Application Architecture

3.1 Concepts Used in Implementation

Alongside the technologies mentioned in the previous section, a couple of concepts have also been designed for the ease of use of the application, for the prevention of server flooding and for the ease of data handling.

First of all, a login bootstrap has been implemented, ensuring the following :

1. A user can only log on if he has an existing account registered in the database
2. A user cannot log on with an account that is already logged on and using the application
3. A user cannot flood the login system as he will be timed out after too many unsuccessful attempts
4. The server cannot be overloaded with users as it has a user capacity (flexible, adjustable based on the machine that runs the server-side application
5. Only once a user has entered the correct credentials and has not been timed out, he can access the application.

Therefore, the server's security is ensured.

Second, data containing book information, be it a search filter or a search result, has been stored in intuitive structures and codified accordingly in messages sent between the server and the client.

1. Data containing the filter information, which is passed from the client to the server is in a string, each filter criteria being separated by semi-columns. An example of this would be : 2;10;Loghin;C++;1970;2010;4;2047. By deciphering this, the server can state the following :
   (a) The client is looking for books on the second page
   (b) The client requests a maximum of 10 entries per page
   (c) The client is looking for books written by any author whose full name contains 'Loghin'
   (d) The client is looking for books which contain 'C++' in the title
   (e) The client is looking for books published between 1970 and 2010
   (f) The client is looking for books rated 4/5 and higher
   (g) The client is looking for books of all genres (Below there will be an explanation for this number)

Fields in this passed strings can be void, as in the following example : 1;20;;;1950;;;; . In this example, the client has only specified that he wants the first 20 books published after 1950.

2. **The last given item in the string represents the genre filter flag**. A filter flag is represented by a number in base 2. Each byte will represent a specific genre, making filtering these in the database further on much easier. Example : 1000000011 represents books of the fantasy, comedy and drama genres, as 512 represents the fantasy flag, 2 represents the comedy flag and 1 represents the drama flag. A value of $2^n - 1$ represents all genres.
3. The server will decode the string into a database filter structure, for ease of use. Each item in the structure represents a value used in the filter.
4. Once the database query has finished, information will be stored in a different structure, containing different information. For each structure value, as it will be an array of entries, it will contain :
   (a) The full title of the book
   (b) The full author's name
   (c) The ISBN of the book
   (d) The Rating of the book
   (e) The Year in which the book was published
   (f) (Server Only) The location of the book on the storage device of the server's machine. It will be used once the client requests a download of the specific book.

   This structure will be passed on to the client, so as it can print the results of his search.

Finally, the transfer of the book to the client requesting the download is realized through a locally defined File Transfer Protocol, implemented in a way so it can generate a progress bar in the client application. The thread will send the size of the book file first, then send the book's data in packets of 1 Kilobyte each. In order to ensure that more clients can download the same file at the same time, a copy is created on the server each time one requests a download, resulting in the copy being sent to the user. Afterwards, the sender thread will delete the temporary file.

3.2 Logical Implementation. Project Diagram

The flowcharts of both client and server-side applications are on the following pages, starting with the server diagram.

