Whiteboard application

2019/2020

Introduction

Goal

The goal of the project is to write a “whiteboard” application, where a community of users could interact and exchange messages about common interests, by creating topics and threads. The components of the project are three, a server, a client and an admin. The server and the admin are unique while the clients can be more than one.

 A server will wait for connections, on a INET socket, forking new processes to handle interactions with clients. The principal operations in this system are the registration, authentication, operations related to the topics, threads and messages management (will talk about later)

Data flux

Registration

The client registers by entering a username and a password, the server then checks if the username entered is available, if so, the user is registered otherwise the client is notified that the username is not available, sending an error message.

(image)

Authentication

The client authenticates by entering a username and a password with which he previously registers, if the credentials are right the server proceeds to authenticate the user, otherwise the user is notified with an error message.

(image)

Topics, threads and messages

Once a user is authenticated, he can create topics about his interests, in which other users or himself can create threads on the main subject (Topic) then discuss about their interests by sending messages in the threads.

(image)

Whiteboard Structure

The Whiteboard folder is structured in the following way

```
├── admin.c
├── client
├── counter.txt
├── server
├── topics
|  └── topic_name
|         ├── owner.txt    // creator of the topic
|         └── thread_name
|                  ├── message_id // message written in the thread
|
└── users
```

- The **file admin.c** representing the admin
- The **client folder** containing all the necessary scripts to the functioning of the client
- The **file counter.txt** used to manage the messages id distribution
- The **server folder** containing all the necessary scripts to the functioning of the server
- The **topic folder** with all the topics,
- The **user folder** containing all the created users

Admin specifications

Admin

The damin can perform two operations

- create_user()
  - Create a user in the users folder, return **1** if **success** and **0** if **fail**

- delete_user()
  - Delete a user in the users folder, return **1** if **success** and **0** if **fail**

And a **main** function.

Client specifications

Client

The client can perform the following tasks

- 1 - Registration
- 2 - Authentication
- 3 - Create topic9
- 4 - Create thread
- 5 - Delete topic
- 6 - List (Topic, Thread, message)
- 7 - New message
- 8 - Reply message

- 9 - Print message
- 10 - Message status
- 11 - Subscribe to topic
- 12 - Logout

Each operation has an associated number the client needs to enter, in order to execute the corresponding operation.

The client interacts with the server like in a chat, (Just to be”**original**”) so, for example during the registration the client will first enter the operation number, let say 1, then the server will ask for a username and then for the password but only after the client enters the username.

(image)

Registration

For the registration the user has to enter the username and the password, the username must only contain alphabets characters, and the password must be composed only with numbers. Otherwise the server will answer with the appropriate message error.
The username must be unique no two users have the same username (No case sensitive).

(image)

Authentication

Same as the registration, username password, if the password is wrong the client is prompted with an error.

(image)

The client has a special mechanism that allows it to receive a continuous flux of data from the server, the mechanism is triggered by the server, by sending the string **“start_sequence”** to the client, and stops when the server sends the string **“end_sequence”.** This mechanism is mostly used in the listing of topics, threads and messages.

After a reading of the socket, …

```
if(strcmp(command, "start_sequence") == 0) {
                    while (1) {
                              read(sock, command, BUF_SIZE);
                              if (strcmp(command, "end_sequence") == 0) break;
                              else printf("%s", command);
                              }
```

```
        } else printf("%s\n", command);
```

Server specifications

First the server begins by clearing the shared memory for the access of the topics “space” if there's any (from precedent execution) and creates a new one, with the function ***shm_clear();*** Then after setting the parameter for the socket and the shared memory id the server will put everything belonging to the ../topics folder in the shared memory, with the **function read_topics**

```
  if(read_topics(shm_id, "../topics/") == 0) {
                printf("error!\n");
                exit(0);
        }
```

After that the server will set the parameters for the semaphores

```
int sem_id;
        key_t key = ftok("./server.c", 'J');
        sem_id = semget(key, 2, 0777 | IPC_CREAT);

        // sem 1: ?
        // sem 2: ?
        semctl(sem_id, 0, SETVAL, 1);
        semctl(sem_id, 1, SETVAL, 1);
        // sem 1: 1
        // sem 2: 1

        struct sembuf operation;
```

Basically the server receives in input the operation numbers from the client corresponding in the associated operation

1- Registration

If the server receive the operation number 1, first of all it checks if the client has already effectuate the registration by verifying the **registrationStatus** variable if it’s equal to 1 then the user has already been registered

```
        if (strcmp(command, "1") == 0) {

                //Check registration status
                if (registrationStatus == 1) {
                write(socket, "\t\t\t (ツ)_/¯ User already registered to the
WHITEBOARD!\n",BUF_SIZE );
                continue;
                }//End if

                char username[BUF_SIZE];
                char password[BUF_SIZE];

                get_input(username, password, socket);
                registrationStatus = registration(username, password, socket);
        }//End REGISTRATION
```

Otherwise the server proceed to get the username and password with the **get_input(char* username, char* password, int socket)** function, then successively proceed with the **registration(char* username, char* password, int socket)** function.

