Université Sultan Moulay Slimane
Ecole Nationale des Sciences Appliquées
Khouribga

Génie Informatique

2eme Année du Cycle ingénieur

Programmation Systèmes et Réseau

Mini-Projet

Développement d' un client/serveur

- Réalisé par :

ZKIM Youssef

LEMLIH Aymane

- Encadré par :

Pr. SAADI MUSTAFA

Année universitaire : 2020/2021

Remerciement :

Nous tenons à remercier non seulement comme devoir mais par grand respect et gratitude profonde notre professeur Dr. SAADI MUSTAFA à qui nous adressons nos sentiments de reconnaissance et de respect pour nous avoir guidé dans l'élaboration de ce travail, avec la patience et le dynamisme qui la caractérisent, son soutien, ainsi ses directives précieuses durant le déroulement du projet.

Table de matières

Description du projet :

L'objectif principal de ce projet est de fournir un couple de programmes, un client et un serveur, permettant de simuler des « jeux en lignes » très simples. L'idée est basée sur le principe de questions basiques, posées aux clients, auxquelles ceux-ci doivent répondre pour essayer de gagner.

Le présent rapport décrit les différentes étapes de réalisation de ce projet.

Présentation des diffèrent chapitre traite dans notre rapport

Pour un bon travail il nous faut un rapport bien structuré qui peut être exploité après la mise en place de ce site, pour cela nous allons organiser notre présent rapport de la manière suivante :

Dans le premier chapitre « Introduction Générale », nous allons mettre notre projet dans son cadre général en définissant Oracle et en présentant le sujet.

Dans le deuxième chapitre « Conception et réalisation » nous abordons la phase de conception.

Enfin et au niveau du cinquième et dernier chapitre intitulé « Réalisation », nous allons présenter notre site web, en mentionnant les différents environnements de travail matériels et logiciels utilisés pour entamer le projet, ainsi qu'en citant des remarques sur les fonctions utiliser dans la programmation du notre projet

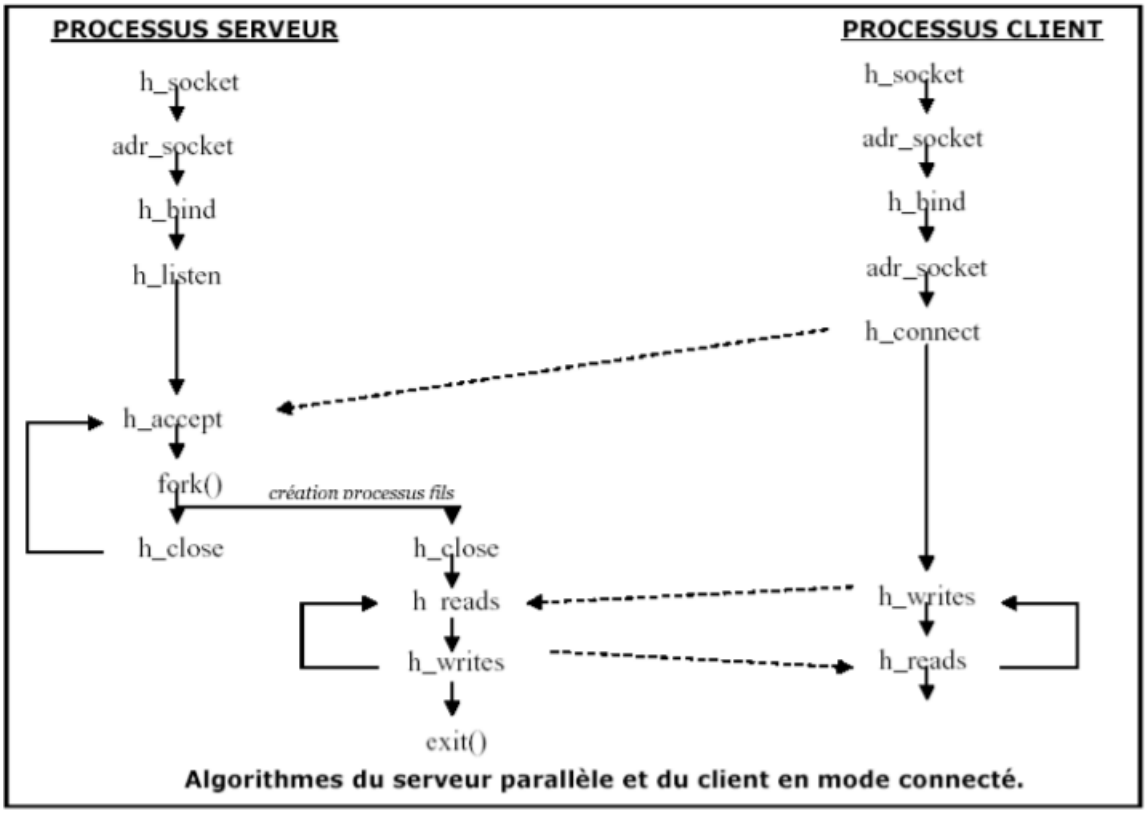
Chapitre 1. Introduction Générale :

