



UNIVERSITÉ ABDELMALEK ESSAADI
FACULTÉ DES SCIENCES ET TECHNIQUES
DEPARTEMENT GENIE INFORMATIQUE
TANGER



Filière : Cycle licence en Génie Informatique (S5)
Module : Architecture C/S

Rapport du projet : Application Annuaire de contacts (Multi-Client/Serveur)

Réalisé par :
El Mesaouri Tarik

Encadré par :
M. Ait Kbir

Table des matières

Introduction	4
But du Projet	5
Les Fonctions de base du socket	6
Coté Client	6
1. Création du socket Client	6
2. La primitive connect()	6
3. Les fonctions de dialogue Client/Serveur	7
4. La primitive close()	7
Coté Serveur	8
Création du socket serveur	8
1. La primitive bind()	8
2. La primitive listen()	9
3. La primitive accept()	9
4. Les fonctions de dialogue Client/Serveur	10
5. La primitive close()	10
Fonctionnement de notre programme	11
Coté Client	11
Coté Serveur	13
1. Le cas d'un administrateur.....	16
Menu principal d'un administrateur.....	16
Lister les contacts.....	17
Lister les contacts selon leurs noms.....	18
Lister les contacts selon leurs villes.....	19
Rechercher un contact.....	20
Ajouter un contact.....	21
Modifier un contact.....	21
Supprimer un contact.....	23

Supprimer tous les contacts.....	24
Manuel d'utilisation.....	25
Se connecter en tant qu'utilisateur.....	25
Menu apres fonction.....	26
Exit.....	26
2. Le cas d'un utilisateur	27
Conclusion	28

Introduction

L'architecture client-serveur, est une architecture logicielle dans laquelle les programmes d'application, dits *clients*, font appel, dans le cadre d'un réseau, à des services génériques distants fournis par des ordinateurs appelés *serveurs*. On parle d'architecture de un à trois niveaux. Internet propose lui une architecture client-serveur "Multi-niveaux".

Les serveurs sont des ordinateurs dédiés au logiciel serveur qu'ils abritent, et dotés de capacités supérieures à celles des ordinateurs personnels en termes de puissance, d'entrées/Sorties et de connexions réseau.

Les clients sont souvent des ordinateurs personnels ou des appareils individuels (smartphone, smart tv, tablette), mais différent du fonctionnement d'un serveur. Ce dernier peut répondre aux requêtes d'un grand nombre de clients. En fait, il faut un certain nombre de choses pour que deux consoles puissent communiquer entre eux. Ils ont besoin au minimum des trois éléments suivants : Un protocole de communication commun.
Une adresse IP ; Un port libre et ouvert ;

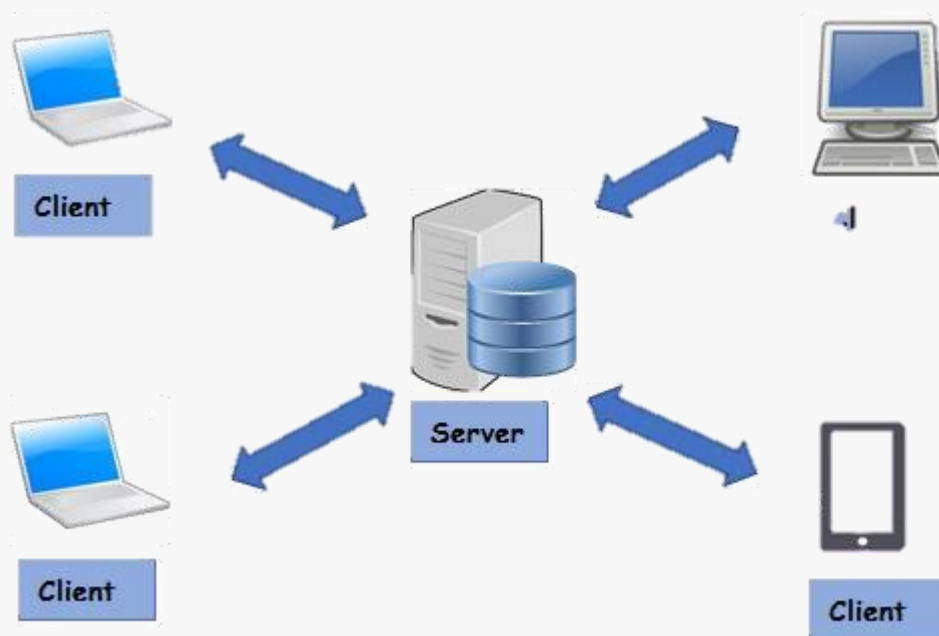
Les sockets sont des points d'accès de bas niveau par lequel un processus peut envoyer ou recevoir des données. Le socket se présente sous la forme d'un descripteur .Les sockets sont une interface d'accès au réseau au-dessus du service transport.