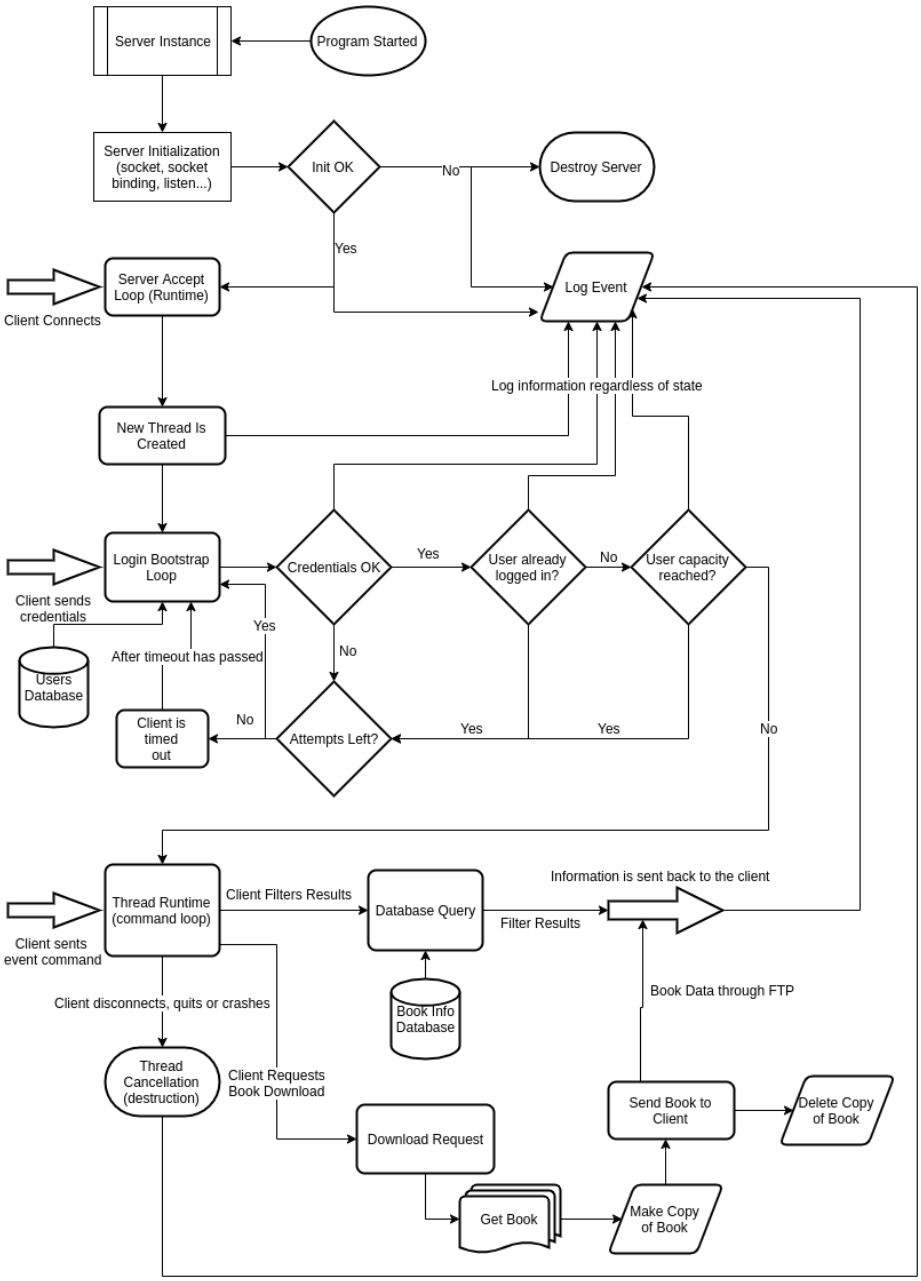

Fig. 2. Above is the logical flowchart of the server-side application.