Introduction :

En programmation réseau, nous allons utiliser les sockets. C'est un moyen de communication qui se définit par un port et une adresse. Elle est représentée sous la forme d'un descripteur de fichier (un entier). Les fonctions permettant de manipuler des sockets prennent en argument une structure d'adresse de socket.

Fonctionnement d'un client et d'un serveur TCP :

- **Les clients :** ils prennent l'initiative du début de la communication (demande d'ouverture de connexion, envoi de requête, attente de réponse)
- **Les serveurs :** ils sont en permanence à l'écoute, en attente de demande de communications (attente de demande d'ouverture de connexion, de requête, émission de réponse). La procédure générale de dialogue entre un serveur et son ou ses clients est la suivante :
    1. le serveur initialisé se place en écoute (attente de requête client) ;
    2. le client émet une requête à destination du serveur. Il demande un service ou l'accès à une ressource ;
    3. le serveur renvoie une réponse au(x) client(s). Il rend le service ou octroie la ressource, éventuellement il engage même un dialogue assez conséquent avec son (ses) client(s) ;
    4. le serveur se replace en écoute (attente de nouvelle requête client).

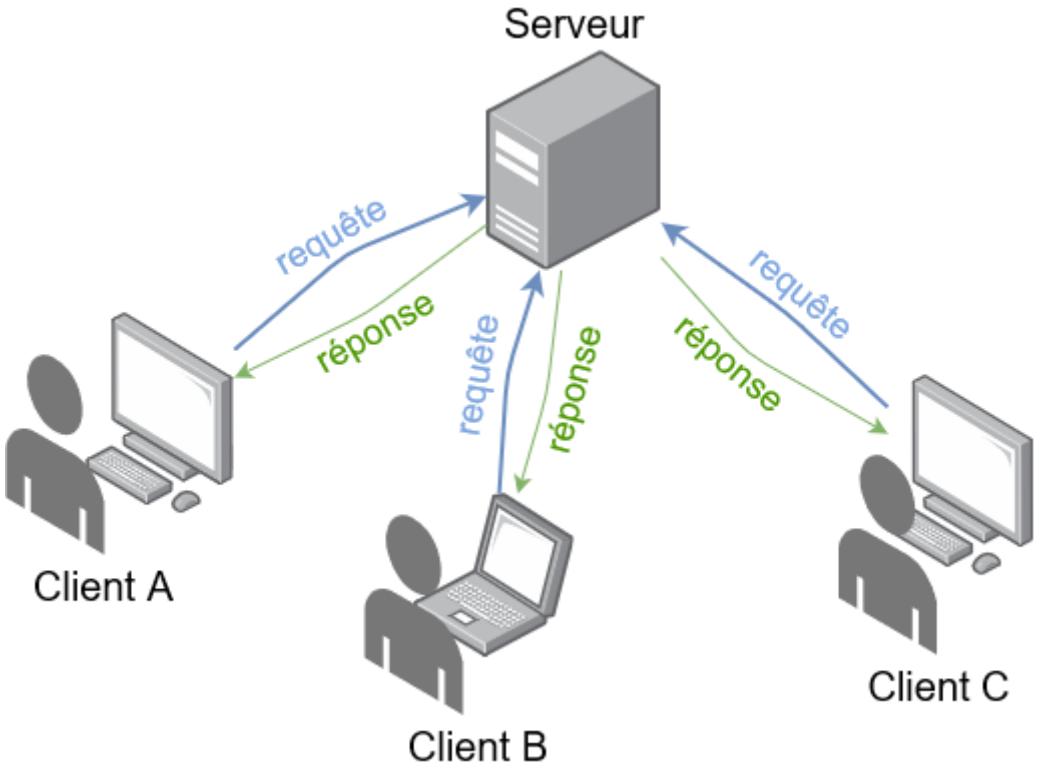


Algorithmes du serveur parallèle et du client en mode connecté.