Un réseau permet de relier des terminaux entre eux afin qu'ils puissent s'échanger des informations à but de communiquer. Le plus connu des réseaux est Internet qui est un outil de communication utilisé au niveau mondial. Mais un réseau peut très bien être beaucoup plus petit, comme les réseaux LAN qu'on utilise dans notre maison ou dans les sociétés ou universités, généralement la liaison entre deux machines se fait à l'aide d'un câble RJ45.il y a aussi d'autres types de réseaux chacun a son unique utilisation par exemple: le réseau PAN et MAN et RAN...

But du Projet

Le but de ce projet est de réaliser un serveur de gestion de fichier en développant deux applications respectant l'architecture Client/serveur. En effet, le serveur gère une liste de contacts et les fichiers de ces derniers. Concernant le client, il permet à l'aide de deux codes d'accès, un pour les invités et un pour les administrateurs.

Il permet aux administrateurs de lister, ajouter, modifier, supprimer les contacts, et aux invités de se connecter au serveur, de consulter et recherche les contacts.



Les Fonctions de base du socket :

Coté Client

1. Création du socket Client

La primitive **socket()** crée un socket rattachée à la machine qui la crée.

```
sClient=socket(AF_INET,SOCK_STREAM,IPPROTO_TCP);
```

AF_INET : Protocoles Internet

SOCK_STREAM : Utilisé en mode connecté au-dessus de TCP.

IPPROTO_TCP : Utilisation du mode TCP

2. La primitive connect()

La primitive **connect()** en mode connecté établit la connexion avec une adresse de destination.

```
ret_fct=connect(sClient,(struct sockaddr*)&addrServeur,sizeof(struct sockaddr));
```

sClient : numéro du socket créé, renvoyé par la primitive **socket()**.

Struct sockaddr &addrServeur : pointeur vers une structure générique de type **struct sockaddr** contenant l'adresse du serveur.

Sizeof (struct sockaddr_in) : taille de la zone réservée pour la variable pointée par **addrServeur**

3. Les fonctions de dialogue Client/Serveur

Les primitives `write()`, `read()` ont pour but d'aller respectivement écrire `write()` et lire `read()` des octets dans un socket.

```
longMessage=read(sClient,msg, TAILLEMESSAGE+1);  
write(sClient,msg,strlen(msg)+1);
```

sClient : numéro du socket créé, renvoyé par la primitive `accept()`

msg : variable contenant le message à envoyer ou à lire

TAILLEMESSAGE : Taille de message à envoyer ou à lire

4. La primitive `close()`

Fermer le socket Client

```
close(sClient);
```

sClient : numéro du socket créé, renvoyé par la primitive `socket()`.

Coté Serveur

1. Création du socket serveur

La primitive `socket()` crée un socket rattachée à la machine qui la crée.

```
sServeur = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
```

AF_INET : Protocoles Internet

SOCK_STREAM : Utilisé en mode connecté au-dessus de TCP.

IPPROTO_TCP : Utilisation du mode TCP

2. La primitive bind()

La primitive **bind()** rattache le socket créé au réseau. Ce rattachement est obligatoire pour le serveur.

```
ret_fct = bind(sServeur, (struct sockaddr*)&addrServeur, sizeof(struct sockaddr_in));
```

sServeur : numéro du socket créé, renvoyé par la primitive **socket()**.

Struct sockaddr &addrServeur : pointeur vers une structure générique de type **struct sockaddr** contenant l'adresse du serveur.

Sizeof(struct sockaddr_in) : taille de la zone réservée pour la variable pointée par **addrServeur**

ret_fct : un code de retour égal à 0 en cas de succès et -1 en cas d'erreur.

3. La primitive listen()

La primitive **listen()** prépare un socket à l'écoute des connexions entrants.

```
ret_fct=listen(sServeur,NBRECON);
```

sServeur : numéro du socket créé, renvoyé par la primitive **socket()**.

NBRECON : nombre maximal de demande de connexions.

ret_fct : un code de retour égal à 0 en cas de succès et -1 en cas d'erreur.