2- Authentication

If the server receive the operation number 2, first it checks for the **authenticationStatus** to see if the client is already authenticated, if authenticationStatus is equal to 2 then the client has already been authenticated, so by the same occasion the already registered so it sets the **registrationStatus** to 1 in order to avoid an authenticated user to use the registration operation.

```
  else if (strcmp(command, "2") == 0){

                if (authenticationStatus == 2) {
                registrationStatus = 1; //to avoid already authenticated user to use the register
command
                write(socket, "\t\t\t (ツ)_/¯ User already authenticated to the
WHITEBOARD!\n",BUF_SIZE );
                continue; //one client can't authenticate more than one user per session
                }
```

```
        get_input(username, password, socket);
        authenticationStatus = authentication(username, password, socket);



    }
```

Otherwise the server proceed to get the username and password with the **get_input(char* username, char* password, int socket)** function, then successively proceed with the **authentication(char* username, char* password, int socket)** function.

3- Create topic

By receiving the operation number 3 the server will first check if the user is authenticated, since this operation can't be performed by an unauthenticated user. If the user is authenticated it will then ask for the topic name to be created. First it set a semaphore in order to avoid any concurrence problem then create the topic in the shared memory with the function **create_topic(char* topic_name, char* username, int socket, int shmId)** then releases the semaphore.

```
    else if (strcmp(command, "3") == 0) {


        if (authenticationStatus == 0) {
        conn_status = write(socket, is_not_authenticated, BUF_SIZE);
        if(check_conn(conn_status) == 0) exit(0);
        continue;
        }

        char topic_name[BUF_SIZE];

        conn_status = write(socket, "\t\t\t <[^_^]> Insert a topic name!\n", BUF_SIZE);
        if(check_conn(conn_status) == 0) exit(0); //TO-DO -Will check if necessary
        conn_status = read(socket, topic_name, BUF_SIZE);
        if(check_conn(conn_status) == 0) exit(0);

        strtok(topic_name, "\n");

        // Lock semaphore (Topics handling)
        operation.sem_num = 0;   // 1st semaphore
        operation.sem_op  = -1;  // Lock operation (Decrement)
```

```
            operation.sem_flg = 0;   // Sem flag
            semop(sem_id, &operation, 1);
            // Sem is locked!
            create_topic(topic_name, username, socket, shm_id);
            // Lock semaphore (Topics handling)
            operation.sem_num = 0;   // 1st semaphore
            operation.sem_op  = 1;  // Unlock operation (Increment)
            operation.sem_flg = 0;   // Sem flag
            semop(sem_id, &operation, 1);
            // Sem is unlocked!

    }//END CREATE TOPIC
```

4 - Create Thread

Upon receiving the operation number 4, the server will check the authentication status of the client then proceed to the creation of the thread in the selected topic with the function **create_thread(char* topic_name, int socket, int shm_id, char* username).**

```
    else if (strcmp(command, "4") == 0) {

            char topic_name[BUF_SIZE];

            if (authenticationStatus == 0) {
            conn_status = write(socket, is_not_authenticated, BUF_SIZE_LARGE);
            if(check_conn(conn_status) == 0) exit(0);
            continue;
            }

            conn_status = write(socket, "\t\t\t <[^_^]>  In which Topic?\n", BUF_SIZE);
            if(check_conn(conn_status) == 0) exit(0); //TO-DO -Will check if necessary
            conn_status = read(socket, topic_name, BUF_SIZE_LARGE);
            if(check_conn(conn_status) == 0) exit(0);
            strtok(topic_name, "\n");

            // Lock semaphore (Topics handling)
            operation.sem_num = 0;   // 1st semaphore
            operation.sem_op  = 1;  // Unlock operation (Increment)
            operation.sem_flg = 0;   // Sem flag
            semop(sem_id, &operation, 1);
            // Sem is unlocked!
```

```
                create_thread(topic_name, socket, shm_id, username);
                // Lock semaphore (Topics handling)
                operation.sem_num = 0;   // 1st semaphore
                operation.sem_op  = 1;  // Unlock operation (Increment)
                operation.sem_flg = 0;   // Sem flag
                semop(sem_id, &operation, 1);
                // Sem is unlocked!

        }//END CREATE THREAD
```