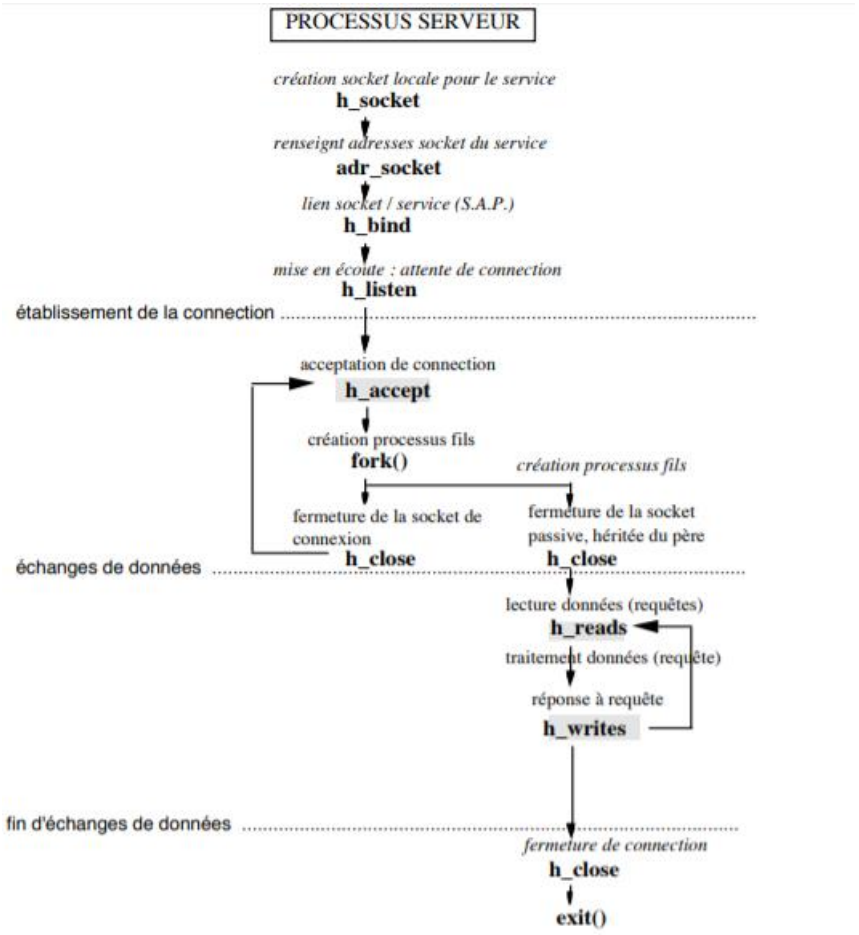
Schéma du notre projet :

Pour qu'on puisse mieux comprendre notre projet on va travailles avec des schémas alors dans notre projet il nous faut de programmer deux programme, le premier sera servir le côté serveur et l'autre pour le côté.

Après les clients vont essayer de se connecter au serveur, Donc on peut déduire que le schéma suivant décrit ceci.



Maintenant on peut décrira notre projet d'une autre façon tell que les fonctions principales et le séquencement de leurs utilisations pour rendre notre projet exécutable.



Threads :

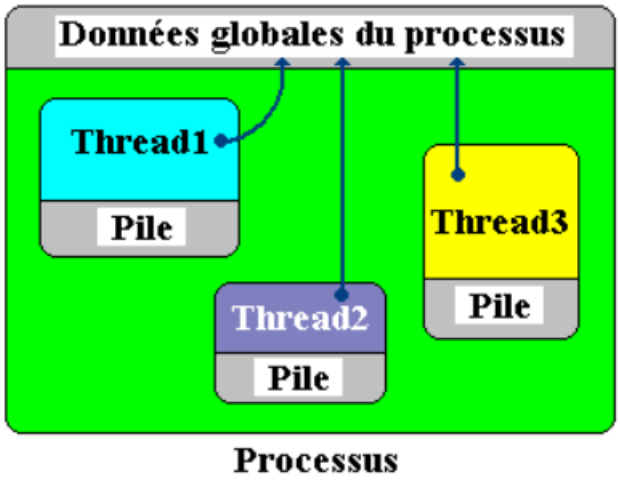
Dans notre projet on a utilisé les threads car l'utilisation des processus nous ramené à un autre problème qui rendre notre travailler difficile et aussi a une application de communication de qui peuvent ne fonctionne pas dans certain cas, Alors pour remédier ce problème major on est pensé au Threads pour les avantages qui va nous fournir.

