

ecet4640-lab5

1.0

Generated by Doxygen 1.9.8



<b>1 ecet4640-lab5</b>	<b>1</b>
1.1 Intro	1
1.2 Contributions	1
1.3 Overview	2
<b>2 Compilation</b>	<b>3</b>
2.1 Compilation Pipelines	3
2.2 Compiling and running	3
2.3 Screenshot of Compilation	4
2.4 Cleaning	4
<b>3 Topic Index</b>	<b>5</b>
3.1 Topics	5
<b>4 Data Structure Index</b>	<b>7</b>
4.1 Data Structures	7
<b>5 File Index</b>	<b>9</b>
5.1 File List	9
<b>6 Topic Documentation</b>	<b>11</b>
6.1 Build	11
6.1.1 Detailed Description	11
6.1.2 Function Documentation	11
6.1.2.1 CreateUsersArray()	11
6.1.2.2 CreateUsersMap()	12
6.2 Cipher	12
6.2.1 Detailed Description	13
6.2.2 Function Documentation	13
6.2.2.1 GenerateCipher()	13
6.2.2.2 PrintCipher()	13
6.2.2.3 EncryptString()	14
6.2.2.4 DecryptString()	14
6.3 Connection	15
6.3.1 Detailed Description	16
6.3.2 Function Documentation	16
6.3.2.1 InitializeShared()	16
6.3.2.2 StartUpdateThread()	16
6.3.2.3 StartConnectionThread()	17
6.3.2.4 MessageOrClose()	18
6.3.2.5 MessageAndClose()	20
6.3.2.6 _help()	20
6.3.2.7 _register()	21
6.3.2.8 _myinfo()	22

6.3.2.9 _who()	23
6.3.2.10 _rand_gpa()	24
6.3.2.11 _rand_age()	25
6.3.2.12 _advertisement()	26
6.3.2.13 _password()	27
6.3.2.14 _authenticate()	28
6.4 Data	29
6.4.1 Detailed Description	29
6.4.2 Macro Definition Documentation	29
6.4.2.1 RECORD_COUNT	29
6.4.2.2 ID_MAX_LENGTH	29
6.4.2.3 NAME_MAX_LENGTH	30
6.4.2.4 IP_LENGTH	30
6.4.2.5 PASSWORD_LENGTH	30
6.4.3 Variable Documentation	30
6.4.3.1 accepted_userIDs [1/2]	30
6.4.3.2 userFullNames [1/2]	31
6.4.3.3 accepted_userIDs [2/2]	31
6.4.3.4 userFullNames [2/2]	31
6.5 Files	31
6.5.1 Detailed Description	32
6.5.2 Macro Definition Documentation	32
6.5.2.1 LOCKFILE	32
6.5.2.2 REGISTERED_FILE	32
6.5.2.3 SERVER_SETTINGS_FILE	33
6.5.2.4 KEY_FILE	33
6.5.2.5 ADS_DIR	33
6.5.3 Function Documentation	34
6.5.3.1 FileStatus()	34
6.5.3.2 CreateOrOpenFileVerbose()	34
6.5.3.3 ReadKeyIntoSettingsMap()	35
6.5.3.4 ReadRegisteredFileIntoUsersMap()	36
6.5.3.5 UpdateRegisteredFileFromUsersMap()	37
6.5.3.6 NumberOfFilesInDirectory()	38
6.5.3.7 GetRandomFileNameFromDir()	38
6.5.3.8 ReadSettingsFileIntoSettingsMap()	39
6.5.3.9 CatFileToBuffer()	40
6.6 Logfile	41
6.6.1 Detailed Description	41
6.6.2 Function Documentation	41
6.6.2.1 SetLogfileName()	41
6.6.2.2 LogfileError()	41

6.6.2.3 LogfileMessage()	42
6.7 Map	43
6.7.1 Detailed Description	44
6.7.2 Function Documentation	44
6.7.2.1 NewMap()	44
6.7.2.2 Map_Set()	45
6.7.2.3 Map_Get()	45
6.7.2.4 Map_Delete()	46
6.8 Server	46
6.8.1 Detailed Description	47
6.8.2 Function Documentation	47
6.8.2.1 StartServer()	47
6.8.2.2 NextAvailableConnection()	48
6.8.2.3 CloseServer()	49
6.8.2.4 InitializeServer()	49
6.8.2.5 InitializeCipher()	49
6.9 Util	50
6.9.1 Detailed Description	51
6.9.2 Macro Definition Documentation	51
6.9.2.1 COLOR_RED	51
6.9.2.2 COLOR_GREEN	51
6.9.2.3 COLOR_YELLOW	51
6.9.2.4 COLOR_BLUE	51
6.9.2.5 COLOR_RESET	51
6.9.3 Function Documentation	51
6.9.3.1 printRed()	51
6.9.3.2 printGreen()	52
6.9.3.3 printYellow()	53
6.9.3.4 printBlue()	53
6.9.3.5 RandomInteger()	54
6.9.3.6 RandomFloat()	55
6.9.3.7 RandomFlag()	55
6.10 Process	56
6.10.1 Detailed Description	56
6.10.2 Function Documentation	56
6.10.2.1 InitializeCipher()	56
6.10.2.2 RunHeadless()	57
6.10.3 Variable Documentation	58
6.10.3.1 users_array	58
6.10.3.2 users_map	58
6.10.3.3 settings_map	58
6.10.3.4 default_settings	58

6.10.3.5 active_clients	58
<b>7 Data Structure Documentation</b>	<b>59</b>
7.1 _map_bucket Struct Reference	59
7.1.1 Detailed Description	59
7.2 ClientShared Struct Reference	59
7.2.1 Detailed Description	60
7.2.2 Field Documentation	60
7.2.2.1 users	60
7.2.2.2 mutex	60
7.2.2.3 dirty	60
7.2.2.4 shutting_down	60
7.2.2.5 send_buffer_size	61
7.2.2.6 receive_buffer_size	61
7.2.2.7 cipher	61
7.2.2.8 start	61
7.2.2.9 end	61
7.3 Connection Struct Reference	62
7.3.1 Detailed Description	62
7.3.2 Field Documentation	62
7.3.2.1 status	62
7.3.2.2 socket	63
7.3.2.3 address	63
7.3.2.4 address_length	63
7.3.2.5 thread_id	63
7.3.2.6 time_connected	63
7.3.2.7 state	63
7.3.2.8 user	64
7.4 map Struct Reference	64
7.4.1 Detailed Description	64
7.5 map_result Struct Reference	65
7.5.1 Detailed Description	65
7.6 ServerProperties Struct Reference	65
7.6.1 Detailed Description	65
7.6.2 Field Documentation	65
7.6.2.1 port	65
7.6.2.2 send_buffer_size	66
7.6.2.3 receive_buffer_size	66
7.6.2.4 socket_id	66
7.6.2.5 backlog	66
7.6.2.6 active_connections	66
7.6.2.7 max_connections	66

---

7.6.2.8 time_started . . . . .	67
7.6.2.9 cipher . . . . .	67
7.6.2.10 start . . . . .	67
7.6.2.11 end . . . . .	67
7.7 User Struct Reference . . . . .	67
7.7.1 Detailed Description . . . . .	68
<b>8 File Documentation</b>	<b>69</b>
8.1 Build.c . . . . .	69
8.2 Build.h . . . . .	69
8.3 Cipher.c . . . . .	70
8.4 Cipher.h . . . . .	71
8.5 colors.h . . . . .	71
8.6 Connection.c . . . . .	71
8.7 Connection.h . . . . .	77
8.8 Data.c . . . . .	78
8.9 Data.h . . . . .	78
8.10 File.c . . . . .	79
8.11 File.h . . . . .	82
8.12 Logfile.c . . . . .	82
8.13 Logfile.h . . . . .	84
8.14 main.c . . . . .	84
8.15 map.c . . . . .	84
8.16 map.h . . . . .	86
8.17 Process.c . . . . .	87
8.18 Process.h . . . . .	90
8.19 Server.c . . . . .	91
8.20 Server.h . . . . .	93
8.21 Util.c . . . . .	94
8.22 Util.h . . . . .	95
<b>Index</b>	<b>97</b>





# Chapter 1

## ecet4640-lab5

### 1.1 Intro

This program starts a Ipv4 server that listens on a port for incoming connections. For each new connection, it starts a thread. Connecting users send strings to execute various actions on the server. It uses a substitution cipher to encrypt messages sent to clients, and the same cipher to decrypt messages sent from clients.

### 1.2 Contributions

*Note: This includes the contributions to lab 5, which lab 6 built off of.*

- On 9/29, Christian made the initial repository.
- On 9/29, Karl made the Makefile, copied the example client and server, created the Build, Data, File, Process, Util, and map modules.
- On 9/30, Karl added logic for reading the registered users file into a map and initializing/binding the server socket; created the Server and [Connection](#) modules.
- On 10/1, Paul and Karl did work on reading the server-settings.txt file, and creating a new thread for a client connection.
- On 10/2, Christian added a Log module and added some functionality to support various log levels.
- On 10/5, Paul, Karl, and Christian began adding different command line arguments the server could take.
- On 10/5, Karl and Paul fixed some bugs related to the multithreading.
- On 10/5, Paul added the myinfo and register commands and fixed typos.
- On 10/8, Christian added features for initializing the logger and fixed a bug related to disconnects.
- On 10/12, Karl, Paul and Christian changed the message format to the current format, set commands to be lowercase, and improved the messages sent to users.
- On 10/16, Christian added features to the `_rand_age` function.
- On 10/18, Karl fixed a segfault bug that occurred when users entered an invalid ID.
- On 10/19, Christian added the call to updating the registered file as users register
- On 10/19, Christian, Karl, and Paul added more command line arguments to the server.

- On 10/21, Christian implemented the advertisement feature and removed debug prints.
- On 11/1, Christian created [Cipher.c](#) and [Cipher.h](#), and added procedures in Process, Server, and [Connection](#) to manage encryption and decryption.
- On 11/2, Karl added [Logfile.h](#) and [Logfile.c](#) to enable logging activity to a file. Christian condensed the code and fixed a decryption bug.
- On 11/9, Christian and Paul worked together on decryption, and began the procedures for handling user passwords.
- On 11/10, Paul finished the password.
- On 11/12, Paul and Karl fixed a bug with registration, improved logging coverage, and added colored ads.
- On 12/3, Karl and Paul removed unused logging functions.

## 1.3 Overview

Argument	Description	Calls
none	Defaults to RunCommand; runs server attached to terminal	RunCommand()
headless	Runs the server with .nohup, as a background process.	<a href="#">RunHeadless()</a>
stop	Stops an existing server process if it is running.	<a href="#">StopCommand()</a>

### Author

Karl Miller  
Paul Shriner  
Christian Messmer

## Chapter 2

# Compilation

### 2.1 Compilation Pipelines

There are several compilation pipelines, which are described in more detail in the Makefile comments.

The first is for making and running the regular server process. Calling `make` executes this. It uses the files in `src/server` to generate the binary and runs it. This will run the binary after it is built, and the default command will cause it to run in the server. Executing `make server` will make the server binary without running it.

The second is for making the test binary. This compiles the files in `tests` and the files in `src/server`, but excludes `src/main.c` so that `tests/main_test.c` will be the program entry point instead. The tests use `CuTest`. The tests are not documented here in order to not inflate the documentation size any further.

### 2.2 Compiling and running

1. Copy the .zip file to the server.
2. Extract the zip file.
3. Enter the unzipped folder.
4. Run `make server`
5. Run `./server` to run the server attached to the shell.
6. Press `ctrl+c` to exit and close the server.
7. Run `./server headless` to run the server headlessly.
8. Run `./server stop` to stop the headless server.
9. If a better client is not available, you can use the example client to connect.
10. `cd` into the `/example` folder
11. run `gcc client.c`
12. run `./a.out` and input '3001' as the port.

## 2.3 Screenshot of Compilation

```

[shr5683@draco1 ecet4640-lab6]$ make server
mkdir -p bin/src/server/
gcc -Wall -Itests -Itests/lib -Isrc -Isrc/server -Isrc/client -c src/server/Util.c -o bin/src/server/Util.c.o
mkdir -p bin/src/server/
gcc -Wall -Itests -Itests/lib -Isrc -Isrc/server -Isrc/client -c src/server/Connection.c -o bin/src/server/Connection.c.o
mkdir -p bin/src/server/
gcc -Wall -Itests -Itests/lib -Isrc -Isrc/server -Isrc/client -c src/server/File.c -o bin/src/server/File.c.o
mkdir -p bin/src/server/
gcc -Wall -Itests -Itests/lib -Isrc -Isrc/server -Isrc/client -c src/server/Data.c -o bin/src/server/Data.c.o
mkdir -p bin/src/server/
gcc -Wall -Itests -Itests/lib -Isrc -Isrc/server -Isrc/client -c src/server/Logfile.c -o bin/src/server/Logfile.c.o
mkdir -p bin/src/server/
gcc -Wall -Itests -Itests/lib -Isrc -Isrc/server -Isrc/client -c src/server/map.c -o bin/src/server/map.c.o
mkdir -p bin/src/server/
gcc -Wall -Itests -Itests/lib -Isrc -Isrc/server -Isrc/client -c src/server/main.c -o bin/src/server/main.c.o
mkdir -p bin/src/server/
gcc -Wall -Itests -Itests/lib -Isrc -Isrc/server -Isrc/client -c src/server/Process.c -o bin/src/server/Process.c.o
mkdir -p bin/src/server/
gcc -Wall -Itests -Itests/lib -Isrc -Isrc/server -Isrc/client -c src/server/Server.c -o bin/src/server/Server.c.o
mkdir -p bin/src/server/
gcc -Wall -Itests -Itests/lib -Isrc -Isrc/server -Isrc/client -c src/server/Cipher.c -o bin/src/server/Cipher.c.o
src/server/Cipher.c: In function 'PrintCipher':
src/server/Cipher.c:61:16: warning: universal character names are only valid in C++ and C99 [enabled by default]
   printf("%s%s%s%s%s \u00BB %s%s%s%s%s", COLOR_GRAY, COLOR_RED, i + start, COLOR_GRAY, COLOR_RESET);
                   ^
src/server/Cipher.c:61:16: warning: universal character names are only valid in C++ and C99 [enabled by default]
   printf("%s%s%s%s%s \u00BB %s%s%s%s%s", COLOR_GRAY, COLOR_RED, i + start, COLOR_GRAY, COLOR_RESET);
                   ^
mkdir -p bin/src/server/
gcc -Wall -Itests -Itests/lib -Isrc -Isrc/server -Isrc/client -c src/server/Build.c -o bin/src/server/Build.c.o
gcc bin/src/server/Util.c.o bin/src/server/Connection.c.o bin/src/server/File.c.o bin/src/server/Data.c.o bin/src/server/Logfile.c.o bin/src/server/map.c.o bin/src/server/main.c.o bin/src/server/Process.c.o bin/src/server/Server.c.o bin/src/server/Cipher.c.o bin/src/server/Build.c.o -lpthread -o server
[shr5683@draco1 ecet4640-lab6]$

```

Figure 2.1 Compiling on draco1

## 2.4 Cleaning

`make clean` will clean all .o files and binaries.

## Chapter 3

# Topic Index

### 3.1 Topics

Here is a list of all topics with brief descriptions:

Build . . . . .	11
Cipher . . . . .	12
Connection . . . . .	15
Data . . . . .	29
Files . . . . .	31
Logfile . . . . .	41
Map . . . . .	43
Server . . . . .	46
Util . . . . .	50
Process . . . . .	56



## Chapter 4

# Data Structure Index

### 4.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">_map_bucket</a>	Map_bucket is an endpoint in the map. It is also a node in a linked list; if there were collisions, then the buckets are appended to the linked list at that location, then traversed until the matching key is found . . . . .	59
<a href="#">ClientShared</a>	. . . . .	59
<a href="#">Connection</a>	. . . . .	62
<a href="#">map</a>	A map. Stores key-value pairs for near constant lookup and insertion time . . . . .	64
<a href="#">map_result</a>	The result of a map retrieval . . . . .	65
<a href="#">ServerProperties</a>	. . . . .	65
<a href="#">User</a>	. . . . .	67





# Chapter 5

## File Index

### 5.1 File List

Here is a list of all documented files with brief descriptions:

src/server/Build.c	69
src/server/Build.h	69
src/server/Cipher.c	70
src/server/Cipher.h	71
src/server/colors.h	71
src/server/Connection.c	71
src/server/Connection.h	77
src/server/Data.c	78
src/server/Data.h	78
src/server/File.c	79
src/server/File.h	82
src/server/Logfile.c	82
src/server/Logfile.h	84
src/server/main.c	84
src/server/map.c	84
src/server/map.h	86
src/server/Process.c	87
src/server/Process.h	90
src/server/Server.c	91
src/server/Server.h	93
src/server/Util.c	94
src/server/Util.h	95



# Chapter 6

## Topic Documentation

### 6.1 Build

Functions for creating and populating data structures.

#### Functions

- [User](#) \* [CreateUsersArray](#) (char \*\*userIDs, char \*\*userNames, int recordsCount)
- [map](#) \* [CreateUsersMap](#) ([User](#) \*usersArray, int recordsCount)

#### 6.1.1 Detailed Description

#### 6.1.2 Function Documentation

##### 6.1.2.1 CreateUsersArray()

```
User * CreateUsersArray (  
    char ** userIDs,  
    char ** userNames,  
    int recordsCount )
```

Mallocs a new array of [User](#) structs.

#### Parameters

<i>userIDs</i>	An array of userIDs to set.
<i>userNames</i>	An array of userNames corresponding to the userIDs.
<i>recordsCount</i>	The number of records in userIDs and userNames, and the size of the created an array.

#### Returns

A malloced array of user structs.

Definition at line 10 of file [Build.c](#).

### 6.1.2.2 CreateUsersMap()

```
map * CreateUsersMap (
    User * usersArray,
    int recordsCount )
```

Given a user's array, initializes a new map that points to the underlying data in the array, using the user's ID as a key.

#### Parameters

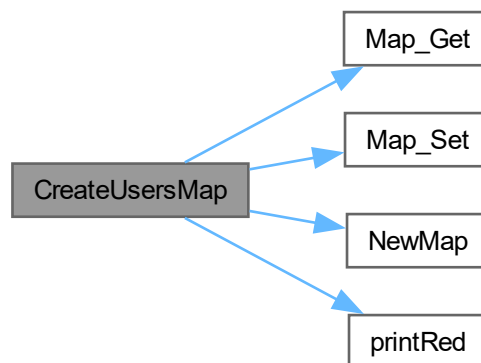
<i>usersArray</i>	The array used to build the user's map.
<i>recordsCount</i>	The number of records in the user's array.

#### Returns

A map

Definition at line 24 of file [Build.c](#).

Here is the call graph for this function:



## 6.2 Cipher

This module handles encryption and decryption.

#### Functions

- void [GenerateCipher](#) (char \*cipher, char start, char end)  
*Generates a random cipher.*
- void [PrintCipher](#) (char \*cipher, char start, char length)  
*Prints the cipher.*
- void [EncryptString](#) (char \*string, int length, char \*cipher, char start, char end)  
*Encrypts the given string.*
- void [DecryptString](#) (char \*string, int length, char \*cipher, char start, char end)  
*Decrypts the given string.*

## 6.2.1 Detailed Description

## 6.2.2 Function Documentation

### 6.2.2.1 GenerateCipher()

```
void GenerateCipher (
    char * cipher,
    char start,
    char end )
```

Modular, intended to be reusable.

The array is first filled with the characters between start and end. Then the array is traversed. Each element is swapped with some random other element. Each element is swapped at least once.

#### Parameters

<i>cipher</i>	The cipher to fill.
<i>start</i>	The character to start.
<i>end</i>	The character the cipher will end on (inclusive).

#### Attention

Cipher is at least end-start in length.

Mutates: Fills cipher randomly with characters between start and end

Definition at line 42 of file [Cipher.c](#).

### 6.2.2.2 PrintCipher()

```
void PrintCipher (
    char * cipher,
    char start,
    char length )
```

Modular, intended to be reusable.

Prints the cipher in a series of columns describing what each character will be transformed into.

Uses [colors.h](#).

#### Parameters

<i>cipher</i>	The cipher to print.
<i>start</i>	The character started on cipher
<i>length</i>	The length of the cipher

Definition at line 60 of file [Cipher.c](#).

### 6.2.2.3 EncryptString()

```
void EncryptString (
    char * string,
    int length,
    char * cipher,
    char start,
    char end )
```

Modular, intended to be reusable.

Replaces the string in place, mutating it. Anything out of bounds of the cipher will not be encrypted and will stay as its original character.

#### Attention

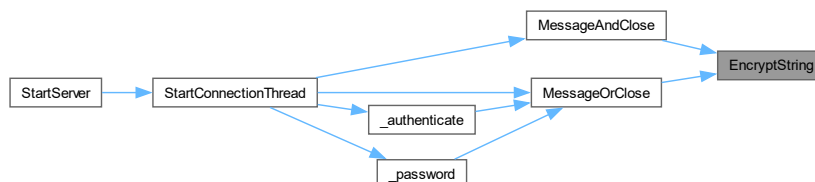
mutatates: encrypts the string in place, destroying the original characters

#### Parameters

<i>string</i>	String to ecrypt.
<i>length</i>	Length of <i>string</i>
<i>cipher</i>	Cipher to use for encrypting the string. Must be (start-end)+1 in size.
<i>start</i>	The first character the cipher uses
<i>end</i>	The last character the cipher uses

Definition at line 76 of file [Cipher.c](#).

Here is the caller graph for this function:



### 6.2.2.4 DecryptString()

```
void DecryptString (
    char * string,
    int length,
    char * cipher,
    char start,
    char end )
```

Modular, intended to be reusable.