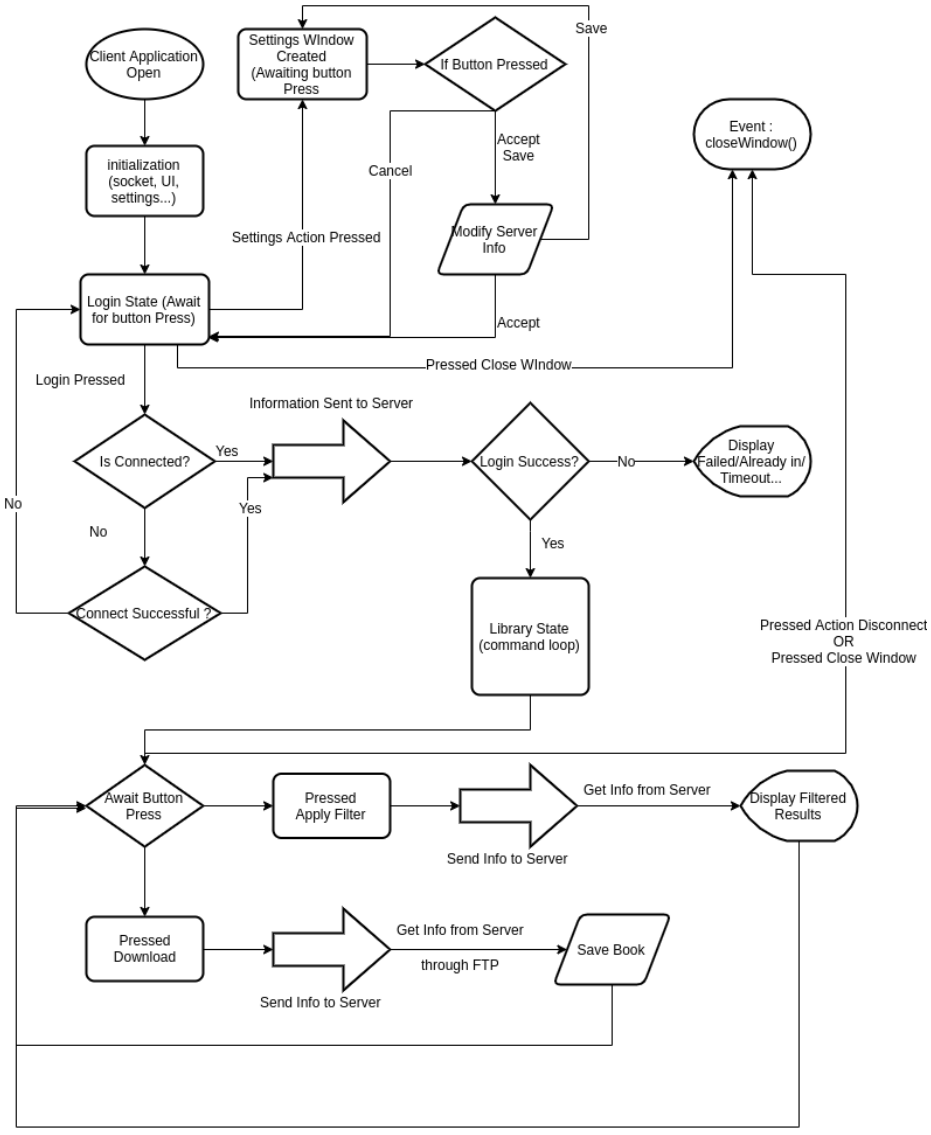

Fig. 3. Above is the logical flowchart of the client-side application.

4 Relevant Code. Implementation Details. Critical Scenarios.

4.1 Implementation Details of the Solutions Devised to Solve Critical Scenarios.

As stated above in sections no. 2 and 3, a few concepts have been used/implemented to ensure the ease of use and smooth runtime of the application. in this section a few design details and quirks will be described alongside the code for their implementation. Code examples will go in the order they are implemented in the flowchart, not in the order they were presented in the previous sections.

Server Relevant Code:

```
void Server::Init()
{
    this->logFile = this->CreateLog();
    if(-1 == (this->socketDescriptor = socket(AF_INET, SOCK_STREAM, NO_DEFINED_PROTOCOL)))
        this->ReportLog(socketError);
    int on = 1;
    setsockopt(this->socketDescriptor, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(int));
    if(-1 == bind(this->socketDescriptor, (sockaddr *)&this->serverInfo, sizeof(sockaddr)))
        this->ReportLog(bindError);
    if(-1 == listen(this->socketDescriptor, MAX_CLIENTS_IN_QUEUE))
        this->ReportLog(listenError);
}

Server::Server(const int& givenPort)
{
    this->sentInfo = nullptr;
    this->pid = 0;
    this->nLoggedUsers = 0;
    this->socketDescriptor = 0;
    this->serverPort = givenPort;
    this->threads = (pthread_t *)malloc(sizeof(pthread_t) * MAX_THREADS);
    this->loggedUsers = (char**)malloc(MAX_USERS * sizeof(size_t));
    memset(this->loggedUsers, 0, MAX_USERS * sizeof(size_t));
    memset(this->threads, 0, MAX_THREADS*sizeof(pthread_t));
    memset(&this->serverInfo, 0, sizeof(sockaddr_in));
    memset(&this->receivedInfo, 0, sizeof(sockaddr_in));
    this->serverInfo.sin_family = AF_INET;
    this->serverInfo.sin_addr.s_addr = htonl(INADDR_ANY);
    this->serverInfo.sin_port = htons(this->serverPort);
}
```

