ecet4640-lab5

1.0

Generated by Doxygen 1.9.8

Chapter 1

ecet4640-lab5

1.1 Intro

This program starts a Ipv4 server that listens on a port for incomming 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.	RunHeadless()
stop	Stops an existing server process if it is running.	StopCommand()

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.

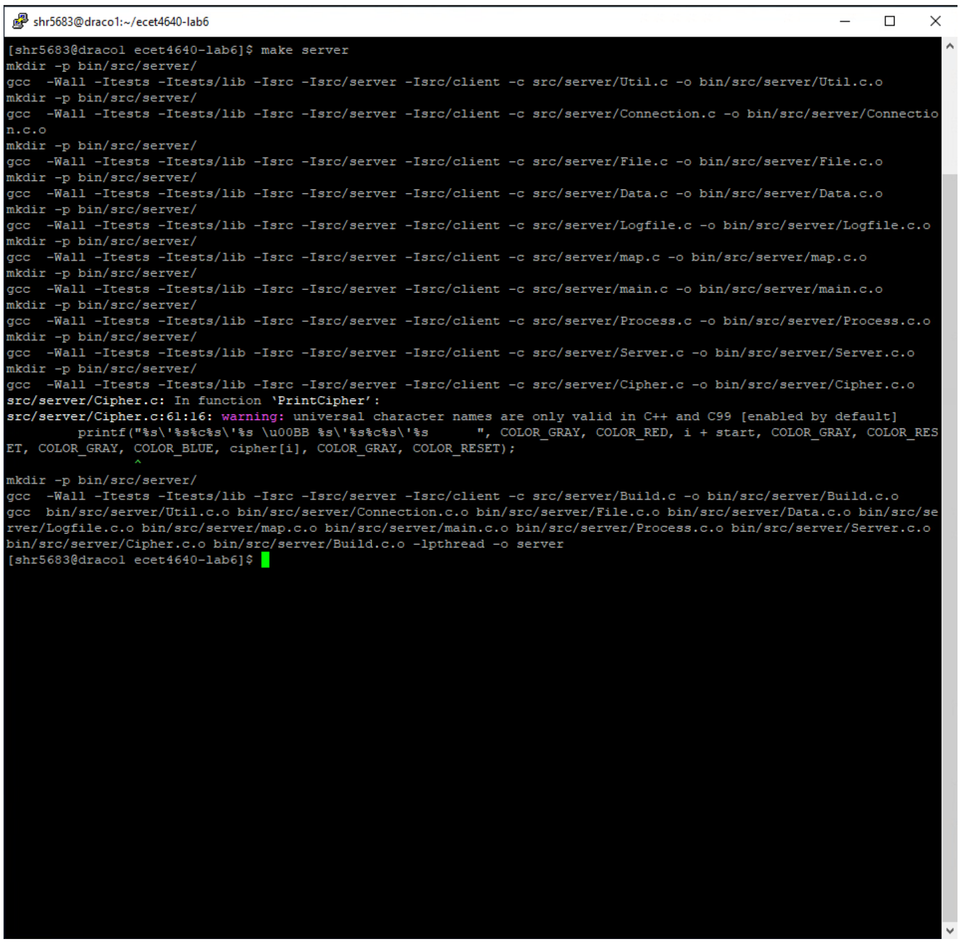
The first is for making and running the regular server process. Calling `make` executes this. It uses the files in `src/server` to generate the binary and runs it. This will run the binary after it is built, and the default command will cause it to run in the server. Executing `make server` will make 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/Connectio
n.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%c%s\'%s \u00BB %s\'%s%c%s\'%s       ", COLOR_GRAY, COLOR_RED, i + start, COLOR_GRAY, COLOR_RES
ET, COLOR_GRAY, COLOR_BLUE, cipher[i], 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/se
rver/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:

Chapter 4

Data Structure Index

4.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 5

File Index

5.1 File List

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

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

userIDs	An array of userIDs to set.
userNames	An array of userNames corresponding to the userIDs.
recordsCount	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

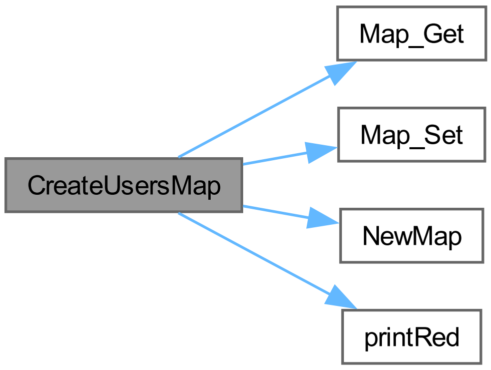
usersArray	The array used to build the user's map.
recordsCount	The number of records in the user's array.

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

cipher	The cipher to fill.
start	The character to start.
end	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

cipher	The cipher to print.
start	The character started on cipher
length	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

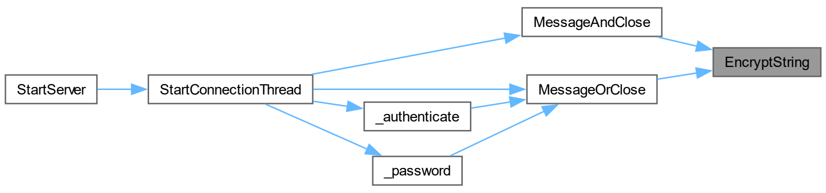
mutatates: encypts the string in place, destroying the original characters

Parameters

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

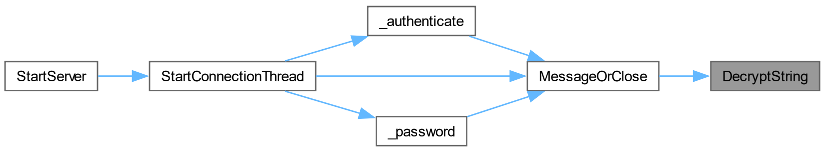
mutatates: encypts the string in place, destroying the original characters

Parameters

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

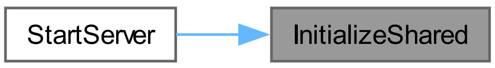
users_map	The map of User structs.

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

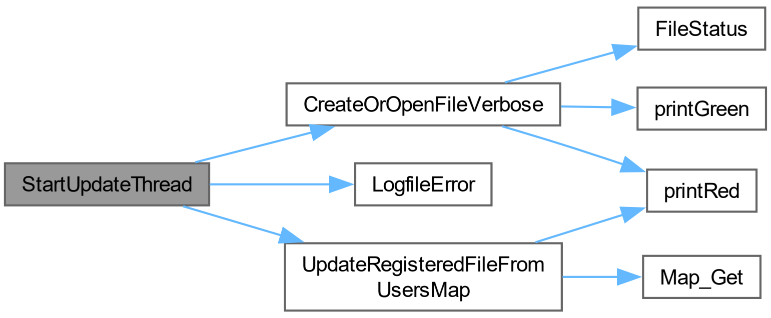
paramter	None.

Returns

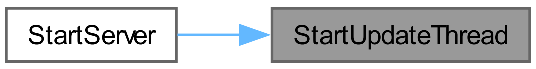
NULL

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

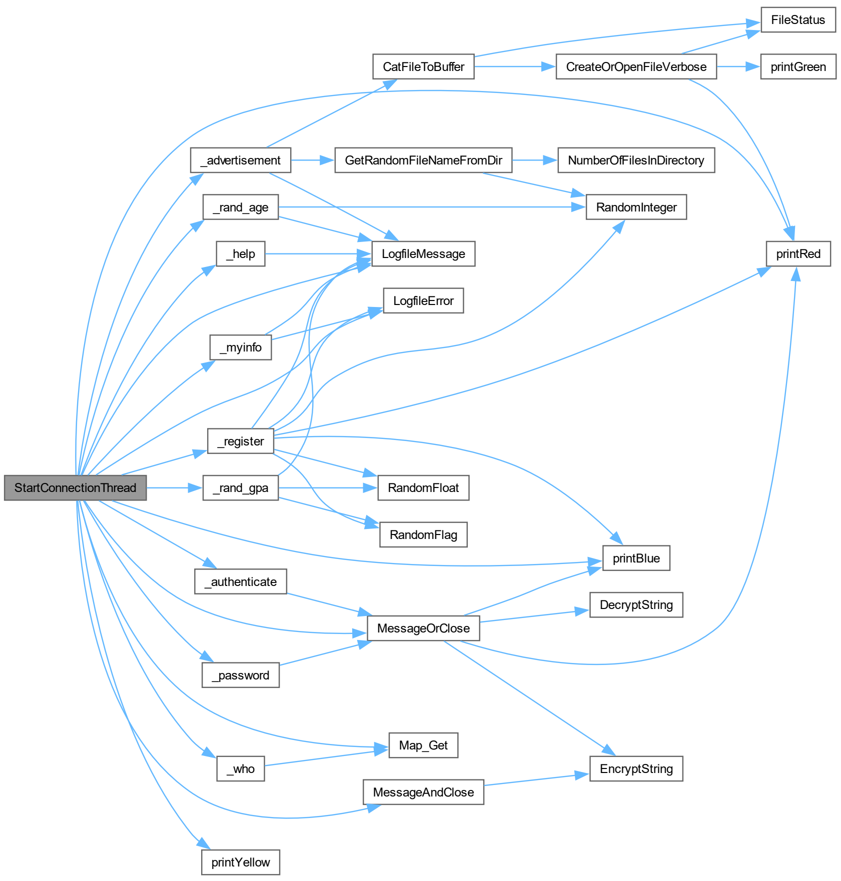
connection	A pointer to a Connection structure from the server's connections array.

Returns

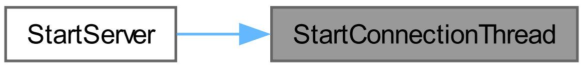
NULL

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

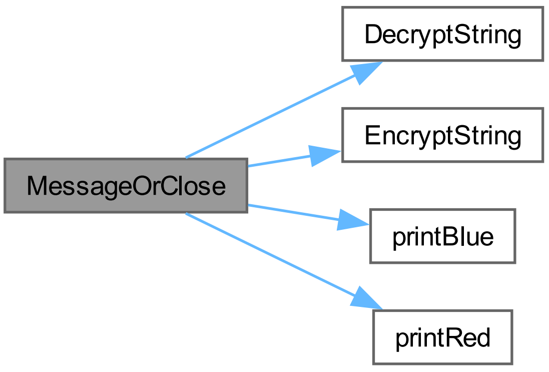
send_buffer	A message to send to the client.
receive_buffer	The message received by the client.
connection	The socket's Connection

Returns

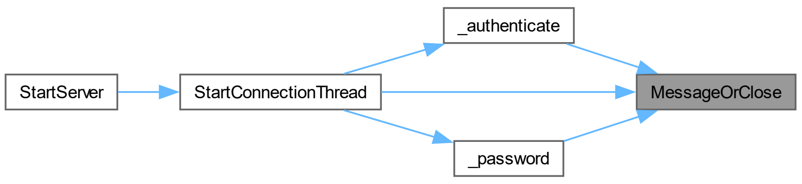
The number of bytes read into receive_buffer, or 0 if the connection closed.