4. La primitive accept()

La primitive **accept()** extrait une demande de connexion de la file d'attente. Lorsque la connexion est acceptée, le serveur crée un second socket sur lequel se feront les échanges de données.


```
sClient=accept(sServeur,(struct sockaddr *)&addrSocket,&longAddr);
```

sServeur : numéro du socket créé, renvoyé par la primitive **socket()**.

Struct sockaddr &addrSocket : pointeur vers une structure générique de type contenant l'adresse du serveur.

struct sockaddr & longAddr : taille de la zone réservée pour la variable pointée par **addrSocket**

5. Les fonctions de dialogue Client/Serveur

Les primitives **write()**,**read()** ont pour but d'aller respectivement écrire **write()** et lire **read()** des octets dans un socket.

```
longMessage=read(sClient,msg,TAILLEMESSAGE+1);  
write(sClient,msg,strlen(msg)+1);
```

sClient : numéro du socket créé, renvoyé par la primitive **accept()**

msg : variable contenant le message à envoyer ou à lire

TAILLEMESSAGE : Taille de message à envoyer ou à lire

6. La primitive close()

Fermer le socket serveur et le socket de dialogue

```

        close(sServeur);
        close(sClient);
        return 0;
    }

```

sServeur : numéro du socket créé, renvoyé par la primitive *socket()*.

sClient : numéro du socket créé, renvoyé par la primitive *accept()*.

Fonctionnement de notre programme

Coté Client

```

while(1){

    if(gets(msg)>0){
        strcpy(msg1," ");
        write(sClient,msg,strlen(msg)+1);
        longMessage = read(sClient,msg1,20);
        printf("%s",msg1);
        strcpy(msg1," ");
    }

    longMessage=read(sClient,msg,500);
    if(longMessage>0){
        printf("\n%s\n",msg);
        if(!strcmp(msg,"exit") || !strcmp(msg,"EXIT"))break;
    }
}

```

Comme vous voyez au-dessus le code client ne dépasse pas 80 lignes et son fonctionnement est très simple par rapport au code source du serveur, notre programme **Client** se comporte comme un logiciel de messagerie (*chat avec un serveur*), donc le client ne fait qu'entrer des données (`gets(msg)`) et il les envoie au serveur avec la fonction `write`,

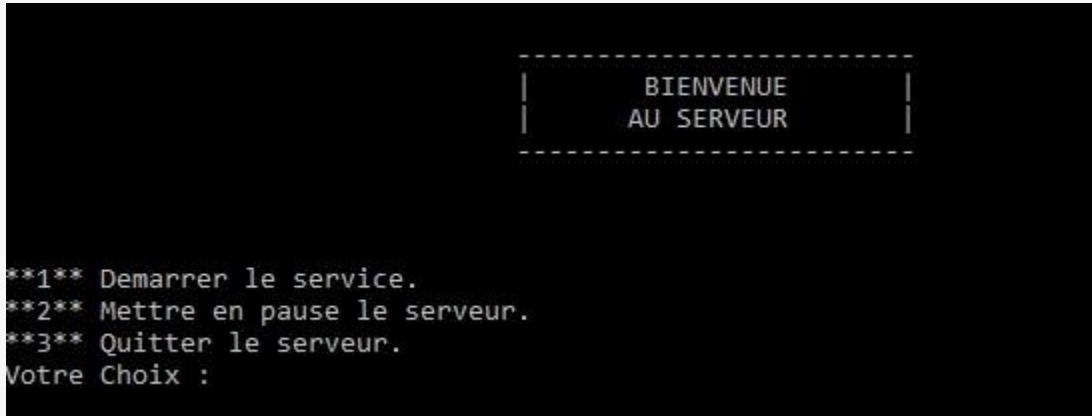
Le serveur lit ce qu'a écrit le client avec la fonction `read` et fait appel aux fonctions nécessaires pour réaliser la demande du client. Une fois les fonctions et les opérations sont réalisées le serveur envoie le résultat au client avec la fonction `write` et le client à son tour lit ce résultat avec la fonction `read` et l'affiche dans le terminal avec `printf`.

Pourquoi on a structuré le programme client de cette manière ?

On a trouvé ce style de programmation d'architecture Client/serveur d'être plus sécurisé car il ne donne au client aucun accès à la base des données, ou déroulement des fonctions, Les codes de l'administrateur et de l'invitée sont déclarés au niveau du serveur donc le client n'aura aucune option de les connaître, sauf si il est l'un des deux. Le client ne fait que de suivre les commandes offert par le serveur. Personne ne peut changer le fonctionnement des fonctions et du code sauf le développeur qui a l'accès au serveur.