Replaces the string in place, mutating it. Anything out of bounds of the cipher will not be decrypted and will stay as its original character.

**Attention**

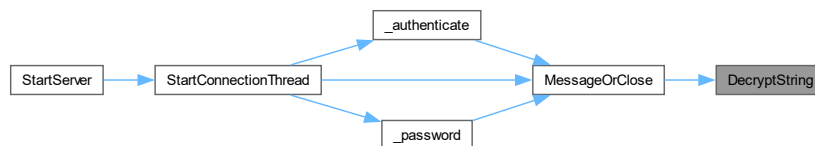
mutatates: encrypts the string in place, destroying the original characters

**Parameters**

<i>string</i>	String to decrypt.
<i>length</i>	Length of <i>string</i>
<i>cipher</i>	Cipher to use for encrypting the string. Must be (start-end)+1 in size.
<i>start</i>	The first character the cipher uses
<i>end</i>	The last character the cipher uses

Definition at line 95 of file [Cipher.c](#).

Here is the caller graph for this function:



## 6.3 Connection

This module handles an individual user's active connection.

**Data Structures**

- struct [ClientShared](#)
- struct [Connection](#)

**Functions**

- [ClientShared](#) \* [InitializeShared](#) ([map](#) \*[users\\_map](#), [size\\_t](#) [send\\_buffer\\_size](#), [size\\_t](#) [receive\\_buffer\\_size](#), [char](#) \*[cipher](#), [char](#) [start](#), [char](#) [end](#))
- [void](#) \* [StartUpdateThread](#) ([void](#) \*[parameter](#))
- [void](#) \* [StartConnectionThread](#) ([void](#) \*[p\\_connection](#))
- [int](#) [MessageOrClose](#) ([char](#) \*[send\\_buffer](#), [char](#) \*[receive\\_buffer](#), [Connection](#) \*[connection](#))
- [void](#) [MessageAndClose](#) ([char](#) \*[send\\_buffer](#), [Connection](#) \*[connection](#))
- [void](#) [\\_help](#) ([Connection](#) \*[connection](#), [char](#) \*[response](#))
- [int](#) [\\_register](#) ([Connection](#) \*[connection](#), [char](#) \*[response](#))
- [int](#) [\\_myinfo](#) ([Connection](#) \*[connection](#), [char](#) \*[response](#))
- [void](#) [\\_who](#) ([char](#) \*[response](#))
- [void](#) [\\_rand\\_gpa](#) ([Connection](#) \*[connection](#), [char](#) \*[response](#))
- [void](#) [\\_rand\\_age](#) ([Connection](#) \*[connection](#), [char](#) \*[response](#))
- [void](#) [\\_advertisement](#) ([Connection](#) \*[connection](#), [char](#) \*[response](#))  
*responds with a random ascii art*
- [int](#) [\\_password](#) ([Connection](#) \*[connection](#))  
*Sends password changing request to user.*
- [int](#) [\\_authenticate](#) ([Connection](#) \*[connection](#))  
*Authenticates a login request.*

### 6.3.1 Detailed Description

### 6.3.2 Function Documentation

#### 6.3.2.1 InitializeShared()

```
ClientShared * InitializeShared (
    map * users_map,
    size_t send_buffer_size,
    size_t receive_buffer_size,
    char * cipher,
    char start,
    char end )
```

Initializes the structure that shares data between connections and the server.

##### Parameters

<i>users_map</i>	The map of <a href="#">User</a> structs.
------------------	--

##### Returns

A pointer to the same [ClientShared](#) object seen by the connection threads.

Definition at line 20 of file [Connection.c](#).

Here is the caller graph for this function:



#### 6.3.2.2 StartUpdateThread()

```
void * StartUpdateThread (
    void * parameter )
```

Starts an update thread. This thread is responsible for checking `shared.dirty`. If it is, it writes the user's data to a file and sets `dirty` to 0.

##### Parameters

<i>parameter</i>	None.
------------------	-------

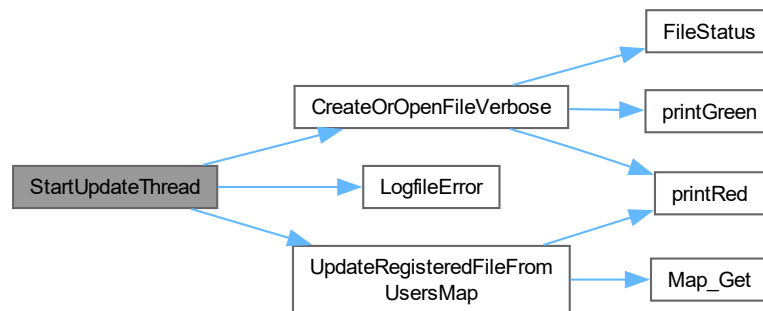


**Returns**

NULL

Definition at line 34 of file [Connection.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**6.3.2.3 StartConnectionThread()**

```
void * StartConnectionThread (
    void * connection )
```

Starts a connection thread

**Parameters**

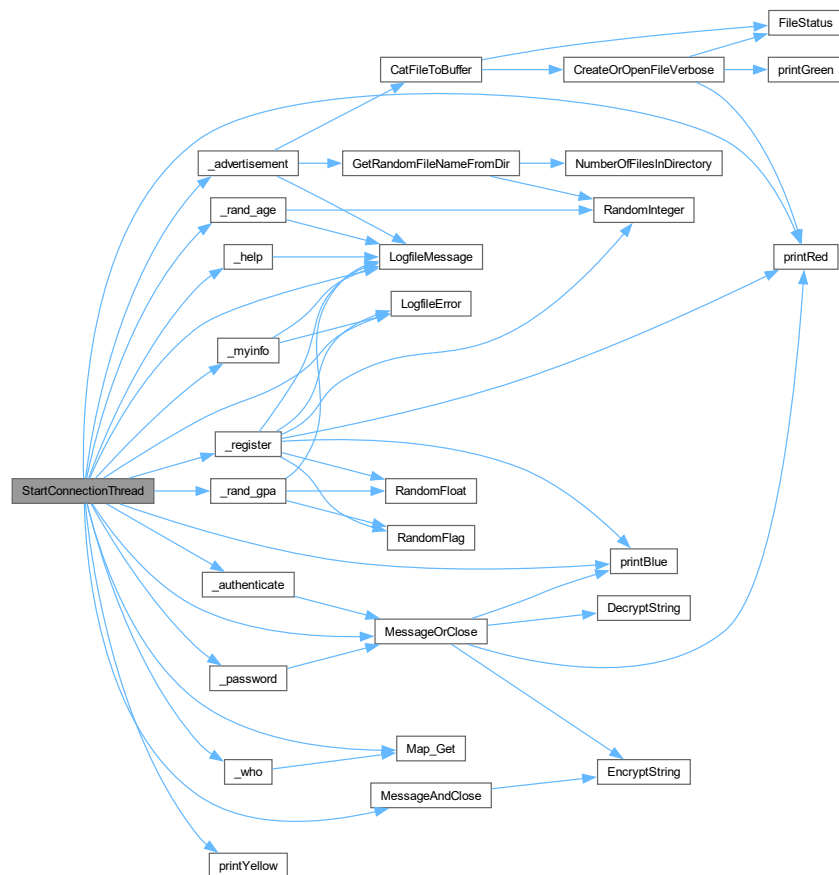
<i>connection</i>	A pointer to a <a href="#">Connection</a> structure from the server's connections array.
-------------------	--

**Returns**

NULL

Definition at line 56 of file [Connection.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 6.3.2.4 MessageOrClose()

```

int MessageOrClose (
    char * send_buffer,
    char * receive_buffer,
    Connection * connection )
  
```

Sends send\_buffer to the socket referenced by connection, then memsets send\_buffer to 0. Memsets receive\_buffer to 0, then receives a message from the client. If this length is 0, assumes the connection was closed and sets connection->active to 0.

**Warning**

send\_buffer and receive\_buffer must be the size specified in shared.

**Parameters**

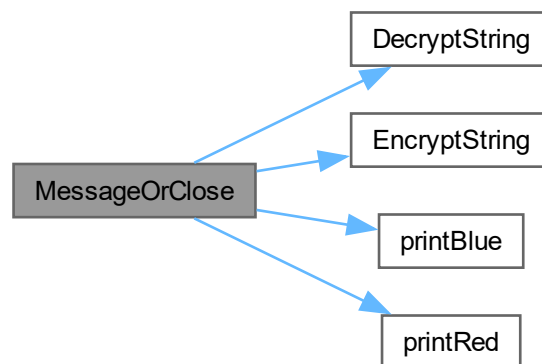
<i>send_buffer</i>	A message to send to the client.
<i>receive_buffer</i>	The message received by the client.
<i>connection</i>	The socket's <a href="#">Connection</a>

**Returns**

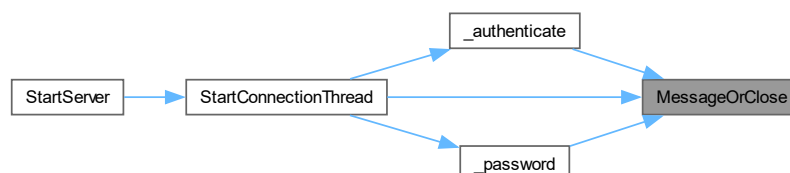
The number of bytes read into receive\_buffer, or 0 if the connection closed.

Definition at line 206 of file [Connection.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.3.2.5 MessageAndClose()

```
void MessageAndClose (
    char * send_buffer,
    Connection * connection )
```

Sends send\_buffer to the socket referenced by connection, then sets connection.active to 0.

#### Parameters

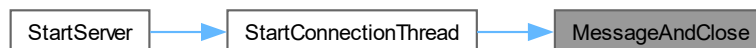
<i>send_buffer</i>	The send buffer. Should be shared.send_length in size.
<i>connection</i>	The socket's <a href="#">Connection</a> .

Definition at line [237](#) of file [Connection.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.3.2.6 \_help()

```
void _help (
    Connection * connection,
    char * response )
```

Returns the functions available to the user

#### Parameters

<i>connection</i>	connection the user is on
<i>response</i>	fills the response buffer with what to send to the client

Definition at line 247 of file [Connection.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 6.3.2.7 \_register()

```
int _register (
    Connection * connection,
    char * response )
```

Registers the user from connection

##### Parameters

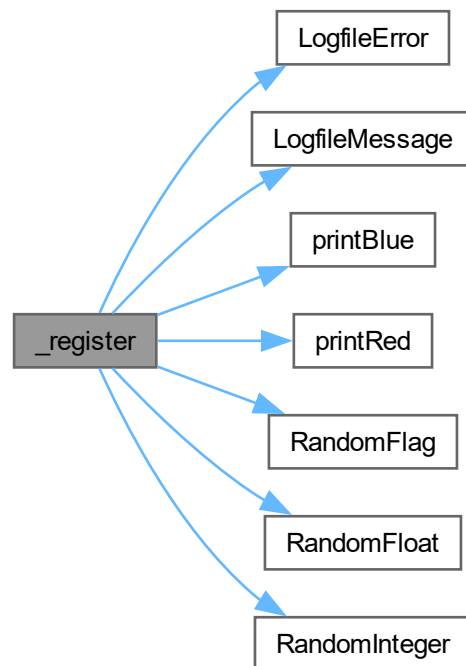
<i>connection</i>	connection the users is on
<i>response</i>	fills the response buffer with what to send to the client

**Returns**

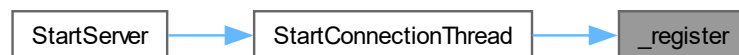
int 1 if successful, 0 if not

Definition at line 271 of file [Connection.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**6.3.2.8 \_myinfo()**

```
int _myinfo (  
    Connection * connection,  
    char * response )
```

Returns the info of the user to the client

## Parameters

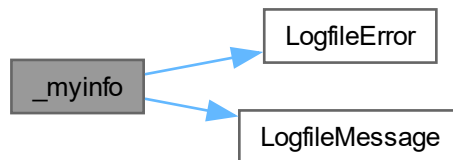
<i>connection</i>	connection the user is on
<i>response</i>	fills the response buffer with what to send to the client

## Returns

int 1 if successful, 0 if not

Definition at line 308 of file [Connection.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.3.2.9 \_who()

```
void _who (
    char * response )
```

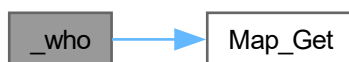
Sets response buffer to be a list a userIDs that are connected.

## Parameters

<i>response</i>	fills the response buffer with what to send to the client
-----------------	---

Definition at line 329 of file [Connection.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 6.3.2.10 `_rand_gpa()`

```
void _rand_gpa (  
    Connection * connection,  
    char * response )
```

Randomly changes the gpa of the user

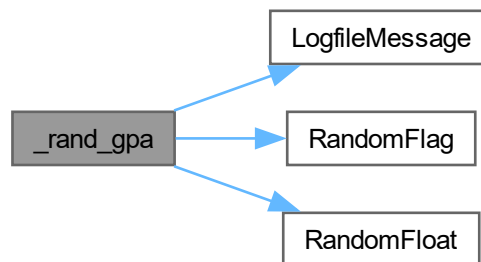
##### Parameters

<i>connection</i>	connection the user is on
<i>response</i>	fills the response buffer with what to send to the client

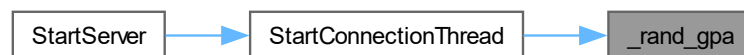
Definition at line [344](#) of file [Connection.c](#).



Here is the call graph for this function:



Here is the caller graph for this function:



#### 6.3.2.11 `_rand_age()`

```
void _rand_age (
    Connection * connection,
    char * response )
```

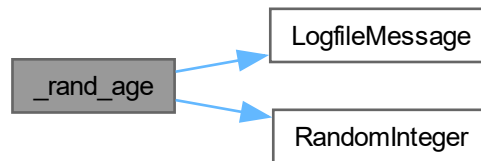
Randomly changes the age of the user

##### Parameters

<i>connection</i>	connection the user is on
<i>response</i>	fills the response buffer with what to send to the client

Definition at line 360 of file [Connection.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.3.2.12 \_advertisement()

```

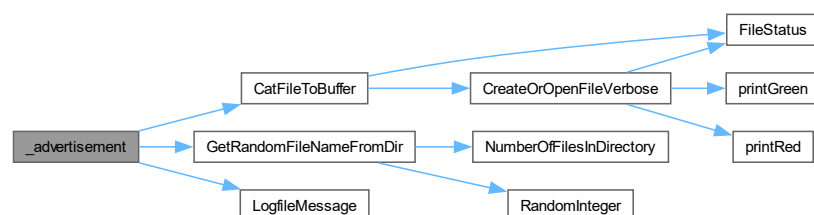
void _advertisement (
    Connection * connection,
    char * response )
  
```

#### Parameters

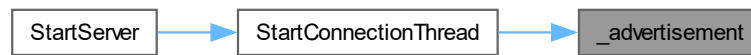
<i>connection</i>	connection the user is on
<i>response</i>	fills the response buffer with what to send to the client

Definition at line 372 of file [Connection.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.3.2.13 `_password()`

```
int _password (
    Connection * connection )
```

#### Parameters

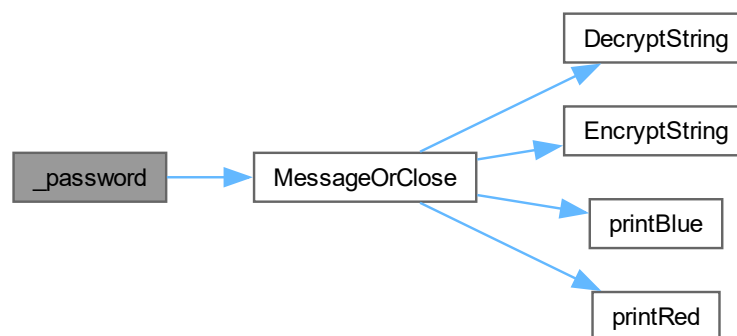
<code>connection</code>	connection the user is on
-------------------------	---------------------------

#### Returns

0 if valid password was entered, 1 if invalid

Definition at line 390 of file [Connection.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 6.3.2.14 \_authenticate()

```
int _authenticate (
    Connection * connection )
```

##### Parameters

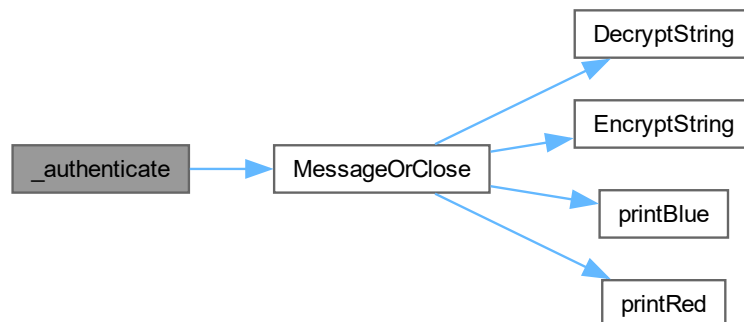
<i>connection</i>	connection the user is on
-------------------	---------------------------

##### Returns

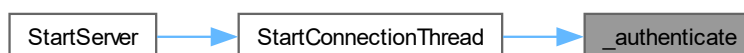
0 if valid password was entered, 1 if invalid

Definition at line 431 of file [Connection.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



## 6.4 Data

This module describes structures used in this program.

### Data Structures

- struct [User](#)

### Macros

- `#define` [RECORD\\_COUNT](#) 17
- `#define` [ID\\_MAX\\_LENGTH](#) 9
- `#define` [NAME\\_MAX\\_LENGTH](#) 21
- `#define` [IP\\_LENGTH](#) 16
- `#define` [PASSWORD\\_LENGTH](#) 20

### Variables

- `char *` [accepted\\_userIDs](#) []
- `char *` [userFullNames](#) []
- `char *` [accepted\\_userIDs](#) []
- `char *` [userFullNames](#) []

### 6.4.1 Detailed Description

### 6.4.2 Macro Definition Documentation

#### 6.4.2.1 RECORD\_COUNT

```
#define RECORD_COUNT 17
```

The total count of records.

Definition at line 12 of file [Data.h](#).

#### 6.4.2.2 ID\_MAX\_LENGTH

```
#define ID_MAX_LENGTH 9
```

The amount of memory (bytes) required to be allocated for the ID field. Equal to the longest name in `Data_IDs`, "mes08346", plus the null terminator

Definition at line 17 of file [Data.h](#).

#### 6.4.2.3 NAME\_MAX\_LENGTH

```
#define NAME_MAX_LENGTH 21
```

The amount of memory (bytes) required to be allocated for the Name field. Equal to the longest name in Data\_Names, "Assefa Ayalew Yoseph", plus the null terminator

Definition at line 22 of file [Data.h](#).

#### 6.4.2.4 IP\_LENGTH

```
#define IP_LENGTH 16
```

The amount of memory (bytes) required to be allocated for the IP field. Large enough to store '111.111.111.111' plus the null terminator.

Definition at line 28 of file [Data.h](#).

#### 6.4.2.5 PASSWORD\_LENGTH

```
#define PASSWORD_LENGTH 20
```

Max length of password

Definition at line 33 of file [Data.h](#).

### 6.4.3 Variable Documentation

#### 6.4.3.1 accepted\_userIDs [1/2]

```
char* accepted_userIDs[]
```

**Initial value:**

```
= {  
    "chen",  
    "bea1389",  
    "bol4559",  
    "cal6258",  
    "kre5277",  
    "lon1150",  
    "mas9309",  
    "mes08346",  
    "mil7233",  
    "nef9476",  
    "nov7488",  
    "pan9725",  
    "rac3146",  
    "rub4133",  
    "shr5683",  
    "vay3083",  
    "yos2327"}  
}
```

An array of the accepted userIDs.

Definition at line 7 of file [Data.c](#).

#### 6.4.3.2 userFullNames [1/2]

```
char* userFullNames[ ]
```

##### Initial value:

```
= {  
    "Weifeng Chen",  
    "Christian Beatty",  
    "Emily Bolles",  
    "Cameron Calhoun",  
    "Ty Kress",  
    "Cody Long",  
    "Caleb Massey",  
    "Christian Messmer",  
    "Karl Miller",  
    "Jeremiah Neff",  
    "Kaitlyn Novacek",  
    "Joshua Panaro",  
    "Caleb Rachocki",  
    "Caleb Ruby",  
    "Paul Shriner",  
    "Alan Vayansky",  
    "Assefa Ayalew Yoseph"}  
}
```

An array of the full names, where the index of the name corresponds to the id in `accepted_userIDs`.

Definition at line 26 of file [Data.c](#).

#### 6.4.3.3 accepted\_userIDs [2/2]

```
char* accepted_userIDs[ ] [extern]
```

An array of the accepted userIDs.

Definition at line 7 of file [Data.c](#).

#### 6.4.3.4 userFullNames [2/2]

```
char* userFullNames[ ] [extern]
```

An array of the full names, where the index of the name corresponds to the id in `accepted_userIDs`.

Definition at line 26 of file [Data.c](#).

## 6.5 Files

This module contains functions that interact with files.