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 )
```

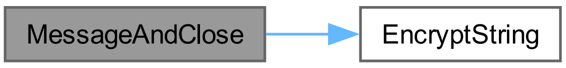
Sends send_buffer to the socket referenced by connection, then sets connection.active to 0.

Parameters

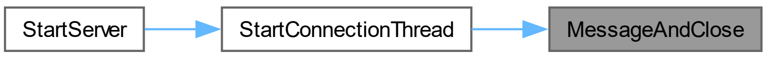
send_buffer	The send buffer. Should be shared.send_length in size.
connection	The socket's Connection.

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 )
```

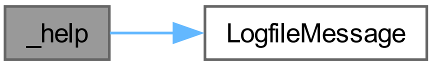
Returns the functions available to the user

Parameters

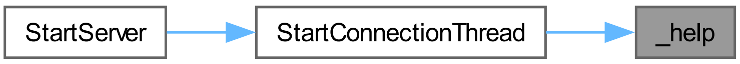
connection	connection the user is on
response	fills the response buffer with what to send to the client

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

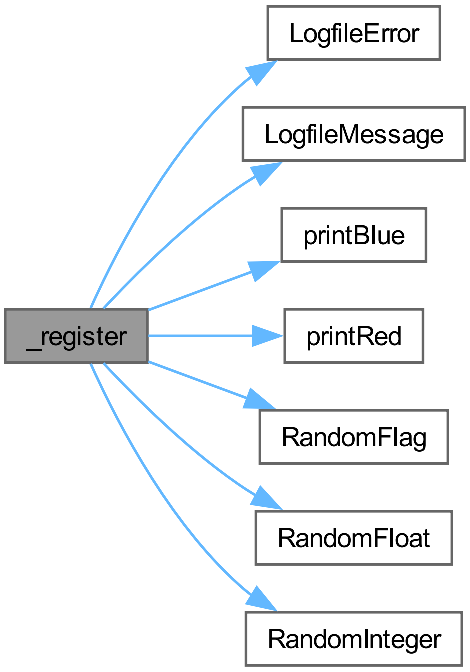
connection	connection the users is on
response	fills the response buffer with what to send to the client

Returns

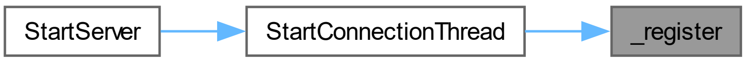
int 1 if successful, 0 if not

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

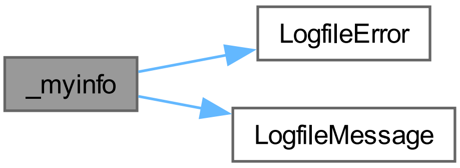
connection	connection the user is on
response	fills the response buffer with what to send to the client

Returns

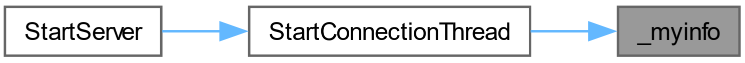
int 1 if successful, 0 if not

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 )
```

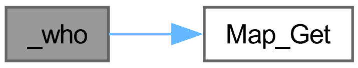
Sets response buffer to be a list a userIDs that are connected.

Parameters

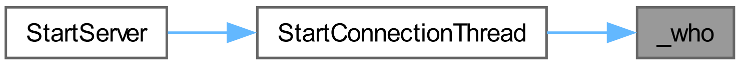
response	fills the response buffer with what to send to the client

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 )
```

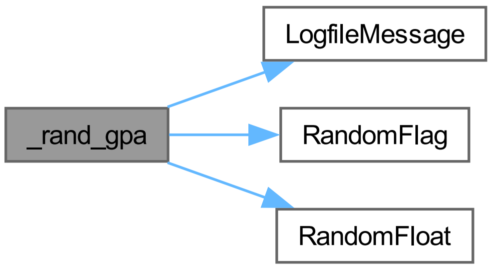
Randomly changes the gpa of the user

Parameters

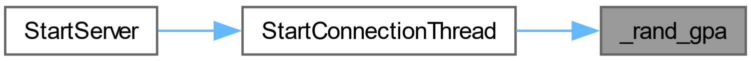
connection	connection the user is on
response	fills the response buffer with what to send to the client

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 )
```

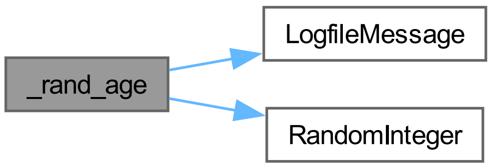
Randomly changes the age of the user

Parameters

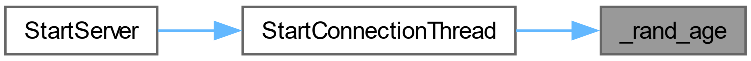
connection	connection the user is on
response	fills the response buffer with what to send to the client

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()

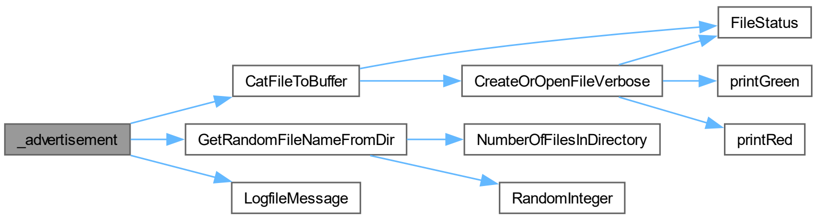
```
void _advertisement (
            Connection * connection,
            char * response )
```

Parameters

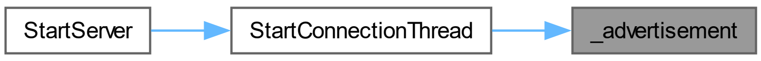
connection	connection the user is on
response	fills the response buffer with what to send to the client

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

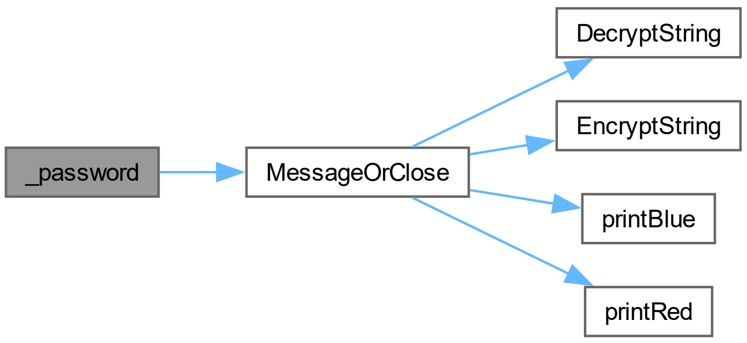
connection	connection the user is on

Returns

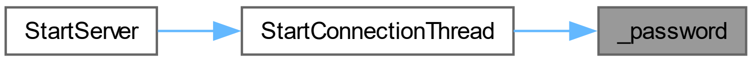
0 if valid password was entered, 1 if invalid

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

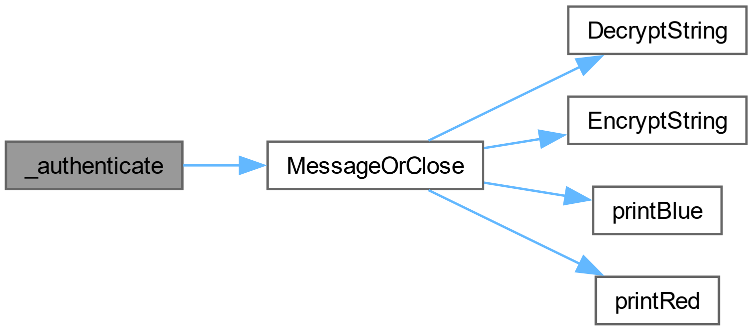
connection	connection the user is on

Returns

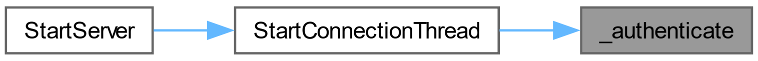
0 if valid password was entered, 1 if invalid

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 )
```

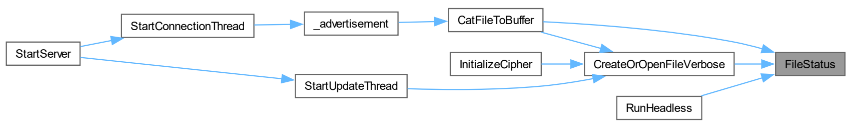
Determines if a file indicated by filename exists and is accesible by the user.

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

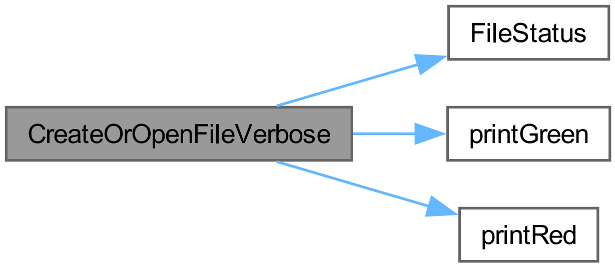
filename	The file name to create or open.
defaultContents	The contents to put in the file, if creating a default file, or NULL if no contents should be added.

Returns

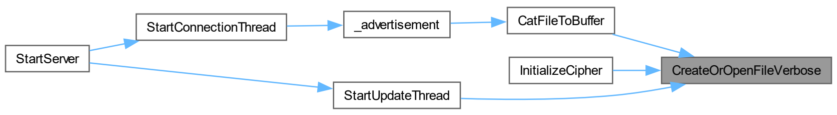
The opened file, or NULL on failure.

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

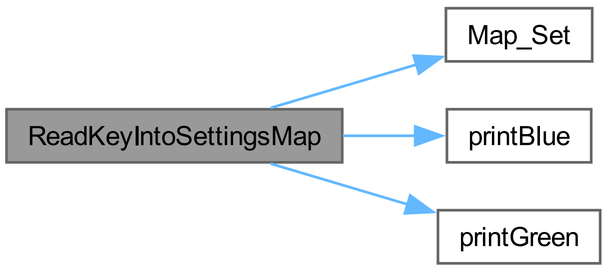
key_file	file to read
settings_map	map to put the settings into

Returns

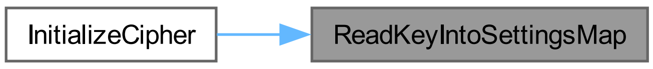
1 if successful, 0 otherwise

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

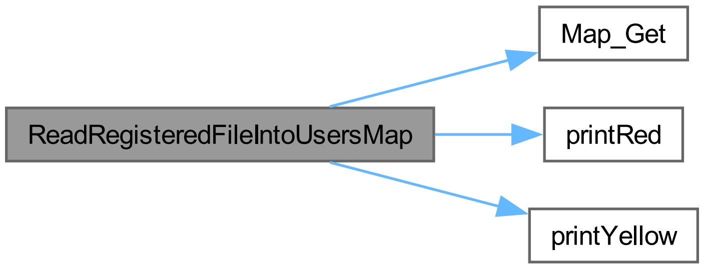
reg_file	The registered users file, open for reading.
users_map	The user's map to read into.

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()

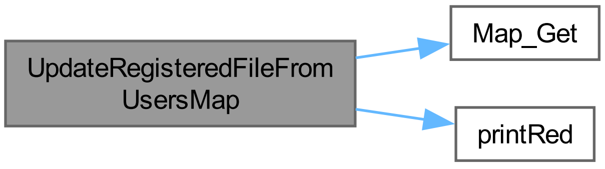
```
void UpdateRegisteredFileFromUsersMap (
            FILE * reg_file,
            map * users_map )
```

Parameters

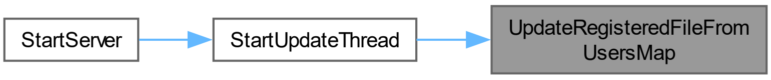
reg_file	file to update to
users_map	the map of users to use to update

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

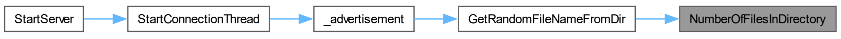
dir_name	directory to count files from

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()

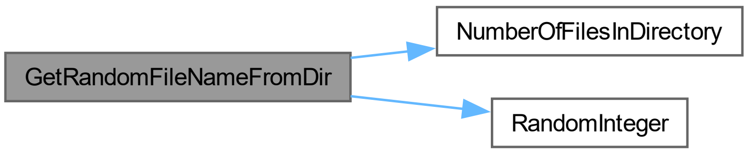
```
void GetRandomFileNameFromDir (
             char * dir_name,
             char * file_name )
```

Parameters

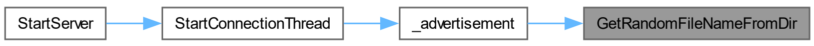
dir_name	name of the director to get a file name of
file_name	sets the name of the file into file_name

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

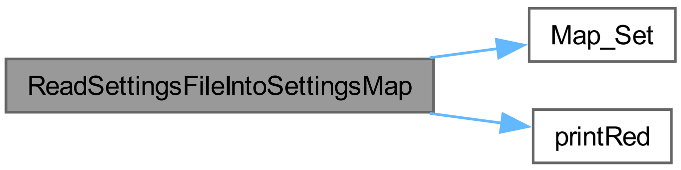
settings_file	The settings file, open for reading.
users_map	The settings_map to read into.