Fig. 4. Server initialization and creation of TCP socket

```
void Server::Runtime()
{
    int client;
    char * serverInput = (char* )malloc(128);
    memset(serverInput,0 ,128);
    unsigned int addrLen;
    int threadIndex = 0;
    Server::thData *threadData;
    while(true)
    {
        addrLen = sizeof(this->receivedInfo);
        this->ReportLog(serverPortWait);
        if( 0 > (client = accept(this->socketDescriptor, (sockaddr*)&this->receivedInfo,(socklen_t*) &addrLen)))
        {
            this->ReportLog(acceptError);
            continue;
        }
        threadData = (thData*)malloc(sizeof(thData));
        threadData->idThread = threadIndex;
        threadData->clientDescriptor = client;
        threadData->parentServer = this;
        pthread_create(&this->threads[threadIndex++], NULL, this->ThreadInitCall, (void*)threadData);
    }
}
```

Fig. 5. Server Client Accept Loop and Thread Creation

```
if(!strcmp(clientRequest, "CLIENTEVENTQUIT"))
    return false;
if(!strcmp(clientRequest, "CLIENTEVENTDISC"))
    return false;
p = strtok(clientRequest, ";");
p = strtok(NULL, ";");

isLoggedOn = usersDB.ConfirmLogon(clientRequest, p);

strcpy(username, clientRequest);
strcpy(clientRequest,  isLoggedOn ? "LOGONACCMPL" : "LOGONFAILED");

if(ThreadTimeout(timeout, timeoutSeconds, timeoutSecondsLeft))
{
    this->ReportLog(threadWriteNotify, threadDataLocal.idThread);
    if( 0 >= write(threadDataLocal.clientDescriptor, "ERRTIMEDOUT", 12))
        this->ReportLog(threadWriteError, threadDataLocal.idThread);
    else
        this->ReportLog(threadWriteSuccess, threadDataLocal.idThread);

    this->ReportLog(threadWriteNotify, threadDataLocal.idThread);
    if( 0 >= write(threadDataLocal.clientDescriptor, &timeoutSecondsLeft, 4))
        this->ReportLog(threadWriteError, threadDataLocal.idThread);
    else
        this->ReportLog(threadWriteSuccess, threadDataLocal.idThread);

    usersDB.~Database();
    continue;
}
```

Fig. 6. Thread Login Timeout Event

```
int Server::GetDateInSeconds(const char* givenDate)
{
    printf("%s0\n", givenDate);
    char *date = (char*)malloc(64);
    strcpy(date, givenDate);
    *strchr(date, '<') = 0;
    int seconds = atoi(strchr(strchr(date, '-') + 1, '-')+1);
    *strchr(strchr(date, '-') + 1,'-') = 0;
    int minutes = atoi(strchr(date, '-')+1);
    *strchr(date, '-') = 0;
    int hours = atoi(date);
    free(date);
    return hours * 3600 + minutes * 60 + seconds;
}

bool Server::ThreadTimeout(const char* startDate, const int& seconds, int& secondsLeft)
{
    if(*startDate == 0)
        return false;
    char *currentDate = (char*)malloc(64);
    strcpy(currentDate, this->GetLocaltime());
    int startSeconds = this->GetDateInSeconds(startDate);
    int currentSeconds = this->GetDateInSeconds(currentDate);
    free(currentDate);
    secondsLeft = seconds - (currentSeconds - startSeconds);
    if(secondsLeft <= 0)
        return false;
    return true;
}
```

Fig. 7. Thread Login Timeout Event cont. (Implementation of ThreadTimeout())

```
void* Server::ThreadInitCall(void* arg)
{
    Server::thData threadDataLocal;
    threadDataLocal = *((thData*)arg);
    threadDataLocal.parentServer->ReportLog(threadMessageWait, threadDataLocal.idThread);
    if(threadDataLocal.parentServer->ThreadLoginBootstrap((thData*)&threadDataLocal))
    {
        threadDataLocal.parentServer->ThreadRuntime(threadDataLocal);
    }
    close(threadDataLocal.clientDescriptor);
    pthread_detach(pthread_self());
    pthread_cancel(pthread_self());
    return (NULL);

}
```