### Macros

- `#define LOCKFILE "/tmp/lab6.lock"`
- `#define REGISTERED_FILE "registered.txt"`
- `#define SERVER_SETTINGS_FILE "server-settings.txt"`
- `#define KEY_FILE "sub.key"`  
*contains the key for the cipher*
- `#define ADS_DIR "ads"`

## Functions

- short [FileStatus](#) (char \*filename)
- FILE \* [CreateOrOpenFileVerbose](#) (char \*filename, char \*defaultContents)
- int [ReadKeyIntoSettingsMap](#) (FILE \*key\_file, map \*settings\_map)  
*Reads the cipher file into the settings map.*
- int [ReadRegisteredFileIntoUsersMap](#) (FILE \*reg\_file, map \*users\_map)
- void [UpdateRegisteredFileFromUsersMap](#) (FILE \*reg\_file, map \*users\_map)  
*Updates the registered file with of all users from user map that are marked as registered.*
- int [NumberOfFilesInDirectory](#) (char \*dir\_name)  
*Finds the number of files/directories in a given directory.*
- void [GetRandomFileNameFromDir](#) (char \*dir\_name, char \*file\_name)  
*Get the Random File Name From Dir object.*
- int [ReadSettingsFileIntoSettingsMap](#) (FILE \*settings\_file, map \*settings\_map)
- void [CatFileToBuffer](#) (char \*file\_name, char \*buffer, size\_t buffer\_size)  
*Concatinates the contents of file\_name into the buffer string.*

### 6.5.1 Detailed Description

### 6.5.2 Macro Definition Documentation

#### 6.5.2.1 LOCKFILE

```
#define LOCKFILE "/tmp/lab6.lock"
```

The presence of a lockfile indicates that a server process is already running. The lockfile contains the process ID of the running process.

Definition at line 16 of file [File.h](#).

#### 6.5.2.2 REGISTERED\_FILE

```
#define REGISTERED_FILE "registered.txt"
```

This file contains a list of registered users and their data, with fields tab-delimited.

#### Note

- (1) The userID of the user.
- (2) The age of the user.
- (3) The GPA of the user.
- (4) The IP address of the user.
- (5) The last connection time of a user.

Definition at line 26 of file [File.h](#).



### 6.5.2.3 SERVER\_SETTINGS\_FILE

```
#define SERVER_SETTINGS_FILE "server-settings.txt"
```

Contains settings for the server. Each setting row contains a key, 0 or more space, an '=' symbol, and a value. Valid keys:

#### Note

- port; the port the server will listen on.
- send\_buffer\_size; the size of the send buffer
- receive\_buffer\_size; the size of the receive buffer
- backlog; the quantity of allowed backlogged unprocessed connections.
- log\_file; the name of the logging file

Definition at line 39 of file [File.h](#).

### 6.5.2.4 KEY\_FILE

```
#define KEY_FILE "sub.key"
```

#### Note

starts with the first character that it starts substituting followed by  
followed by the last chracter in substitution range followed by a  
and then the cipher in ascii order fro mthe start character to the last character of the characters to use instead  
at that spot

Definition at line 45 of file [File.h](#).

### 6.5.2.5 ADS\_DIR

```
#define ADS_DIR "ads"
```

Contains files of ascii art to sent to clients.

#### Note

should be the relative directory to the file the ads are in

Definition at line 52 of file [File.h](#).

### 6.5.3 Function Documentation

#### 6.5.3.1 FileStatus()

```
short FileStatus (
    char * filename )
```

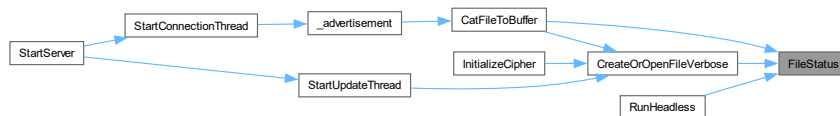
Determines if a file indicated by filename exists and is accesible by the user.

##### Returns

0 if the file does not exist. 1 if the file exists and the user has access. 2 if the file exists and the user does not have read and write permissions.

Definition at line 16 of file [File.c](#).

Here is the caller graph for this function:



#### 6.5.3.2 CreateOrOpenFileVerbose()

```
FILE * CreateOrOpenFileVerbose (
    char * filename,
    char * defaultContents )
```

Will call fopen() on a file and put default data inside, or nothing if defaultContents is NULL. Will print the results of its attempt.

##### Warning

Does not close the file; returns the open file.

##### Note

Prints successes and errors.

##### Parameters

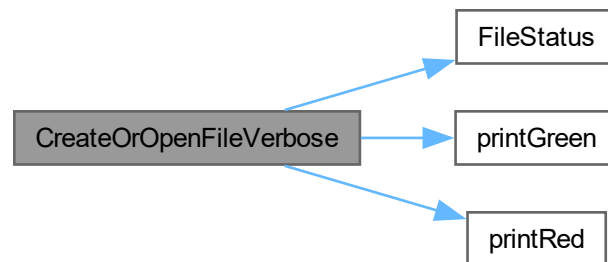
<i>filename</i>	The file name to create or open.
<i>defaultContents</i>	The contents to put in the file, if creating a default file, or NULL if no contents should be added.

**Returns**

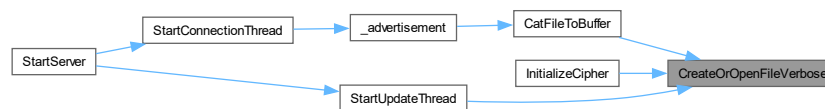
The opened file, or NULL on failure.

Definition at line 28 of file [File.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**6.5.3.3 ReadKeyIntoSettingsMap()**

```

int ReadKeyIntoSettingsMap (
    FILE * key_file,
    map * settings_map )
  
```

**Parameters**

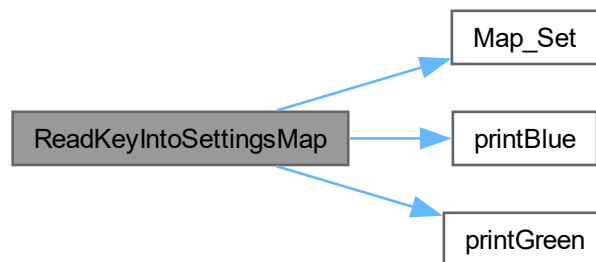
<i>key_file</i>	file to read
<i>settings_map</i>	map to put the settings into

**Returns**

1 if successful, 0 otherwise

Definition at line 76 of file [File.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 6.5.3.4 ReadRegisteredFileIntoUsersMap()

```

int ReadRegisteredFileIntoUsersMap (
    FILE * reg_file,
    map * users_map )
  
```

Reads the registered file into the user's map, by checking the IDs in the first field and setting the data at that location.

##### Note

Prints warnings and errors.

##### Parameters

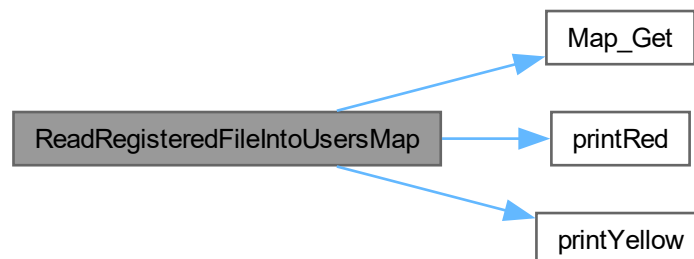
<i>reg_file</i>	The registered users file, open for reading.
<i>users_map</i>	The user's map to read into.

##### Returns

0 if success, error code if there was an error.

Definition at line 105 of file [File.c](#).

Here is the call graph for this function:



#### 6.5.3.5 UpdateRegisteredFileFromUsersMap()

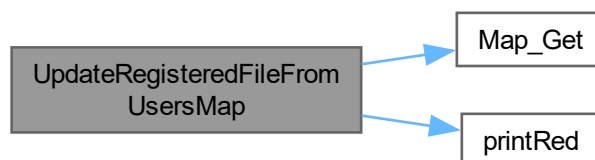
```
void UpdateRegisteredFileFromUsersMap (
    FILE * reg_file,
    map * users_map )
```

##### Parameters

<i>reg_file</i>	file to update to
<i>users_map</i>	the map of users to use to update

Definition at line 139 of file [File.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 6.5.3.6 NumberOfFilesInDirectory()

```
int NumberOfFilesInDirectory (
    char * dir_name )
```

##### Parameters

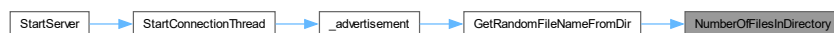
<i>dir_name</i>	directory to count files from
-----------------	-------------------------------

##### Returns

int number of files in the directory

Definition at line 155 of file [File.c](#).

Here is the caller graph for this function:



#### 6.5.3.7 GetRandomFileNameFromDir()

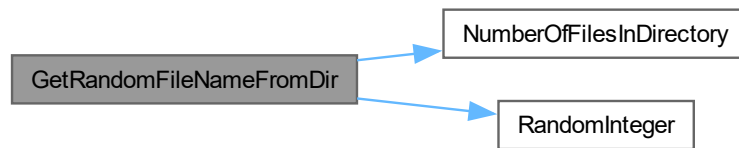
```
void GetRandomFileNameFromDir (
    char * dir_name,
    char * file_name )
```

##### Parameters

<i>dir_name</i>	name of the director to get a file name of
<i>file_name</i>	sets the name of the file into file_name

Definition at line 174 of file [File.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.5.3.8 ReadSettingsFileIntoSettingsMap()

```
int ReadSettingsFileIntoSettingsMap (
    FILE * settings_file,
    map * settings_map )
```

Reads the settings file into the settings map, by checking each line for a key value pair separated by a "=". It mallocs each key and value string it finds.

#### Note

Prints warnings and errors.

#### Parameters

<i>settings_file</i>	The settings file, open for reading.
<i>users_map</i>	The settings_map to read into.

#### Returns

0 if success, an error code if there was an error.

Definition at line 193 of file [File.c](#).

Here is the call graph for this function:



### 6.5.3.9 CatFileToBuffer()

```

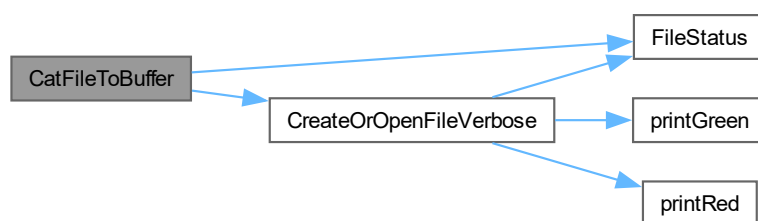
void CatFileToBuffer (
    char * file_name,
    char * buffer,
    size_t buffer_size )
  
```

#### Parameters

<i>file_name</i>	file to concatenate
<i>buffer</i>	string to copy it to
<i>buffer_size</i>	max size of buffer

Definition at line 218 of file [File.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:





## 6.6 Logfile

Logs server activity to a file, so we can see who has been using the server.

### Functions

- void [SetLogfileName](#) (char \*logfile\_name\_param)
- void [LogfileError](#) (const char \*format,...)
- void [LogfileMessage](#) (const char \*format,...)

### 6.6.1 Detailed Description

### 6.6.2 Function Documentation

#### 6.6.2.1 SetLogfileName()

```
void SetLogfileName (  
    char * logfile_name )
```

Sets the name of the logfile, as retrieved from the server-settings.txt file.

This file will be appended to on each log.

Definition at line 47 of file [Logfile.c](#).

#### 6.6.2.2 LogfileError()

```
void LogfileError (  
    const char * format,  
    ... )
```

Logs an error to the logfile. Will be in the format:

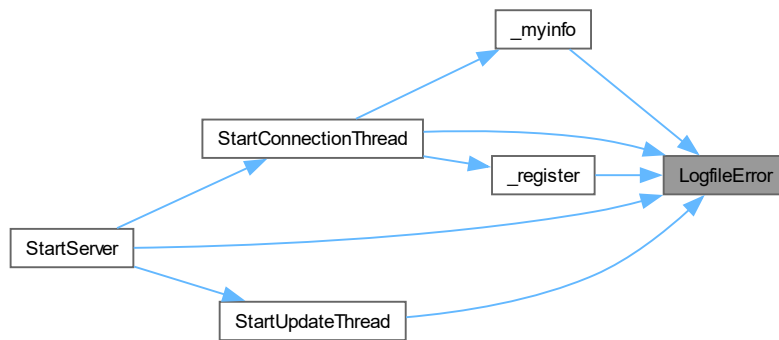
MM-DD-HH-MM-SS ERR <message>

#### Parameters

<i>format</i>	A format string, as with printf
...	Additional args

Definition at line 55 of file [Logfile.c](#).

Here is the caller graph for this function:



### 6.6.2.3 LogfileMessage()

```
void LogfileMessage (
    const char * format,
    ... )
```

Logs a message to the logfile. Will be in the format:

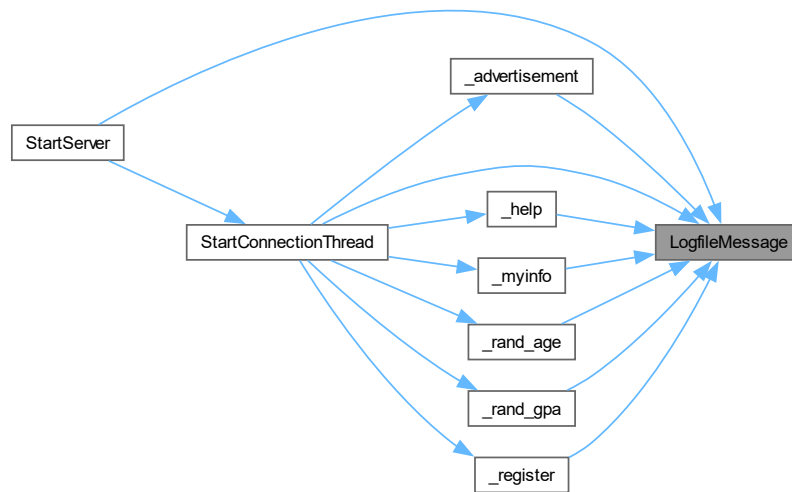
MM-DD-HH-MM-SS MSG <message>

#### Parameters

<i>format</i>	A format string, as with printf
...	Additional args

Definition at line 72 of file [Logfile.c](#).

Here is the caller graph for this function:



## 6.7 Map

Functions that implement a hash map data structure.

### Data Structures

- `struct _map_bucket`  
*map\_bucket is an endpoint in the map. It is also a node in a linked list; if there were collisions, then the buckets are appended to the linked list at that location, then traversed until the matching key is found.*
- `struct map`  
*A map. Stores key-value pairs for near constant lookup and insertion time.*
- `struct map_result`  
*The result of a map retrieval.*

### Functions

- `map * NewMap (int capacity)`
- `void Map_Set (map *a_map, char *key, void *value)`  
*Sets a value in the map.*
- `map_result Map_Get (map *a_map, char *key)`  
*Gets a value from the map. It will return a map\_get\_result describing whether it was succesful, and possibly containing the data sought, or NULL if it was unsuccessful.*
- `map_result Map_Delete (map *a_map, char *key, short free_it)`  
*Deletes a key from the map. Returns a map\_get\_result describing whether the delete was succesful and containing the removed data, if extant.*

## 6.7.1 Detailed Description

Karl's take on a simple hash map structure, which maps strings to void pointers. You can use casting to convert the void pointers into most of whatever else is needed.

Example usage, casting an int into the data part of the map.

```
int myfunc() {
    map *mymap = NewMap(100);
    Map_Set(mymap, "age", (void*)55);
    map_result result = Map_Get(mymap, "age");
    int age;
    if(result.found) {
        age = (int) map_result.data;
    }
}
```

Note, with this simple implementation, the map cannot change its capacity. A change to its capacity would change the hashing.

Ultimately there are really only three things you need to do with the map.

Initialize it, with some capacity larger than you will use. EG `map * mymap = NewMap(100)`. The bigger it is, the fewer collisions (which are pretty rare anyway).

Set some values in it. Eg `Map_Set(mymap, "key", &value)`;

You can cast numbers to void pointers to put them in the map, or you can use the pointers as references to, for example, strings malloced somewhere.

Get some values from it. Eg `void* myval = Map_Get(mymap, "key")`;

Delete some values from it. For example `Map_Delete(mymap, "key", 0)`;

Note that the last parameter, 'free it', tells the map whether it should call 'free' on the underlying data in memory. If this is 1, and the underlying data is not a reference to a malloced part of the heap, errors will result.

## 6.7.2 Function Documentation

### 6.7.2.1 NewMap()

```
map * NewMap (
    int capacity )
```

Creates a new map. The map capacity will be a power of 2 that is large enough to contain the estimated size.

#### Parameters

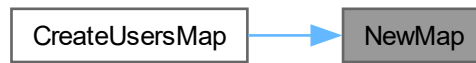
<i>capacity</i>	The estimated required capacity of the map.
-----------------	---

#### Returns

A pointer to the heap allocated map.

Definition at line 49 of file [map.c](#).

Here is the caller graph for this function:



### 6.7.2.2 Map\_Set()

```

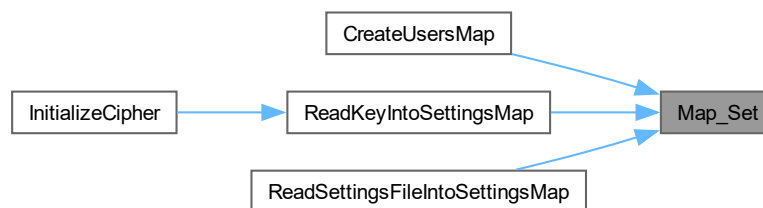
void Map_Set (
    map * a_map,
    char * key,
    void * value )
  
```

#### Parameters

<i>map</i>	The map to set a key in.
<i>key</i>	The key to use.
<i>keylen</i>	The length of the key.
<i>value</i>	The pointer to the data stored at that location.

Definition at line 89 of file [map.c](#).

Here is the caller graph for this function:



### 6.7.2.3 Map\_Get()

```

map_result Map_Get (
    map * a_map,
    char * key )
  
```

**Parameters**

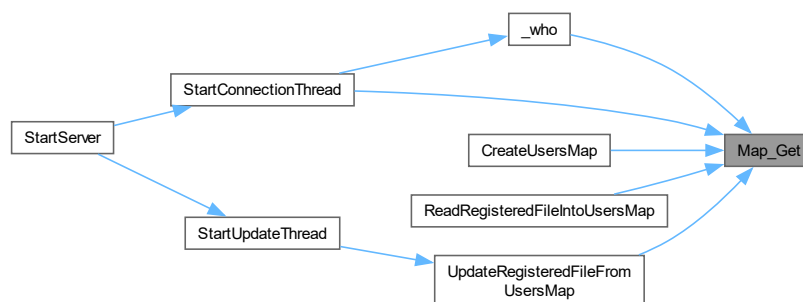
<i>map</i>	The map to retrieve from.
<i>key</i>	The key of the item.

**Returns**

A `map_get_result` containing the sought data.

Definition at line 119 of file [map.c](#).

Here is the caller graph for this function:

**6.7.2.4 Map\_Delete()**

```

map_result Map_Delete (
    map * a_map,
    char * key,
    short free_it )

```

**Parameters**

<i>map</i>	The map to delete the key from.
<i>key</i>	The key to delete.
<i>free_it</i>	Whether to call <code>free()</code> on the underlying data.

**Returns**

A `map_get_result` with the data that was removed.

Definition at line 154 of file [map.c](#).

**6.8 Server**

Functions for running the server.

## Data Structures

- struct [ServerProperties](#)

## Functions

- int [StartServer](#) (map \*[users\\_map](#))
- [Connection](#) \* [NextAvailableConnection](#) ()
- int [CloseServer](#) ()
- int [InitializeServer](#) ()
- int [InitializeCipher](#) ()

*Initializes the cipher from the key file.*

### 6.8.1 Detailed Description

### 6.8.2 Function Documentation

#### 6.8.2.1 StartServer()

```
int StartServer (  
    map * users\_map )
```

Starts the server.

#### Note

This is a blocking call that will start a loop until SIGINT is received.

#### Parameters

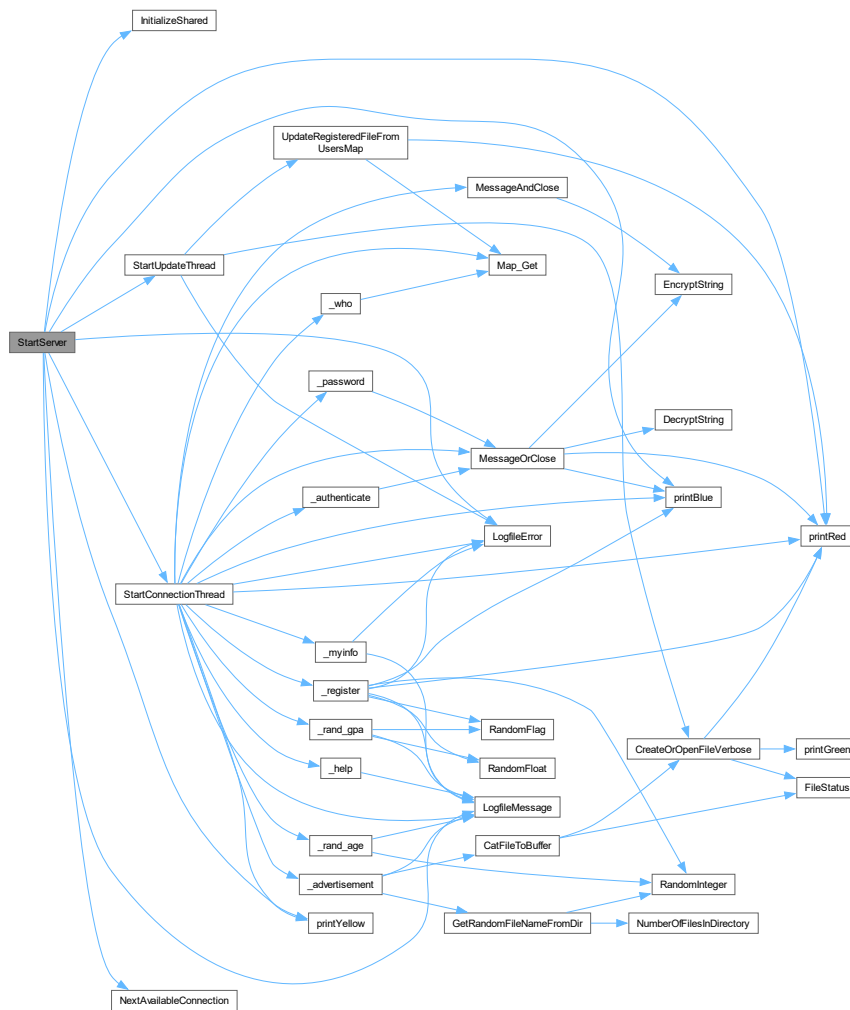
<a href="#">users_map</a>	The user's map.
---------------------------	-----------------

#### Returns

1 if the server ran and shutdown gracefully, 0 if there was an error during setup.