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()

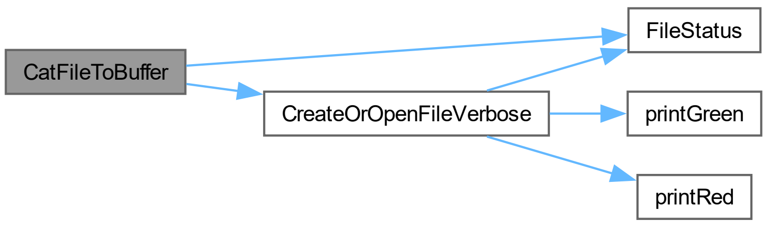
```
void CatFileToBuffer (
            char * file_name,
            char * buffer,
            size_t buffer_size )
```

Parameters

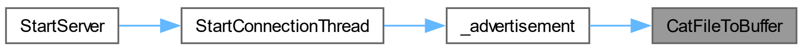
file_name	file to concatinate
buffer	string to copy it to
buffer_size	max size of buffer

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:

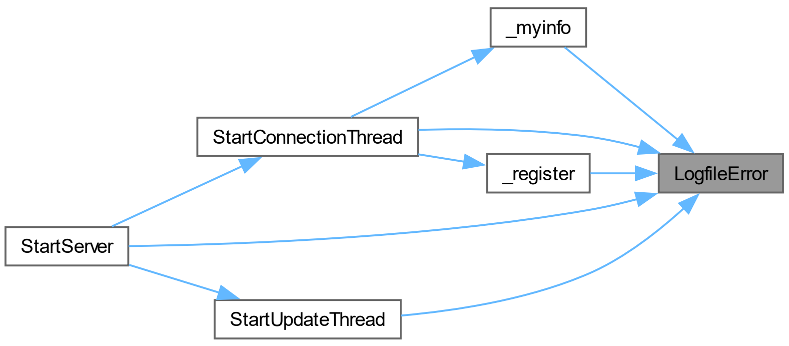
MM-DD-HH-MM-SS ERR <message>

Parameters

format	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:

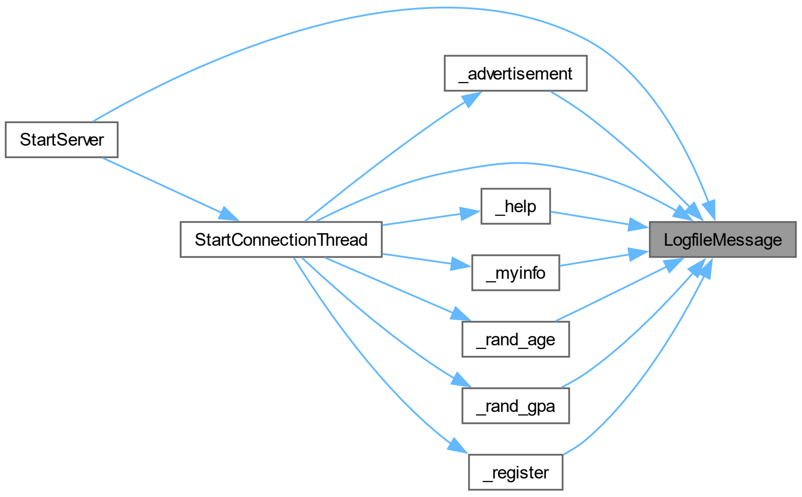
MM-DD-HH-MM-SS MSG <message>

Parameters

format	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 unsuccesful.
- 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

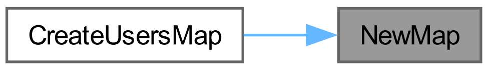
capacity	The estimated required capacity of the map.

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()

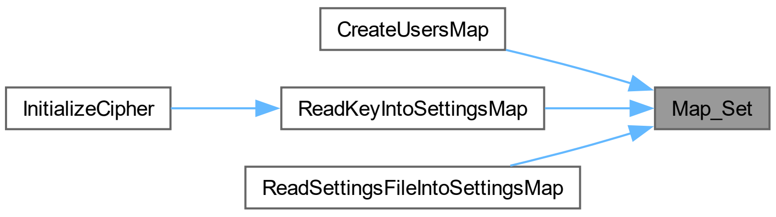
```
void Map_Set (
            map * a_map,
            char * key,
            void * value )
```

Parameters

map	The map to set a key in.
key	The key to use.
keylen	The length of the key.
value	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

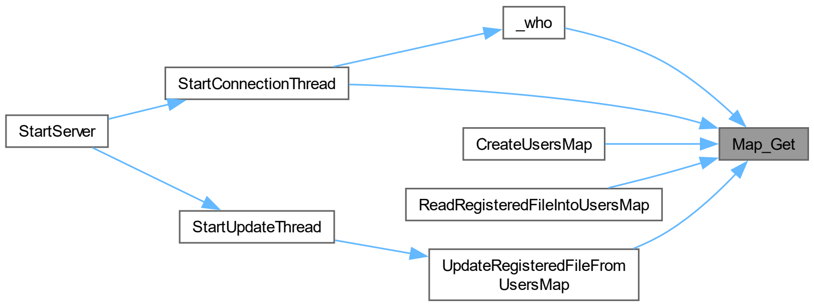
map	The map to retrieve from.
key	The key of the item.

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

map	The map to delete the key from.
key	The key to delete.
free←↩ _it	Whether to call free() 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 ()

  Initilizes 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

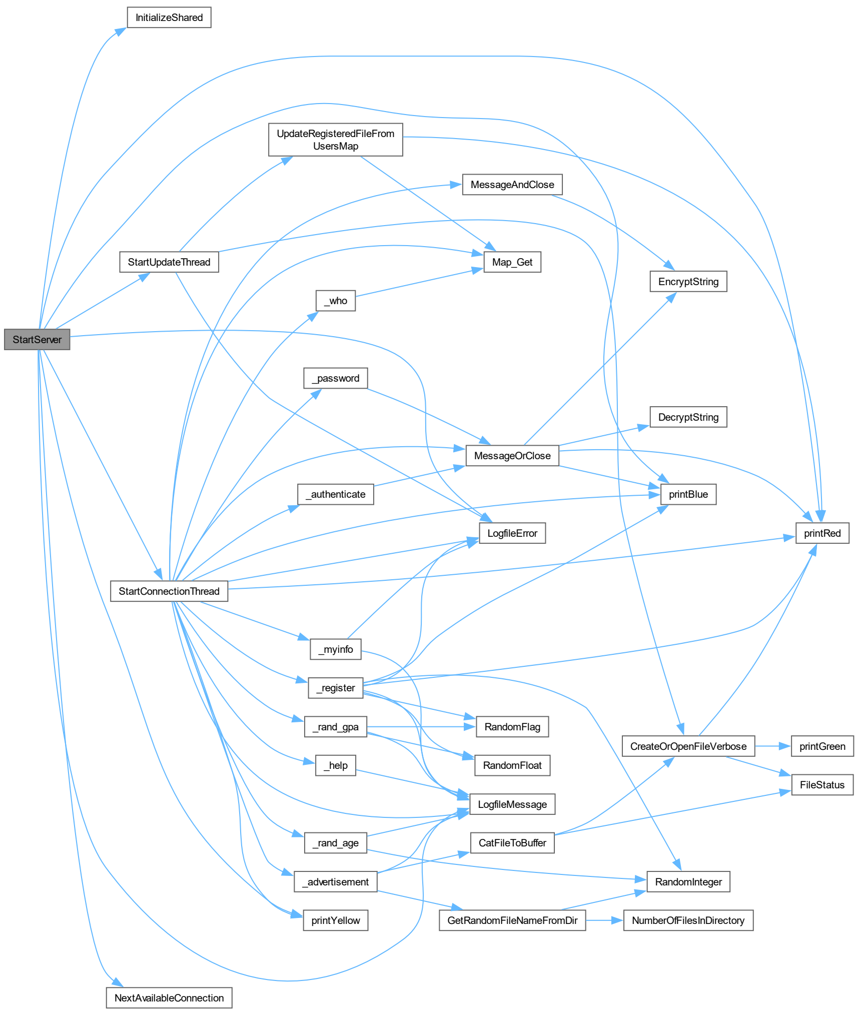
users_map	The user's map.

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 ( )
```

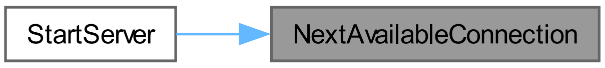
Iterates through the Connections array until it finds one whose 'active' field is false and returns it. If it iterates through the array and fails to find a connection, it returns NULL.

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 of 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

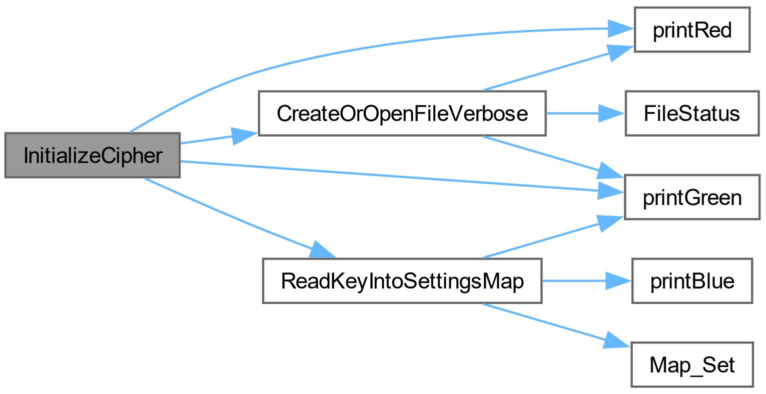
Initializes the Cipher.

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,
              ...  )
```

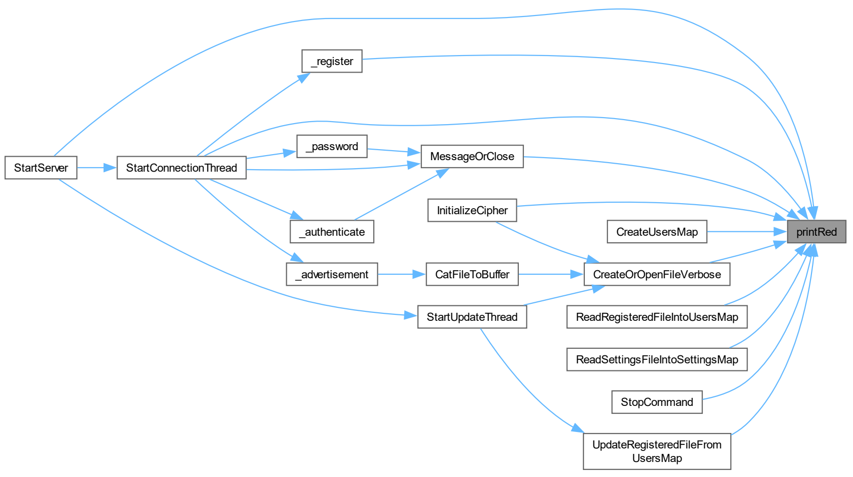
Prints to the console in red.

Parameters

format	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,
             ...  )
```

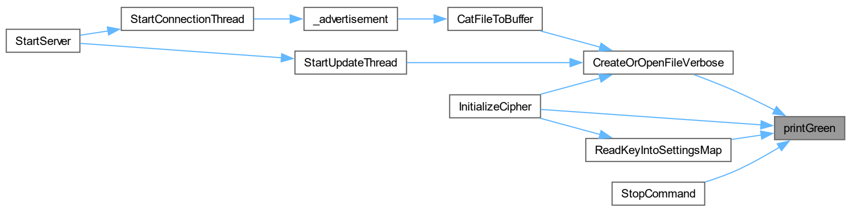
Prints to the console in green.

Parameters

format	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,
              ... )
```

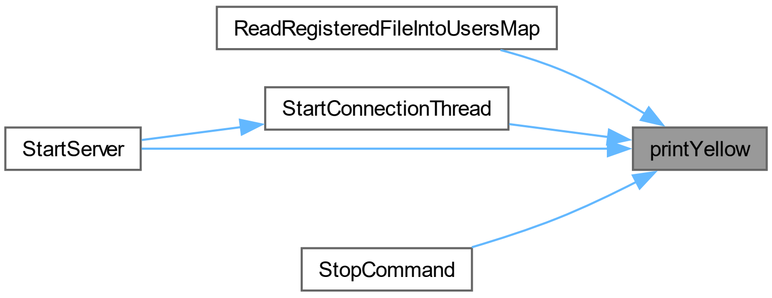
Prints to the console in yellow.