- Premièrement on peut savoir facilement le processus d'un ou plusieurs threads à la fois ceci pour rendre la communication entre le serveur et le client facile à gère et aussi de recevoir d'une manier très facile les valeurs du thread vers le processus correspondant savoir sans utiliser plusieurs processus fils à l'aide de fork.
- Finalement, on peut minimiser le nombre des processus fils crée lors d'Une connexion multiple avec le serveur qui nous ramené à un code lisible, facile à le modifier et optimale.

Un processus travaille et gère, pendant le quantum de temps qui lui est alloué, des ressources et exécute des actions sur et avec ces ressources. Un thread constitue la partie exécution d'un processus alliée à un minimum de variables qui sont propres au thread.

- Un processus peut comporter plusieurs threads.
- Les threads situés dans un même processus partagent les mêmes variables générales de données et les autres ressources allouées au processus englobant. (Ce qui est très intéressant pour notre cas)
- Un thread possède en propre un contexte d'exécution (registres du processeur, code, données)

Cette répartition du travail entre thread et processus, permet de charger le processus de la gestion des ressources (fichiers, périphériques, variables globales, mémoire,) et de dédier le thread à l'exécution du code proprement dit sur le processeur central (à travers ses registres, sa pile lifo etc…).



Les fonctions nouvelles utiliser dans notre projet :

```
int pthread_create (
        pthread_t *thread ,
        pthread_attr_t *a t tr ,
        void *(* s tar t _rou tine ) (void *) , void *arg ) ;
void pthread_exit (void * re tval ) ;
int pthread_join ( pthread_t th , void ** thread_return ) ;
```

La fonction pthread_create demande le lancement d'un nouveau processus léger, avec les attributs indiqués par la structure pointée par attr (NULL = attributs par défaut). Ce processus exécutera la fonction start_routine, en lui donnant le pointeur arg en paramètre. L'identifiant du processus léger est rangé à l'endoit pointé par thread.

Ce processus léger se termine (avec un code de retour) lorsque la fonction qui lui est associée se termine par return retcode, ou lorsque le processus léger exécute un pthread_exit (retcode).

La fonction pthread_join permet au processus père d'attendre la fin d'un processus léger, et de récupérer éventuellement son code de retour

Chapitre 2. Réalisation du Mini-Projet:

Cote serveur :

On commence tout d'abbord par les include que le programme demande pour son bon fonctionnement :

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

#include <netinet/in.h>

#include <arpa/inet.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <errno.h>

#include <string.h>

#include <pthread.h>

#include <sys/types.h>

#include <signal.h>



#define NB_MAX_CLIENTS 5

#define BUFFER_SZ 2048



static _Atomic unsigned int cli_count = 0;

static int uid = 10;
```

Passant maintenant a la structure du client : elle contient adresse du client, sa socket son uid, son nom, le quiz qu'il a choisi, le numero de la question et son score

```
typedef struct{

  struct sockaddr_in address;

  int sockfd;

  int uid;
```

```
    char name[50];
    int Score;
    int qwiz_num;
    int num_question;
} client_t;

client_t *clients[NB_MAX_CLIENTS];
pthread_mutex_t clients_mutex = PTHREAD_MUTEX_INITIALIZER;
```

l'instruction client_t *clients[NB_MAX_CLIENTS]; défini le nombre maximale des clients ayant le droit de se connecter avec le serveur.

Fonctions et procedures utilisées

Free_buffer

```
void free_buffer() {
  printf("\r%s", "> ");
  fflush(stdout);
}
```

Cette fonction permet de liberer le buffer apres utilisation.

str_trim_lf

```
void str_trim_lf (char* arr, int length) {
 int i;
 for (i = 0; i < length; i++) { // trim \n
  if (arr[i] == '\n') {
   arr[i] = '\0';
   break;
  }
 }
}
```

Cette fonction permet de liberer le buffer apres utilisation.

Print_client_addr

```
void print_client_addr(struct sockaddr_in addr){
  printf("%d.%d.%d.%d",
    addr.sin_addr.s_addr & 0xff,
    (addr.sin_addr.s_addr & 0xff00) >> 8,
    (addr.sin_addr.s_addr & 0xff0000) >> 16,
    (addr.sin_addr.s_addr & 0xff000000) >> 24);
}
```

Cette fonction permet de liberer le buffer apres utilisation.

Ajout et effacement des clients