Definition at line [135](#) of file [Server.c](#).

Here is the call graph for this function:



### 6.8.2.2 NextAvailableConnection()

```
Connection * NextAvailableConnection ( )
```

Iterates through the Connections array until it finds one whose 'active' field is false and returns it. If it iterates through the array and fails to find a connection, it returns NULL.

#### Returns

A [Connection](#) struct or null.

Definition at line 198 of file [Server.c](#).



Here is the caller graph for this function:



### 6.8.2.3 CloseServer()

```
int CloseServer ( )
```

Unbinds the socket interface and closes the server.

#### Returns

0 on success or a number on error

Definition at line 210 of file [Server.c](#).

### 6.8.2.4 InitializeServer()

```
int InitializeServer ( )
```

Initializes the server properties structure and the structures for holding [Connection](#) objects.

#### Note

Prints initialization status.

#### Returns

1 if it was able to initialize, otherwise 0.

### 6.8.2.5 InitializeCipher()

```
int InitializeCipher ( )
```

#### Returns

int 1 if it was able to initialize, otherwise 0

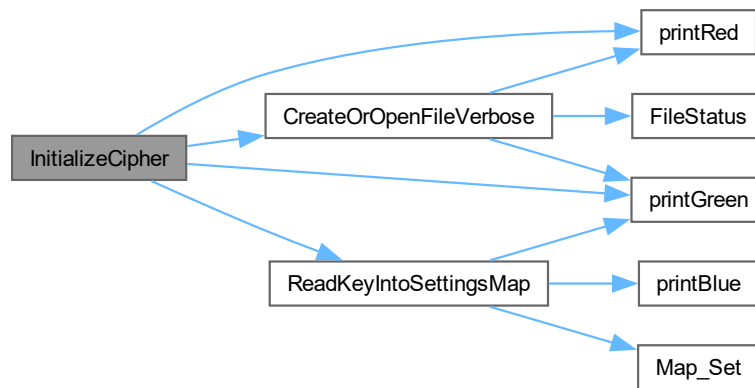
Initializes the Cipher.

**Returns**

1 on success, 0 if it can't set up key

Definition at line 96 of file [Process.c](#).

Here is the call graph for this function:



## 6.9 Util

Utility functions used by various modules but not dependent on any other modules.

**Macros**

- `#define COLOR_RED "\e[38;2;255;75;75m"`
- `#define COLOR_GREEN "\e[38;2;0;240;0m"`
- `#define COLOR_YELLOW "\e[38;2;255;255;0m"`
- `#define COLOR_BLUE "\e[38;2;0;240;240m"`
- `#define COLOR_RESET "\e[0m"`

**Functions**

- void `printRed` (const char \*format,...)
- void `printGreen` (const char \*format,...)
- void `printYellow` (const char \*format,...)
- void `printBlue` (const char \*format,...)
- int `RandomInteger` (int min, int max)
- float `RandomFloat` (float min, float max)
- short `RandomFlag` (float percentage\_chance)

## 6.9.1 Detailed Description

## 6.9.2 Macro Definition Documentation

### 6.9.2.1 COLOR\_RED

```
#define COLOR_RED "\e[38;2;255;75;75m"
```

A virtual terminal escape sequence to print foreground red.

Definition at line 12 of file [Util.h](#).

### 6.9.2.2 COLOR\_GREEN

```
#define COLOR_GREEN "\e[38;2;0;240;0m"
```

A VTE for green.

Definition at line 14 of file [Util.h](#).

### 6.9.2.3 COLOR\_YELLOW

```
#define COLOR_YELLOW "\e[38;2;255;255;0m"
```

A VTE for yellow.

Definition at line 16 of file [Util.h](#).

### 6.9.2.4 COLOR\_BLUE

```
#define COLOR_BLUE "\e[38;2;0;240;240m"
```

A VTE for blue.

Definition at line 18 of file [Util.h](#).

### 6.9.2.5 COLOR\_RESET

```
#define COLOR_RESET "\e[0m"
```

A VTE to reset the printing color.

Definition at line 20 of file [Util.h](#).

## 6.9.3 Function Documentation

### 6.9.3.1 printRed()

```
void printRed (  
    const char * format,  
    ... )
```

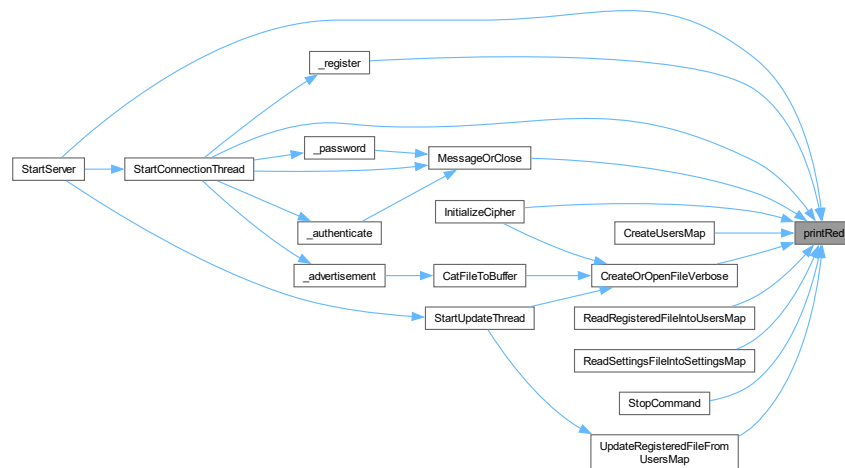
Prints to the console in red.

## Parameters

<i>format</i>	A format, as printf.
...	args, as printf.

Definition at line 11 of file [Util.c](#).

Here is the caller graph for this function:



### 6.9.3.2 printGreen()

```
void printGreen (
    const char * format,
    ... )
```

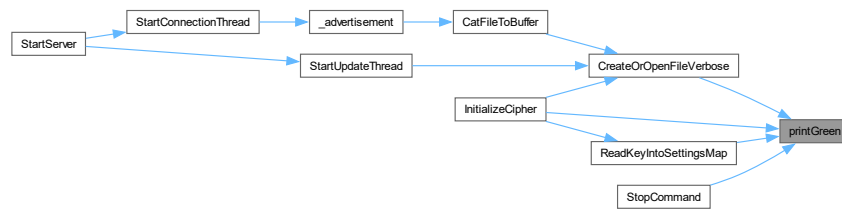
Prints to the console in green.

## Parameters

<i>format</i>	A format, as printf.
...	args, as printf.

Definition at line 20 of file [Util.c](#).

Here is the caller graph for this function:



### 6.9.3.3 printYellow()

```
void printYellow (
    const char * format,
    ... )
```

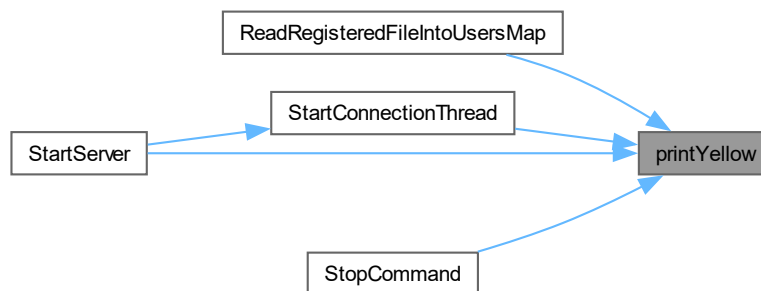
Prints to the console in yellow.

#### Parameters

<i>format</i>	A format, as printf.
...	args, as printf.

Definition at line 29 of file [Util.c](#).

Here is the caller graph for this function:



### 6.9.3.4 printBlue()

```
void printBlue (
    const char * format,
    ... )
```

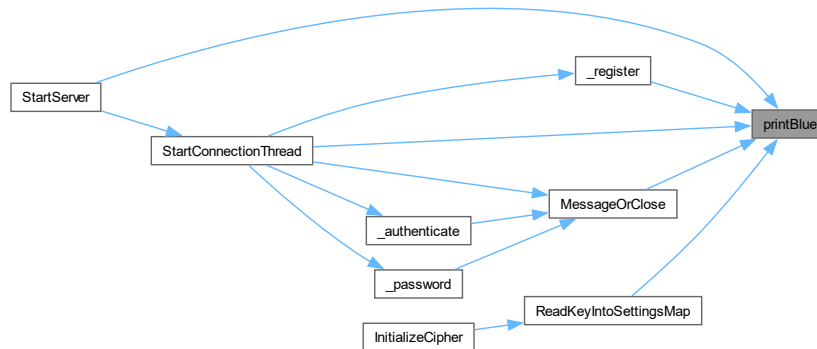
Prints to the console in blue.

**Parameters**

<i>format</i>	A format, as printf.
...	args, as printf.

Definition at line 38 of file [Util.c](#).

Here is the caller graph for this function:

**6.9.3.5 RandomInteger()**

```
int RandomInteger (
    int min,
    int max )
```

Returns an integer between min and max.

**Parameters**

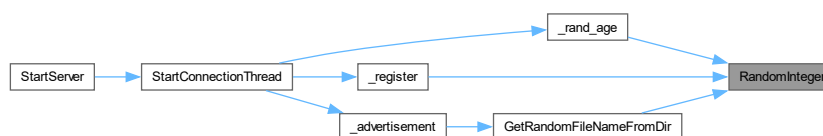
<i>min</i>	The minimum, inclusive.
<i>max</i>	The maximum, inclusive.

**Returns**

A random integer between min and max.

Definition at line 47 of file [Util.c](#).

Here is the caller graph for this function:



### 6.9.3.6 RandomFloat()

```
float RandomFloat (
    float min,
    float max )
```

Returns a float between min and max.

#### Parameters

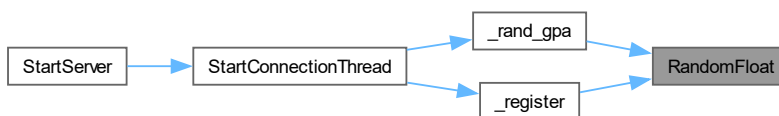
<i>min</i>	The minimum, inclusive.
<i>max</i>	The maximum, inclusive.

#### Returns

A random integer between min and max.

Definition at line 53 of file [Util.c](#).

Here is the caller graph for this function:



### 6.9.3.7 RandomFlag()

```
short RandomFlag (
    float percentage_chance )
```

Returns 1, `percentage_chance` of the time.

#### Parameters

<i>percentage_chance</i>	The chance to return 1.
--------------------------	-------------------------

#### Note

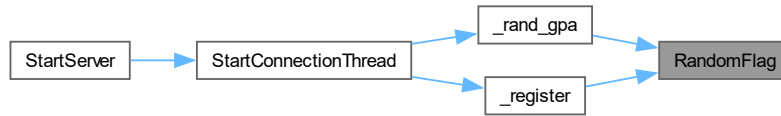
If `percentage_chance > 1`, this will always return true.

#### Returns

1 or 0

Definition at line 60 of file [Util.c](#).

Here is the caller graph for this function:



## 6.10 Process

### Functions

- int [InitializeCipher](#) ()  
*Initilizes the cipher from the key file.*
- void [RunHeadless](#) (char \*processName)
- void **StopCommand** ()  
*Stops the server that is running headlessly and prints the results of running the command.*

### Variables

- User \* [users\\_array](#)
- map \* [users\\_map](#)
- map \* [settings\\_map](#)
- char \* [default\\_settings](#)
- int [active\\_clients](#)

### 6.10.1 Detailed Description

### 6.10.2 Function Documentation

#### 6.10.2.1 InitializeCipher()

```
int InitializeCipher ( )
```

Initializes the Cipher.

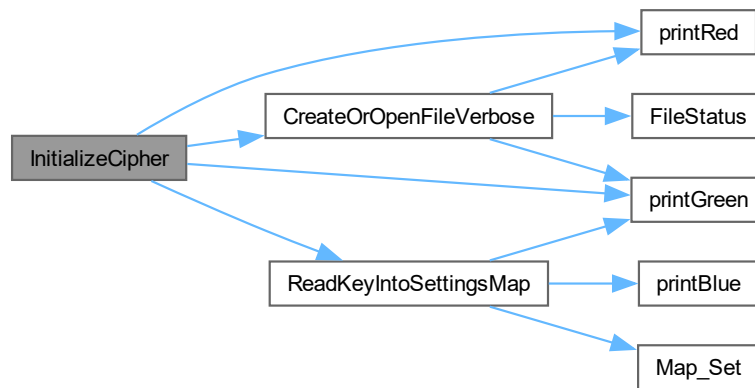


**Returns**

1 on success, 0 if it can't set up key

Definition at line 96 of file [Process.c](#).

Here is the call graph for this function:

**6.10.2.2 RunHeadless()**

```
void RunHeadless (
    char * processName )
```

Uses `nohup ./{processName} run` to run the process headlessly.

**Parameters**

<i>processName</i>	The name of the currently running process, by default, 'server'.
--------------------	--

Definition at line 217 of file [Process.c](#).

Here is the call graph for this function:



### 6.10.3 Variable Documentation

#### 6.10.3.1 users\_array

```
User* users_array
```

The array of users. This will be populated on initialize by functions in Build.

Definition at line 20 of file [Process.c](#).

#### 6.10.3.2 users\_map

```
map* users_map
```

The map of userIDs to users. Populated on Initialize by functions in Build.

Definition at line 22 of file [Process.c](#).

#### 6.10.3.3 settings\_map

```
map* settings_map
```

The map of settings stored in the server settings file.

Definition at line 24 of file [Process.c](#).

#### 6.10.3.4 default\_settings

```
char* default_settings
```

##### Initial value:

```
= "port          = 3000\n"
   "send_buffer_size  = 1024\n"
   "receive_buffer_size = 1024\n"
   "backlog           = 10\n"
   "max_connections   = 20\n"
   "log_file           = log.txt\n"
   "log_level          = 1\n"
   "log_to_console     = true"
```

The default contents of the settings file, if it doesn't exist.

Definition at line 26 of file [Process.c](#).

#### 6.10.3.5 active\_clients

```
int active_clients
```

The number of active clients.

Definition at line 40 of file [Process.c](#).

## Chapter 7

# Data Structure Documentation

### 7.1 `_map_bucket` Struct Reference

`map_bucket` is an endpoint in the map. It is also a node in a linked list; if there were collisions, then the buckets are appended to the linked list at that location, then traversed until the matching key is found.

#### 7.1.1 Detailed Description

Definition at line 81 of file [map.h](#).

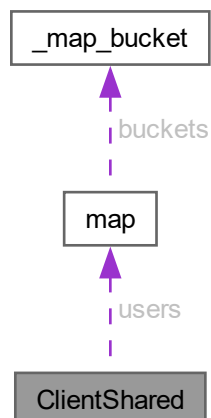
The documentation for this struct was generated from the following file:

- `src/server/map.h`

### 7.2 `ClientShared` Struct Reference

```
#include <Connection.h>
```

Collaboration diagram for `ClientShared`:



## Data Fields

- `map * users`
- `pthread_mutex_t mutex`
- `short dirty`
- `short shutting_down`
- `size_t send_buffer_size`
- `size_t receive_buffer_size`
- `char * cipher`
- `char start`
- `char end`

## 7.2.1 Detailed Description

Shared between the Connections and the Server.

Definition at line 17 of file [Connection.h](#).

## 7.2.2 Field Documentation

### 7.2.2.1 users

```
map* users
```

The user's map.

Definition at line 19 of file [Connection.h](#).

### 7.2.2.2 mutex

```
pthread_mutex_t mutex
```

A mutex to provide mutual-exclusion to connection threads operating on the user's map.

Definition at line 21 of file [Connection.h](#).

### 7.2.2.3 dirty

```
short dirty
```

Whether there were changes to the user's map that need to be saved in a file.

Definition at line 23 of file [Connection.h](#).

### 7.2.2.4 shutting\_down

```
short shutting_down
```

Whether the server is shutting down.

Definition at line 25 of file [Connection.h](#).

#### 7.2.2.5 send\_buffer\_size

```
size_t send_buffer_size
```

Passed along from server settings at the time shared is initialized

Definition at line 28 of file [Connection.h](#).

#### 7.2.2.6 receive\_buffer\_size

```
size_t receive_buffer_size
```

Passed along from server settings at the time shared is initialized

Definition at line 30 of file [Connection.h](#).

#### 7.2.2.7 cipher

```
char* cipher
```

A string containing the cipher to use when encoding and decoding messages

Definition at line 33 of file [Connection.h](#).

#### 7.2.2.8 start

```
char start
```

Starting character of cipher

Definition at line 35 of file [Connection.h](#).

#### 7.2.2.9 end

```
char end
```

ending character of cipher

Definition at line 37 of file [Connection.h](#).

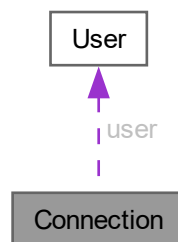
The documentation for this struct was generated from the following file:

- [src/server/Connection.h](#)

## 7.3 Connection Struct Reference

```
#include <Connection.h>
```

Collaboration diagram for Connection:



### Data Fields

- ConnectionState [status](#)
- int [socket](#)
- struct sockaddr\_in [address](#)
- socklen\_t [address\\_length](#)
- pthread\_t [thread\\_id](#)
- time\_t [time\\_connected](#)
- ClientState [state](#)
- User \* [user](#)

### 7.3.1 Detailed Description

Data for a single client socket connection to the server. Passed into the thread runner as the parameter.

Definition at line 54 of file [Connection.h](#).

### 7.3.2 Field Documentation

#### 7.3.2.1 status

```
ConnectionState status
```

Whether this connection is closed (0) or active (1) or closing (2). This is set by the SERVER just prior to starting the thread. The thread sets it back to 0 when it is completely done.

Definition at line 56 of file [Connection.h](#).

#### 7.3.2.2 socket

```
int socket
```

The underlying socket file descriptor.

Definition at line 58 of file [Connection.h](#).

#### 7.3.2.3 address

```
struct sockaddr_in address
```

The socket address of the connection.

Definition at line 60 of file [Connection.h](#).

#### 7.3.2.4 address\_length

```
socklen_t address_length
```

The actual size of the client address; send by accept.

Definition at line 62 of file [Connection.h](#).

#### 7.3.2.5 thread\_id

```
pthread_t thread_id
```

The pthread ID of this client thread.

Definition at line 64 of file [Connection.h](#).

#### 7.3.2.6 time\_connected

```
time_t time_connected
```

When the client connected.

Definition at line 66 of file [Connection.h](#).

#### 7.3.2.7 state

```
ClientState state
```

The client state.

Definition at line 68 of file [Connection.h](#).

### 7.3.2.8 user

```
User* user
```

The user associated with this client.

Definition at line 70 of file [Connection.h](#).

The documentation for this struct was generated from the following file:

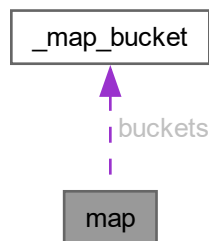
- [src/server/Connection.h](#)

## 7.4 map Struct Reference

A map. Stores key-value pairs for near constant lookup and insertion time.

```
#include <map.h>
```

Collaboration diagram for map:



### 7.4.1 Detailed Description

#### Note

Use [NewMap\(\)](#) to create a new map.

Use [Map\\_Set\(\)](#) to set a key in the map.

Use [Map\\_Get\(\)](#) to get a value from the map.

The values stored are of type void pointer.

Definition at line 101 of file [map.h](#).

The documentation for this struct was generated from the following file:

- [src/server/map.h](#)



## 7.5 map\_result Struct Reference

The result of a map retrieval.

```
#include <map.h>
```

### 7.5.1 Detailed Description

Definition at line 111 of file [map.h](#).

The documentation for this struct was generated from the following file:

- [src/server/map.h](#)

## 7.6 ServerProperties Struct Reference

```
#include <Server.h>
```

### Data Fields

- `uint16_t` [port](#)
- `size_t` [send\\_buffer\\_size](#)
- `size_t` [receive\\_buffer\\_size](#)
- `int` [socket\\_id](#)
- `int` [backlog](#)
- `int` [active\\_connections](#)
- `int` [max\\_connections](#)
- `time_t` [time\\_started](#)
- `char *` [cipher](#)
- `char` [start](#)
- `char` [end](#)

### 7.6.1 Detailed Description

Defines the properties for the server.

Defined in [server-settings.txt](#), a configuration file.

Definition at line 19 of file [Server.h](#).

### 7.6.2 Field Documentation

#### 7.6.2.1 port

```
uint16_t port
```

The port the server will connect on.

Definition at line 21 of file [Server.h](#).

#### 7.6.2.2 `send_buffer_size`

```
size_t send_buffer_size
```

The size of each send buffer.

Definition at line 23 of file [Server.h](#).

#### 7.6.2.3 `receive_buffer_size`

```
size_t receive_buffer_size
```

The size of each receive buffer.

Definition at line 25 of file [Server.h](#).

#### 7.6.2.4 `socket_id`

```
int socket_id
```

The socket ID for the bound interface

Definition at line 27 of file [Server.h](#).

#### 7.6.2.5 `backlog`

```
int backlog
```

The size of the backlog of unprocessed connections.

Definition at line 29 of file [Server.h](#).