Coté Serveur

Le serveur est le centre de notre projet, c'est lui qui gère tous les fonctions et tous les fichiers. Quand on lance l'application serveur il nous demande de taper entrer pour le démarrer cela veut dire démarrer la fonction liste qui permet de préparer le serveur à l'écoute de nouvelles connexions.



```

      BIENVENUE
    AU SERVEUR

**1** Demarrer le service.
**2** Mettre en pause le serveur.
**3** Quitter le serveur.
Votre Choix :
```

Il offre la possibilité de démarrer le service pour allouer à plusieurs clients de se connecter, il peut aussi se mettre en pause jusqu'à la saisie d'un caractère précis, et aussi de quitter le serveur qui, tout simplement, ferme la fenêtre du serveur et arrête le service.

Après avoir établi une connexion avec un client distant ou local, on lui demande d'entrer en code accès, si il saisit un code d'administrateur, Le serveur lui donne la permission de lister, ajouter, modifier, supprimer les contacts, mais si il saisit le code d'invite le serveur lui donne seulement la permission de lister les contacts et de rechercher un contact, s'il saisit un code ni d'administrateur ni d'invité, le serveur ne lui donne aucune permission.

Toutes les opérations faites par un administrateur ou un invité sont stockées dans un fichier texte qui s'appelle **Operations**, ces opérations sont stockées accompagnée de la date et de l'heure de l'opération et avec l'adresse IP du client qui a fait cette opération.

```

void saveoper(char msg[],struct sockaddr_in addr){
    time_t t;
    char msg0[100];

    FILE *f3;
    f3=fopen("Operations.txt","a");
    if (f3 == NULL)
    {
        printf("Error au niveau d'ouvrir le fichier de contact !! \n");
    }
    else
    {
        time(&t);
        while(!feof(f3)){
            if(getc(f3)== EOF){
                fprintf(f3,"\n%s",ctime(&t));
                strcpy(msg0,"IP:");
                strcpy(msg0,(char *)inet_ntoa(addr.sin_addr.s_addr));
                fprintf(f3,"%s",msg0);
                fprintf(f3,"\t%s",msg);

                break;
            }else{
                break;
            }
        }
        fclose(f3);
    }
}

```

Les opérations sont affichées dans le fichier Operations.txt comme ci-dessous :

```

Tue Oct 29 13:35:26 2019
192.168.137.219 L'administrateur vient de se connecter.

Tue Oct 29 13:35:29 2019
192.168.137.219 >420/1//L'administrateur a choisi de lister tout les contacts.

Tue Oct 29 13:35:43 2019
192.168.137.210 >420/7//L'administrateur a choisi de suprmier un contact :El_Mesaouri Tarik

Tue Oct 29 13:35:47 2019
192.168.137.210 >420/1//L'administrateur a choisi de lister tout les contacts.

Tue Oct 29 13:35:51 2019
192.168.137.219 >420/7//L'administrateur a choisi de suprmier un contact :Zeggaf Amine

Tue Oct 29 13:35:57 2019
192.168.137.219 >420/1//L'administrateur a choisi de lister tout les contacts.

Tue Oct 29 13:36:45 2019
192.168.137.219 >420/5//L'administrateur a choisi d'ajouter un contact: El_Msaouri Tarik

Tue Oct 29 13:36:48 2019
192.168.137.219 >420/1//L'administrateur a choisi de lister tout les contacts.

```

Notre serveur a aussi l'avantage de permettre la multi-connexions de plusieurs clients dans le même temps, cette caractéristique de multi-user a été rendue possible grâce à la fonction **fork()**. L'appel système Fork est utilisé pour créer un nouveau processus, appelé **child process**, qui s'exécute simultanément avec le processus qui effectue l'appel fork () (parent process). Après la création d'un nouveau processus enfant, les deux processus exécuteront l'instruction suivante à la suite de l'appel système fork () .

```
pid_t childpid;

while(1){

sClient=accept(sServeur,(struct sockaddr *)&addrSocket,&longAddr);
if(sClient==ERROR){
    afficheErrorSortie("accept()");
    exit(1);}

printf("\nConnexion ...");
aff_addr(addrSocket);
printf("\n");

if ((childpid = fork()) == 0 ) { // On a utiliser fork() pour avoir un multi-client
    close(sServeur);
```