Parameters

format	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,
              ... )
```

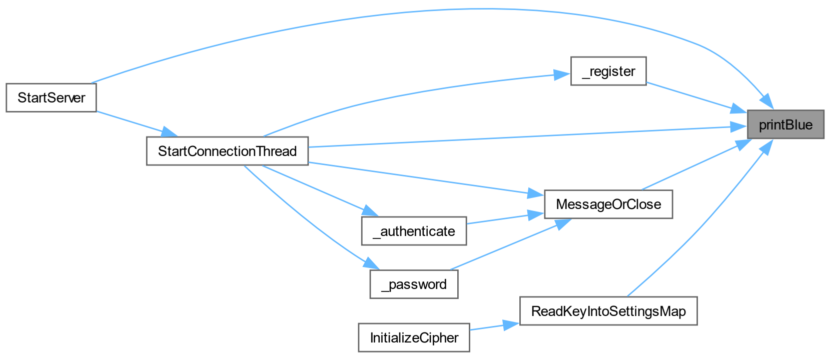
Prints to the console in blue.

Parameters

format	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

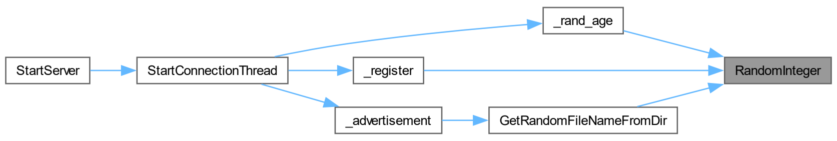
min	The minimum, inclusive.
max	The maximum, inclusive.

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

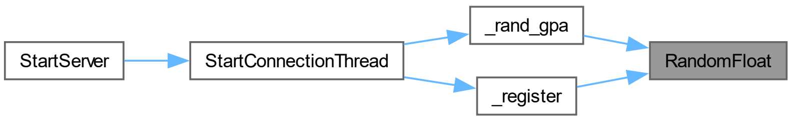
min	The minimum, inclusive.
max	The maximum, inclusive.

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

percentage_chance	The chance to return 1.

Note

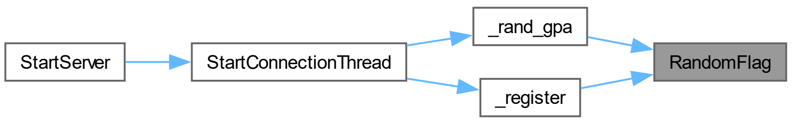
If percentage_chance > 1, this will always return true.

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 ( )
```

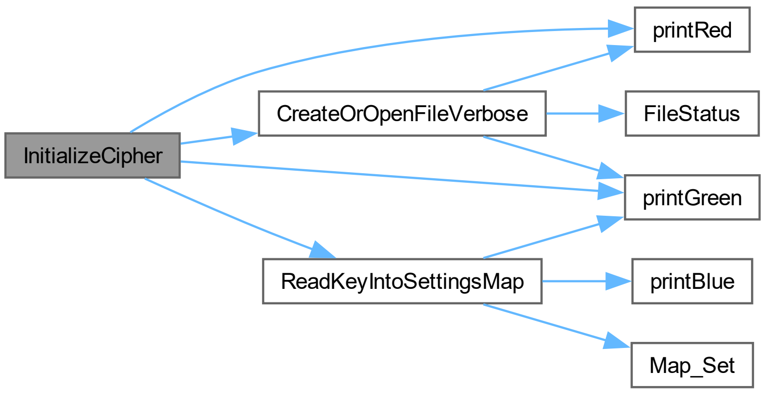
Initializes the Cipher.

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 )
```

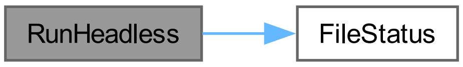
Uses nohup `./{processName} run` to run the process headlessly.

Parameters

processName	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.

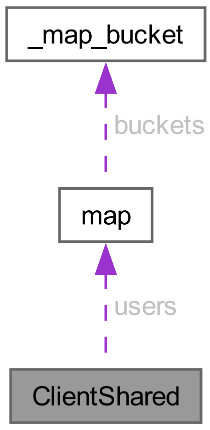
The documentation for this struct was generated from the following file:

- 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.

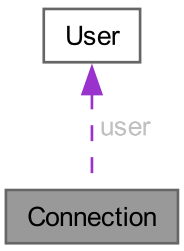
The documentation for this struct was generated from the following file:

- 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:

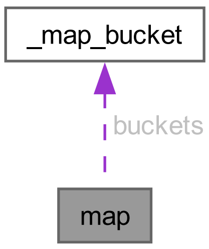
- src/server/Connection.h

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 substituion 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

```c

#include <stdlib.h>
#include <string.h>
#include "Build.h"
#include "Util.h"

User * CreateUsersArray(char ** userIDs, char ** userNames, int recordsCount)
{
    size_t uarr_size = sizeof(User) * recordsCount;
    User * uarr = malloc(uarr_size);
    memset(uarr, 0, uarr_size);
    int i;
    for(i = 0; i < recordsCount; i++)
    {
        strcpy(uarr[i].id, userIDs[i]);
        strcpy(uarr[i].name, userNames[i]);
    }
    return uarr;
}

map * CreateUsersMap(User * usersArray, int recordsCount)
{
    map * umap = NewMap(recordsCount * 3);
    int i;
    for(i = 0; i < recordsCount; i++) {
        Map_Set(umap, usersArray[i].id, &usersArray[i]);
        map_result mr = Map_Get(umap, usersArray[i].id);
        if(!mr.found) {
            printRed("Map failed on user: %s... your program may have issues.\n", usersArray[i].id);
        }
    }
    return umap;
}


```

8.2 Build.h

```c
#ifndef Build_h
#define Build_h
#include "Data.h"
#include "map.h"

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

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



#endif
```

8.3 Cipher.c

```c

/*
    Class: ECET 4640-002
    Assignment: Lab Assignment 3
    Authors: Christian Messmer, Karl Miller, Paul Shriner

    Cipher.c: Functions used for generating the cipher, printing it out, and encrypting a string.
*/

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
#include "colors.h"
#include "Util.h"

void FillArraySequential(char *array, char start, char end)
{
    //char length = end - start + 1;
    char i;
    for (i = start; i <= end; i++)
    {
        array[i - start] = i;
    }
}

// See Cipher.h for header comments
void GenerateCipher(char *cipher, char start, char end)
{
    time_t t;
    srand((unsigned)time(&t));

    FillArraySequential(cipher, start, end);
    int length = end - start + 1;
    int hold, swap_index, i;
    for (i = 0; i < length; i++)
    {
        swap_index = rand() % length;
        hold = cipher[swap_index];
        cipher[swap_index] = cipher[i];
        cipher[i] = hold;
    }
}

// See Cipher.h for header comments
void PrintCipher(char *cipher, char start, char length)
{
    printf("\nCipher:\n");
    int i;
    for (i = 0; i < length; i++)
    {
        printf("%s\'%s%c%s\'%s \u00BB %s\'%s%c%s\'%s       ", COLOR_GRAY, COLOR_RED, i + start,
COLOR_GRAY, COLOR_RESET, COLOR_GRAY, COLOR_BLUE, cipher[i], COLOR_GRAY, COLOR_RESET);
        if ((i + 1) % 5 == 0)
        {
            printf("\n");
        }
    }
    printf("\n");
}

// See Cipher.h for header comments
void EncryptString(char *string, int length, char *cipher, char start, char end)
{

    //char cipher_l = end - start + 1;
    int i;
    for (i = 0; i < length; i++)
    {
        // printf("Encrypting string[%d] , was %c\n", i, string[i]);
        // printf("String in range between %c and %c?\n", start, end);
        if (!(string[i] - start > end || string[i] < start))
        {
            // printf("String in range\n");
            // printf("String[%d] - %d(start) is: %d\n", i, start, string[i]-start);
            string[i] = cipher[string[i] - start];
            // printf("String[%d] is now %c\n", i, string[i]);
        }
    }
}

void DecryptString(char* string, int length, char* cipher, char start, char end) {
    //char cipher_l = end - start + 1;

    int i;
    for(i = 0; i < length; i++) {
        if(!(string[i] - start > end || string[i] < start)) {
            char* c = strchr(cipher, string[i]);
            string[i] = c - cipher + start;
        }
    }
}


```

8.4 Cipher.h

```
#ifndef Cipher_h
#define Cipher_h
/*
    Class: ECET 4640-002
    Assignment: Lab Assignment 3
    Authors: Christian Messmer, Karl Miller, Paul Shriner

    Cipher.h: Function prototypes for Cipher.c
*/

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

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

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

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

#endif
```

8.5 colors.h

```
/*
    Class: ECET 4640-002
    Assignment: Lab Assignment 3
    Authors: Christian Messmer, Karl Miller, Paul Shriner

    colors.h: Define color macros for use with printing text to the console

    Acknowledgements/Credits:
        1. https://www.man7.org/linux/man-pages/man4/console_codes.4.html
*/

#ifndef colors_h
#define colors_h
/*
    Karl's magic color macros.

    These use Virtual Terminal escape sequences to trigger color changes on the console when printed.

    See 1 in Acknowledgements/Credits for more information.

*/
#define COLOR_RED "\e[38;2;255;75;75m"
#define COLOR_BLUE "\e[38;2;0;240;240m"
#define COLOR_GREEN "\e[38;2;0;240;0m"
#define COLOR_YELLOW "\e[38;2;255;255;0m"
#define COLOR_GRAY "\e[38;2;224;224;224m"
#define COLOR_BOLD "\e[1m"
#define COLOR_RESET "\e[0m"

#endif
```

8.6 Connection.c

```

#include "Connection.h"
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <string.h>
#include <arpa/inet.h>
#include <unistd.h>
```

```
#include "Util.h"
#include "Data.h"
#include "File.h"
#include "Logfile.h"
#include "Cipher.h"

ClientShared shared;

ClientShared * InitializeShared(map * users_map, size_t send_buffer_size, size_t receive_buffer_size,
      char* cipher, char start, char end)
{
    shared.users = users_map;
    shared.dirty = 0;
    shared.shutting_down = 0;
    shared.send_buffer_size = send_buffer_size;
    shared.receive_buffer_size = receive_buffer_size;
    shared.cipher = cipher;
    shared.start = start;
    shared.end = end;
    pthread_mutex_init(&(shared.mutex), NULL);
    return &shared;
}

void * StartUpdateThread(void * parameter)
{
    while(shared.shutting_down == 0) {
        if(shared.dirty) {
            pthread_mutex_lock(&(shared.mutex));
            shared.dirty = 0;
            FILE * reg_file = CreateOrOpenFileVerbose(REGISTERED_FILE, NULL);
            if(reg_file != NULL) {
                UpdateRegisteredFileFromUsersMap(reg_file, shared.users);
                fclose(reg_file);
            } else {
                LogfileError("FAILED TO OPEN REGISTERED FILE - NO DATA WILL BE UPDATED");
                shared.dirty = 1;
            }
            pthread_mutex_unlock(&(shared.mutex));
        }
        sleep(1);

    }
    return NULL;
}

void * StartConnectionThread(void * p_connection)
{
    Connection * connection = (Connection *) p_connection;
    connection->state = ClientState_ENTRY;
    connection->user = NULL;
    time(&(connection->time_connected));
    // allocate send and receive buffers.
    char * send_buffer = malloc(shared.send_buffer_size);
    char * receive_buffer = malloc(shared.receive_buffer_size);
    map_result result;