```
/* ajouter clients  */
void queue_add(client_t *cl){
        pthread_mutex_lock(&clients_mutex);

        for(int i=0; i < NB_MAX_CLIENTS; ++i){
                if(!clients[i]){
                        clients[i] = cl;
                        break;
                }
        }

        pthread_mutex_unlock(&clients_mutex);
}

/* effacer les clients */
void queue_remove(int uid){
        pthread_mutex_lock(&clients_mutex);

        for(int i=0; i < NB_MAX_CLIENTS; ++i){
                if(clients[i]){
                        if(clients[i]->uid == uid){
```

```
                                        clients[i] = NULL;
                                        break;
                                }
                        }
                }

                pthread_mutex_unlock(&clients_mutex);
}
```

Ces fonctions permet l'ajout et l'efacement des clients

Envoie de tous les clients sauf l'emetteur

```
/* Send message to all clients except sender */
void send_message(char *s, int uid){
        pthread_mutex_lock(&clients_mutex);

        for(int i=0; i<NB_MAX_CLIENTS; ++i){
                if(clients[i]){
                        if(clients[i]->uid != uid){
                                if(write(clients[i]->sockfd, s, strlen(s)) < 0){
                                        perror("Erreur dans l'écriture dans le fichier descripteur");
                                        break;
                                }
                        }
                }
        }

        pthread_mutex_unlock(&clients_mutex);
}
```

Communication avec client

Toute la communication avec le client est assurer par la fonction suivante

```
/* Handle all communication with the client */
```

```
void *handle_client(void *arg){
	char buff_out[BUFFER_SZ];
	char name[50];
	int leave_flag = 0;
	int cnt;

	cli_count++;
	client_t *cli = (client_t *)arg;

	// Name
	if(recv(cli->sockfd, name, 50, 0) <= 0 || strlen(name) <  2 || strlen(name) >= 50-1){
		printf("N'as pas entrez son nom.\n");
		leave_flag = 1;
	} else{
		strcpy(cli->name, name);
		sprintf(buff_out, "%s viens de rejoindre\n", cli->name);
		printf("%s", buff_out);
		send_message(buff_out, cli->uid);
	}
	bzero(buff_out, BUFFER_SZ);
	while(1){
		if (leave_flag)
			break;
		int receive = recv(cli->sockfd, buff_out, BUFFER_SZ, 0);
		if (receive > 0){
			if(strlen(buff_out) > 0){
				send_message(buff_out, cli->uid);
				str_trim_lf(buff_out, strlen(buff_out));
				if ((cnt = 0)){
					if (strcmp(buff_out, "1")){
						cli->qwiz_num = 1;
```

```
                                        cli->num_question = 0;
                                        cli->Score = 0;
                                        printf(".... \n");
                                }
                                if (strcmp(buff_out, "2"))
                                        cli->qwiz_num = 2;
                                if (strcmp(buff_out, "3"))
                                        cli->qwiz_num = 3;
                                cnt++;
                        }
                        if ((cli->qwiz_num = 1)){
                                if (strcmp(buff_out, "C") == 0 && cli->num_question == 0)
                                        cli->Score++;
                                if (strcmp(buff_out, "C") == 0 && cli->num_question == 1)
                                        cli->Score++;
                                if (strcmp(buff_out, "D") == 0 && cli->num_question == 2)
                                        cli->Score++;
                                if (strcmp(buff_out, "A") == 0 && cli->num_question == 3)
                                        cli->Score++;
                                cli->num_question++;
                        }
                        printf("%s -> %s ->score : %d\n", buff_out, cli->name, cli->Score);
                }
        } else if (receive == 0 || strcmp(buff_out, "exit") == 0){
                sprintf(buff_out, "%s viens de quitter\n", cli->name);
                printf("%s", buff_out);
                send_message(buff_out, cli->uid);
                leave_flag = 1;
        } else {
                printf("Erreur: -1\n");
                leave_flag = 1;
```

```
                }


                bzero(buff_out, BUFFER_SZ);
        }
```

Fonction main

```
int main(int argc, char **argv){
        if(argc != 2){
                printf("Usage: %s <port>\n", argv[0]);
                return EXIT_FAILURE;
        }

        char *ip = "127.0.0.1";
        int port = atoi(argv[1]);
        int option = 1;
        int listenfd = 0, connfd = 0;
        struct sockaddr_in serv_addr;
        struct sockaddr_in cli_addr;
        pthread_t tid;

 /* Socket settings */
 listenfd = socket(AF_INET, SOCK_STREAM, 0);
 serv_addr.sin_family = AF_INET;
 serv_addr.sin_addr.s_addr = inet_addr(ip);
 serv_addr.sin_port = htons(port);