Dans le terminal :

```
Serveur en ecoute...
Connexion ...
L'adresseIP : 192.168.137.50 dans le port : 49316

Connexion ...
L'adresseIP : 192.168.137.219 dans le port : 50192

Connexion ...
L'adresseIP : 192.168.137.210 dans le port : 51655
```

*La connexion de plusieurs clients dans le serveur en utilisant la fonction **fork()***

1. Le cas d'un administrateur

1. Menu Principal d'un administrateur

```
Connexion ...
Votre adresse IP : 127.0.0.1 connectee au port : 5555
```

```
-----
|      BIENVENUE      |
|      CHER CLIENT    |
|-----|
```

```
>>Veuillez entrer le code d'accès:
$> 420
```

```
-----
>>-Menu Administrateur-<<
```

```
** 1 ** pour lister les contacts.
** 2 ** pour lister les contacts selon leurs nom.
** 3 ** pour lister les contacts selon leurs ville.
** 4 ** pour rechercher un contact.
** 5 ** pour ajouter un contact.
** 6 ** pour modifier un contact.
** 7 ** pour supprimer un contact.
** 8 ** pour supprimer tout les contacts.
** M ** pour le manuel d'utilisation.
** S ** pour se connecter en tant qu'utilisateur.
** EXIT ** pour quitter le programme.
```

```
-----
>> Votre choix :
```

Une fois le client tape le code d'accès de l'administrateur (420), un menu est affiché. L'administrateur peut lister les contacts selon leur nom et les filtrer selon leur ville, chercher un contact, ajouter, modifier, ou supprimer un ou tous les contacts. Il peut également voir le manuel d'utilisation pour notre application, changer d'utilisateur, ou il peut aussi choisir de quitter le programme.

Dans les prochaines fonctions, vous allez remarquer que a chaque le variable de retour on le déclare avec **calloc** , un outil d'allocation dynamique des tableaux qui permet de a chaque fois qu'on utilise le variable de le réinitialiser ,au contraire d'allouer dynamiquement avec **malloc** ,la différence entre **malloc** et **calloc** est que **malloc** ne met pas la mémoire à zéro, alors que **calloc** met la mémoire allouée à zéro.

2. Lister les contacts

Si l'administrateur choisit de lister tous les contacts le programme fait appelle à la fonction qui va afficher tous les contacts enregistrées dans un fichier texte des fonctions. Le serveur lit tout le contenu du fichier avec **fread** puis envoie ses données au client en utilisant la fonction **write()** et ce dernier les affiche en respectant la tabulation enregistrée dans le fichier.


```

////////////////////////////////////
// Fonction lister tout les contacts //
////////////////////////////////////

char * consulter_contacts(void){
    FILE* f1;

    char ligne[255];
    char * msg= calloc(1000,sizeof(char));

    f1 = fopen("contacts.txt", "r");
    if (f1 == NULL)
    {
        printf("!! Erreur au niveau de l'ouverture du fichier des contacts !! \n");
    }
    else
    {
        while(!feof(f1))
        {
            fread(msg,1000,30,f1);
        }
        fclose(f1);
    }

    return msg;
}

```

Dans le terminal :

```

-----
>> Votre choix : 1
Nom          PreNom      Numero      Ville      Email
Zeggaf       Amine         0655841012  Tanger     amine.zeggaf99@gmail.com
Medaghri_Alaoui Amine       0691408600  Tanger     aminealaouim98@gmail.com
El_Msaouri   Tarik        0621533465  Tanger     tarik.elmsaouri@gmail.com
Berdey       Yousra       0623133292  Tetouan    youssraberdey@gmail.com
-----

```

3. Lister les contacts selon leurs noms

Cette fonction permet d'utilisateur de chercher un contact selon la première lettre de leur nom ou bien une partie de leur nom. Il donne à l'utilisateur l'option d'entrer la première lettre du nom du contact et cherche tous les noms des contacts qui commencent par cette lettre. La technique utilisée dans cette fonction est une technique simple qui consiste de lire la première lettre de chaque ligne et de la comparer avec la lettre qu'on cherche s'il réussit de trouver la lettre, il enregistre toute la ligne dans une variable de type char, s'il ne trouve aucun contact qui commence avec cette lettre il lui retourne le message qu'il n'existe aucun contact qui commence par cette lettre.


```

char * premierlettre(char first)

{
    char * msg= calloc(1000,sizeof(char));
    char tmp[120]="";
    FILE* f1 ;
    char ligne[100];
    char c;
    long b=0;
    int k=0;
    f1 = fopen("contacts.txt", "r");
    if (f1 == NULL)
    {
        printf("error au niveau d'ouvrir le fichier de contact !! \n");
    }else{
        while(!feof(f1)){
            c=fgetc(f1); //va lire caractere par caractere
            if(c==first ){
                k++;
                b=fseek(f1,-1, SEEK_CUR );
                fgets(ligne,100,f1);
                strcat(msg,ligne);
            }else {
                fgets(tmp,120,f1); strcpy(tmp,"") ; //pour perdu le reste de ligne qui nous s'inte
                continue; }
        }
        if (k==0) strcpy(msg,"Aucun nom de contact commence par cette lettre"); }
    fclose(f1);
    return msg;
}