    // ask for their user ID initially, or disconnect them.
    strcpy(send_buffer, "<Message>Welcome. Please send your user ID.");
    MessageOrClose(send_buffer, receive_buffer, connection);
    if(connection->status == ConnectionStatus_ACTIVE) {
        result = Map_Get(shared.users, receive_buffer);
        if(!result.found)
        {
            printYellow("Unauthorized access attempt by %s with name '%s'.\n",
      inet_ntoa(connection->address.sin_addr), receive_buffer);
            strcpy(send_buffer, "<Error>No such user");
            MessageAndClose(send_buffer, connection);
            LogfileError("Unauthorized access attempt by unknown user %s from %s.", receive_buffer,
      inet_ntoa(connection->address.sin_addr));
            // send a one-way message to the client
        } else {
            User * user = (User *) result.data;
            if(user->connected) {
                printYellow("User %s attempted to double connect from IP %s.\n", user->id,
      inet_ntoa(connection->address.sin_addr));
                strcpy(send_buffer, "<Error>You are already connected.");
                LogfileError("User %s attempted to double connect from IP %s.\n", user->id,
      inet_ntoa(connection->address.sin_addr));
                MessageAndClose(send_buffer, connection);
                // send the other connected user an informative message?
            } else {
                connection->user = user;
                connection->user->connected = 1;
                strcpy(connection->user->ip, inet_ntoa(connection->address.sin_addr));
                if(connection->user->registered) {
                    connection->state = ClientState_UNAUTHENTICATED;
                    LogfileMessage("User %s is attempting a login from ip %s.",
connection->user->name, inet_ntoa(connection->address.sin_addr));
                } else {
                    connection->state = ClientState_ACCESSING;
                }
            }
        }
    }

    if(connection->state == ClientState_ACCESSING && connection->status == ConnectionStatus_ACTIVE) {
        strcpy(send_buffer, "<Message>Say something, unregistered user!");
    } else if (connection->state == ClientState_UNAUTHENTICATED && connection->status ==
ConnectionStatus_ACTIVE) {
        strcpy(send_buffer, "<Message>Say something, registered user (logged out)!");
    } else if (connection->state == ClientState_REGISTERED && connection->status ==
ConnectionStatus_ACTIVE) {
        strcpy(send_buffer, "<Message>Say something, registered user (logged in)!");
    }

    while(connection->status == ConnectionStatus_ACTIVE)
    {
        if(connection->state == ClientState_ACCESSING) {
            MessageOrClose(send_buffer, receive_buffer, connection);
            if (strcmp(receive_buffer, "help") == 0) {
                _help(connection, send_buffer);
            } else if (strcmp(receive_buffer, "exit") == 0) {
                strcpy(send_buffer, "<Message>Goodbye.");
                MessageAndClose(send_buffer, connection);
            } else if (strcmp(receive_buffer, "register") == 0) {
                if (_password(connection) == 0) {
                    _register(connection, send_buffer);
                    LogfileMessage("User %s registered.", connection->user->id);
                } else {
                    strcpy(send_buffer, "<Error>Invalid password entered, cannot register");
                    LogfileError("User %s attempted to register with an invalid password.",
connection->user->id);
                }
            } else {
                strcpy (send_buffer, "<Error>Invalid command, use 'help' for list of commands");
            }
        } else if (connection -> state == ClientState_UNAUTHENTICATED) {
            MessageOrClose(send_buffer, receive_buffer, connection);
            if (strcmp(receive_buffer, "help") == 0) {
                _help(connection, send_buffer);
            } else if (strcmp(receive_buffer, "exit") == 0) {
                strcpy(send_buffer, "<Message>Goodbye.");
                MessageAndClose(send_buffer, connection);
            } else if (strcmp(receive_buffer, "login") == 0) {
                if (_authenticate(connection) == 0) {
                    strcpy(send_buffer, "<Message>You have logged in!");
                    connection->user->lastConnection = time(NULL);
                    printBlue("Setting connection time_connected to: %d\n",
connection->time_connected);
                    shared.dirty = 1;
                    LogfileMessage("User %s logged in.", connection->user->id);
                } else {
                    strcpy(send_buffer, "<Message>Login failed!");
                    LogfileError("Failed login attempt for user %s.", connection->user->id);
                }
            } else {
                strcpy (send_buffer, "<Error>Invalid command, use 'help' for list of commands");
            }
        } else if(connection->state == ClientState_REGISTERED) {
            MessageOrClose(send_buffer, receive_buffer, connection);
            if (strcmp(receive_buffer, "help") == 0) {
                _help(connection, send_buffer);
            } else if (strcmp(receive_buffer, "exit") == 0) {
                strcpy(send_buffer, "<Message>Goodbye.");
                MessageAndClose(send_buffer, connection);
            } else if (strcmp(receive_buffer, "myinfo") == 0) {
                _myinfo(connection, send_buffer);
            } else if (strcmp(receive_buffer, "who") == 0) {
                _who(send_buffer);
            } else if (strcmp(receive_buffer, "random-gpa") == 0) {
                _rand_gpa(connection, send_buffer);
            } else if (strcmp(receive_buffer, "random-age") == 0) {
                _rand_age(connection, send_buffer);
            } else if (strcmp(receive_buffer, "advertisement") == 0) {
                _advertisement(connection, send_buffer);
            } else if (strcmp(receive_buffer, "change-password") == 0) {
                if (_password(connection) == 0) {
                    strcpy(send_buffer, "<Message>Password has been changed");
                } else {
                    strcpy(send_buffer, "<Error>Invalid password entered, no action taken");
                }
            } else {
                strcpy(send_buffer, "<Error>Invalid command, use 'help' for list of commands");
            }
            // call a function for processing this state.
        } else {
            printRed("Client entered invalid state. Disconnecting. \n");
            strcpy(send_buffer, "<Error>You entered an invalid state!");
            MessageAndClose(send_buffer, connection);
            connection->status = ConnectionStatus_CLOSING;
        }
    }

    if(connection->user != NULL) {
        connection->user->connected = 0;
        printf("User %s from ip %s disconnected.\n", connection->user->id, connection->user->ip);
        LogfileMessage("User %s from ip %s disconnected.", connection->user->id,
connection->user->ip);
    } else {
        printf("Ip %s disconnected.\n", inet_ntoa(connection->address.sin_addr));
        LogfileMessage("Ip %s disconnected.\n", inet_ntoa(connection->address.sin_addr));
    }


    free(send_buffer);
    free(receive_buffer);
    close(connection->socket);
    if(connection->user != NULL) {
        connection->user->connected = 0;
    }
    connection->status = ConnectionStatus_CLOSED;
    return NULL;
}


int MessageOrClose(char * send_buffer, char * receive_buffer, Connection * connection) {
    receive_buffer[0] = '\0';
    EncryptString(send_buffer, strlen(send_buffer), shared.cipher, shared.start, shared.end);
    memset(receive_buffer, 0, shared.receive_buffer_size);
    if(send(connection->socket, send_buffer, shared.send_buffer_size, 0) < 0) {
        printRed("Failed to send message to %s. Disconnecting.\n",
inet_ntoa(connection->address.sin_addr));
        perror("Error:");
        connection->status = ConnectionStatus_CLOSING;
        return 0;
    }
    int received_size = recv(connection->socket, receive_buffer, shared.receive_buffer_size, 0);
    if(received_size < 0) {
        printRed("Failed to receive message from %s. Disconnecting.\n",
inet_ntoa(connection->address.sin_addr));
        perror("Error: ");
        connection->status = ConnectionStatus_CLOSING;
        return 0;
    }
    if(received_size == 0 ) {
        printBlue("%s disconnected.\n", inet_ntoa(connection->address.sin_addr));
        connection->status = ConnectionStatus_CLOSING;
        return 0;
    }
    send_buffer[0] = '\0';
    // memset(send_buffer, 0, shared.send_buffer_size);

    DecryptString(receive_buffer, strlen(receive_buffer), shared.cipher, shared.start, shared.end);
    return received_size;
}



void MessageAndClose(char * send_buffer, Connection * connection) {
    strcat(send_buffer, "<Disconnect>");
    EncryptString(send_buffer, strlen(send_buffer), shared.cipher, shared.start, shared.end);
    send(connection->socket, send_buffer, shared.send_buffer_size, 0);
    connection->status = ConnectionStatus_CLOSING;
    if (connection -> user != NULL) {
        connection->user->connected = 0;
    }
}

void _help(Connection* connection, char* response) {
    if (connection -> state == ClientState_UNAUTHENTICATED) {
        strcpy(response, "<Message>help - get a list of available commands\n");
        strcat(response, "login - login to the server\n");
        strcat(response, "exit - disconnect from the server");
        LogfileMessage("%s asked for help.", inet_ntoa(connection->address.sin_addr));
    } else if(connection->state != ClientState_REGISTERED) {
        strcpy(response, "<Message>help - get a list of available commands\n");
        strcat(response, "register - register your user\n");
        strcat(response, "exit - disconnect from the server");
        LogfileMessage("%s asked for help.", inet_ntoa(connection->address.sin_addr));
    } else if(connection->state == ClientState_REGISTERED) {
        strcpy(response, "<Message>help - get a list of available commands\n");
        strcat(response, "exit - disconnect from the server\n");
        strcat(response, "who - get a list of online users\n");
        strcat(response, "random-gpa - set your gpa to a new random value\n");
        strcat(response, "random-age - set your age to a new random value\n");
        strcat(response, "advertisement - get a colorful advertisement\n");
        strcat(response, "myinfo - get info about yourself\n");
        strcat(response, "change-password - change your current password\n");
        LogfileMessage("%s asked for help.", connection->user->name);
    }
}

int _register(Connection * connection, char* response) {
    if(connection->user->registered) {
        strcpy(response, "<Error>");
        strcat(response, connection->user->id);
        strcat(response, " is already registered.");

        printRed("%s from ip %s has attempted to register a second time.\n", connection->user->id,
inet_ntoa(connection->address.sin_addr));
        LogfileError("%s from ip %s has attempted to register a second time.\n", connection->user->id,
inet_ntoa(connection->address.sin_addr));
        return 0;
    }

    pthread_mutex_lock(&(shared.mutex));

    connection->user->registered = 1;

    connection->user->age = RandomInteger(18, 22);

    if(RandomFlag(.4)) {
        connection->user->gpa = 4.0;
    } else {
        connection->user->gpa = RandomFloat(2.5, 4);
    }

    connection->state = ClientState_REGISTERED;

    LogfileMessage("%s registered from ip %s.", connection->user->id,
inet_ntoa(connection->address.sin_addr));
    printBlue("%s registered.\n", connection->user->id);

    shared.dirty = 1;
    pthread_mutex_unlock(&(shared.mutex));

    strcpy(response, "<Message>You have been registered ");
    strcat(response, connection->user->name);

    return 1;
}

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

    if (!(connection->user->registered)) {
        strcpy(response, "<Error>");
        strcat(response, connection->user->id);
        strcat(response, " is not registered.");

        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));

        return 1;
    }

    //Referenced snprintf from https://cplusplus.com/reference/cstdio/snprintf/
    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));

    printf("%s viewed their information.\n", connection->user->id);
    LogfileMessage("%s viewed their information.", connection->user->name);

    return 0;
}

void _who(char * response) {
    int i;
    for(i = 0; i < RECORD_COUNT; i++) {
        map_result result = Map_Get(shared.users, accepted_userIDs[i]);
        if(result.found) {
            User* user = (User *) result.data;

            if(user->connected) {
                strcat(response, "<OnlineUser>");
                strcat(response, user->id);
            }
        }
    }
}

void _rand_gpa(Connection* connection, char* response) {
    char gpa_str[5];
    pthread_mutex_lock(&(shared.mutex));
    if(RandomFlag(.4)) {
        connection->user->gpa = 4.0;
    } else {
        connection->user->gpa = RandomFloat(2.2, 4.0);
    }
    shared.dirty = 1;
    pthread_mutex_unlock(&(shared.mutex));
    sprintf(gpa_str, "%.2f", connection->user->gpa);
    strcat(response, "<User.GPA>");
    strcat(response, gpa_str);
    LogfileMessage("%s randomized their gpa.", connection->user->name);
}

void _rand_age(Connection* connection, char * response) {
    char age_str[5];
    pthread_mutex_lock(&(shared.mutex));
    connection->user->age = RandomInteger(18, 22);
    shared.dirty = 1;
    pthread_mutex_unlock(&(shared.mutex));
    sprintf(age_str, "%d", connection->user->age);
    strcat(response, "<User.Age>");
    strcat(response, age_str);
    LogfileMessage("%s randomized their age.", connection->user->name);
}

void _advertisement(Connection * connection, char * response) {
    char filename[FILENAME_MAX];

    GetRandomFileNameFromDir(ADS_DIR, filename);

    char* filepath = malloc(FILENAME_MAX + sizeof(ADS_DIR));
    strcpy(filepath, ADS_DIR);
    strcat(filepath, "/");
    strcat(filepath, filename);

    strcat(response, "<Message>");

    LogfileMessage("User %s viewed advertisement %s.", connection->user->name, filepath);
    CatFileToBuffer(filepath, response, shared.send_buffer_size);

    free(filepath); //always free malloced strings to prevent mem leaks!
}

int _password (Connection* connection) {
    // Create send/rcv buffers, password buffer
    char * send_buffer = malloc(shared.send_buffer_size);
    char * receive_buffer1 = malloc(shared.receive_buffer_size);
    char * receive_buffer2 = malloc(shared.receive_buffer_size);
    char * password = malloc(PASSWORD_LENGTH);

    // Prompt user for password two times
    strcpy(send_buffer, "<Message>Enter a password");
    MessageOrClose(send_buffer, receive_buffer1, connection);
    strcpy(send_buffer, "<Message>Enter the same password");
    MessageOrClose(send_buffer, receive_buffer2, connection);