#### 7.6.2.6 `active_connections`

```
int active_connections
```

The number of active connections.

Definition at line 31 of file [Server.h](#).

#### 7.6.2.7 `max_connections`

```
int max_connections
```

The maximum number of active connections the server supports.

Definition at line 33 of file [Server.h](#).

#### 7.6.2.8 time\_started

```
time_t time_started
```

The time the server was started.

Definition at line 35 of file [Server.h](#).

#### 7.6.2.9 cipher

```
char* cipher
```

The cipher to use when encrypting and decrypting messages

Definition at line 37 of file [Server.h](#).

#### 7.6.2.10 start

```
char start
```

first character in cipher substitutions range

Definition at line 39 of file [Server.h](#).

#### 7.6.2.11 end

```
char end
```

last character in cipher substitution range

Definition at line 41 of file [Server.h](#).

The documentation for this struct was generated from the following file:

- [src/server/Server.h](#)

## 7.7 User Struct Reference

```
#include <Data.h>
```

## Data Fields

- char **id** [[ID\\_MAX\\_LENGTH](#)]  
*The user ID; equal to an element in `accepted_userIDs`.*
- char **name** [[NAME\\_MAX\\_LENGTH](#)]  
*The user's real name; equal to an element in `userFullNames`.*
- int **age**  
*The user's age, randomized between 18 and 22.*
- float **gpa**  
*The user's gpa, randomized between 2.5 and 4.0.*
- short **connected**  
*Whether the user is connected.*
- char **ip** [[IP\\_LENGTH](#)]  
*The last IP used by the user; set on connection.*
- long **lastConnection**  
*A unix timestamp representing the last time a user connected.*
- short **registered**  
*Whether user has executed the 'register' command.*
- char **password** [[PASSWORD\\_LENGTH](#)]  
*Passsword of the user.*

### 7.7.1 Detailed Description

A [User](#) of this server. The ID and Name fields are populated initially. GPA and age are populated at the time a user is registered, and saved and loaded from a file. Active is set and unset when a user connects. IP is set each time a user connects, and saved in the file.

Definition at line [48](#) of file [Data.h](#).

The documentation for this struct was generated from the following file:

- `src/server/Data.h`

# Chapter 8

## File Documentation

### 8.1 Build.c

```
00001
00005 #include <stdlib.h>
00006 #include <string.h>
00007 #include "Build.h"
00008 #include "Util.h"
00009
00010 User * CreateUsersArray(char ** userIDs, char ** userNames, int recordsCount)
00011 {
00012     size_t uarr_size = sizeof(User) * recordsCount;
00013     User * uarr = malloc(uarr_size);
00014     memset(uarr, 0, uarr_size);
00015     int i;
00016     for(i = 0; i < recordsCount; i++)
00017     {
00018         strcpy(uarr[i].id, userIDs[i]);
00019         strcpy(uarr[i].name, userNames[i]);
00020     }
00021     return uarr;
00022 }
00023
00024 map * CreateUsersMap(User * usersArray, int recordsCount)
00025 {
00026     map * umap = NewMap(recordsCount * 3);
00027     int i;
00028     for(i = 0; i < recordsCount; i++) {
00029         Map_Set(umap, usersArray[i].id, &usersArray[i]);
00030         map_result mr = Map_Get(umap, usersArray[i].id);
00031         if(!mr.found) {
00032             printRed("Map failed on user: %s... your program may have issues.\n", usersArray[i].id);
00033         }
00034     }
00035     return umap;
00036 }
00037
00038
```

### 8.2 Build.h

```
00001 #ifndef Build_h
00002 #define Build_h
00008 #include "Data.h"
00009 #include "map.h"
00010
00018 User * CreateUsersArray(char ** userIDs, char ** userNames, int recordsCount);
00019
00027 map * CreateUsersMap(User * usersArray, int recordsCount);
00028
00029
00030
00034 #endif
```

## 8.3 Cipher.c

```

00001
00006 /*
00007     Class: ECET 4640-002
00008     Assignment: Lab Assignment 3
00009     Authors: Christian Messmer, Karl Miller, Paul Shriner
00010
00011     Cipher.c: Functions used for generating the cipher, printing it out, and encrypting a string.
00012 */
00013
00014 #include <stdio.h>
00015 #include <stdlib.h>
00016 #include <time.h>
00017 #include <string.h>
00018 #include "colors.h"
00019 #include "Util.h"
00020
00031 void FillArraySequential(char *array, char start, char end)
00032 {
00033     //char length = end - start + 1;
00034     char i;
00035     for (i = start; i <= end; i++)
00036     {
00037         array[i - start] = i;
00038     }
00039 }
00040
00041 // See Cipher.h for header comments
00042 void GenerateCipher(char *cipher, char start, char end)
00043 {
00044     time_t t;
00045     srand((unsigned)time(&t));
00046
00047     FillArraySequential(cipher, start, end);
00048     int length = end - start + 1;
00049     int hold, swap_index, i;
00050     for (i = 0; i < length; i++)
00051     {
00052         swap_index = rand() % length;
00053         hold = cipher[swap_index];
00054         cipher[swap_index] = cipher[i];
00055         cipher[i] = hold;
00056     }
00057 }
00058
00059 // See Cipher.h for header comments
00060 void PrintCipher(char *cipher, char start, char length)
00061 {
00062     printf("\nCipher:\n");
00063     int i;
00064     for (i = 0; i < length; i++)
00065     {
00066         printf("%s'%s%c%s'\\" %s \u00BB %s'\\" %s%c%s'\\" %s      ", COLOR_GRAY, COLOR_RED, i + start,
00067             COLOR_GRAY, COLOR_RESET, COLOR_GRAY, COLOR_BLUE, cipher[i], COLOR_GRAY, COLOR_RESET);
00068         if ((i + 1) % 5 == 0)
00069         {
00070             printf("\n");
00071         }
00072     }
00073     printf("\n");
00074 }
00075 // See Cipher.h for header comments
00076 void EncryptString(char *string, int length, char *cipher, char start, char end)
00077 {
00078     //char cipher_l = end - start + 1;
00079     int i;
00080     for (i = 0; i < length; i++)
00081     {
00082         // printf("Encrypting string[%d] , was %c\n", i, string[i]);
00083         // printf("String in range between %c and %c?\n", start, end);
00084         if (!(string[i] - start > end || string[i] < start))
00085         {
00086             // printf("String in range\n");
00087             // printf("String[%d] - %d(start) is: %d\n", i, start, string[i]-start);
00088             string[i] = cipher[string[i] - start];
00089             // printf("String[%d] is now %c\n", i, string[i]);
00090         }
00091     }
00092 }
00093 }
00094
00095 void DecryptString(char* string, int length, char* cipher, char start, char end) {
00096     //char cipher_l = end - start + 1;
00097     int i;

```

```

00099     for(i = 0; i < length; i++) {
00100         if(!(string[i] - start > end || string[i] < start)) {
00101             char* c = strchr(cipher, string[i]);
00102             string[i] = c - cipher + start;
00103         }
00104     }
00105 }
00106
00107

```

## 8.4 Cipher.h

```

00001 #ifndef Cipher_h
00002 #define Cipher_h
00009 /*
00010     Class: ECET 4640-002
00011     Assignment: Lab Assignment 3
00012     Authors: Christian Messmer, Karl Miller, Paul Shriner
00013
00014     Cipher.h: Function prototypes for Cipher.c
00015 */
00016
00031 void GenerateCipher(char *cipher, char start, char end);
00032
00046 void PrintCipher(char *cipher, char start, char length);
00047
00062 void EncryptString(char *string, int length, char *cipher, char start, char end);
00063
00078 void DecryptString(char* string, int length, char* cipher, char start, char end);
00079
00083 #endif

```

## 8.5 colors.h

```

00001 /*
00002     Class: ECET 4640-002
00003     Assignment: Lab Assignment 3
00004     Authors: Christian Messmer, Karl Miller, Paul Shriner
00005
00006     colors.h: Define color macros for use with printing text to the console
00007
00008     Acknowledgements/Credits:
00009         1. https://www.man7.org/linux/man-pages/man4/console\_codes.4.html
00010 */
00011
00012 #ifndef colors_h
00013 #define colors_h
00014 /*
00015     Karl's magic color macros.
00016
00017     These use Virtual Terminal escape sequences to trigger color changes on the console when printed.
00018
00019     See 1 in Acknowledgements/Credits for more information.
00020
00021 */
00022 #define COLOR_RED "\e[38;2;255;75;75m"
00023 #define COLOR_BLUE "\e[38;2;0;240;240m"
00024 #define COLOR_GREEN "\e[38;2;0;240;0m"
00025 #define COLOR_YELLOW "\e[38;2;255;255;0m"
00026 #define COLOR_GRAY "\e[38;2;224;224;224m"
00027 #define COLOR_BOLD "\e[1m"
00028 #define COLOR_RESET "\e[0m"
00029
00030 #endif

```

## 8.6 Connection.c

```

00001
00005 #include "Connection.h"
00006 #include <stdio.h>
00007 #include <stdlib.h>
00008 #include <strings.h>
00009 #include <string.h>
00010 #include <arpa/inet.h>
00011 #include <unistd.h>

```

```

00012 #include "Util.h"
00013 #include "Data.h"
00014 #include "File.h"
00015 #include "Logfile.h"
00016 #include "Cipher.h"
00017
00018 ClientShared shared;
00019
00020 ClientShared * InitializeShared(map * users_map, size_t send_buffer_size, size_t receive_buffer_size,
char* cipher, char start, char end)
00021 {
00022     shared.users = users_map;
00023     shared.dirty = 0;
00024     shared.shutting_down = 0;
00025     shared.send_buffer_size = send_buffer_size;
00026     shared.receive_buffer_size = receive_buffer_size;
00027     shared.cipher = cipher;
00028     shared.start = start;
00029     shared.end = end;
00030     pthread_mutex_init(&(shared.mutex), NULL);
00031     return &shared;
00032 }
00033
00034 void * StartUpdateThread(void * parameter)
00035 {
00036     while(shared.shutting_down == 0) {
00037         if(shared.dirty) {
00038             pthread_mutex_lock(&(shared.mutex));
00039             shared.dirty = 0;
00040             FILE * reg_file = CreateOrOpenFileVerbose(REGISTERED_FILE, NULL);
00041             if(reg_file != NULL) {
00042                 UpdateRegisteredFileFromUsersMap(reg_file, shared.users);
00043                 fclose(reg_file);
00044             } else {
00045                 LogfileError("FAILED TO OPEN REGISTERED FILE - NO DATA WILL BE UPDATED");
00046                 shared.dirty = 1;
00047             }
00048             pthread_mutex_unlock(&(shared.mutex));
00049         }
00050         sleep(1);
00051     }
00052     return NULL;
00053 }
00054
00055 void * StartConnectionThread(void * p_connection)
00056 {
00057     Connection * connection = (Connection *) p_connection;
00058     connection->state = ClientState_ENTRY;
00059     connection->user = NULL;
00060     time(&(connection->time_connected));
00061     // allocate send and receive buffers.
00062     char * send_buffer = malloc(shared.send_buffer_size);
00063     char * receive_buffer = malloc(shared.receive_buffer_size);
00064     map_result result;
00065
00066     // ask for their user ID initially, or disconnect them.
00067     strcpy(send_buffer, "<Message>Welcome. Please send your user ID.");
00068     MessageOrClose(send_buffer, receive_buffer, connection);
00069     if(connection->status == ConnectionStatus_ACTIVE) {
00070         result = Map_Get(shared.users, receive_buffer);
00071         if(!result.found)
00072         {
00073             printYellow("Unauthorized access attempt by %s with name '%s'.\n",
inet_ntoa(connection->address.sin_addr), receive_buffer);
00074             strcpy(send_buffer, "<Error>No such user");
00075             MessageAndClose(send_buffer, connection);
00076             LogfileError("Unauthorized access attempt by unknown user %s from %s.", receive_buffer,
inet_ntoa(connection->address.sin_addr));
00077             // send a one-way message to the client
00078         } else {
00079             User * user = (User *) result.data;
00080             if(user->connected) {
00081                 printYellow("User %s attempted to double connect from IP %s.\n", user->id,
inet_ntoa(connection->address.sin_addr));
00082                 strcpy(send_buffer, "<Error>You are already connected.");
00083                 LogfileError("User %s attempted to double connect from IP %s.\n", user->id,
inet_ntoa(connection->address.sin_addr));
00084                 MessageAndClose(send_buffer, connection);
00085                 // send the other connected user an informative message?
00086             } else {
00087                 connection->user = user;
00088                 connection->user->connected = 1;
00089                 strcpy(connection->user->ip, inet_ntoa(connection->address.sin_addr));
00090                 if(connection->user->registered) {
00091                     connection->state = ClientState_UNAUTHENTICATED;
00092                     LogfileMessage("User %s is attempting a login from ip %s.",
00093

```



```

        connection->user->name, inet_ntoa(connection->address.sin_addr));
00094         } else {
00095             connection->state = ClientState_ACCESSING;
00096         }
00097     }
00098 }
00099 }
00100
00101 if(connection->state == ClientState_ACCESSING && connection->status == ConnectionStatus_ACTIVE) {
00102     strcpy(send_buffer, "<Message>Say something, unregistered user!");
00103 } else if (connection->state == ClientState_UNAUTHENTICATED && connection->status ==
ConnectionStatus_ACTIVE) {
00104     strcpy(send_buffer, "<Message>Say something, registered user (logged out)!");
00105 } else if (connection->state == ClientState_REGISTERED && connection->status ==
ConnectionStatus_ACTIVE) {
00106     strcpy(send_buffer, "<Message>Say something, registered user (logged in)!");
00107 }
00108
00109 while(connection->status == ConnectionStatus_ACTIVE)
00110 {
00111     if(connection->state == ClientState_ACCESSING) {
00112         MessageOrClose(send_buffer, receive_buffer, connection);
00113         if (strcmp(receive_buffer, "help") == 0) {
00114             _help(connection, send_buffer);
00115         } else if (strcmp(receive_buffer, "exit") == 0) {
00116             strcpy(send_buffer, "<Message>Goodbye.");
00117             MessageAndClose(send_buffer, connection);
00118         } else if (strcmp(receive_buffer, "register") == 0) {
00119             if (_password(connection) == 0) {
00120                 _register(connection, send_buffer);
00121                 LogfileMessage("User %s registered.", connection->user->id);
00122             } else {
00123                 strcpy(send_buffer, "<Error>Invalid password entered, cannot register");
00124                 LogfileError("User %s attempted to register with an invalid password.",
connection->user->id);
00125             }
00126         } else {
00127             strcpy (send_buffer, "<Error>Invalid command, use 'help' for list of commands");
00128         }
00129     } else if (connection->state == ClientState_UNAUTHENTICATED) {
00130         MessageOrClose(send_buffer, receive_buffer, connection);
00131         if (strcmp(receive_buffer, "help") == 0) {
00132             _help(connection, send_buffer);
00133         } else if (strcmp(receive_buffer, "exit") == 0) {
00134             strcpy(send_buffer, "<Message>Goodbye.");
00135             MessageAndClose(send_buffer, connection);
00136         } else if (strcmp(receive_buffer, "login") == 0) {
00137             if (_authenticate(connection) == 0) {
00138                 strcpy(send_buffer, "<Message>You have logged in!");
00139                 connection->user->lastConnection = time(NULL);
00140                 printBlue("Setting connection time_connected to: %d\n",
connection->time_connected);
00141                 shared.dirty = 1;
00142                 LogfileMessage("User %s logged in.", connection->user->id);
00143             } else {
00144                 strcpy(send_buffer, "<Message>Login failed!");
00145                 LogfileError("Failed login attempt for user %s.", connection->user->id);
00146             }
00147         } else {
00148             strcpy (send_buffer, "<Error>Invalid command, use 'help' for list of commands");
00149         }
00150     } else if (connection->state == ClientState_REGISTERED) {
00151         MessageOrClose(send_buffer, receive_buffer, connection);
00152         if (strcmp(receive_buffer, "help") == 0) {
00153             _help(connection, send_buffer);
00154         } else if (strcmp(receive_buffer, "exit") == 0) {
00155             strcpy(send_buffer, "<Message>Goodbye.");
00156             MessageAndClose(send_buffer, connection);
00157         } else if (strcmp(receive_buffer, "myinfo") == 0) {
00158             _myinfo(connection, send_buffer);
00159         } else if (strcmp(receive_buffer, "who") == 0) {
00160             _who(send_buffer);
00161         } else if (strcmp(receive_buffer, "random-gpa") == 0) {
00162             _rand_gpa(connection, send_buffer);
00163         } else if (strcmp(receive_buffer, "random-age") == 0) {
00164             _rand_age(connection, send_buffer);
00165         } else if (strcmp(receive_buffer, "advertisement") == 0) {
00166             _advertisement(connection, send_buffer);
00167         } else if (strcmp(receive_buffer, "change-password") == 0) {
00168             if (_password(connection) == 0) {
00169                 strcpy(send_buffer, "<Message>Password has been changed");
00170             } else {
00171                 strcpy(send_buffer, "<Error>Invalid password entered, no action taken");
00172             }
00173         } else {
00174             strcpy(send_buffer, "<Error>Invalid command, use 'help' for list of commands");
00175         }
00176     }
00177 }

```

```

00176         // call a function for processing this state.
00177     } else {
00178         printRed("Client entered invalid state. Disconnecting. \n");
00179         strcpy(send_buffer, "<Error>You entered an invalid state!");
00180         MessageAndClose(send_buffer, connection);
00181         connection->status = ConnectionStatus_CLOSING;
00182     }
00183 }
00184
00185 if(connection->user != NULL) {
00186     connection->user->connected = 0;
00187     printf("User %s from ip %s disconnected.\n", connection->user->id, connection->user->ip);
00188     LogfileMessage("User %s from ip %s disconnected.", connection->user->id,
connection->user->ip);
00189 } else {
00190     printf("Ip %s disconnected.\n", inet_ntoa(connection->address.sin_addr));
00191     LogfileMessage("Ip %s disconnected.\n", inet_ntoa(connection->address.sin_addr));
00192 }
00193
00194
00195 free(send_buffer);
00196 free(receive_buffer);
00197 close(connection->socket);
00198 if(connection->user != NULL) {
00199     connection->user->connected = 0;
00200 }
00201 connection->status = ConnectionStatus_CLOSED;
00202 return NULL;
00203 }
00204
00205
00206 int MessageOrClose(char * send_buffer, char * receive_buffer, Connection * connection) {
00207     receive_buffer[0] = '\0';
00208     EncryptString(send_buffer, strlen(send_buffer), shared.cipher, shared.start, shared.end);
00209     memset(receive_buffer, 0, shared.receive_buffer_size);
00210     if(send(connection->socket, send_buffer, shared.send_buffer_size, 0) < 0) {
00211         printRed("Failed to send message to %s. Disconnecting.\n",
inet_ntoa(connection->address.sin_addr));
00212         perror("Error:");
00213         connection->status = ConnectionStatus_CLOSING;
00214         return 0;
00215     }
00216     int received_size = recv(connection->socket, receive_buffer, shared.receive_buffer_size, 0);
00217     if(received_size < 0) {
00218         printRed("Failed to receive message from %s. Disconnecting.\n",
inet_ntoa(connection->address.sin_addr));
00219         perror("Error: ");
00220         connection->status = ConnectionStatus_CLOSING;
00221         return 0;
00222     }
00223     if(received_size == 0 ) {
00224         printBlue("%s disconnected.\n", inet_ntoa(connection->address.sin_addr));
00225         connection->status = ConnectionStatus_CLOSING;
00226         return 0;
00227     }
00228     send_buffer[0] = '\0';
00229     // memset(send_buffer, 0, shared.send_buffer_size);
00230
00231     DecryptString(receive_buffer, strlen(receive_buffer), shared.cipher, shared.start, shared.end);
00232     return received_size;
00233 }
00234
00235
00236
00237 void MessageAndClose(char * send_buffer, Connection * connection) {
00238     strcat(send_buffer, "<Disconnect>");
00239     EncryptString(send_buffer, strlen(send_buffer), shared.cipher, shared.start, shared.end);
00240     send(connection->socket, send_buffer, shared.send_buffer_size, 0);
00241     connection->status = ConnectionStatus_CLOSING;
00242     if (connection -> user != NULL) {
00243         connection->user->connected = 0;
00244     }
00245 }
00246
00247 void _help(Connection* connection, char* response) {
00248     if (connection -> state == ClientState_UNAUTHENTICATED) {
00249         strcpy(response, "<Message>help - get a list of available commands\n");
00250         strcat(response, "login - login to the server\n");
00251         strcat(response, "exit - disconnect from the server");
00252         LogfileMessage("%s asked for help.", inet_ntoa(connection->address.sin_addr));
00253     } else if(connection->state != ClientState_REGISTERED) {
00254         strcpy(response, "<Message>help - get a list of available commands\n");
00255         strcat(response, "register - register your user\n");
00256         strcat(response, "exit - disconnect from the server");
00257         LogfileMessage("%s asked for help.", inet_ntoa(connection->address.sin_addr));
00258     } else if(connection->state == ClientState_REGISTERED) {
00259         strcpy(response, "<Message>help - get a list of available commands\n");