```

4. Lister les contacts selon leurs ville

Cette fonction permet de l'utilisateur de choisir une ville et puis lister tous les contacts qui appartiennent à cette ville.

```
while(!feof(f1)){
    fscanf(f1,"%s",ligne);          // on lit le nom
    strcpy(tmp,ligne);
    for( i=strlen(tmp);i<17;i++) strcat(tmp," ");

    fscanf(f1,"%s",ligne);          // on lit le prenom
    strcat(tmp,ligne);
    for( i=strlen(tmp);i<32;i++) strcat(tmp," ");

    fscanf(f1,"%s",ligne);          // on lit le telephone
    strcat(tmp,ligne);
    for( i=strlen(tmp);i<44;i++) strcat(tmp," ");
    fscanf(f1,"%s",ligne);          // on lit la ville

    if(strcmp(ligne,ville)==0 ){
        k++;

        strcat(tmp,ligne);
        for( i=strlen(tmp);i<56;i++) strcat(tmp," ");

        fgets(ligne,90,f1);
        strcat(tmp,ligne);
        strcat(msg,tmp);
    }else {
        fgets(tmp1,90,f1);
        strcpy(tmp,"");
        continue; }
    }
    if (k==0){ strcpy(msg,"Aucun contact dans cette ville"); }
```

Dans le terminal :

```
>>-Menu Administrateur-<<

** 1 ** pour lister les contacts.
** 2 ** pour lister les contacts selon leurs nom.
** 3 ** pour lister les contacts selon leurs ville.
** 4 ** pour rechercher un contact.
** 5 ** pour ajouter un contact.
** 6 ** pour modifier un contact.
** 7 ** pour supprimer un contact.
** M ** pour le manuel d'utilisation.
** EXIT ** pour quitter le programme.
-----
>> Votre choix : 3

>>Entrez la ville des contacts desirees :
$> Tanger
Medaghri_Alaoui Amine 0666341418 Tanger
```

5. Rechercher un contact

Cette fonction permet à l'administrateur d'entrer un nom et un prénom, et puis d'afficher le contact désignés s'il existe. Si le contact n'existe pas alors le serveur va renvoyer un message qui annonce que ce contact n'existe pas dans la liste.

```
while(!feof(f1)){
    fscanf(f1,"%s ",ligne); //va lire mots par motss

    if(strcmp(ligne,nom_chercher)==0 ){
        fscanf(f1,"%s",ligne);

        if(strcmp(ligne, prenom_chercher )==0){ k=3;
            strcpy(msg,"\n");
            strcat(msg,nom_chercher);
            strcat(msg,"\t\t");
            strcat(msg,prenom_chercher);
            strcat(msg,"\t");
            fgets(ligne,90,f1); //va lire jusqu'a la fin de la ligne
            strcat(msg,ligne);
            strcat(msg,"\n-----");
            break;

        }else {
            continue;
        }
    }
}
```

6. Ajouter un contact

Cette fonction permet à l'administrateur d'ajouter un contact à la liste des contacts. Si l'administrateur choisit d'ajouter un contact. Le programme demande de lui d'entrer les informations nécessaires qui sont : Nom, Prénom, numéro, Ville, et E-mail.

```
strcpy(msgn1,"\n>>Entrez le nom du nouveau contact :\n$> ");
strcpy(msgp1,">>Entrez le prenom du nouveau contact :\n$> ");
strcpy(msgt1,">>Entrez le numero de telephone du nouveau contact :\n$> ");
strcpy(msga1,">>Entrez la ville du nouveau contact :\n$> ");
strcpy(msge1,">>Entrez l'e-mail du nouveau contact :\n$> ");
strcpy(msger,"Ce contact existe deja!\n-----");
```

Dans le terminal :

7. Modifier un contact

La fonction de **modifier un contact** permet à l'administrateur de modifier un contact après la demande d'entrer un nom et prénom d'un contact qui existe déjà et le modifier de la liste des contacts. Il donne l'option de modifier seulement un élément du contact (**le nom, le prénom, la ville etc...**), ou bien de modifier l'intégralité du contact.