    // Check if passwords match and is within PASSWORD_LENGTH
    if ((strcmp(receive_buffer1, receive_buffer2) != 0) || (strlen(receive_buffer1) >
PASSWORD_LENGTH)) {
        free(send_buffer);
        free(receive_buffer1);
        free(receive_buffer2);
        free(password);
        return 1;
    }

    // If so, fill password with buffer up to PASSWORD_LENGTH
    int i = 0;
    for (i = 0; i < PASSWORD_LENGTH; ++i) {
        password[i] = receive_buffer1[i];
    }

    // Set user’s password
    pthread_mutex_lock(&(shared.mutex));
    strcpy(connection -> user -> password, password);
    shared.dirty = 1;
    pthread_mutex_unlock(&(shared.mutex));

    free(send_buffer);
    free(receive_buffer1);
    free(receive_buffer2);
```

```c
    free(password);
    return 0;
}

int _authenticate (Connection* connection) {
    // Create send/rcv buffers
    char * send_buffer = malloc(shared.send_buffer_size);
    char * receive_buffer = malloc(shared.receive_buffer_size);

    // Prompt for password
    strcpy(send_buffer, "<Message>Enter your password");
    MessageOrClose(send_buffer, receive_buffer, connection);

    // Check if password matches current user's password
    if (strcmp(receive_buffer, connection -> user -> password) != 0) {
        free(send_buffer);
        free(receive_buffer);
        return 1;
    }

    // If so, user is now authenticated
    pthread_mutex_lock(&(shared.mutex));
    connection->state = ClientState_REGISTERED;
    shared.dirty = 1;
    pthread_mutex_unlock(&(shared.mutex));

    free(send_buffer);
    free(receive_buffer);
    return 0;
}

```

8.7 Connection.h

```c
#ifndef Connection_h
#define Connection_h
#include <netinet/in.h>
#include <pthread.h>
#include <time.h>
#include <Data.h>
#include "map.h"

typedef struct {
    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;
} ClientShared;

#define ClientState_ENTRY 1
#define ClientState_ACCESSING 2
#define ClientState_REGISTERED 3
#define ClientState_UNAUTHENTICATED 4
typedef short ClientState;

#define ConnectionStatus_CLOSED 0
#define ConnectionStatus_ACTIVE 1
#define ConnectionStatus_CLOSING 2
typedef short ConnectionState;

typedef struct {
    ConnectionState status;
    int socket;
    struct sockaddr_in address;
    socklen_t address_length;
    pthread_t thread_id;
    time_t time_connected;
    ClientState state;
    User * user;

} Connection;

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

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

```c

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

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

void * StartUpdateThread(void * parameter);

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

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

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

void _who(char* response);

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

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

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

int _password(Connection* connection);

int _authenticate(Connection* connection);

#endif
```

8.8 Data.c

```c

#include "Data.h"

char * accepted_userIDs[] = {
    "chen",
    "bea1389",
    "bol4559",
    "cal6258",
    "kre5277",
    "lon1150",
    "mas9309",
    "mes08346",
    "mil7233",
    "nef9476",
    "nov7488",
    "pan9725",
    "rac3146",
    "rub4133",
    "shr5683",
    "vay3083",
    "yos2327"};

char * userFullNames[] = {
    "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"};

```

8.9 Data.h

```c
#ifndef Data_h
#define Data_h
#define RECORD_COUNT 17
#define ID_MAX_LENGTH 9
#define NAME_MAX_LENGTH 21

```

```c
#define IP_LENGTH 16

#define PASSWORD_LENGTH 20

extern char * accepted_userIDs[];

extern char * userFullNames[];

typedef struct
{
    char id[ID_MAX_LENGTH];
    char name[NAME_MAX_LENGTH];
    int age;
    float gpa;
    short connected;
    char ip[IP_LENGTH];
    long lastConnection;
    short registered;
    char password[PASSWORD_LENGTH];
} User;

#endif
```

8.10 File.c

```c

#include <unistd.h>
#include <fcntl.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <dirent.h>
#include "File.h"
#include "Data.h"
#include "Util.h"

short FileStatus(char * filename) {
    int err = access(filename, F_OK);
    if(!err) {
        err = access(filename, F_OK | R_OK | W_OK);
        if(!err) {
            return 1;
        }
        return 2;
    }
    return 0;
}

FILE * CreateOrOpenFileVerbose(char * filename, char * defaultContents) {
    FILE * file = NULL;
    int status = FileStatus(filename);

    if(status == 2) {
        printRed("Error: %s exists but you do not have permission to access it.\n", filename);
        return NULL;
    }

    if(status == 0) {
        printf("Creating %s.\n", filename);
        file = fopen(filename, "w+");
    } else if(status == 1) {
        printf("Opening %s.\n", filename);
        file = fopen(filename, "r+");
    }


    if(file == NULL) {
        printf(COLOR_RED);
        if(status == 0) {
            printf("Failed to create %s.\n", filename);
            perror("Error: ");
        } else if(status == 1) {
            printf("Failed to open %s.\n", filename);
            perror("Error: ");
        } else {
            printf("Unknown error opening %s.", filename);
        }
        printf(COLOR_RESET);
        return NULL;
    }

    if(status == 0) {
        printGreen("Created %s.\n", filename);
        if(defaultContents != NULL) {
            fpos_t start_pos;
            fgetpos(file, &start_pos);
            fprintf(file, defaultContents, 0);
            fsetpos(file, &start_pos);
        }
    } else if(status == 1) {
        printGreen("Opened %s.\n", filename);
    }

    return file;
}

int ReadKeyIntoSettingsMap(FILE* key_file, map* settings_map) {
    char* start = malloc(sizeof(char*));
    fgets(start, sizeof(start), key_file);
    printBlue("Start char is: '%c'\n", start[0]);
    char* end = malloc(sizeof(char*));
    fgets(end, sizeof(end), key_file);
    printBlue("End char is: '%c'\n", end[0]);

    char* start_key = malloc(sizeof("start_char"));
    strcpy(start_key, "start_char");
    char* end_key = malloc(sizeof("end_char"));
    strcpy(end_key, "end_char");
    char* cipher_key = malloc(sizeof("cipher"));
    strcpy(cipher_key, "cipher");

    char* cipher = calloc(end[0] - start[0] + 2, sizeof(char));
    printGreen("Value of end char is: '%d'\n", (int)end[0]);
    printGreen("Value of start char is: '%d'\n", (int)start[0]);
    printBlue("Value of end[0] - start[0] + 1 is: '%d'\n", end[0] - start[0] + 1);
    fgets(cipher, end[0] - start[0] + 2, key_file);
    printBlue("Cipher is: '%s'\n", cipher);

    Map_Set(settings_map, start_key, start);
    Map_Set(settings_map, end_key, end);
    Map_Set(settings_map, cipher_key, cipher);
    return 1;
}


int ReadRegisteredFileIntoUsersMap(FILE * reg_file, map * users_map) {
    char userID[ID_MAX_LENGTH];
    int user_age;
    float user_gpa;
    char userLastIP[IP_LENGTH];
    long lastConnection;
    char userPassword[PASSWORD_LENGTH];

    int scan_items;
    int line = 1;

    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) {
        map_result result = Map_Get(users_map, userID);
        if(result.found == 0) {
            printYellow("Couldn't find user %s. Continuing read.\n", userID);
            continue;
        }
        User * user = (User*)result.data;
        user->age = user_age;
        user->gpa = user_gpa;
        strcpy(user->ip, userLastIP);
        user->lastConnection = lastConnection;
        user->registered = 1;
        strcpy(user->password, userPassword);
        line++;
    }

    if(scan_items != EOF) {
        printRed("Error scanning registered file on line %d. Expected 5 items but had %d.\n", line,
scan_items);
        return 1;
    }
    return 0;
}

void UpdateRegisteredFileFromUsersMap(FILE * reg_file, map * users_map) {
    int i;
    for(i = 0; i < RECORD_COUNT; i++) {
        map_result result = Map_Get(users_map, accepted_userIDs[i]);
        if(!result.found) {
            printRed("User %s was not found in users map.", accepted_userIDs[i]);
            continue;
        }

        User * user = (User *) result.data;
        if(user->registered) {
            fprintf(reg_file, "%s\t%d\t%f\t%s\t%ld\t%s\n", user->id, user->age, user->gpa, user->ip,
user->lastConnection, user -> password);
        }
    }
}

int NumberOfFilesInDirectory(char* dir_name) {
    int count = 0;

    DIR * dirp;
    struct dirent * entry;

    dirp = opendir(dir_name);

    while((entry = readdir(dirp)) != NULL) {
        if(entry->d_type == DT_REG) {
            count++;
        }
    }

    closedir(dirp);

    return count;
}

void GetRandomFileNameFromDir(char * dir_name, char* file_name) {
    int file = RandomInteger(0, NumberOfFilesInDirectory(dir_name) - 1);

    DIR* dirp = opendir(dir_name);
    struct dirent * entry;

    while(file >= 0 && ((entry = readdir(dirp)) != NULL)) {
        if(entry->d_type == DT_REG) {
            file--;
        }
    }

    if(entry != NULL) {
        strcpy(file_name, entry->d_name);
    }

    closedir(dirp);
}

int ReadSettingsFileIntoSettingsMap(FILE * settings_file, map * settings_map) {
    char key_read[100];
    char value_read[100];

    int scan_items;
    int line = 1;

    while( (scan_items = fscanf(settings_file, " %s = %s ", key_read, value_read)) == 2) {
        char * key_alloc = malloc( (strlen(key_read)+1) * sizeof(char));
        memset(key_alloc, 0, strlen(key_read)+1);
        strcpy(key_alloc, key_read);
        char * val_alloc = malloc( (strlen(value_read)+1) * sizeof(char));
        memset(val_alloc, 0, strlen(value_read)+1);
        strcpy(val_alloc, value_read);
        Map_Set(settings_map, key_alloc, val_alloc);
        line++;
    }

    if(scan_items != EOF) {
        printRed("Error scanning settings file on line %d. Expected 2 items but had %d.\n", line,
scan_items);
        return 1;
    }
    return 0;
}

void CatFileToBuffer(char* file_name, char* buffer, size_t buffer_size) {
    if(FileStatus(file_name)) {
        FILE* file = CreateOrOpenFileVerbose(file_name, NULL);
        char* temp = malloc(buffer_size);

        while(fgets(temp, buffer_size - strlen(buffer), file) && buffer_size - strlen(buffer) > 1) {
            strcat(buffer, temp);
        }

        free(temp);
    }
}

int CreateLockfile()
```

```c
{
    FILE * file = fopen(LOCKFILE, "w");
    if(file == NULL) {
        return 0;
    }
    fprintf(file, "0 %d", getpid());
    fclose(file);
    return 1;
}

int DeleteLockfile()
{
    return remove(LOCKFILE);
}

```

8.11 File.h

```c
#ifndef Files_h
#define Files_h
#include <stdio.h>
#include "map.h"

// ~~~~ Macros ~~~~ //

#define LOCKFILE "/tmp/lab6.lock"

#define REGISTERED_FILE "registered.txt"

#define SERVER_SETTINGS_FILE "server-settings.txt"

#define KEY_FILE "sub.key"

#define ADS_DIR "ads"

#define CIPHER_MAX_LENGTH 100

// ~~~~~ General File Functions ~~~~~ //

short FileStatus(char * filename);

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

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


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

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

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

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

int NumberOfFilesInDirectory(char* dir_name);

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

/***
    Creates a lockfile.
    @warning This should only be called by a running server process when a lockfile does not already exist.
    @returns 1 on success, otherwise 0.
*/
int CreateLockfile();

/***
    Deletes a lockfile.
    @returns 1 on success, otherwise 0.
*/
int DeleteLockfile();

#endif
```