 /* Ignore pipe signals */
        signal(SIGPIPE, SIG_IGN);

        if(setsockopt(listenfd, SOL_SOCKET,(SO_REUSEPORT | SO_REUSEADDR),(char*)&option,sizeof(option)) < 0){
                perror("ERROR: dans setsockopt");
```

```
 return EXIT_FAILURE;
        }

        /* Bind */
if(bind(listenfd, (struct sockaddr*)&serv_addr, sizeof(serv_addr)) < 0) {
 perror("ERROR: bind n'as pas réussi");
 return EXIT_FAILURE;
}

/* Listen */
if (listen(listenfd, 5) < 0) {
 perror("ERROR: ecoute n'as pas réussi");
 return EXIT_FAILURE;
        }

        //_____________________________________________________________________________

        printf("\n\n ************************ QUIZ LEMLIH ZKIM / SERVER SIDE ************************");
        printf("\n  ------------------------------------------------------------------------------");
        printf("\n          ..... Les rép des clients se trouvent ici .....\n");

printf("\n\n ****************************** BIENVENU ADMIN ***********************************\n\n");

        while(1){
                socklen_t clilen = sizeof(cli_addr);
                connfd = accept(listenfd, (struct sockaddr*)&cli_addr, &clilen);

                /* Check if max clients is reached */
                if((cli_count + 1) == NB_MAX_CLIENTS){
                        printf("Le nombre maximal des clients atteint. Rejeté: ");
                        print_client_addr(cli_addr);
```

```
                printf(":%d\n", cli_addr.sin_port);
                close(connfd);
                continue;
            }

            /* Client settings */
            client_t *cli = (client_t *)malloc(sizeof(client_t));
            cli->address = cli_addr;
            cli->sockfd = connfd;
            cli->uid = uid++;

            /* Add client to the queue and fork thread */
            queue_add(cli);
            pthread_create(&tid, NULL, &handle_client, (void*)cli);

            /* Reduce CPU usage */
            sleep(1);
        }

        return EXIT_SUCCESS;
}
```

Notre fonction main cree la socket et assure le bind et l'accept du connexion

Cote Client :

Début

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include <unistd.h>
```

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

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <pthread.h>


#define LENGTH 2048


// Global variables

volatile sig_atomic_t flag = 0;//exit****************************************

int sockfd = 0;

char name[32];


void choisir_level(){

   printf("==> Choisissez le niveau : <== \n");

   printf("[+] Niveau 1 : Facile \n");

   printf("[+] Niveau 2 : Moyen \n");

   printf("[+] Niveau 3 : Difficile \n");

}
```

Fonctions pour gestion des chaines de carac

```
void free_buffer() {

 printf("%s", "> ");

 fflush(stdout);//libération des données dans le buffer

}


void str_trim_lf (char* arr, int length) {//chaine de caractére + strlen

 int i;

 for (i = 0; i < length; i++) { // trim \n
```

```
  if (arr[i] == '\n') {
    arr[i] = '\0';
    break;
  }
 }
}

void catch_ctrl_c_and_exit(int sig) {
  flag = 1;
}
```

La premiere fct permet de liberer le buffer

La deuxieme c'est pour lire jusqu'au fin de ligne et arreter

La 3eme c'est pour quitter si CTRL + C

Recuperation du niveau choisis et envoie du nom et niveau choisis

```
//appeler par la fonction pthread_create
void send_answers() {//Récupérer choix niveau et envoie du nom et niveau au serveur (envoie via socket)
        char message[LENGTH] = {};
        char buffer[LENGTH + 32] = {};
        int j, quiz_num = 0;
        int cnt = 0;

        FILE * fp;
        char * line = NULL;
        size_t len = 0;
        ssize_t read;
        fp = fopen("niveau2.txt", "r");
        if (fp == NULL)
```