Son principe, d'abord comme **consulter_contact** est de chercher le nom et le prénom et qu'on les trouve existant (**ex : modifier nom**). On retourne au début de la ligne par fonction **fseek** et on écrit sur l'ancien nom en l'écrasant sachant que le curseur s'arrête à la fin du prénom lors de la recherche.

```
char* modifier_element_contact(char nom_chercher[],char prenom_chercher[],
char newelement[],char operation[]){

FILE* f1;    int i, k=0;  long b=0;
char * msg=calloc(200,sizeof(char)) ;
char mots[30],nom[30]="",tmp[100]="";
    f1 = open("contacts.txt","r+");
if (f1 == NULL) {
printf("Error au niveau d'ouvrir le fichier de contact !! \n");
    }else{
while(!feof(f1)){
char c;
strcpy(nom,""); // vider le var nom
    c=fgetc(f1); // pour arrêter le curseur à la fin de prochaine mot
b=fseek(f1,-1, SEEK_CUR );
    fscanf(f1,"%s",mots);
    strcat(nom,mots);
    c=fgetc(f1);
    fscanf(f1,"%s",mots);
    if(strcmp(nom,nom_chercher)==0 ){

        if(strcmp(mots, prenom_chercher )==0){

            if(strcmp(operation,"1") == 0){ //modifie nom
                b= fseek(f1,-strlen(mots) ,SEEK_CUR );
                b= fseek(f1,-17 ,SEEK_CUR ); // retourne au debut de la ligne
                for( i=strlen(newelement);i<17;i++) strcat(newelement," ");
                fprintf(f1,"%s",newelement);// ecraser l'ancien nom par lr nouveau
                k=1; break; //k=1 pour verifier que la Modifiecation a lien avec succes
            }
        }
    }
}
```

On a utilisé **c=getc(f1)** pour arrêter le curseur à la fin du prochain mot .

Dans le terminal :

```
-----
>> Votre choix : 6

>>Entrez le nom du contact que vous voulez modifier:
$> Medaghri Alaoui

>>Entrez le prenom du contact que vous voulez modifier:
$> Amine

** 1 ** pour modifier le nom.
** 2 ** pour modifier le prenom du contact.
** 3 ** pour modifier le numero de telephone du contact.
** 4 ** pour modifier la ville du contact.
** 5 ** pour modifier l'e-mail du contact.
** 6 ** pour modifier tout les elements du contact.
>> Votre choix:
```

8. Supprimer un contact

La fonction supprimer un contact permet à l'administrateur d'entrer un nom et un prénom d'un contact qui existe et de le supprimer de la liste des contacts.