8.12 Logfile.c

```c

#include <stdlib.h>
#include <string.h>
```

```c
#include <stdarg.h>
#include <stdio.h>
#include <time.h>
#include "Util.h"

char * logfile_name;

char* months[12] = {
    "jan",
    "feb",
    "mar",
    "apr",
    "may",
    "jun",
    "jul",
    "aug",
    "sep",
    "oct",
    "nov",
    "dec"
};

void logline(const char * logtype, char * logmsg) {
    time_t current_time;
    struct tm *timeinfo;
    time(&current_time);
    timeinfo=localtime(&current_time);

    FILE * file = fopen(logfile_name, "a");
    if(file == NULL) {
        printRed("Failed to open logfile %s.\n", logfile_name);
        return;
    }

    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);

    fclose(file);
}

void SetLogfileName(char * logfile_name_param) {
    if(logfile_name != NULL) {
        free(logfile_name);
    }
    logfile_name = malloc(strlen(logfile_name_param)+1);
    strcpy(logfile_name, logfile_name_param);
}

void LogfileError(const char * format, ...) {
    va_list args;
    va_start(args, format);

    va_list args_copy;
    va_copy(args_copy, args);
    int size = vsnprintf(NULL, 0, format, args_copy);
    va_end(args_copy);

    char * file_print_string = (char*) malloc((size+1)*sizeof(char));

    vsprintf(file_print_string, format, args);
    va_end(args);
    logline("ERR", file_print_string);
    free(file_print_string);
}

void LogfileMessage(const char * format, ...) {
    va_list args;
    va_start(args, format);

    va_list args_copy;
    va_copy(args_copy, args);
    int size = vsnprintf(NULL, 0, format, args_copy);
    va_end(args_copy);

    char * file_print_string = (char*) malloc((size+1)*sizeof(char));

    vsprintf(file_print_string, format, args);
    va_end(args);
    logline("MSG", file_print_string);
    free(file_print_string);
}

```

8.13 Logfile.h

```
#ifndef Logfile_h
#define Logfile_h

#include <stdio.h>

void SetLogfileName(char * logfile_name);

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

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

#endif

```

8.14 main.c

```
#include <stdio.h>
#include "Util.h"
#include "Process.h"
#include <string.h>
//#include "Logfile.h"

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

    if(argc <= 1 )
    {
        RunCommand();
    }
    else if (strcmp(argv[1], "headless") == 0)
    {
        RunHeadless(argv[0]);
    }
    else if (strcmp(argv[1], "stop") == 0)
    {
        StopCommand();
    }
    else if (argc > 2)
    {
        // Print command not found
        // HelpCommand()
    }
    else
    {
        RunCommand();
    }
    return 0;
}
```

8.15 map.c

```

#include "stdlib.h"
#include "string.h"
#include "map.h"
#include "math.h"

int hash_log2(int num_to_log)
{
    int t = 1;
    int i = 0;
    do
    {
        num_to_log = num_to_log & ~t;
        t = t « 1;
        i++;
    } while (num_to_log > 0);
    return i;
}

int hash_upperLimit(int bitsize)
{
    return 1 « bitsize;
}

int char_ratio = (int)(sizeof(int) / sizeof(char));

int hash_string(int hash_table_size, char *string, int strlen)
```

```
{
    int i, hash = 2166136261;
    for (i = 0; i < strlen; i += 1)
    {
        hash *= 16777619;
        hash ^= string[i];
    }
    if (hash < 0)
    {
        hash *= -1;
    }
    return hash % hash_table_size;
}

map *NewMap(int capacity)
{
    int log2 = hash_log2(capacity);
    int capac = hash_upperLimit(log2);
    int sz = sizeof(struct _map_bucket) * capac;
    struct _map_bucket *buckets = malloc(sz);
    memset(buckets, 0, sz);
    int i;
    for (i = 0; i < capac; i++)
    {
        buckets[i] = (struct _map_bucket){NULL, NULL, NULL};
    }
    map newm = (map){capac, buckets};
    map *map_p = malloc(sizeof(map));
    *map_p = newm;
    return map_p;
}

void _bucket_insert(struct _map_bucket *bucket, char *key, void *value)
{
    struct _map_bucket *check = bucket;
    while (check->key != NULL)
    {
        if (strcmp(check->key, key) == 0)
        {
            check->data = value;
            return;
        }
        if (check->next == NULL)
        {
            check->next = malloc(sizeof(struct _map_bucket));
            *(check->next) = (struct _map_bucket){NULL, NULL, NULL};
        }
        check = check->next;
    }
    check->key = key;
    check->data = value;
}

void Map_Set(map *a_map, char *key, void *value)
{
    int keyl = (int)strlen(key);
    int hash = hash_string(a_map->size, key, keyl);
    _bucket_insert(&(a_map->buckets[hash]), key, value);
}
void _bucket_get(struct _map_bucket *bucket, char *key, map_result *result)
{
    struct _map_bucket *check = bucket;
    while (check->key != NULL)
    {
        if (strcmp(check->key, key) == 0)
        {
            result->found = 1;
            result->data = check->data;
            return;
        }
        else if (check->next != NULL)
        {
            check = check->next;
        }
        else
        {
            result->found = 0;
            break;
        }
    }
}

map_result Map_Get(map *a_map, char *key)
{
    map_result res = (map_result){0, NULL};
    int keyl = (int)strlen(key);
    int hash = hash_string(a_map->size, key, keyl);
    _bucket_get(&(a_map->buckets[hash]), key, &res);
    return res;
}

void _bucket_delete(struct _map_bucket *bucket, char *key, short free_it, map_result *result)
{
    struct _map_bucket *last = bucket;
    struct _map_bucket *next = bucket->next;
    while (next != NULL)
    {
        if (strcmp(next->key, key) == 0)
        {
            result->found = 1;
            result->data = next->data;
            if (free_it)
            {
                free(next->data);
                result->data = NULL;
            }
            last->next = next->next;
            free(next);
        }
        else
        {
            last = next;
            next = next->next;
        }
    }
}

map_result Map_Delete(map *a_map, char *key, short free_it)
{
    map_result res = (map_result){0, NULL};
    int keyl = (int)strlen(key);
    int hash = hash_string(a_map->size, key, keyl);

    struct _map_bucket top = a_map->buckets[hash];
    if (top.key == NULL)
    {
        return res;
    }
    if (strcmp(top.key, key) == 0)
    {
        res.found = 1;
        res.data = top.data;
        if (free_it)
        {
            free(top.data);
            res.data = NULL;
        }
        if (top.next != NULL)
        {
            a_map->buckets[hash] = *(top.next);
            free(top.next);
        }
        else
        {
            a_map->buckets[hash] = (struct _map_bucket){NULL, NULL, NULL};
        }
        return res;
    }
    if (top.next == NULL)
    {
        return res;
    }
    _bucket_delete(&(a_map->buckets[hash]), key, free_it, &res);

    return res;
}
```

8.16 map.h

```
#ifndef map_h
#define map_h

// -----------------------------
//        Hashing Math
// -----------------------------

int hash_log2(int number_to_log);

int hash_string(int hash_table_capacity, char *string, int strlen);

```

```c
int hash_upperLimit(int bitsize);

// ------------------------------------
//         General Map Operations
// ------------------------------------

struct _map_bucket
{
    char *key;
    void *data;
    struct _map_bucket *next;
};

typedef struct
{
    int size;
    struct _map_bucket *buckets;
} map;

typedef struct
{

    short found;
    void *data;
} map_result;

map *NewMap(int capacity);

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

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

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

#endif
```

8.17 Process.c

```c

#include <stdio.h>
#include <string.h>
#include <signal.h>
#include <stdlib.h>
#include <unistd.h>
#include "Data.h"
#include "Build.h"
#include "map.h"
#include "File.h"
#include "Util.h"
#include "Server.h"
#include "Logfile.h"
#include "Connection.h"

User * users_array;
map * users_map;
map * settings_map;
char * default_settings = "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";

char * default_sub =    " \n"
                        "~\n"
                        "`\"G)YF7A,R2L'@
ZD/E5I<?H:i4NJ&g;rB(f#KobljnW1C{_-Ua]%^cV\\>tOP|pQ$689=+whzS3*Xm!ek~My[}sqduv0.Tx";

int active_clients;

int _initializeLogger() {
    return 1;
    // char* fileName = "log.txt";
    // int printLevel, LogLevel, printAlltoStdOut;
    // map_result result = Map_Get(settings_map, "log_file");
    // if(!result.found) {
    //     printYellow("No output file found. Defaulting to 'log.txt'\n");
    //     SetLogfileName("log.txt");
    // } else {
    //     fileName = result.data;
    //     SetLogfileName(fileName);
    // }

    // result = Map_Get(settings_map, "print_level");
    // if(!result.found) {
    //     printYellow("No print_level found, defaulting to 3\n");
    //     printLevel = 3;
    // } else {
    //     printLevel = atoi(result.data);
    //     if(printLevel < 0 || printLevel > 5) {
    //         printYellow("Invalid print_level of %d, defaulting to 3\n", printLevel);
    //         printLevel = 3;
    //     }
    // }

    // result = Map_Get(settings_map, "log_level");
    // if(!result.found) {
    //     printYellow("No log_level found, defaulting to 3\n");
    //     LogLevel = 3;
    // } else {
    //     LogLevel = atoi(result.data);
    //     if(LogLevel < 0 || LogLevel > 5) {
    //         printYellow("Invalid log_level of %d, defaulting to 3\n", LogLevel);
    //         LogLevel = 3;
    //     }
    // }

    // result = Map_Get (settings_map, "log_to_console");
    // if(!result.found) {
    //     printYellow("No log_to_console found, defaulting to true\n");
    //     printAlltoStdOut = 1;
    // } else {
    //     if(strcmp(result.data, "true") == 0) {
    //         printAlltoStdOut = 1;
    //     } else if(strcmp(result.data, "false") == 0) {
    //         printAlltoStdOut = 0;
    //     } else {
    //         printYellow("invalid data in log_to_console, defaulting to true\n");
    //         printAlltoStdOut = 1;
    //     }
    // }
    // return 1;
}

int InitializeCipher() {
    printf("Reading key file.\n");
    FILE * key_file = CreateOrOpenFileVerbose(KEY_FILE, default_sub);
    if(key_file == NULL) {
        printRed("Initialization failed during access of file: %s.\n", KEY_FILE);
        return 0;
    }
    ReadKeyIntoSettingsMap(key_file, settings_map);
    // int key_read_err = ReadKeyIntoSettingsMap(key_file, settings_map);
    // if(key_read_err) {
    //     printRed("Initialization failed while reading key file %s. Correct this file or delete it
so default can be generated.\n", KEY_FILE);
    // }
    fclose(key_file);
    printGreen("Read %s.\n", KEY_FILE);
    return 1;
}

int Initialize() {

    // Create the data structures on the heap.
    printf("Initializing User data structures.\n");
    users_array = CreateUsersArray(accepted_userIDs, userFullNames, RECORD_COUNT);
    users_map = CreateUsersMap(users_array, RECORD_COUNT);
    active_clients = 0;
    printGreen("User data structures initialized.\n");

    // Create the registered file that tracks registered users.
    printf("Checking for registered file.\n");
    FILE * reg_file = CreateOrOpenFileVerbose(REGISTERED_FILE, NULL);
    if(reg_file == NULL) {
        printRed("Initialization failed during accessing of file: %s.\n", REGISTERED_FILE);
        return 0;
    }

    // Update the User's map with with the data from the registered file.
    printf("Reading registered file.\n");
    int read_error = ReadRegisteredFileIntoUsersMap(reg_file, users_map);
    fclose(reg_file);
    if(read_error) {
        printRed("Initialization failed during reading of file: %s.\n", REGISTERED_FILE);
        return 0;
    }
    printGreen("Loaded %s into users map.\n", REGISTERED_FILE);
```

```c

    printf("Reading settings file.\n");
    settings_map = NewMap(50);
    FILE * settings_file = CreateOrOpenFileVerbose(SERVER_SETTINGS_FILE, default_settings);
    if(settings_file == NULL) {
        printRed("Initialization failed during accessing of file: %s.\n", SERVER_SETTINGS_FILE);
        return 0;
    }
    int settings_read_err = ReadSettingsFileIntoSettingsMap(settings_file, settings_map);
    if(settings_read_err) {
        printRed("Initialization failed while reading settings file %s. Correct this file or delete it so a default can be generated.\n", SERVER_SETTINGS_FILE);
        return 0;
    }
    fclose(settings_file);
    printGreen("Read %s.\n", SERVER_SETTINGS_FILE);

    InitializeCipher();


    printf("Initializing logger.\n");
    int logger_initialized = _initializeLogger();
    if(!logger_initialized) {
        printRed("Failed to initalize logger.\n");
    }


    printf("Initializing server.\n");
    int server_initialized = InitializeServer(settings_map);
    if(!server_initialized) {
        printRed("Failed to initialize server.\n");
        return 0;
    }


    return 1;
}

void SignalHandle(int signo) {
    if(signo == SIGINT || signo == SIGTERM) {
        printYellow("Received signal. Shutting down server.\n");
        int err = CloseServer();
        if(err) {
            printRed("Problem closing server\n");
            perror("Error closing socket interface.\n");
        }
        DeleteLockfile();
        exit(0);
    }

}

int RunCommand() {
    if (FileStatus(LOCKFILE) > 0)
    {
        printf("Server process already running.\n");
        return 0;
    }
    signal(SIGTERM, SignalHandle);
    signal(SIGINT, SignalHandle);
    int init_success = Initialize();
    if(!init_success) {
        printRed("Could not start the server due to failed initialization.\n");
        return 0;
    }
    printf("Running server.\n");
    int server_success = StartServer(users_map);
    if(!server_success) {
        printRed("There was a problem running the server.\n");
        return 0;
    }
    int delete_lockfile_success = DeleteLockfile();
    if(!delete_lockfile_success) {
        printRed("There was a problem deleting the Lockfile.\n");
        return 0;
    }
    return 1;
}

void RunHeadless(char *processName) {
    if (FileStatus(LOCKFILE) > 0)
    {
        printf("Server process already running.\n");
        return;
    }
    char commandFront[] = " nohup ";
    char commandEnd[] = " & exit";
    size_t comm_length = strlen(commandFront) + strlen(commandEnd) + strlen(processName) + 1;
    char *commandFull = malloc(comm_length * sizeof(char));
    memset(commandFull, 0, comm_length * sizeof(char));
    strcpy(commandFull, commandFront);
    strcat(commandFull, processName);
    strcat(commandFull, commandEnd);

    printf("Executing: %s\n", commandFull);
    popen(commandFull, "we");
    printf("Server running headlessly.\n");
}

int TerminateExistingServer()
{
    FILE *file = fopen(LOCKFILE, "r");
    if (file == NULL)
    {
        perror("Error opening lockfile");
        return -1;
    }
    int need_rewrite;
    int pid = 0;
    fscanf(file, "%d %d", &need_rewrite, &pid);
    fclose(file);
    if (pid > 0)
    {
        return kill(pid, SIGTERM);
    }
    return -2;
}

void StopCommand() {
    printYellow("\nStopping server...\n");
    int err = TerminateExistingServer();
    if (err)
    {
        if (err == -1)
        {
            printRed("Server isn't running.\n");
        }
        else if (err == -2)
        {
            printRed("Lockfile did not contain a valid process id!\n");
        }
        else
        {
            printRed("Sending terminate signal failed!\n");
        }
    }
    else
    {
        printGreen("Server terminated.\n");
    }
}

void ResetCommand() {
    if(FileStatus(REGISTERED_FILE)) {
        fclose(fopen(REGISTERED_FILE, "w")); //empties the registered file
    }
}
```