Fig. 8. Thread Initial Function

```
void Server::ReportLog(const int logId, const int threadId,const char* userId)
{
    char* logLine = (char*)malloc(MAXSTRLEN);
    memset(logLine ,0 ,MAXSTRLEN);
    flock fileLock;
    fileLock.l_type = F_WRLCK;
    fileLock.l_len = MAXSTRLEN;
    fileLock.l_start = 0;
    fileLock.l_whence = SEEK_CUR;
    fcntl(this->logFile, F_SETLK, &fileLock);
    switch(logId)
    {…
    }
    fileLock.l_type = F_UNLCK;
    fcntl(this->logFile, F_SETLK, &fileLock);
    close(this->logFile);
    this->logFile = open(this->logName, O_WRONLY,0666);
    lseek(this->logFile, 0, SEEK_END);
}
```

Fig. 9. Usage of Sequence Locks for Logging

```
if(!strcmp(request, "CLIENTEVENTQUER"))
{
    if( 0 > read(threadDataLocal.clientDescriptor, request, 1024) )
        this->ReportLog(threadReadError, threadDataLocal.idThread, threadDataLocal.linkedUser);
    Database locationIndexer("./data/libIndex.json");
    locationIndexer.DecodeFilter(&userFilter, request);
    if(sentInfo != nullptr)
        free(sentInfo);
    sentInfo = (DBResult *)malloc(sizeof(DBResult) * userFilter.entriesPerPage);
    memset(this->sentInfo, 0, sizeof(DBResult) * userFilter.entriesPerPage);
    this->maxEntries = locationIndexer.HandleQuery(this->sentInfo, userFilter.entriesPerPage, &userFilter);
    this->PrintResults(this->sentInfo, this->maxEntries);
    if( 0 >= write(threadDataLocal.clientDescriptor ,this->sentInfo, sizeof(DBResult)*userFilter.entriesPerPage))
        this->ReportLog(threadWriteError, threadDataLocal.idThread, threadDataLocal.linkedUser);
    if( 0 >= write(threadDataLocal.clientDescriptor, &this->maxEntries, 4))
        this->ReportLog(threadWriteError, threadDataLocal.idThread, threadDataLocal.linkedUser);
}
```

Fig. 10. Usage of Database Access for handling client Queries

```
void Server::DisconnectUser(const char* username)
{
    int index = 0;
    for(;index<this->nLoggedUsers;index++)
    {
        if(!strcmp(*(this->loggedUsers+index), username))
            break;
    }
    free(*(this->loggedUsers + index));
    for(; index < this->nLoggedUsers-1; index++)
        *(this->loggedUsers + index) = *(this->loggedUsers + index + 1);
    *(this->loggedUsers + nLoggedUsers-1) = nullptr;
    this->nLoggedUsers--;
}
```