5 - Delete topic

The deletion of a topic is triggered by receiving the operation command number 5. From now and on we will assume that the server will always check for the authentication status since these operations are only feasible if authenticated.
So after checking for the authentication status the server proceed to delete the topic using the function **delete_topic(char* username, int socket, int shm_id)**

```
        else if (strcmp(command, "5") == 0) {

                if (authenticationStatus == 0) {
                conn_status = write(socket, is_not_authenticated, BUF_SIZE);
                if(check_conn(conn_status) == 0) exit(0);
                continue;
                }
                delete_topic(username, socket, shm_id);


        }//END DELETE TOPIC
```

6 - List (Topics - Threads - Messages)

The operation number 6 is the listing of the Whiteboard elements. The server calls for the function **list(int socket, int shm_id)** which will pops up a sub-menu letting the client the choice between listing topics, threads, messages or all of them at the same time (*see Functions specifications section*)

```
        else if (strcmp(command, "6") == 0) {

                if (authenticationStatus == 0) {
                conn_status = write(socket, is_not_authenticated, BUF_SIZE);
                if(check_conn(conn_status) == 0) exit(0);
                continue;
                }
                list(socket,shm_id);

        }//END LIST
```

7- New Message

7 is the operation number for entering a new message, ther server will call the function **write_messgae(char* username, int socket, int shm_id)** (*see Functions specifications section*)

```
 //APPEND NEW MESSAGE TO NEW TOPIC
        else if (strcmp(command, "7") == 0) {

                if (authenticationStatus == 0) {
                conn_status = write(socket, is_not_authenticated, BUF_SIZE);
                if(check_conn(conn_status) == 0) exit(0);
                continue;
                }

                write_messgae(username, socket, shm_id);

        }
```

8- Reply

Reply to a message by entering the operation number 8**,** the function used here is **reply_message(char* username, int socket, int shm_id)** (*see Functions specifications section*)

```
        else if (strcmp(command, "8") == 0) {

                if (authenticationStatus == 0) {
                conn_status = write(socket, is_not_authenticated, BUF_SIZE);
                if(check_conn(conn_status) == 0) exit(0);
                continue;
                }

                reply_message(username, socket, shm_id);

            }
```

9- Print message

Operation 9, the function called is **print_message_informations(int socket, int shm_id)**

```
    else if (strcmp(command, "9") == 0) {

        if (authenticationStatus == 0) {
        conn_status = write(socket, is_not_authenticated, BUF_SIZE);
        if(check_conn(conn_status) == 0) exit(0);
        continue;
        }
        print_message_informations(socket, shm_id);


    }//END SELECT A GIVEN MESSAGE
```

10- Message status

Operation 10, the function called here is **message_status(int socket, int shm_id)**

```
    else if (strcmp(command, "10") == 0) {

        if (authenticationStatus == 0) {
        conn_status = write(socket, is_not_authenticated, BUF_SIZE);
        if(check_conn(conn_status) == 0) exit(0);
        continue;
        }

    message_status(socket, shm_id);
```

```
}//END MESSAGE STATUS
```

11- Subscribe

Operation 11, the function called is **subscribe(char* username, int socket, int shm_id)**

```
else if (strcmp(command, "11") == 0) {

        if (authenticationStatus == 0) {
        conn_status = write(socket, is_not_authenticated, BUF_SIZE);
        if(check_conn(conn_status) == 0) exit(0);
        continue;
        }

subscribe(username, socket, shm_id);

}//END SUBSCRIBE TOPIC
```

12- Logout

Operation number 12, consist in setting the registrationStatus and the authenticationStatus to 0

```
else if (strcmp(command, "12") == 0) {
        /* code */
        if (authenticationStatus == 0) {
        conn_status = write(socket, is_not_authenticated, BUF_SIZE);
        if(check_conn(conn_status) == 0) exit(0);
        continue;
        }
        else {
        registrationStatus = 0;
        authenticationStatus = 0;
        write(socket, "\t\t\t (✖╭╮✖) Logged out!\n", BUF_SIZE);
        }

}//END LOGOUT
```

Functions specifications

Registration functions

The functionalities related to the registration operations are written in the **registration_operatons.c** script

- **int check_username_available(char* username, char* password, int socket);**
    - Checks if the username has already been used (no case sensitive) by reading the **users** folder.Return **1** in case of **success** and **0** if it **fails**