8.18 Process.h

```
#ifndef Process_h
#define Process_h
/***
 * \defgroup Process
 * \brief This module holds functions that realize the primary business logic of the program.
 * \details When command line arguments are parsed by main, functions in this module are called.
 * @{
*/

/***
    Handles an interrupt or quit signal.

    Send server to shutdown mode, resulting in graceful deletion of lockfile.
*/
void SignalHandle(int signo);

int InitializeCipher();

/***
 * Performs initializing activities which must occur prior to a server loop starting.
```

```c
 *
 * Will print errors if there are problems initializing.
 *
 * @returns 1 on success, otherwise 0.
*/
int Initialize();

/***
 * Runs the server.
 * @returns 1 on success, otherwise 0.
*/
int RunCommand();

/***
    Finds the process ID of a running server using the lockfile, and calls kill on it, sending a SIGTERM.
    @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.

*/
int TerminateExistingServer();

void StopCommand();

void RunHeadless(char *processName);

#endif
```

8.19 Server.c

```c

#include <stdlib.h>
#include <netinet/in.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <linux/net.h>
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>

#include "Server.h"
#include "Connection.h"
#include "Util.h"
#include "map.h"
#include "Logfile.h"
#include "File.h"

ServerProperties server;
Connection * connections;

// A private function just for reading the settings map into the server struct and printing warnings as necessary.
void _readSettingsMapIntoServerStruct(map * server_settings) {
    map_result result = Map_Get(server_settings, "port");
    if(!result.found) {
        printYellow("No port setting found. Defaulting to 3000.\n");
        server.port = 3000;
    } else {
        int found_port = atoi(result.data);
        if(found_port <= 0) {
            printYellow("Invalid port setting: %s. Defaulting to 3000.\n", result.data);
            server.port = htons(3000);
        } else {
            server.port = htons(found_port);
        }
    }
    result = Map_Get(server_settings, "send_buffer_size");
    if(!result.found) {
        printYellow("No send_buffer_size setting found. Defaulting to 1024.\n");
        server.send_buffer_size = 1024;
    } else {
        int found_sb_size = atoi(result.data);
        if(found_sb_size <= 0) {
            printYellow("Invalid send_buffer_size setting: %s. Defaulting to 1024.\n", result.data);
            server.send_buffer_size = 1024 * sizeof(char);
        } else {
            server.send_buffer_size = found_sb_size * sizeof(char);
        }
    }
    result = Map_Get(server_settings, "receive_buffer_size");
    if(!result.found) {
        printYellow("No receive_buffer_size setting found. Defaulting to 1024.\n");
        server.send_buffer_size = 1024;
    } else {
        int found_rb_size = atoi(result.data);
        if(found_rb_size <= 0) {
            printYellow("Invalid receive_buffer_size setting: %s. Defaulting to 1024.\n", result.data);
            server.receive_buffer_size = 1024 * sizeof(char);
        } else {
            server.receive_buffer_size = found_rb_size * sizeof(char);
        }
    }
    result = Map_Get(server_settings, "backlog");
    if(!result.found) {
        printYellow("No backlog setting found. Defaulting to 10.\n");
        server.backlog = 10;
    } else {
        int found_backlog = atoi(result.data);
        if(found_backlog <= 0) {
            printYellow("Invalid backlog setting: %s. Defaulting to 10.\n", result.data);
            server.backlog = 10;
        } else {
            server.backlog = found_backlog;
        }
    }
    result = Map_Get(server_settings, "max_connections");
    if(!result.found) {
        printYellow("No max_connections setting found. Defaulting to 20.\n");
        server.max_connections = 20;
    } else {
        int found_max_connections = atoi(result.data);
        if(found_max_connections <= 0) {
            printYellow("Invalid max_connections setting: %s. Defaulting to 20.\n", result.data);
            server.max_connections = 20;
        } else {
            server.max_connections = found_max_connections;
        }
    }
    result = Map_Get(server_settings, "start_char");
    if(!result.found) {
        printYellow("No start_char setting found. Defaulting to ' '\n");
        server.start = ' ';
    } else {
        char* start = result.data;
        server.start = start[0];
    }
    result = Map_Get(server_settings, "end_char");
    if(!result.found) {
        printYellow("No end_char setting found. defaulting to '~'\n");
        server.end = '~';
    } else {
        char* end = result.data;
        server.end = end[0];
    }
    result = Map_Get(server_settings, "cipher");
    if(!result.found) {
        printRed("No cipher found.\n");
    } else {
        server.cipher = result.data;
    }
    result = Map_Get(server_settings, "log_file");
    if(!result.found) {
        printYellow("No log_file specified, defaulting to 'log.txt'.\n");
        SetLogfileName("log.txt");
    } else {
        SetLogfileName((char *) result.data);
        printBlue("Logging to %s.\n", result.data);
    }
};

int InitializeServer(map * server_settings) {
    _readSettingsMapIntoServerStruct(server_settings);
    connections = malloc(server.max_connections * sizeof(Connection));
    int i;
    for(i=0; i < server.max_connections; i++) {
        connections[i].status = ConnectionStatus_CLOSED;
    }
    printGreen("Server initialized with %d max connections.\n", server.max_connections);
    return 1;
}

int StartServer(map * users_map) {
    int serverSocket = 0;
    struct sockaddr_in server_address;
    // Record the time the server started.
    time(&server.time_started);
    // Get a socket file pointer associated with ipv4 internet protocols that represents a two-way connection based byte stream.
    serverSocket = socket(AF_INET, SOCK_STREAM, 0);
    server_address.sin_family = AF_INET;
    // Set the address to bind to all available interfaces.
    server_address.sin_addr.s_addr = htonl(INADDR_ANY);
    // Set the port.
    server_address.sin_port = server.port;
    // Assign a name to the socket.
    int bind_error = bind(serverSocket, (struct sockaddr*)&server_address, sizeof(server_address));
    if(bind_error) {
        printRed("Error binding the server to port %d.\n", ntohs(server.port));
        perror("Bind Error:");
        return 0;
    }
    server.socket_id = serverSocket;
    int lockfile_success = CreateLockfile();
    if(!lockfile_success) {
        printRed("Failed to create Lockfile! Server cannot start.");
        return 0;
    }
    // Initialized a shared space that will be used across threads.
    ClientShared * shared = InitializeShared(users_map, server.send_buffer_size, server.receive_buffer_size, server.cipher, server.start, server.end);
    // The update thread is responsible for checking if there is 'dirty' data that should be saved to the registered user's file.
    pthread_t registered_update_thread;
    pthread_create(&registered_update_thread, NULL, StartUpdateThread, NULL);
    printBlue("Server listening on port: %d\n", ntohs(server.port));
    LogfileMessage("Server started.");
    // begin listening according to the socket settings
    listen(serverSocket, server.backlog);
    while(!shared->shutting_down) {
        // Get an available connection.
        Connection * next_client = NextAvailableConnection();
        if(next_client == NULL) {
            printYellow("Server connections are maxxed.\n");
            LogfileError("Server couldn't accept connection; available connections are maxxed.");
            sleep(1);
            continue;
        }
        // Accept a connection.
        next_client->address_length = sizeof(next_client->address);
        next_client->socket = accept(serverSocket, (struct sockaddr *)&(next_client->address), &(next_client->address_length));
        if(next_client->socket < 0)
        {
            printRed("Failed to accept() client!\n");
            LogfileError("Failed to accept() client.");
            sleep(1);
            continue;
        }
        printBlue("New client connection from IP: %s\n", inet_ntoa(next_client->address.sin_addr));
        LogfileMessage("New client connection from IP: %s", inet_ntoa(next_client->address.sin_addr));
        next_client->status = ConnectionStatus_ACTIVE;
        // Start a thread to handle communication from that connection.
        pthread_create(&(next_client->thread_id), NULL, StartConnectionThread, next_client);
    }

    return 1;
}

Connection * NextAvailableConnection()
{
    int i;
    for(i = 0; i < server.max_connections; i++) {
        if(connections[i].status == ConnectionStatus_CLOSED)
        {
            return &(connections[i]);
        }
    }
    return NULL;
}

int CloseServer() {
    return close(server.socket_id);
}


```

8.20 Server.h

```
#ifndef Server_h
#define Server_h
#include <stdint.h>
```

```c
#include <time.h>
#include "map.h"
#include "Connection.h"

typedef struct {
    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;
} ServerProperties;

int InitializeServer();

int InitializeCipher();

int StartServer(map * users_map);

int CloseServer();

Connection * NextAvailableConnection();

#endif
```

8.21 Util.c

```c

#include <stdio.h>
#include <stdarg.h>
#include <string.h>
#include <stdlib.h>
#include "Util.h"

void printRed(const char * format, ...) {
    printf(COLOR_RED);
    va_list args;
    va_start(args, format);
    vprintf(format, args);
    va_end(args);
    printf(COLOR_RESET);
}

void printGreen(const char * format, ...) {
    printf(COLOR_GREEN);
    va_list args;
    va_start(args, format);
    vprintf(format, args);
    va_end(args);
    printf(COLOR_RESET);
}

void printYellow(const char * format, ...) {
    printf(COLOR_YELLOW);
    va_list args;
    va_start(args, format);
    vprintf(format, args);
    va_end(args);
    printf(COLOR_RESET);
}

void printBlue(const char * format, ...) {
    printf(COLOR_BLUE);
    va_list args;
    va_start(args, format);
    vprintf(format, args);
    va_end(args);
    printf(COLOR_RESET);
}

int RandomInteger(int min, int max)
{
    int r_add = rand() % (max - min + 1);
    return r_add + min;
}

float RandomFloat(float min, float max)
{
    float dif = max - min;
    int rand_int = rand() % (int)(dif * 10000);
    return min + (float)rand_int / 10000.0;
}

short RandomFlag(float percentage_chance)
{
    float random_value = (float)rand() / RAND_MAX;
    if (random_value < percentage_chance)
    {
        return 1;
    }
    return 0;
}

```

8.22 Util.h

```
#ifndef Util_h
#define Util_h
#include <stdarg.h>

#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"

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);


#endif
```

Index