Fig. 11. Thread Disconnect User

Client Relevant Code:

```
MainWindow::MainWindow(QWidget *parent) : QMainWindow(parent), ui(new Ui::MainWindow)
{
    memset(&this->serverInfo, 0 ,sizeof(sockaddr_in));
    this->serverIP = "127.0.0.1";
    this->serverPort = "3000";
    this->entries = nullptr;
    this->serverInfo.sin_family = AF_INET;
    this->serverInfo.sin_addr.s_addr = inet_addr(this->serverIP.toStdString().c_str());
    this->serverInfo.sin_port = htons(atoi(this->serverPort.toStdString().c_str()));    ⚠ implicit conversion loses int
    this->ui->setupUi(this);
    this->ui->actionDisconnect->setEnabled(false);
    this->ui->libGroup->hide();
    this->pressedButton = 0;
    this->downloadDirPath = (char*)malloc(256);
    this->signalMapper = new QSignalMapper(this);
    connect(this->signalMapper, SIGNAL(mapped(int)), this, SLOT(buttonDownload_clicked(int)));
    memset(this->downloadDirPath, 0 ,256);
    strcpy(this->downloadDirPath, "./downloads");
    if(0 == (this->downloadDir = opendir(this->downloadDirPath)))
    {
        mkdir(this->downloadDirPath, 0700);
    }
    closedir(this->downloadDir);
}

void MainWindow::Init()
{
    this->loginAttempts = 3;
    this->logFile = this->CreateLog();
    if( -1 == (this->socketDescriptor = socket(AF_INET, SOCK_STREAM, 0)))
        this->ReportLog(socketError);
}
```

```
void MainWindow::on_pushButtonLogin_clicked()
{
    if(this->GetReconnect() == 1)
    {
        this->GetReconnect() = 0;
        ::close(socketDescriptor);
        socketDescriptor = socket(AF_INET, SOCK_STREAM, 0);
        isConnected = false;
    }
    if(isConnected == false)
    {
        if( -1 == ::connect(this->socketDescriptor, (sockaddr* )&this->serverInfo, sizeof(sockaddr))) // ::connect  (=) extern::connect
        {
            this->ReportLog(connectError);
            QMessageBox *connectFailedBox = new QMessageBox();
            connectFailedBox->setText("Connection failed!");
            connectFailedBox->setInformativeText("Sever might be down. Press Login to try again");
            connectFailedBox->setStandardButtons(QMessageBox::Ok);
            connectFailedBox->setDefaultButton(QMessageBox::Ok);
            connectFailedBox->exec();
            delete connectFailedBox;
            return;
        }
        else
        {
            isConnected = true;
        }
    }
```

Client Initialize and Connect (previous page)

```
void MainWindow::on_pushButtonApplyFilter_clicked()
{
    int flags = GetCurrentGenreFlag();
    int bytesRead = 0;

    this->filterBuffer=(char*)malloc(1024);                                                                    ⚠ use of old-style cast

    memset(this->filterBuffer, 0 , 1024);
    strcpy(this->filterBuffer, "CLIENTEVENTQUER");
    write(this->socketDescriptor, this->filterBuffer, 1024);
    memset(this->filterBuffer, 0 , 1024);
    sprintf(this->filterBuffer, "%d;%d;%s;%s;%d;%d;%d;%d", atoi(ui->comboBoxPage->currentText().toStdString().c_str()), atoi(ui->comboBoxItemsPage->currentTex
            ui->lineEditAuthor->text().toStdString().c_str(), ui->lineEditTitle->text().toStdString().c_str(), ui->dateEditMin->date().year(), ui->dateEditMax
            flags, ui->spinBoxRating->value());
    write(this->socketDescriptor, this->filterBuffer, 1024);
    this->actualNumberEntries = 0;
    int maxNumberEntries = atoi(ui->comboBoxItemsPage->currentText().toStdString().c_str());
    free(this->entries);
    this->entries = (DBResult*)malloc(sizeof(DBResult)*maxNumberEntries);    ⚠ use of old-style cast   ⚠ implicit conversion changes signedness: 'int' to 'u…
    memset(this->entries, 0 ,sizeof(DBResult)*maxNumberEntries);                          ⚠ implicit conversion changes signedness: 'int' to 'unsigned long'
    if( 0 >= (bytesRead = read(socketDescriptor, this->entries, sizeof(DBResult) * maxNumberEntries)))    ⚠ implicit conversion changes signedness: 'int' to…
        if(bytesRead < 0)
            this->ReportLog(readError);
        else                                                                                                   ⚠ add explicit braces to avoid dangling else
            this->ReportLog(connectionLost);

    if( 0 > read(socketDescriptor, &actualNumberEntries, 4))
        this->ReportLog(readError);
    this->PrintResults(this->entries, actualNumberEntries);
    AddToList(this->entries, actualNumberEntries);
    free(this->filterBuffer);
}
```