```
                exit(EXIT_FAILURE);


        free_buffer();
        if ((cnt = 0)){
                free_buffer();//libérer les données dans la fonction
                while (1){//Spécifier le nombre du level (quiz)
                        fgets(message, LENGTH, stdin);
                        if (strcmp(message, "1")){
                                quiz_num = 1;
                                break;
                        }else if (strcmp(message, "2")){
                                quiz_num = 2;
                                break;
                        }else if (strcmp(message, "3")){
                                quiz_num = 3;
                                break;
                        }else
                                printf("il y a que 3 options 1,2 ou 3 !!!");
                }
                cnt++;
        str_trim_lf(message, LENGTH);
        sprintf(buffer, "%s: %s\n", name, message);
        send(sockfd, buffer, strlen(buffer), 0);
        }


while(1) {
        free_buffer();
 fgets(message, LENGTH, stdin);
 str_trim_lf(message, LENGTH);


 if (strcmp(message, "EXIT") == 0) {
```

```
                                        break;
  }else{
                sprintf(buffer, "%s: %s\n", name, message);
                                //lvl_question(quiz_num, j++);
                                read = getline(&line, &len, fp);
                printf("%s", line);
                send(sockfd, buffer, strlen(buffer), 0);
  }
                bzero(message, LENGTH);
  bzero(buffer, LENGTH + 32);
 }
 catch_ctrl_c_and_exit(2);
}
```

Reception du message handler

```
void recv_msg_handler() {
                char message[LENGTH] = {};
 while (1) {
                int receive = recv(sockfd, message, LENGTH, 0);
  if (receive > 0) {
    printf("%s", message);
    free_buffer();
  } else if (receive == 0) {
                                        break;
  } else {
                                        // -1
                }
                memset(message, 0, sizeof(message));
 }
}
```

Fonction main

```
int main(int argc, char **argv){
	if(argc != 2){
		printf("Usage: %s <port>\n", argv[0]);
		return EXIT_FAILURE;
	}
	char *ip = "127.0.0.1";//adress of localhost (local)
	int port = atoi(argv[1]);//Cette fonction permet de transformer une chaîne de caractères, représentant une valeur entière, en une valeur numérique de type int.

	signal(SIGINT, catch_ctrl_c_and_exit); // si client tape Ctr-C ===> exit the execution

	printf("Merci de tapez votre nom : ");
	fgets(name, 32, stdin);//stdin ==> entrée standard = clavier
	str_trim_lf(name, strlen(name));//tapez entrer ===> take name
	// testez si la taille du nom est inf à 2 ou sup à 32
	if (strlen(name) > 50 || strlen(name) < 2){
		printf("Le nom doit avoir entre 2 et 50 caractères !! .\n");
		return EXIT_FAILURE;//quitter l'exécution
	}

	struct sockaddr_in server_addr;

	/* Socket settings */
	//remplir les infos de server_addr
	sockfd = socket(AF_INET, SOCK_STREAM, 0);
	server_addr.sin_family = AF_INET;
	server_addr.sin_addr.s_addr = inet_addr(ip);
	server_addr.sin_port = htons(port);
```

```
// Connect to Server
        int connection = connect(sockfd, (struct sockaddr *)&server_addr, sizeof(server_addr));
        if (connection == -1) {
                printf("Erreur dans la connexion\n");
                exit(1);
        }

        // Send name
        send(sockfd, name, 32, 0);

        printf("\n\n ************************ Avant de commancer le quizz ************************");
        printf("\n  ---------------------------------------------------------------------------------");
        printf("\n                ..... Directives .....\n");
        printf("\n [+] Vous aurez un total de 3 questions, les questions sont choisis aléatoirement");
        printf("\n [+] Vous devez choisir la réponse juste en choisissanr A, B ou C");
        printf("\n [+] Un bonus de 5 points pour chaque réponse juste");
        printf("\n [+] le score ne sera pas débité en cas de fausse réponse");

printf("\n\n ****************************** BONNE CHANCE ********************************\n\n");
        choisir_level();///===> afficher des messages pour choisir un level
        pthread_t msg_in_thread_from_client;// Création de la variable qui va contenir le thread
        if(pthread_create(&msg_in_thread_from_client, NULL, (void *)send_answers, NULL) != 0){//create the thread
                        printf("Erreur dans la création du thread\n");
                return EXIT_FAILURE;
        }

        pthread_t msg_in_thread_from_server;
        if(pthread_create(&msg_in_thread_from_server, NULL, (void *) recv_msg_handler, NULL) != 0){
                printf("Erreur dans la création du thread\n");
                return EXIT_FAILURE;
```