- **int registration(char* username, char* password, int socket);**
    - Register the user by creating a file name with the username and containing the user password, after verifying if the credentials (username) is not already used and if they are respecting the input validity (password must be only numeric and username must be only alphabetic) by using respectively the functions **check_username_available(char* username, char* password, int socket)** and **check_validity(char* username, char* password, int socket)**
      Return **1** in case of **success** and **0** if it **fails**

- **int check_validity(char* username, char* password, int socket);**
    - Checks if the password is only numeric and if the username is only alphabetic using isdigit**(password)** and **isalpha(username).Return 1 in case of success and 0 if it fails**

Authentication functions

The functionalities related to the authentication operations are written in the **authentication_operatons.c** script

- **void is_authenticated(int authenticationStatus, int socket);**
    - Check if the user is authenticated by looking at the **authenticationStatus** variable (if 0 the user is not authenticated)

- **int authentication(char* username, char* password, int socket);**
  - Authenticate the user by setting his status to 2 after successful check of the credentials calling the function **check_credentials,** if one of the credentials is wrong the status is set to 3 and an error message is sent to the client (wrong username or password), otherwise the authentication status remain 0 and in that case an other error message is sent to the client (unknown user …)

- **int check_credentials(char* username, char* password);**
  - Read the users file and compare the username and the password received in input with the ones in the users file if there are both matches on the username and password the user status will be set to authenticated.(set to 2) returns 3 for wrong password or username , **1** if **success**

Topics functions

The functionalities related to the topic management operations are written in the **topic_operatons.c** script

- **void list_topic(int socket, int shm_id);**
  - Lists all the topics present in the shared memory (Whiteboard), using the **start_sequence/end_sequence** previously cited in the client section.

- **int read_topics (int shm_id, char *initial_path);**
  - Scans the entire topics folder, thus all the threads and messages

- **void create_topic(char* topic_name, char* username, int socket, int shmId);**
  - Check if the topic name to be created is already used if not create the topic alling the **new_topic(char *topicName, char* username)** which effectively proceed with the creation of the topic

- **int new_topic(char *topicName, char* username);**
  - Simply create a folder inside the topics folder, with the named with the topic name given in input by the user and create a dile owner.txt containing the name of the user who created the topic, by calling the function **save_owner(char* path, char* owner)**

- **int save_owner(char* path, char* owner);**
  - See description in **new_topic function** description.returns **1** if **successful** and returns **0** if **failed**

- **int is_topic(char* topicName, int shmId);**
  - Checks if the topic name given by the user is present in the shared memory, returns **1** if **successful** and returns **0** if **failed**

- **void delete_topic(char* username, int socket, int shm_id);**
  - Check if the topic name entered by the user actually exist by calling the function **is_topic(char* topicName, int shmId)** then call the function **remove_topic(char* topic_name, char* username)** for the effective deletion of topic

- **int remove_topic(char* topic_name, char* username);**
  - Deletes the topic file from the shared memory.returns **1** if **successful** and returns **0** if **failed**

- **Topics* append_topics(Topics* topics, Topics new_topic);**
  - If the topic is null create a new instance of topic with the function istance_topic(new_topic). The function add topics in a list of topics

- **char* get_owner(char* path)**
  - Returns a character representing the owner of a topic from the peter.txt file of the topic.

- **Topics* istance_topic(Topics new_topic)**
    - Create a new instance of Topics

Thread functions

The functionalities related to the thread management operations are written in the **thread_operatons.c** script

- **void list_thread(int socket, int shm_id);**
    - List all the thread present in the shared memory

- **void create_thread(char* topic_name, int socket, int shm_id, char* username);**
    - Check if the topics in which the thread will be created already exist (**istopic(topic, shm_id)**) if not check if the thread to be created already exist (**is_thread(thread_name, shm_id)**) if not proceed to create the thread (**new_thread(topic_name, thread_name, username**))

- **int is_thread(char* thread_name, int shm_id);**
    - Check if the thread already exist in the shared memory by comparing the thread name with all the thread already present in the shared memory. Returns **1** if **successful** and returns **0** if **failed**

      .