```

```

00260         strcat(response, "exit - disconnect from the server\n");
00261         strcat(response, "who - get a list of online users\n");
00262         strcat(response, "random-gpa - set your gpa to a new random value\n");
00263         strcat(response, "random-age - set your age to a new random value\n");
00264         strcat(response, "advertisement - get a colorful advertisement\n");
00265         strcat(response, "myinfo - get info about yourself\n");
00266         strcat(response, "change-password - change your current password\n");
00267         LogfileMessage("%s asked for help.", connection->user->name);
00268     }
00269 }
00270
00271 int _register(Connection * connection, char* response) {
00272     if(connection->user->registered) {
00273         strcpy(response, "<Error>");
00274         strcat(response, connection->user->id);
00275         strcat(response, " is already registered.");
00276
00277         printRed("%s from ip %s has attempted to register a second time.\n", connection->user->id,
inet_ntoa(connection->address.sin_addr));
00278         LogfileError("%s from ip %s has attempted to register a second time.\n", connection->user->id,
inet_ntoa(connection->address.sin_addr));
00279         return 0;
00280     }
00281
00282     pthread_mutex_lock(&(shared.mutex));
00283
00284     connection->user->registered = 1;
00285
00286     connection->user->age = RandomInteger(18, 22);
00287
00288     if(RandomFlag(.4)) {
00289         connection->user->gpa = 4.0;
00290     } else {
00291         connection->user->gpa = RandomFloat(2.5, 4);
00292     }
00293
00294     connection->state = ClientState_REGISTERED;
00295
00296     LogfileMessage("%s registered from ip %s.", connection->user->id,
inet_ntoa(connection->address.sin_addr));
00297     printBlue("%s registered.\n", connection->user->id);
00298
00299     shared.dirty = 1;
00300     pthread_mutex_unlock(&(shared.mutex));
00301
00302     strcpy(response, "<Message>You have been registered ");
00303     strcat(response, connection->user->name);
00304
00305     return 1;
00306 }
00307
00308 int _myinfo(Connection* connection, char* response) {
00309
00310     if (!(connection->user->registered)) {
00311         strcpy(response, "<Error>");
00312         strcat(response, connection->user->id);
00313         strcat(response, " is not registered.");
00314
00315         LogfileError("%s from ip %s has attempted to view their information as an unregistered
user.\n", connection->user->id, inet_ntoa(connection->address.sin_addr));
00316
00317         return 1;
00318     }
00319
00320     //Referenced snprintf from https://cplusplus.com/reference/cstdio/snprintf/
00321     snprintf(response, shared.send_buffer_size, "<User.Name>%s<User.Age>%d<User.GPA>%.2f<User.IP>%s",
connection->user->name, connection->user->age, connection->user->gpa,
inet_ntoa(connection->address.sin_addr));
00322
00323     printf("%s viewed their information.\n", connection->user->id);
00324     LogfileMessage("%s viewed their information.", connection->user->name);
00325
00326     return 0;
00327 }
00328
00329 void _who(char * response) {
00330     int i;
00331     for(i = 0; i < RECORD_COUNT; i++) {
00332         map_result result = Map_Get(shared.users, accepted_userIDs[i]);
00333         if(result.found) {
00334             User* user = (User *) result.data;
00335
00336             if(user->connected) {
00337                 strcat(response, "<OnlineUser>");
00338                 strcat(response, user->id);
00339             }
00340         }
00341     }

```

```

00341     }
00342 }
00343
00344 void _rand_gpa(Connection* connection, char* response) {
00345     char gpa_str[5];
00346     pthread_mutex_lock(&(shared.mutex));
00347     if(RandomFlag(.4)) {
00348         connection->user->gpa = 4.0;
00349     } else {
00350         connection->user->gpa = RandomFloat(2.2, 4.0);
00351     }
00352     shared.dirty = 1;
00353     pthread_mutex_unlock(&(shared.mutex));
00354     sprintf(gpa_str, "%.2f", connection->user->gpa);
00355     strcat(response, "<User.GPA>");
00356     strcat(response, gpa_str);
00357     LogfileMessage("%s randomized their gpa.", connection->user->name);
00358 }
00359
00360 void _rand_age(Connection* connection, char * response) {
00361     char age_str[5];
00362     pthread_mutex_lock(&(shared.mutex));
00363     connection->user->age = RandomInteger(18, 22);
00364     shared.dirty = 1;
00365     pthread_mutex_unlock(&(shared.mutex));
00366     sprintf(age_str, "%d", connection->user->age);
00367     strcat(response, "<User.Age>");
00368     strcat(response, age_str);
00369     LogfileMessage("%s randomized their age.", connection->user->name);
00370 }
00371
00372 void _advertisement(Connection * connection, char * response) {
00373     char filename[FILENAME_MAX];
00374
00375     GetRandomFileNameFromDir(ADS_DIR, filename);
00376
00377     char* filepath = malloc(FILENAME_MAX + sizeof(ADS_DIR));
00378     strcpy(filepath, ADS_DIR);
00379     strcat(filepath, "/");
00380     strcat(filepath, filename);
00381
00382     strcat(response, "<Message>");
00383
00384     LogfileMessage("User %s viewed advertisement %s.", connection->user->name, filepath);
00385     CatFileToBuffer(filepath, response, shared.send_buffer_size);
00386
00387     free(filepath); //always free malloced strings to prevent mem leaks!
00388 }
00389
00390 int _password (Connection* connection) {
00391     // Create send/rcv buffers, password buffer
00392     char * send_buffer = malloc(shared.send_buffer_size);
00393     char * receive_buffer1 = malloc(shared.receive_buffer_size);
00394     char * receive_buffer2 = malloc(shared.receive_buffer_size);
00395     char * password = malloc(PASSWORD_LENGTH);
00396
00397     // Prompt user for password two times
00398     strcpy(send_buffer, "<Message>Enter a password");
00399     MessageOrClose(send_buffer, receive_buffer1, connection);
00400     strcpy(send_buffer, "<Message>Enter the same password");
00401     MessageOrClose(send_buffer, receive_buffer2, connection);
00402
00403     // Check if passwords match and is within PASSWORD_LENGTH
00404     if ((strcmp(receive_buffer1, receive_buffer2) != 0) || (strlen(receive_buffer1) >
PASSWORD_LENGTH)) {
00405         free(send_buffer);
00406         free(receive_buffer1);
00407         free(receive_buffer2);
00408         free(password);
00409         return 1;
00410     }
00411
00412     // If so, fill password with buffer up to PASSWORD_LENGTH
00413     int i = 0;
00414     for (i = 0; i < PASSWORD_LENGTH; ++i) {
00415         password[i] = receive_buffer1[i];
00416     }
00417
00418     // Set user's password
00419     pthread_mutex_lock(&(shared.mutex));
00420     strcpy(connection->user->password, password);
00421     shared.dirty = 1;
00422     pthread_mutex_unlock(&(shared.mutex));
00423
00424     free(send_buffer);
00425     free(receive_buffer1);
00426     free(receive_buffer2);

```

```

00427     free(password);
00428     return 0;
00429 }
00430
00431 int _authenticate (Connection* connection) {
00432     // Create send/rcv buffers
00433     char * send_buffer = malloc(shared.send_buffer_size);
00434     char * receive_buffer = malloc(shared.receive_buffer_size);
00435
00436     // Prompt for password
00437     strcpy(send_buffer, "<Message>Enter your password");
00438     MessageOrClose(send_buffer, receive_buffer, connection);
00439
00440     // Check if password matches current user's password
00441     if (strcmp(receive_buffer, connection->user->password) != 0) {
00442         free(send_buffer);
00443         free(receive_buffer);
00444         return 1;
00445     }
00446
00447     // If so, user is now authenticated
00448     pthread_mutex_lock(&(shared.mutex));
00449     connection->state = ClientState_REGISTERED;
00450     shared.dirty = 1;
00451     pthread_mutex_unlock(&(shared.mutex));
00452
00453     free(send_buffer);
00454     free(receive_buffer);
00455     return 0;
00456 }
00457

```

## 8.7 Connection.h

```

00001 #ifndef Connection_h
00002 #define Connection_h
00008 #include <netinet/in.h>
00009 #include <pthread.h>
00010 #include <time.h>
00011 #include <Data.h>
00012 #include "map.h"
00013
00017 typedef struct {
00019     map * users;
00021     pthread_mutex_t mutex;
00023     short dirty;
00025     short shutting_down;
00026
00028     size_t send_buffer_size;
00030     size_t receive_buffer_size;
00031
00033     char* cipher;
00035     char start;
00037     char end;
00038 } ClientShared;
00039
00040 #define ClientState_ENTRY 1
00041 #define ClientState_ACCESSING 2
00042 #define ClientState_REGISTERED 3
00043 #define ClientState_UNAUTHENTICATED 4
00044 typedef short ClientState;
00045
00046 #define ConnectionStatus_CLOSED 0
00047 #define ConnectionStatus_ACTIVE 1
00048 #define ConnectionStatus_CLOSING 2
00049 typedef short ConnectionState;
00050
00054 typedef struct {
00056     ConnectionState status;
00058     int socket;
00060     struct sockaddr_in address;
00062     socklen_t address_length;
00064     pthread_t thread_id;
00066     time_t time_connected;
00068     ClientState state;
00070     User * user;
00071
00072 } Connection;
00073
00079 ClientShared * InitializeShared(map * users_map, size_t send_buffer_size, size_t receive_buffer_size,
    char * cipher, char start, char end);
00080
00085 void * StartConnectionThread(void * connection);

```

```

00086
00096 int MessageOrClose(char * send_buffer, char * receive_buffer, Connection * connection);
00097
00103 void MessageAndClose(char * send_buffer, Connection * connection);
00104
00111 void * StartUpdateThread(void * parameter);
00112
00120 int _register(Connection * connection, char* response);
00121
00128 void _help(Connection* connection, char* response);
00129
00137 int _myinfo(Connection* connection, char* response);
00138
00144 void _who(char* response);
00145
00152 void _rand_age(Connection* connection, char* response);
00153
00160 void _rand_gpa(Connection* connection, char* response);
00161
00168 void _advertisement(Connection * connection, char * response);
00169
00176 int _password(Connection* connection);
00177
00184 int _authenticate(Connection* connection);
00185
00189 #endif

```

## 8.8 Data.c

```

00001
00005 #include "Data.h"
00006
00007 char * accepted_userIDs[] = {
00008     "chen",
00009     "beal389",
00010     "bol4559",
00011     "cal6258",
00012     "kre5277",
00013     "lon1150",
00014     "mas9309",
00015     "mes08346",
00016     "mil7233",
00017     "nef9476",
00018     "nov7488",
00019     "pan9725",
00020     "rac3146",
00021     "rub4133",
00022     "shr5683",
00023     "vay3083",
00024     "yos2327"};
00025
00026 char * userFullNames[] = {
00027     "Weifeng Chen",
00028     "Christian Beatty",
00029     "Emily Bolles",
00030     "Cameron Calhoun",
00031     "Ty Kress",
00032     "Cody Long",
00033     "Caleb Massey",
00034     "Christian Messmer",
00035     "Karl Miller",
00036     "Jeremiah Neff",
00037     "Kaitlyn Novacek",
00038     "Joshua Panaro",
00039     "Caleb Rachocki",
00040     "Caleb Ruby",
00041     "Paul Shriner",
00042     "Alan Vayansky",
00043     "Assefa Ayalew Yoseph"};
00044

```

## 8.9 Data.h

```

00001 #ifndef Data_h
00002 #define Data_h
00012 #define RECORD_COUNT 17
00017 #define ID_MAX_LENGTH 9
00022 #define NAME_MAX_LENGTH 21
00023

```

```

00028 #define IP_LENGTH 16
00029
00033 #define PASSWORD_LENGTH 20
00034
00038 extern char * accepted_userIDs[];
00039
00043 extern char * userFullNames[];
00044
00048 typedef struct
00049 {
00051     char id[ID_MAX_LENGTH];
00053     char name[NAME_MAX_LENGTH];
00055     int age;
00057     float gpa;
00059     short connected;
00061     char ip[IP_LENGTH];
00063     long lastConnection;
00065     short registered;
00067     char password[PASSWORD_LENGTH];
00068 } User;
00069
00073 #endif

```

## 8.10 File.c

```

00001
00005 #include <unistd.h>
00006 #include <fcntl.h>
00007 #include <stdio.h>
00008 #include <string.h>
00009 #include <stdlib.h>
00010 #include <unistd.h>
00011 #include <dirent.h>
00012 #include "File.h"
00013 #include "Data.h"
00014 #include "Util.h"
00015
00016 short FileStatus(char * filename) {
00017     int err = access(filename, F_OK);
00018     if(!err) {
00019         err = access(filename, F_OK | R_OK | W_OK);
00020         if(!err) {
00021             return 1;
00022         }
00023         return 2;
00024     }
00025     return 0;
00026 }
00027
00028 FILE * CreateOrOpenFileVerbose(char * filename, char * defaultContents) {
00029     FILE * file = NULL;
00030     int status = FileStatus(filename);
00031
00032     if(status == 2) {
00033         printRed("Error: %s exists but you do not have permission to access it.\n", filename);
00034         return NULL;
00035     }
00036
00037     if(status == 0) {
00038         printf("Creating %s.\n", filename);
00039         file = fopen(filename, "w+");
00040     } else if(status == 1) {
00041         printf("Opening %s.\n", filename);
00042         file = fopen(filename, "r+");
00043     }
00044
00045     if(file == NULL) {
00046         printf(COLOR_RED);
00047         if(status == 0) {
00049             printf("Failed to create %s.\n", filename);
00050             perror("Error: ");
00051         } else if(status == 1) {
00052             printf("Failed to open %s.\n", filename);
00053             perror("Error: ");
00054         } else {
00055             printf("Unknown error opening %s.", filename);
00056         }
00057         printf(COLOR_RESET);
00058         return NULL;
00059     }
00060
00061     if(status == 0) {

```

```

00062         printGreen("Created %s.\n", filename);
00063         if(defaultContents != NULL) {
00064             fpos_t start_pos;
00065             fgetpos(file, &start_pos);
00066             fprintf(file, defaultContents, 0);
00067             fsetpos(file, &start_pos);
00068         }
00069     } else if(status == 1) {
00070         printGreen("Opened %s.\n", filename);
00071     }
00072 }
00073 return file;
00074 }
00075
00076 int ReadKeyIntoSettingsMap(FILE* key_file, map* settings_map) {
00077     char* start = malloc(sizeof(char*));
00078     fgets(start, sizeof(start), key_file);
00079     printBlue("Start char is: '%c'\n", start[0]);
00080     char* end = malloc(sizeof(char*));
00081     fgets(end, sizeof(end), key_file);
00082     printBlue("End char is: '%c'\n", end[0]);
00083
00084     char* start_key = malloc(sizeof("start_char"));
00085     strcpy(start_key, "start_char");
00086     char* end_key = malloc(sizeof("end_char"));
00087     strcpy(end_key, "end_char");
00088     char* cipher_key = malloc(sizeof("cipher"));
00089     strcpy(cipher_key, "cipher");
00090
00091     char* cipher = calloc(end[0] - start[0] + 2, sizeof(char));
00092     printGreen("Value of end char is: '%d'\n", (int)end[0]);
00093     printGreen("Value of start char is: '%d'\n", (int)start[0]);
00094     printBlue("Value of end[0] - start[0] + 1 is: '%d'\n", end[0] - start[0] + 1);
00095     fgets(cipher, end[0] - start[0] + 2, key_file);
00096     printBlue("Cipher is: '%s'\n", cipher);
00097
00098     Map_Set(settings_map, start_key, start);
00099     Map_Set(settings_map, end_key, end);
00100     Map_Set(settings_map, cipher_key, cipher);
00101     return 1;
00102 }
00103
00104
00105 int ReadRegisteredFileIntoUsersMap(FILE * reg_file, map * users_map) {
00106     char userID[ID_MAX_LENGTH];
00107     int user_age;
00108     float user_gpa;
00109     char userLastIP[IP_LENGTH];
00110     long lastConnection;
00111     char userPassword[PASSWORD_LENGTH];
00112
00113     int scan_items;
00114     int line = 1;
00115
00116     while( (scan_items = fscanf(reg_file, "%s\t%d\t%f\t%s\t%ld\t%s", userID, &user_age, &user_gpa,
userLastIP, &lastConnection, userPassword)) == 6) {
00117         map_result result = Map_Get(users_map, userID);
00118         if(result.found == 0) {
00119             printYellow("Couldn't find user %s. Continuing read.\n", userID);
00120             continue;
00121         }
00122         User * user = (User*)result.data;
00123         user->age = user_age;
00124         user->gpa = user_gpa;
00125         strcpy(user->ip, userLastIP);
00126         user->lastConnection = lastConnection;
00127         user->registered = 1;
00128         strcpy(user->password, userPassword);
00129         line++;
00130     }
00131
00132     if(scan_items != EOF) {
00133         printRed("Error scanning registered file on line %d. Expected 5 items but had %d.\n", line,
scan_items);
00134         return 1;
00135     }
00136     return 0;
00137 }
00138
00139 void UpdateRegisteredFileFromUsersMap(FILE * reg_file, map * users_map) {
00140     int i;
00141     for(i = 0; i < RECORD_COUNT; i++) {
00142         map_result result = Map_Get(users_map, accepted_userIDs[i]);
00143         if(!result.found) {
00144             printRed("User %s was not found in users map.", accepted_userIDs[i]);
00145             continue;
00146         }

```



```

00147
00148     User * user = (User *) result.data;
00149     if(user->registered) {
00150         fprintf(reg_file, "%s\t%d\t%f\t%s\t%ld\t%s\n", user->id, user->age, user->gpa, user->ip,
00151             user->lastConnection, user->password);
00152     }
00153 }
00154
00155 int NumberOfFilesInDirectory(char* dir_name) {
00156     int count = 0;
00157
00158     DIR * dirp;
00159     struct dirent * entry;
00160
00161     dirp = opendir(dir_name);
00162
00163     while((entry = readdir(dirp)) != NULL) {
00164         if(entry->d_type == DT_REG) {
00165             count++;
00166         }
00167     }
00168
00169     closedir(dirp);
00170
00171     return count;
00172 }
00173
00174 void GetRandomFileNameFromDir(char * dir_name, char* file_name) {
00175     int file = RandomInteger(0, NumberOfFilesInDirectory(dir_name) - 1);
00176
00177     DIR* dirp = opendir(dir_name);
00178     struct dirent * entry;
00179
00180     while(file >= 0 && ((entry = readdir(dirp)) != NULL)) {
00181         if(entry->d_type == DT_REG) {
00182             file--;
00183         }
00184     }
00185
00186     if(entry != NULL) {
00187         strcpy(file_name, entry->d_name);
00188     }
00189
00190     closedir(dirp);
00191 }
00192
00193 int ReadSettingsFileIntoSettingsMap(FILE * settings_file, map * settings_map) {
00194     char key_read[100];
00195     char value_read[100];
00196
00197     int scan_items;
00198     int line = 1;
00199
00200     while( (scan_items = fscanf(settings_file, " %s = %s ", key_read, value_read)) == 2) {
00201         char * key_alloc = malloc( (strlen(key_read)+1) * sizeof(char));
00202         memset(key_alloc, 0, strlen(key_read)+1);
00203         strcpy(key_alloc, key_read);
00204         char * val_alloc = malloc( (strlen(value_read)+1) * sizeof(char));
00205         memset(val_alloc, 0, strlen(value_read)+1);
00206         strcpy(val_alloc, value_read);
00207         Map_Set(settings_map, key_alloc, val_alloc);
00208         line++;
00209     }
00210
00211     if(scan_items != EOF) {
00212         printRed("Error scanning settings file on line %d. Expected 2 items but had %d.\n", line,
00213             scan_items);
00214         return 1;
00215     }
00216     return 0;
00217 }
00218
00219 void CatFileToBuffer(char* file_name, char* buffer, size_t buffer_size) {
00220     if(FileStatus(file_name)) {
00221         FILE* file = CreateOrOpenFileVerbose(file_name, NULL);
00222         char* temp = malloc(buffer_size);
00223
00224         while(fgets(temp, buffer_size - strlen(buffer), file) && buffer_size - strlen(buffer) > 1) {
00225             strcat(buffer, temp);
00226         }
00227         free(temp);
00228     }
00229 }
00230
00231 int CreateLockfile()

```

```

00232 {
00233     FILE * file = fopen(LOCKFILE, "w");
00234     if(file == NULL) {
00235         return 0;
00236     }
00237     fprintf(file, "0 %d", getpid());
00238     fclose(file);
00239     return 1;
00240 }
00241
00242 int DeleteLockfile()
00243 {
00244     return remove(LOCKFILE);
00245 }
00246

```

## 8.11 File.h

```

00001 #ifndef Files_h
00002 #define Files_h
00008 #include <stdio.h>
00009 #include "map.h"
00010
00011 // ~~~~ Macros ~~~~ //
00012
00016 #define LOCKFILE "/tmp/lab6.lock"
00017
00026 #define REGISTERED_FILE "registered.txt"
00027
00039 #define SERVER_SETTINGS_FILE "server-settings.txt"
00040
00045 #define KEY_FILE "sub.key"
00046
00052 #define ADS_DIR "ads"
00053
00054 #define CIPHER_MAX_LENGTH 100
00055
00056 // ~~~~~ General File Functions ~~~~~ //
00057
00062 short FileStatus(char * filename);
00063
00072 FILE * CreateOrOpenFileVerbose(char * filename, char * defaultContents);
00073
00081 int ReadRegisteredFileIntoUsersMap(FILE * reg_file, map * users_map);
00082
00083
00091 int ReadSettingsFileIntoSettingsMap(FILE * settings_file, map * settings_map);
00092
00099 void UpdateRegisteredFileFromUsersMap(FILE * reg_file, map * users_map);
00100
00108 int ReadKeyIntoSettingsMap(FILE* key_file, map* settings_map);
00109
00116 void GetRandomFileNameFromDir(char* dir_name, char* file_name);
00117
00124 int NumberOfFilesInDirectory(char* dir_name);
00125
00133 void CatFileToBuffer(char* file_name, char* buffer, size_t buffer_size);
00134
00135 /**
00136     Creates a lockfile.
00137     @warning This should only be called by a running server process when a lockfile does not already
00138     exist.
00139     @returns 1 on success, otherwise 0.
00140 */
00140 int CreateLockfile();
00141
00142 /**
00143     Deletes a lockfile.
00144     @returns 1 on success, otherwise 0.
00145 */
00146 int DeleteLockfile();
00147
00151 #endif