Fig. 12. Send Filter to Server

```
void SettingsWindow::on_settingsPushButtonSave_clicked()
{
    this->parent->GetIpString() = this->serverIPLineEdit->text();
    this->parent->GetPortString() = this->serverPortLineEdit->text();
    this->parent->GetServerInfo().sin_addr.s_addr = inet_addr(this->parent->GetIpString().toStdString().c_str());
    this->parent->GetServerInfo().sin_port = htons(atoi(this->parent->GetPortString().toStdString().c_str()));    ⚠ implicit conversion loses
    this->parent->GetReconnect() = 1;
}

void SettingsWindow::on_settingsPushButtonAccept_clicked()
{
    this->parent->GetIpString() = this->serverIPLineEdit->text();
    this->parent->GetPortString() = this->serverPortLineEdit->text();
    this->parent->GetServerInfo().sin_addr.s_addr = inet_addr(this->parent->GetIpString().toStdString().c_str());
    this->parent->GetServerInfo().sin_port = htons(atoi(this->parent->GetPortString().toStdString().c_str()));    ⚠ implicit conversion loses
    this->parent->GetReconnect() = 1;

    this->settingsWindow->close();
    this->parent->setEnabled(true);
    delete this;
}

void SettingsWindow::on_settingsPushButtonCancel_clicked()
{
    this->settingsWindow->close();
    this->parent->setEnabled(true);
    //this->~SettingsWindow();
    delete this;
}
```

Fig. 13. Client Settings Window Implementation Example

```
void MainWindow::buttonDownload_clicked(int index)
{
    this->filterBuffer = (char*)malloc(1024);

    memset(this->filterBuffer, 0 ,1024);
    strcpy(this->filterBuffer, "CLIENTEVENTDOWN");
    write(socketDescriptor, this->filterBuffer, 1024);

    write(socketDescriptor, &index, 4);
    char *downloadFileName = (char*)malloc(256);
    memset(downloadFileName, 0 ,256);
    read(socketDescriptor, this->filterBuffer, 1024);
    strcpy(downloadFileName, this->downloadDirPath);
    strcat(downloadFileName, "/");
    strcat(downloadFileName, this->filterBuffer);
    int bookDescriptor = open(downloadFileName, O_WRONLY | O_CREAT | O_TRUNC, 0666);

    long long fileSize = 0;

    read(socketDescriptor, &fileSize, 8);
    int lbytes = 0;
    while((lbytes = read(socketDescriptor, this->filterBuffer, 1024)))          implicit conv
    {
        if(*this->filterBuffer == 0)
            break;
        write(bookDescriptor, this->filterBuffer, strlen(this->filterBuffer));
    }
    ::close(bookDescriptor);

    free(downloadFileName);
    free(this->filterBuffer);
}
```

Fig. 14. Client Download Book Implementation Example

5 Conclusions. Future Updates.

5.1 Conclusions. Near Future Updates.

The client-server application can still be improved, both in design and in implementation. Updates that are to be done until the project's deadline are, as follows:

1. Implementation of mutex locks on threads for data protection in simultaneous access of the same shared data. (i.e. disconnecting an user)
2. Implementation of a system tracking an user's preferred book genres, authors, etc.
3. Implementation of super-users (administrators) that can temporarily block access to a book.
4. Implementation of an user review system.

5.2 Far Future Updates

If the project were to be updated past the deadline, the following features could be implemented to improve it further:

1. Implementation of remove/ban/kick accounts features, only available to administrator users.
2. Implementation of a feature that allows users to upload their own published works.
3. Implementation of a feature that allows users to recommend books to other users.
4. Implementation of an Internet Browser client, and HTTP protocol as client base.
5. Implementation of a mobile phone Client for the application.
6. Implementation of a top list which contains the 3-10 most downloaded books.
7. Implementation of an automatically updating book database, using machine learning, that adds all accessible books on the internet into the database.
8. Implementation of a read-in-app function, allowing the user to read a specific book inside of the app.

6 References

References

1. LNCS Conference proceedings guidelines, http://www.springer.com/lncs/conference-proceedings-guidelines
2. localtime_r(3), https://linux.die.net/man/3/localtime_r
3. QSignalMapper(), https://doc.qt.io/qt-5/qsignalmapper.html
4. File locking in Linux, https://gavv.github.io/articles/file-locks/
5. Basic dynamic multi-threaded server, https://profs.info.uaic.ro/ computernetworks/files/NetEx/S12/ServerConcThread/servTcpConcTh2.c