- **int new_thread(char *topicName, char *threadName, char* username);**
    - Creates a folder in the chosen topic with the name of the thread to be createdReturns **1** if **successful** and returns exit(**0)** if **failed**

- **Threads* instance_thread(Threads new_thread)**
    - Create an instance of a thread

- **Threads* append_thread(Threads* threads, Threads new_thread);**
    - Append instances of threads in a list

Messaging functions

The functionalities related to the message management operations are written in the **message_operatons.c** script

- **void list_message(int socket, int shm_id);**
    - Lists all the message in a given thread, first it checks if the thread exist, if yes, it proceeds to print all the messages in that thread

- **Messages* instance_message(Messages new_message);**
    - Creates an instance of a message, returns a **Messages** "object"

- **Messages* append_message(Messages* messages, Messages new_message);**
    - Append an instance of a message in a list

- **void write_messgae(char* username, int socket, int shm_id);**
    - Writes a message to a thread , first it checks if the thread exists if yes , finds it in the corresponding topic then. Then increment the number int the **counter.txt** file in order to assign that new number as a message id (**increment_id()**) and to finish aska the user to enter the message, which going to be save calling the function **save_message(path, new_message)**

- **int save_message(char* path, Messages message);**
    - Simply create the file containing the message written by the user, returns **1** if **successful** and **0** if it **fails**

- **Messages read_message(char* path);**
    - Read the content of the message file , returns an instance of **Messages with message.id = 0** if **successful** or an instance of **MEssages** with **message.id = -1** if it **fails**

- **void reply_message(char* username, int socket, int shm_id);**
    - Finds the message id entered by the user calling the function **get_message_id(target_id, shm_id)** if the id exists then proceeds by collecting the message information in order to build a path to the message itself ***sprintf(path, "../topics/%s/%s", fnd_message.topic_name, fnd_message.thread_name);*** and set the message **status** to **1** (**message read**), *then increments the counter for a new message id (id for the reply to the message) ad proceed by writing the reply calling the function* ***save_message(path, new_message)***

- **CMessages get_message_id(int target_id, int shm_id);**
    - Creates an instance of **CMessages** which consist in a message "objet" with all the information associated to that message (id, author, content, topic, thread,) and then checks if there is a message with the **target_id** given in input if no returns the **instance** with an **id = -1** otherwise returns the **founded message structure**

- **void message_status(int socket, int shm_id);**
    - Check the status of the message if read or less. If the status is set to 1 the message has been read otherwise the message hasn't been read yet.

- **void print_message_informations(int socket, int shm_id);**
    - Find a message by his id then create an instance of **CMessage** then print all the informations related to the message

Subscribing functions

The functionalities related to the subscription management operations are written in the **threadOperatons.c** script

- **void subscribe(char* username, int socket, int shm_id);**
    - Subscribes the user to a topic, first check if the topic exist calling the function **is_topic(topic_name, shm_id),** next locks the shared memory in order to create effectively add the subscriber to the topic using the function **add_subscriber(topic_name, username)** then release the shared memory

- **int add_subscriber(char* topic_name, char* username);**
    - Creates a file **subscribers.txt** in the topic folder , containing the subscribers to the topic, first it checks if the user is already subscribed to the topic if **no** returns **2 (already subscribed)** otherwise returns **1 (subscribed)**

- **Subscribers* print_subscribers(char* path);**
    - Red the file subscribers.txt, create subscribers instance then add them to a lis of subscribers calling the function **append_subscribers(subs_list, token)** then **returns the list of subscribers**

- **Subscribers* instance_subscriber(char* subscriber);**
    - Create an instance of **Subscribers** then **returns** it

- **int is_subscribed(int fd, char* username);**
    - Check if the user is subscribed to a topic

- **Subscribers* append_subscribers(Subscribers* subscribers, char* username);**
    - Appends instance of subscriber in a list of subscribers

Utilities

Some useful functions is the **utility.c** script

- **void menu(int socket); (Not used)**
    - Suppose to print the menu

- **void get_input(char* username, char* password, int socket);**
    - Used to read the input from the user in the registration and authentication fases.

- **int get_id();**
    - Return the current id number to be assigned to a message from the **counter.txt file,** then **returns the current id**

- **int increment_id();**
    - Increment the current id number to create a new id number for the next message, and returns the new id otherwise returns **-1**

- **void list(int socket, int shm_id);**
    - Prompt the sub menu to choose between the listing of the topics, thread message or all of them, respectively activated by the operation numbers **1**, **2**, **3** and **4** and at each number correspond respectively a call to the following functions **list_topic(socket, shm_id), list_thread(socket, shm_id), list_message(socket, shm_id)** and **map_whiteboard(shm_id, socket);**

- **int check_conn(int conn_status);**
    - Check if the user is still connected to the server, returns **0** if **not** otherwise returns **1**

- **void map_whiteboard(int shmId, int socket);**
    - Scan recursively the topic folder then print in order all the topics, with their respectives message and threads

- **void shm_clear();**
    - Clear the shared memory at **KEY_SHM** with the parameter **IPC_RMID** of the shmctl function

Nota bene:

All the include, **define, structures and function headers** are presents in the file **serverUtils.h**

BUGS: sometimes you need to restart the client if a command is not working.