```

## 8.12 Logfile.c

```

00001
00006 #include <stdlib.h>
00007 #include <string.h>

```

```

00008 #include <stdarg.h>
00009 #include <stdio.h>
00010 #include <time.h>
00011 #include "Util.h"
00012
00013 char * logfile_name;
00014
00015 char* months[12] = {
00016     "jan",
00017     "feb",
00018     "mar",
00019     "apr",
00020     "may",
00021     "jun",
00022     "jul",
00023     "aug",
00024     "sep",
00025     "oct",
00026     "nov",
00027     "dec"
00028 };
00029
00030 void logline(const char * logtype, char * logmsg) {
00031     time_t current_time;
00032     struct tm *timeinfo;
00033     time(&current_time);
00034     timeinfo=localtime(&current_time);
00035
00036     FILE * file = fopen(logfile_name, "a");
00037     if(file == NULL) {
00038         printRed("Failed to open logfile %s.\n", logfile_name);
00039         return;
00040     }
00041
00042     fprintf(file, "%s-%02d %02d:%02d:%02d %s %s\n", months[timeinfo->tm_mon], timeinfo->tm_mday,
timeinfo->tm_hour, timeinfo->tm_min, timeinfo->tm_sec, logtype, logmsg);
00043
00044     fclose(file);
00045 }
00046
00047 void SetLogfileName(char * logfile_name_param) {
00048     if(logfile_name != NULL) {
00049         free(logfile_name);
00050     }
00051     logfile_name = malloc(strlen(logfile_name_param)+1);
00052     strcpy(logfile_name, logfile_name_param);
00053 }
00054
00055 void LogfileError(const char * format, ...) {
00056     va_list args;
00057     va_start(args, format);
00058
00059     va_list args_copy;
00060     va_copy(args_copy, args);
00061     int size = vsnprintf(NULL, 0, format, args_copy);
00062     va_end(args_copy);
00063
00064     char * file_print_string = (char*) malloc((size+1)*sizeof(char));
00065
00066     vsprintf(file_print_string, format, args);
00067     va_end(args);
00068     logline("ERR", file_print_string);
00069     free(file_print_string);
00070 }
00071
00072 void LogfileMessage(const char * format, ...) {
00073     va_list args;
00074     va_start(args, format);
00075
00076     va_list args_copy;
00077     va_copy(args_copy, args);
00078     int size = vsnprintf(NULL, 0, format, args_copy);
00079     va_end(args_copy);
00080
00081     char * file_print_string = (char*) malloc((size+1)*sizeof(char));
00082
00083     vsprintf(file_print_string, format, args);
00084     va_end(args);
00085     logline("MSG", file_print_string);
00086     free(file_print_string);
00087 }
00088

```

## 8.13 Logfile.h

```

00001 #ifndef Logfile_h
00002 #define Logfile_h
00003
00004 #include <stdio.h>
00005
00017 void SetLogfileName(char * logfile_name);
00018
00027 void LogfileError(const char * format, ...);
00028
00037 void LogfileMessage(const char * format, ...);
00038
00042 #endif
00043

```

## 8.14 main.c

```

00001 #include <stdio.h>
00002 #include "Util.h"
00003 #include "Process.h"
00004 #include <string.h>
00005 // #include "Logfile.h"
00006
00097 int main(int argc, char **argv) {
00098
00099     if(argc <= 1 )
00100     {
00101         RunCommand();
00102     }
00103     else if (strcmp(argv[1], "headless") == 0)
00104     {
00105         RunHeadless(argv[0]);
00106     }
00107     else if (strcmp(argv[1], "stop") == 0)
00108     {
00109         StopCommand();
00110     }
00111     else if (argc > 2)
00112     {
00113         // Print command not found
00114         // HelpCommand()
00115     }
00116     else
00117     {
00118         RunCommand();
00119     }
00120     return 0;
00121 }

```

## 8.15 map.c

```

00001
00005 #include "stdlib.h"
00006 #include "string.h"
00007 #include "map.h"
00008 #include "math.h"
00009
00011 int hash_log2(int num_to_log)
00012 {
00013     int t = 1;
00014     int i = 0;
00015     do
00016     {
00017         num_to_log = num_to_log & ~t;
00018         t = t << 1;
00019         i++;
00020     } while (num_to_log > 0);
00021     return i;
00022 }
00023
00025 int hash_upperLimit(int bitsize)
00026 {
00027     return 1 << bitsize;
00028 }
00029
00031 int char_ratio = (int)(sizeof(int) / sizeof(char));
00032
00034 int hash_string(int hash_table_size, char *string, int strlen)

```

```

00035 {
00036     int i, hash = 2166136261;
00037     for (i = 0; i < strlen; i += 1)
00038     {
00039         hash *= 16777619;
00040         hash ^= string[i];
00041     }
00042     if (hash < 0)
00043     {
00044         hash *= -1;
00045     }
00046     return hash % hash_table_size;
00047 }
00048
00049 map *NewMap(int capacity)
00050 {
00051     int log2 = hash_log2(capacity);
00052     int capac = hash_upperLimit(log2);
00053     int sz = sizeof(struct _map_bucket) * capac;
00054     struct _map_bucket *buckets = malloc(sz);
00055     memset(buckets, 0, sz);
00056     int i;
00057     for (i = 0; i < capac; i++)
00058     {
00059         buckets[i] = (struct _map_bucket){NULL, NULL, NULL};
00060     }
00061     map newm = (map){capac, buckets};
00062     map *map_p = malloc(sizeof(map));
00063     *map_p = newm;
00064     return map_p;
00065 }
00066
00067 void _bucket_insert(struct _map_bucket *bucket, char *key, void *value)
00068 {
00069     struct _map_bucket *check = bucket;
00070     while (check->key != NULL)
00071     {
00072         if (strcmp(check->key, key) == 0)
00073         {
00074             check->data = value;
00075             return;
00076         }
00077         if (check->next == NULL)
00078         {
00079             check->next = malloc(sizeof(struct _map_bucket));
00080             *(check->next) = (struct _map_bucket){NULL, NULL, NULL};
00081         }
00082         check = check->next;
00083     }
00084     check->key = key;
00085     check->data = value;
00086 }
00087
00088
00089 void Map_Set(map *a_map, char *key, void *value)
00090 {
00091     int keyl = (int)strlen(key);
00092     int hash = hash_string(a_map->size, key, keyl);
00093     _bucket_insert(&(a_map->buckets[hash]), key, value);
00094 }
00095
00096 void _bucket_get(struct _map_bucket *bucket, char *key, map_result *result)
00097 {
00098     struct _map_bucket *check = bucket;
00099     while (check->key != NULL)
00100     {
00101         if (strcmp(check->key, key) == 0)
00102         {
00103             result->found = 1;
00104             result->data = check->data;
00105             return;
00106         }
00107         else if (check->next != NULL)
00108         {
00109             check = check->next;
00110         }
00111         else
00112         {
00113             result->found = 0;
00114             break;
00115         }
00116     }
00117 }
00118
00119 map_result Map_Get(map *a_map, char *key)
00120 {
00121     map_result res = (map_result){0, NULL};
00122     int keyl = (int)strlen(key);
00123     int hash = hash_string(a_map->size, key, keyl);

```

```

00124     _bucket_get(&(a_map->buckets[hash]), key, &res);
00125     return res;
00126 }
00127
00128 void _bucket_delete(struct _map_bucket *bucket, char *key, short free_it, map_result *result)
00129 {
00130     struct _map_bucket *last = bucket;
00131     struct _map_bucket *next = bucket->next;
00132     while (next != NULL)
00133     {
00134         if (strcmp(next->key, key) == 0)
00135         {
00136             result->found = 1;
00137             result->data = next->data;
00138             if (free_it)
00139             {
00140                 free(next->data);
00141                 result->data = NULL;
00142             }
00143             last->next = next->next;
00144             free(next);
00145         }
00146         else
00147         {
00148             last = next;
00149             next = next->next;
00150         }
00151     }
00152 }
00153
00154 map_result Map_Delete(map *a_map, char *key, short free_it)
00155 {
00156     map_result res = (map_result){0, NULL};
00157     int keyl = (int)strlen(key);
00158     int hash = hash_string(a_map->size, key, keyl);
00159
00160     struct _map_bucket top = a_map->buckets[hash];
00161     if (top.key == NULL)
00162     {
00163         return res;
00164     }
00165     if (strcmp(top.key, key) == 0)
00166     {
00167         res.found = 1;
00168         res.data = top.data;
00169         if (free_it)
00170         {
00171             free(top.data);
00172             res.data = NULL;
00173         }
00174         if (top.next != NULL)
00175         {
00176             a_map->buckets[hash] = *(top.next);
00177             free(top.next);
00178         }
00179         else
00180         {
00181             a_map->buckets[hash] = (struct _map_bucket){NULL, NULL, NULL};
00182         }
00183         return res;
00184     }
00185     if (top.next == NULL)
00186     {
00187         return res;
00188     }
00189     _bucket_delete(&(a_map->buckets[hash]), key, free_it, &res);
00190
00191     return res;
00192 }

```

## 8.16 map.h

```

00001 #ifndef map_h
00002 #define map_h
00003
00041 // -----
00042 //          Hashing Math
00043 // -----
00044
00051 int hash_log2(int number_to_log);
00052
00062 int hash_string(int hash_table_capacity, char *string, int strlen);
00063

```

```

00070 int hash_upperLimit(int bitsize);
00071
00072 // -----
00073 //           General Map Operations
00074 // -----
00075
00081 struct _map_bucket
00082 {
00084     char *key;
00086     void *data;
00088     struct _map_bucket *next;
00089 };
00090
00101 typedef struct
00102 {
00103     int size;
00104     struct _map_bucket *buckets;
00105 } map;
00106
00111 typedef struct
00112 {
00113
00114     short found;
00115     void *data;
00116 } map_result;
00117
00124 map *NewMap(int capacity);
00125
00133 void Map_Set(map *a_map, char *key, void *value);
00134
00141 map_result Map_Get(map *a_map, char *key);
00142
00150 map_result Map_Delete(map *a_map, char *key, short free_it);
00151
00152 #endif

```

## 8.17 Process.c

```

00001
00005 #include <stdio.h>
00006 #include <string.h>
00007 #include <signal.h>
00008 #include <stdlib.h>
00009 #include <unistd.h>
00010 #include "Data.h"
00011 #include "Build.h"
00012 #include "map.h"
00013 #include "File.h"
00014 #include "Util.h"
00015 #include "Server.h"
00016 #include "Logfile.h"
00017 #include "Connection.h"
00018
00020 User * users_array;
00022 map * users_map;
00024 map * settings_map;
00026 char * default_settings = "port           = 3000\n"
00027                          "send_buffer_size = 1024\n"
00028                          "receive_buffer_size = 1024\n"
00029                          "backlog          = 10\n"
00030                          "max_connections   = 20\n"
00031                          "log_file          = log.txt\n"
00032                          "log_level         = 1\n"
00033                          "log_to_console    = true";
00034
00035 char * default_sub = " \n"
00036                   "~\n"
00037                   "`\"G) YF7A,R2L'@
00038   ZD/E5I<?H:i4NJ&g;rB(f#KobljnWlC{_-Ua}%^cV\\>tOP|pQ$689=+whzS3*Xm!ek~My[ ]sqduv0.Tx";
00040 int active_clients;
00041
00042 int _initializeLogger() {
00043     return 1;
00044     // char* fileName = "log.txt";
00045     // int printLevel, LogLevel, printAlltoStdOut;
00046     // map_result result = Map_Get(settings_map, "log_file");
00047     // if(!result.found) {
00048     //     printYellow("No output file found. Defaulting to 'log.txt'\n");
00049     //     SetLogfileName("log.txt");
00050     // } else {
00051     //     fileName = result.data;
00052     //     SetLogfileName(fileName);

```

```

00053 // }
00054
00055 // result = Map_Get(settings_map, "print_level");
00056 // if(!result.found) {
00057 //     printYellow("No print_level found, defaulting to 3\n");
00058 //     printLevel = 3;
00059 // } else {
00060 //     printLevel = atoi(result.data);
00061 //     if(printLevel < 0 || printLevel > 5) {
00062 //         printYellow("Invalid print_level of %d, defaulting to 3\n", printLevel);
00063 //         printLevel = 3;
00064 //     }
00065 // }
00066
00067 // result = Map_Get(settings_map, "log_level");
00068 // if(!result.found) {
00069 //     printYellow("No log_level found, defaulting to 3\n");
00070 //     LogLevel = 3;
00071 // } else {
00072 //     LogLevel = atoi(result.data);
00073 //     if(LogLevel < 0 || LogLevel > 5) {
00074 //         printYellow("Invalid log_level of %d, defaulting to 3\n", LogLevel);
00075 //         LogLevel = 3;
00076 //     }
00077 // }
00078
00079 // result = Map_Get(settings_map, "log_to_console");
00080 // if(!result.found) {
00081 //     printYellow("No log_to_console found, defaulting to true\n");
00082 //     printAlltoStdOut = 1;
00083 // } else {
00084 //     if(strcmp(result.data, "true") == 0) {
00085 //         printAlltoStdOut = 1;
00086 //     } else if(strcmp(result.data, "false") == 0) {
00087 //         printAlltoStdOut = 0;
00088 //     } else {
00089 //         printYellow("invalid data in log_to_console, defaulting to true\n");
00090 //         printAlltoStdOut = 1;
00091 //     }
00092 // }
00093 // return 1;
00094 }
00095
00096 int InitializeCipher() {
00097     printf("Reading key file.\n");
00098     FILE * key_file = CreateOrOpenFileVerbose(KEY_FILE, default_sub);
00099     if(key_file == NULL) {
00100         printRed("Initialization failed during access of file: %s.\n", KEY_FILE);
00101         return 0;
00102     }
00103     ReadKeyIntoSettingsMap(key_file, settings_map);
00104     // int key_read_err = ReadKeyIntoSettingsMap(key_file, settings_map);
00105     // if(key_read_err) {
00106     //     printRed("Initialization failed while reading key file %s. Correct this file or delete it
so default can be generated.\n", KEY_FILE);
00107     // }
00108     fclose(key_file);
00109     printGreen("Read %s.\n", KEY_FILE);
00110     return 1;
00111 }
00112
00113 int Initialize() {
00114
00115     // Create the data structures on the heap.
00116     printf("Initializing User data structures.\n");
00117     users_array = CreateUsersArray(accepted_userIDs, userFullNames, RECORD_COUNT);
00118     users_map = CreateUsersMap(users_array, RECORD_COUNT);
00119     active_clients = 0;
00120     printGreen("User data structures initialized.\n");
00121
00122     // Create the registered file that tracks registered users.
00123     printf("Checking for registered file.\n");
00124     FILE * reg_file = CreateOrOpenFileVerbose(REGISTERED_FILE, NULL);
00125     if(reg_file == NULL) {
00126         printRed("Initialization failed during accessing of file: %s.\n", REGISTERED_FILE);
00127         return 0;
00128     }
00129
00130     // Update the User's map with with the data from the registered file.
00131     printf("Reading registered file.\n");
00132     int read_error = ReadRegisteredFileIntoUsersMap(reg_file, users_map);
00133     fclose(reg_file);
00134     if(read_error) {
00135         printRed("Initialization failed during reading of file: %s.\n", REGISTERED_FILE);
00136         return 0;
00137     }
00138     printGreen("Loaded %s into users map.\n", REGISTERED_FILE);

```



```

00139
00140     printf("Reading settings file.\n");
00141     settings_map = NewMap(50);
00142     FILE * settings_file = CreateOrOpenFileVerbose(SERVER_SETTINGS_FILE, default_settings);
00143     if(settings_file == NULL) {
00144         printRed("Initialization failed during accessing of file: %s.\n", SERVER_SETTINGS_FILE);
00145         return 0;
00146     }
00147     int settings_read_err = ReadSettingsFileIntoSettingsMap(settings_file, settings_map);
00148     if(settings_read_err) {
00149         printRed("Initialization failed while reading settings file %s. Correct this file or delete it
so a default can be generated.\n", SERVER_SETTINGS_FILE);
00150         return 0;
00151     }
00152     fclose(settings_file);
00153     printGreen("Read %s.\n", SERVER_SETTINGS_FILE);
00154
00155     InitializeCipher();
00156
00157     printf("Initializing logger.\n");
00158     int logger_initialized = _initializeLogger();
00159     if(!logger_initialized) {
00160         printRed("Failed to initialize logger.\n");
00161     }
00162
00163
00164
00165     printf("Initializing server.\n");
00166     int server_initialized = InitializeServer(settings_map);
00167     if(!server_initialized) {
00168         printRed("Failed to initialize server.\n");
00169         return 0;
00170     }
00171
00172
00173     return 1;
00174 }
00175
00176 void SignalHandle(int signo) {
00177     if(signo == SIGINT || signo == SIGTERM) {
00178         printYellow("Received signal. Shutting down server.\n");
00179         int err = CloseServer();
00180         if(err) {
00181             printRed("Problem closing server\n");
00182             perror("Error closing socket interface.\n");
00183         }
00184         DeleteLockfile();
00185         exit(0);
00186     }
00187
00188 }
00189
00190 int RunCommand() {
00191     if (FileStatus(LOCKFILE) > 0)
00192     {
00193         printf("Server process already running.\n");
00194         return 0;
00195     }
00196     signal(SIGTERM, SignalHandle);
00197     signal(SIGINT, SignalHandle);
00198     int init_success = Initialize();
00199     if(!init_success) {
00200         printRed("Could not start the server due to failed initialization.\n");
00201         return 0;
00202     }
00203     printf("Running server.\n");
00204     int server_success = StartServer(users_map);
00205     if(!server_success) {
00206         printRed("There was a problem running the server.\n");
00207         return 0;
00208     }
00209     int delete_lockfile_success = DeleteLockfile();
00210     if(!delete_lockfile_success) {
00211         printRed("There was a problem deleting the Lockfile.\n");
00212         return 0;
00213     }
00214     return 1;
00215 }
00216
00217 void RunHeadless(char *processName) {
00218     if (FileStatus(LOCKFILE) > 0)
00219     {
00220         printf("Server process already running.\n");
00221         return;
00222     }
00223     char commandFront[] = " nohup ";
00224     char commandEnd[] = " & exit";

```

```

00225     size_t comm_length = strlen(commandFront) + strlen(commandEnd) + strlen(processName) + 1;
00226     char *commandFull = malloc(comm_length * sizeof(char));
00227     memset(commandFull, 0, comm_length * sizeof(char));
00228     strcpy(commandFull, commandFront);
00229     strcat(commandFull, processName);
00230     strcat(commandFull, commandEnd);
00231
00232     printf("Executing: %s\n", commandFull);
00233     popen(commandFull, "we");
00234     printf("Server running headlessly.\n");
00235 }
00236
00237 int TerminateExistingServer()
00238 {
00239     FILE *file = fopen(LOCKFILE, "r");
00240     if (file == NULL)
00241     {
00242         perror("Error opening lockfile");
00243         return -1;
00244     }
00245     int need_rewrite;
00246     int pid = 0;
00247     fscanf(file, "%d %d", &need_rewrite, &pid);
00248     fclose(file);
00249     if (pid > 0)
00250     {
00251         return kill(pid, SIGTERM);
00252     }
00253     return -2;
00254 }
00255
00256 void StopCommand() {
00257     printYellow("\nStopping server...\n");
00258     int err = TerminateExistingServer();
00259     if (err)
00260     {
00261         if (err == -1)
00262         {
00263             printRed("Server isn't running.\n");
00264         }
00265         else if (err == -2)
00266         {
00267             printRed("Lockfile did not contain a valid process id!\n");
00268         }
00269         else
00270         {
00271             printRed("Sending terminate signal failed!\n");
00272         }
00273     }
00274     else
00275     {
00276         printGreen("Server terminated.\n");
00277     }
00278 }
00279
00280 void ResetCommand() {
00281     if (FileStatus(REGISTERED_FILE)) {
00282         fclose(fopen(REGISTERED_FILE, "w")); //empties the registered file
00283     }
00284 }

```

## 8.18 Process.h

```

00001 #ifndef Process_h
00002 #define Process_h
00003 /**
00004  * \defgroup Process
00005  * \brief This module holds functions that realize the primary business logic of the program.
00006  * \details When command line arguments are parsed by main, functions in this module are called.
00007  * @{
00008  */
00009
00010 /**
00011  * Handles an interrupt or quit signal.
00012
00013  * Send server to shutdown mode, resulting in graceful deletion of lockfile.
00014  */
00015 void SignalHandle(int signo);
00016
00021 int InitializeCipher();
00022
00023 /**
00024  * Performs initializing activities which must occur prior to a server loop starting.