```
        }


        while (1){
                if(flag){
                        printf("\nSignal de quittance CTRL+C Reçu, au revoir...\n");
                        break;
        }
        }
        close(sockfd);
        return EXIT_SUCCESS;
}
```

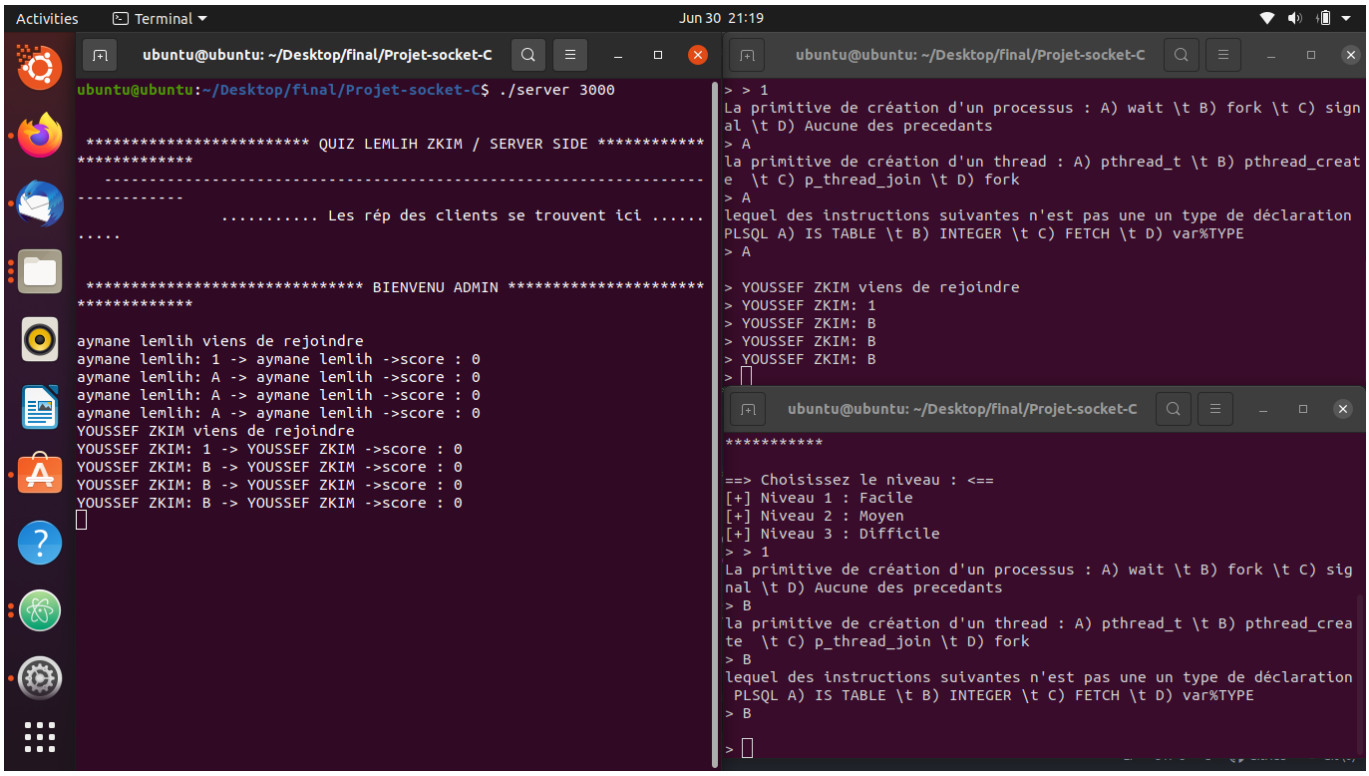
La fonction main contient le menu qui s'affiche pour l user et asssure la communication client serveur en créant les element necessaire pour cette conneection (socket, adresse serveur …) et envoie les réponses du Client au serveur

REMARQUE

La connection en mode single est la même procedure que la connection mode multi la seul difference c'est qu'un seul terminal client est ouvert dans le mode single

Execution

L'execution du programme est comme suit il suffit d'ouvrir un terminal serveur avec le port et des terminals clients avec le meme port



Conclusion :

Pour conclure, nous ne pouvons que nous estimer fiers d'avoir contribuer à la réalisation de ce projet, qui nous a été d'une grande aide et dont nous nous sommes bien enrichis.

Ce projet nous a été d'une aide capitale pour assimiler les objectifs du module. Et nous a donné une opportunité d'appliquer les notions étudiées en cours, ni au niveau de création des processus et les sockets ou la réalisation d'une communication serveur clients ceci va nous servir autant que des administrateur système dans le future est aussi. Sans oublié le contact avec les threads et les comprendre plus grâce a ceci

Ce travail nous a permis de développer notre sens d'analyse et de réflexion. En surmonté tous les obstacles qui nous ont fait face, nous avons pu bénéficier d'une expérience qui nous sera d'une aide capitale dans nos carrières professionnelles.