```

```

00025  *
00026  * Will print errors if there are problems initializing.
00027  *
00028  * @returns 1 on success, otherwise 0.
00029  */
00030  int Initialize();
00031
00032  /**
00033   * Runs the server.
00034   * @returns 1 on success, otherwise 0.
00035   */
00036  int RunCommand();
00037
00038  /**
00039   Finds the process ID of a running server using the lockfile, and calls kill on it, sending a
  SIGTERM.
00040   @returns -1 if the file doesn't exist, -2 if no valid process ID existed in the file, 1 if sending
  the kill signal failed, or 0 on success.
00041
00042  */
00043  int TerminateExistingServer();
00044
00048  void StopCommand();
00049
00054  void RunHeadless(char *processName);
00055
00060  #endif

```

## 8.19 Server.c

```

00001
00005  #include <stdlib.h>
00006  #include <netinet/in.h>
00007  #include <sys/types.h>
00008  #include <sys/socket.h>
00009  #include <arpa/inet.h>
00010  #include <linux/net.h>
00011  #include <stdio.h>
00012  #include <pthread.h>
00013  #include <unistd.h>
00014
00015  #include "Server.h"
00016  #include "Connection.h"
00017  #include "Util.h"
00018  #include "map.h"
00019  #include "Logfile.h"
00020  #include "File.h"
00021
00022  ServerProperties server;
00023  Connection * connections;
00024
00025  // A private function just for reading the settings map into the server struct and printing warnings
  as necessary.
00026  void _readSettingsMapIntoServerStruct(map * server_settings) {
00027      map_result result = Map_Get(server_settings, "port");
00028      if(!result.found) {
00029          printYellow("No port setting found. Defaulting to 3000.\n");
00030          server.port = 3000;
00031      } else {
00032          int found_port = atoi(result.data);
00033          if(found_port <= 0) {
00034              printYellow("Invalid port setting: %s. Defaulting to 3000.\n", result.data);
00035              server.port = htons(3000);
00036          } else {
00037              server.port = htons(found_port);
00038          }
00039      }
00040      result = Map_Get(server_settings, "send_buffer_size");
00041      if(!result.found) {
00042          printYellow("No send_buffer_size setting found. Defaulting to 1024.\n");
00043          server.send_buffer_size = 1024;
00044      } else {
00045          int found_sb_size = atoi(result.data);
00046          if(found_sb_size <= 0) {
00047              printYellow("Invalid send_buffer_size setting: %s. Defaulting to 1024.\n", result.data);
00048              server.send_buffer_size = 1024 * sizeof(char);
00049          } else {
00050              server.send_buffer_size = found_sb_size * sizeof(char);
00051          }
00052      }
00053      result = Map_Get(server_settings, "receive_buffer_size");
00054      if(!result.found) {
00055          printYellow("No receive_buffer_size setting found. Defaulting to 1024.\n");

```

```

00056     server.send_buffer_size = 1024;
00057 } else {
00058     int found_rb_size = atoi(result.data);
00059     if(found_rb_size <= 0) {
00060         printYellow("Invalid receive_buffer_size setting: %. Defaulting to 1024.\n",
result.data);
00061         server.receive_buffer_size = 1024 * sizeof(char);
00062     } else {
00063         server.receive_buffer_size = found_rb_size * sizeof(char);
00064     }
00065 }
00066 result = Map_Get(server_settings, "backlog");
00067 if(!result.found) {
00068     printYellow("No backlog setting found. Defaulting to 10.\n");
00069     server.backlog = 10;
00070 } else {
00071     int found_backlog = atoi(result.data);
00072     if(found_backlog <= 0) {
00073         printYellow("Invalid backlog setting: %. Defaulting to 10.\n", result.data);
00074         server.backlog = 10;
00075     } else {
00076         server.backlog = found_backlog;
00077     }
00078 }
00079 result = Map_Get(server_settings, "max_connections");
00080 if(!result.found) {
00081     printYellow("No max_connections setting found. Defaulting to 20.\n");
00082     server.max_connections = 20;
00083 } else {
00084     int found_max_connections = atoi(result.data);
00085     if(found_max_connections <= 0) {
00086         printYellow("Invalid max_connections setting: %. Defaulting to 20.\n", result.data);
00087         server.max_connections = 20;
00088     } else {
00089         server.max_connections = found_max_connections;
00090     }
00091 }
00092 result = Map_Get(server_settings, "start_char");
00093 if(!result.found) {
00094     printYellow("No start_char setting found. Defaulting to ' '\n");
00095     server.start = ' ';
00096 } else {
00097     char* start = result.data;
00098     server.start = start[0];
00099 }
00100 result = Map_Get(server_settings, "end_char");
00101 if(!result.found) {
00102     printYellow("No end_char setting found. defaulting to '~'\n");
00103     server.end = '~';
00104 } else {
00105     char* end = result.data;
00106     server.end = end[0];
00107 }
00108 result = Map_Get(server_settings, "cipher");
00109 if(!result.found) {
00110     printRed("No cipher found.\n");
00111 } else {
00112     server.cipher = result.data;
00113 }
00114 result = Map_Get(server_settings, "log_file");
00115 if(!result.found) {
00116     printYellow("No log_file specified, defaulting to 'log.txt'.\n");
00117     SetLogfileName("log.txt");
00118 } else {
00119     SetLogfileName((char *) result.data);
00120     printBlue("Logging to %s.\n", result.data);
00121 }
00122 };
00123
00124 int InitializeServer(map * server_settings) {
00125     _readSettingsMapIntoServerStruct(server_settings);
00126     connections = malloc(server.max_connections * sizeof(Connection));
00127     int i;
00128     for(i=0; i < server.max_connections; i++) {
00129         connections[i].status = ConnectionStatus_CLOSED;
00130     }
00131     printGreen("Server initialized with %d max connections.\n", server.max_connections);
00132     return 1;
00133 }
00134
00135 int StartServer(map * users_map) {
00136     int serverSocket = 0;
00137     struct sockaddr_in server_address;
00138     // Record the time the server started.
00139     time(&server.time_started);
00140     // Get a socket file pointer associated with ipv4 internet protocols that represents a two-way
connection based byte stream.

```

```

00141     serverSocket = socket(AF_INET, SOCK_STREAM, 0);
00142     server_address.sin_family = AF_INET;
00143     // Set the address to bind to all available interfaces.
00144     server_address.sin_addr.s_addr = htonl(INADDR_ANY);
00145     // Set the port.
00146     server_address.sin_port = server.port;
00147     // Assign a name to the socket.
00148     int bind_error = bind(serverSocket, (struct sockaddr*)&server_address, sizeof(server_address));
00149     if(bind_error) {
00150         printRed("Error binding the server to port %d.\n", ntohs(server.port));
00151         perror("Bind Error:");
00152         return 0;
00153     }
00154     server.socket_id = serverSocket;
00155     int lockfile_success = CreateLockfile();
00156     if(!lockfile_success) {
00157         printRed("Failed to create Lockfile! Server cannot start.");
00158         return 0;
00159     }
00160     // Initialized a shared space that will be used across threads.
00161     ClientShared * shared = InitializeShared(users_map, server.send_buffer_size,
server.receive_buffer_size, server.cipher, server.start, server.end);
00162     // The update thread is responsible for checking if there is 'dirty' data that should be saved to
the registered user's file.
00163     pthread_t registered_update_thread;
00164     pthread_create(&registered_update_thread, NULL, StartUpdateThread, NULL);
00165     printBlue("Server listening on port: %d\n", ntohs(server.port));
00166     LogfileMessage("Server started.");
00167     // begin listening according to the socket settings
00168     listen(serverSocket, server.backlog);
00169     while(!shared->shutting_down) {
00170         // Get an available connection.
00171         Connection * next_client = NextAvailableConnection();
00172         if(next_client == NULL) {
00173             printYellow("Server connections are maxxed.\n");
00174             LogfileError("Server couldn't accept connection; available connections are maxxed.");
00175             sleep(1);
00176             continue;
00177         }
00178         // Accept a connection.
00179         next_client->address_length = sizeof(next_client->address);
00180         next_client->socket = accept(serverSocket, (struct sockaddr *)&(next_client->address),
&(next_client->address_length));
00181         if(next_client->socket < 0)
00182         {
00183             printRed("Failed to accept() client!\n");
00184             LogfileError("Failed to accept() client.");
00185             sleep(1);
00186             continue;
00187         }
00188         printBlue("New client connection from IP: %s\n", inet_ntoa(next_client->address.sin_addr));
00189         LogfileMessage("New client connection from IP: %s", inet_ntoa(next_client->address.sin_addr));
00190         next_client->status = ConnectionStatus_ACTIVE;
00191         // Start a thread to handle communication from that connection.
00192         pthread_create(&(next_client->thread_id), NULL, StartConnectionThread, next_client);
00193     }
00194
00195     return 1;
00196 }
00197
00198 Connection * NextAvailableConnection()
00199 {
00200     int i;
00201     for(i = 0; i < server.max_connections; i++) {
00202         if(connections[i].status == ConnectionStatus_CLOSED)
00203         {
00204             return &(connections[i]);
00205         }
00206     }
00207     return NULL;
00208 }
00209
00210 int CloseServer() {
00211     return close(server.socket_id);
00212 }
00213
00214

```

## 8.20 Server.h

```

00001 #ifndef Server_h
00002 #define Server_h
00008 #include <stdint.h>

```

```

00009 #include <time.h>
00010 #include "map.h"
00011 #include "Connection.h"
00012
00019 typedef struct {
00021     uint16_t port;
00023     size_t send_buffer_size;
00025     size_t receive_buffer_size;
00027     int socket_id;
00029     int backlog;
00031     int active_connections;
00033     int max_connections;
00035     time_t time_started;
00037     char* cipher;
00039     char start;
00041     char end;
00042 } ServerProperties;
00043
00049 int InitializeServer();
00050
00056 int InitializeCipher();
00057
00064 int StartServer(map * users_map);
00065
00070 int CloseServer();
00071
00077 Connection * NextAvailableConnection();
00078
00082 #endif

```

## 8.21 Util.c

```

00001
00005 #include <stdio.h>
00006 #include <stdarg.h>
00007 #include <string.h>
00008 #include <stdlib.h>
00009 #include "Util.h"
00010
00011 void printRed(const char * format, ...) {
00012     printf(COLOR_RED);
00013     va_list args;
00014     va_start(args, format);
00015     vprintf(format, args);
00016     va_end(args);
00017     printf(COLOR_RESET);
00018 }
00019
00020 void printGreen(const char * format, ...) {
00021     printf(COLOR_GREEN);
00022     va_list args;
00023     va_start(args, format);
00024     vprintf(format, args);
00025     va_end(args);
00026     printf(COLOR_RESET);
00027 }
00028
00029 void printYellow(const char * format, ...) {
00030     printf(COLOR_YELLOW);
00031     va_list args;
00032     va_start(args, format);
00033     vprintf(format, args);
00034     va_end(args);
00035     printf(COLOR_RESET);
00036 }
00037
00038 void printBlue(const char * format, ...) {
00039     printf(COLOR_BLUE);
00040     va_list args;
00041     va_start(args, format);
00042     vprintf(format, args);
00043     va_end(args);
00044     printf(COLOR_RESET);
00045 }
00046
00047 int RandomInteger(int min, int max)
00048 {
00049     int r_add = rand() % (max - min + 1);
00050     return r_add + min;
00051 }
00052
00053 float RandomFloat(float min, float max)
00054 {

```

```

00055     float dif = max - min;
00056     int rand_int = rand() % (int)(dif * 10000);
00057     return min + (float)rand_int / 10000.0;
00058 }
00059
00060 short RandomFlag(float percentage_chance)
00061 {
00062     float random_value = (float)rand() / RAND_MAX;
00063     if (random_value < percentage_chance)
00064     {
00065         return 1;
00066     }
00067     return 0;
00068 }
00069

```

## 8.22 Util.h

```

00001 #ifndef Util_h
00002 #define Util_h
00009 #include <stdarg.h>
00010
00012 #define COLOR_RED "\e[38;2;255;75;75m"
00014 #define COLOR_GREEN "\e[38;2;0;240;0m"
00016 #define COLOR_YELLOW "\e[38;2;255;255;0m"
00018 #define COLOR_BLUE "\e[38;2;0;240;240m"
00020 #define COLOR_RESET "\e[0m"
00021
00027 void printRed(const char * format, ...);
00028
00034 void printGreen(const char * format, ...);
00035
00041 void printYellow(const char * format, ...);
00042
00048 void printBlue(const char * format, ...);
00049
00050
00057 int RandomInteger(int min, int max);
00058
00065 float RandomFloat(float min, float max);
00066
00073 short RandomFlag(float percentage_chance);
00074
00075
00079 #endif

```





# Index

- [\\_advertisement](#)
    - [Connection, 26](#)
  - [\\_authenticate](#)
    - [Connection, 28](#)
  - [\\_help](#)
    - [Connection, 20](#)
  - [\\_map\\_bucket, 59](#)
  - [\\_myinfo](#)
    - [Connection, 22](#)
  - [\\_password](#)
    - [Connection, 27](#)
  - [\\_rand\\_age](#)
    - [Connection, 25](#)
  - [\\_rand\\_gpa](#)
    - [Connection, 24](#)
  - [\\_register](#)
    - [Connection, 21](#)
  - [\\_who](#)
    - [Connection, 23](#)
- [accepted\\_userIDs](#)
  - [Data, 30, 31](#)
- [active\\_clients](#)
  - [Process, 58](#)
- [active\\_connections](#)
  - [ServerProperties, 66](#)
- [address](#)
  - [Connection, 63](#)
- [address\\_length](#)
  - [Connection, 63](#)
- [ADS\\_DIR](#)
  - [Files, 33](#)
- [backlog](#)
  - [ServerProperties, 66](#)
- [Build, 11](#)
  - [CreateUsersArray, 11](#)
  - [CreateUsersMap, 11](#)
- [CatFileToBuffer](#)
  - [Files, 40](#)
- [Cipher, 12](#)
  - [DecryptString, 14](#)
  - [EncryptString, 13](#)
  - [GenerateCipher, 13](#)
  - [PrintCipher, 13](#)
- [cipher](#)
  - [ClientShared, 61](#)
  - [ServerProperties, 67](#)
- [ClientShared, 59](#)
- [cipher, 61](#)
- [dirty, 60](#)
- [end, 61](#)
- [mutex, 60](#)
- [receive\\_buffer\\_size, 61](#)
- [send\\_buffer\\_size, 60](#)
- [shutting\\_down, 60](#)
- [start, 61](#)
- [users, 60](#)
- [CloseServer](#)
  - [Server, 49](#)
- [COLOR\\_BLUE](#)
  - [Util, 51](#)
- [COLOR\\_GREEN](#)
  - [Util, 51](#)
- [COLOR\\_RED](#)
  - [Util, 51](#)
- [COLOR\\_RESET](#)
  - [Util, 51](#)
- [COLOR\\_YELLOW](#)
  - [Util, 51](#)
- [Compilation, 3](#)
- [Connection, 15, 62](#)
  - [\\_advertisement, 26](#)
  - [\\_authenticate, 28](#)
  - [\\_help, 20](#)
  - [\\_myinfo, 22](#)
  - [\\_password, 27](#)
  - [\\_rand\\_age, 25](#)
  - [\\_rand\\_gpa, 24](#)
  - [\\_register, 21](#)
  - [\\_who, 23](#)
  - [address, 63](#)
  - [address\\_length, 63](#)
  - [InitializeShared, 16](#)
  - [MessageAndClose, 19](#)
  - [MessageOrClose, 18](#)
  - [socket, 62](#)
  - [StartConnectionThread, 17](#)
  - [StartUpdateThread, 16](#)
  - [state, 63](#)
  - [status, 62](#)
  - [thread\\_id, 63](#)
  - [time\\_connected, 63](#)
  - [user, 63](#)
- [CreateOrOpenFileVerbose](#)
  - [Files, 34](#)
- [CreateUsersArray](#)
  - [Build, 11](#)

- CreateUsersMap
  - Build, [11](#)
- Data, [29](#)
  - accepted\_userIDs, [30](#), [31](#)
  - ID\_MAX\_LENGTH, [29](#)
  - IP\_LENGTH, [30](#)
  - NAME\_MAX\_LENGTH, [29](#)
  - PASSWORD\_LENGTH, [30](#)
  - RECORD\_COUNT, [29](#)
  - userFullNames, [30](#), [31](#)
- DecryptString
  - Cipher, [14](#)
- default\_settings
  - Process, [58](#)
- dirty
  - ClientShared, [60](#)
- ecet4640-lab5, [1](#)
- EncryptString
  - Cipher, [13](#)
- end
  - ClientShared, [61](#)
  - ServerProperties, [67](#)
- Files, [31](#)
  - ADS\_DIR, [33](#)
  - CatFileToBuffer, [40](#)
  - CreateOrOpenFileVerbose, [34](#)
  - FileStatus, [34](#)
  - GetRandomFileNameFromDir, [38](#)
  - KEY\_FILE, [33](#)
  - LOCKFILE, [32](#)
  - NumberOfFilesInDirectory, [38](#)
  - ReadKeyIntoSettingsMap, [35](#)
  - ReadRegisteredFileIntoUsersMap, [36](#)
  - ReadSettingsFileIntoSettingsMap, [39](#)
  - REGISTERED\_FILE, [32](#)
  - SERVER\_SETTINGS\_FILE, [32](#)
  - UpdateRegisteredFileFromUsersMap, [37](#)
- FileStatus
  - Files, [34](#)
- GenerateCipher
  - Cipher, [13](#)
- GetRandomFileNameFromDir
  - Files, [38](#)
- ID\_MAX\_LENGTH
  - Data, [29](#)
- InitializeCipher
  - Process, [56](#)
  - Server, [49](#)
- InitializeServer
  - Server, [49](#)
- InitializeShared
  - Connection, [16](#)
- IP\_LENGTH
  - Data, [30](#)
- KEY\_FILE
  - Files, [33](#)
- LOCKFILE
  - Files, [32](#)
- Logfile, [41](#)
  - LogfileError, [41](#)
  - LogfileMessage, [42](#)
  - SetLogfileName, [41](#)
- LogfileError
  - Logfile, [41](#)
- LogfileMessage
  - Logfile, [42](#)
- Map, [43](#)
  - Map\_Delete, [46](#)
  - Map\_Get, [45](#)
  - Map\_Set, [45](#)
  - NewMap, [44](#)
- map, [64](#)
  - Map\_Delete
    - Map, [46](#)
  - Map\_Get
    - Map, [45](#)
- map\_result, [65](#)
- Map\_Set
  - Map, [45](#)
- max\_connections
  - ServerProperties, [66](#)
- MessageAndClose
  - Connection, [19](#)
- MessageOrClose
  - Connection, [18](#)
- mutex
  - ClientShared, [60](#)
- NAME\_MAX\_LENGTH
  - Data, [29](#)
- NewMap
  - Map, [44](#)
- NextAvailableConnection
  - Server, [48](#)
- NumberOfFilesInDirectory
  - Files, [38](#)
- PASSWORD\_LENGTH
  - Data, [30](#)
- port
  - ServerProperties, [65](#)
- printBlue
  - Util, [53](#)
- PrintCipher
  - Cipher, [13](#)
- printGreen
  - Util, [52](#)
- printRed
  - Util, [51](#)
- printYellow
  - Util, [53](#)

- Process, 56
  - active\_clients, 58
  - default\_settings, 58
  - InitializeCipher, 56
  - RunHeadless, 57
  - settings\_map, 58
  - users\_array, 58
  - users\_map, 58
- RandomFlag
  - Util, 55
- RandomFloat
  - Util, 55
- RandomInteger
  - Util, 54
- ReadKeyIntoSettingsMap
  - Files, 35
- ReadRegisteredFileIntoUsersMap
  - Files, 36
- ReadSettingsFileIntoSettingsMap
  - Files, 39
- receive\_buffer\_size
  - ClientShared, 61
  - ServerProperties, 66
- RECORD\_COUNT
  - Data, 29
- REGISTERED\_FILE
  - Files, 32
- RunHeadless
  - Process, 57
- send\_buffer\_size
  - ClientShared, 60
  - ServerProperties, 65
- Server, 46
  - CloseServer, 49
  - InitializeCipher, 49
  - InitializeServer, 49
  - NextAvailableConnection, 48
  - StartServer, 47
- SERVER\_SETTINGS\_FILE
  - Files, 32
- ServerProperties, 65
  - active\_connections, 66
  - backlog, 66
  - cipher, 67
  - end, 67
  - max\_connections, 66
  - port, 65
  - receive\_buffer\_size, 66
  - send\_buffer\_size, 65
  - socket\_id, 66
  - start, 67
  - time\_started, 66
- SetLogfileName
  - Logfile, 41
- settings\_map
  - Process, 58
- shutting\_down
  - ClientShared, 60
- socket
  - Connection, 62
- socket\_id
  - ServerProperties, 66
- src/server/Build.c, 69
- src/server/Build.h, 69
- src/server/Cipher.c, 70
- src/server/Cipher.h, 71
- src/server/colors.h, 71
- src/server/Connection.c, 71
- src/server/Connection.h, 77
- src/server/Data.c, 78
- src/server/Data.h, 78
- src/server/File.c, 79
- src/server/File.h, 82
- src/server/Logfile.c, 82
- src/server/Logfile.h, 84
- src/server/main.c, 84
- src/server/map.c, 84
- src/server/map.h, 86
- src/server/Process.c, 87
- src/server/Process.h, 90
- src/server/Server.c, 91
- src/server/Server.h, 93
- src/server/Util.c, 94
- src/server/Util.h, 95
- start
  - ClientShared, 61
  - ServerProperties, 67
- StartConnectionThread
  - Connection, 17
- StartServer
  - Server, 47
- StartUpdateThread
  - Connection, 16
- state
  - Connection, 63
- status
  - Connection, 62
- thread\_id
  - Connection, 63
- time\_connected
  - Connection, 63
- time\_started
  - ServerProperties, 66
- UpdateRegisteredFileFromUsersMap
  - Files, 37
- User, 67
- user
  - Connection, 63
- userFullNames
  - Data, 30, 31
- users
  - ClientShared, 60
- users\_array
  - Process, 58

users\_map  
    Process, [58](#)  
Util, [50](#)  
    COLOR\_BLUE, [51](#)  
    COLOR\_GREEN, [51](#)  
    COLOR\_RED, [51](#)  
    COLOR\_RESET, [51](#)  
    COLOR\_YELLOW, [51](#)  
    printBlue, [53](#)  
    printGreen, [52](#)  
    printRed, [51](#)  
    printYellow, [53](#)  
    RandomFlag, [55](#)  
    RandomFloat, [55](#)  
    RandomInteger, [54](#)