Son principe est de créer un nouveau fichier et copier tous le contenu du fichier origine dans le nouveau fichier sauf pour la ligne qui contient le contact désigné (**le nom_chercher et le prenom_chercher**). Enfin on va supprimer l'ancien fichier et le remplacer par nouveau fichier en le renommant.

```
fscanf(f1,"%s ",mots); //fscanf() : va lire mot par mot //Et fgets() :va lire ligne ou le reste de ligne
strcpy(nom,mots); // il va stocker chaque nom qui est dans le fichier dans un variable temporaire : nom

fscanf(f1,"%s ",mots) ; //on va lire chaque prenom dans le fichier
if(strcmp(nom,nom_chercher)!=0 || strcmp(mots,prenom_chercher )!=0){

if( strcmp(mots,nom )!=0 ){
for(i=strlen(nom);i<17;i++) { strcat(nom," "); } //pour conserver sur la tabulation

strcat(nom,mots); //on va stocker le prenom a cote den nom
for(i=strlen(nom);i<32;i++) {strcat(nom," "); }

} else { //dans le cas ou le nom et le prenom qu'on les veux de supprimer !
fgets(tmp,90,f1); // lire le reste de la ligne dans un var temporaire pour deplacer le curseur vers le nouveau ligne
strcpy(nom,""); //pour vider le var : nom qui contient nom et prenom qu'on les veux de supprimer
}

strcpy(resteligne,""); // pour vider le variable : resteligne
fgets(resteligne,90,f1); //on va lire le reste de ligne
strcat(nom,resteligne); //coller le reste de ligne dans 'nom' qui contient nom et prenom qu'on les veux de laisser
strcat(nom,"");
fprintf(f2,nom);
//else : on va stocker le reste de ligne dans un var temporaire pour seulement bouger le curseur
}else{ k=1; fgets(tmp,100,f1); }
}
fclose(f2);
}
fclose(f1);
}
remove("contacts.txt");
rename("copie.txt","contacts.txt");
if(k!=0){strcpy(mots,"\nLa suppression est terminee.\n-----\n** R
} else { strcpy(mots,"\nLe contact n'existe pas - Ou il a deja ete supprime.\n-----\n** R
```



```

char * supprimer_tout_contact()
{
    FILE* f1;
    FILE* f2;
    char * msg=calloc(500,sizeof(msg) );
    int k=0;
    f1 = fopen("contacts.txt","r");
    if (f1 == NULL){
        printf("\n Erreur au niveau d'ouvrir le fichier contact origine \n");
    }else {
        f2 = fopen("copie.txt", "w+");
        if (f2 == NULL){
            printf("Erreur au niveau d'ouvrir le copie fichier");
        }else{
            strcpy(msg,"Nom          Prenom          Numero          Ville          Email          \n");
            fprintf(f2,msg);
            k-1;

            fclose(f2);
        }
        fclose(f1);
    }

    remove("contacts.txt");
    rename("copie.txt","contacts.txt");

    if(k!=0){strcpy(msg,"\nLa suppression est terminee.\n");
    }
    else { strcpy(msg,"\nLe contact n'existe pas - Ou il a deja ete supprime.\n"); }

    return msg ;
}

```

Dans le terminal :

```

-----
>> Votre choix : 8
Attention! cette action est irreversible. Etes-vous sur de vouloir continuer?
'Oui' ou 'Non'
$> Oui

La suppression est terminee.
-----

```

10. Manuel d'utilisation

L'administrateur peut choisir de voir le manuel de travail de notre programme. Il contient des notes sur le fonctionnement de programme et les noms des créateurs et de l'encadrant.

Dans le terminal :

```

>> Votre choix : m

-----> Manuel d'utilisation <-----

* Faites attention au majuscules lors de rechercher/supprimer/ajouter un contact.
* Lors de l'ajout d'un contact. Il est preferable de ne pas dépasser :
  - 17 caracteres pour le nom.
  - 15 caracteres pour le prenom.
  - 12 caracteres pour le numero de telephone.
  - 13 caracteres pour la ville.
  - 26 caracteres pour l'adresse e-mail.
Cela pour eviter des problemes de tabulation.
Ce programme a ete cree par Amine Zeggaf, Tarik El Mesaouri, Yousra Berdey
et Amine Medaghri Alaoui.

Encarde par M. Ait Kbir

-----
Pour revenir au menu, appuyez sur la touche * ENTRER *.

```

11. Se connecter en tant qu'utilisateur

Permet de revenir à la saisie du code administrateur/utilisateur, en appelant la fonction **goto**.

Dans le terminal :

```

>> Votre choix : s

>>Donnez le code de l'utilisateur : 007

-----

>>-Menu Utilisateur-<<

** 1 ** pour lister les contacts selon leurs nom.
** 2 ** pour rechercher un contact.
** 5 ** pour se connecter en tant qu'administrateur.
** EXIT ** pour quitter le programme.

-----

>> Votre choix :

```

12. Menu après une fonction

Après chaque opération, le programme donne à l'administrateur un menu avec les options de répéter la commande pour ne pas revisiter le menu en entier. Et aussi de revenir au menu principal ou bien de quitter le programme.

```
*** R ** pour repeter la commande.  
*** 0 ** pour revenir au Menu Principal.  
*** EXIT ** pour quitter le programme.
```

En cas d'un choix non valide le programme va afficher un message.

```
if ( (strcmp(msga,"R")==0) || (strcmp(msga,"r")==0) ){  
    goto repeata5;  
    continue;  
}else if (strcmp(msga,"0")==0){  
    menuA(sClient,msga);  
    break;  
}else if ( (!strcmp(msga,"exit")) || (!strcmp(msga,"EXIT")) ){  
    write(sClient,msga,strlen(msga)+1);  
    exit(1);  
}else{  
    (strcpy(msga,"\nChoix non disponible! Donnez un choix valide : "));  
    write(sClient,msga,strlen(msga)+1);  
    continue;  
}
```

13. Exit

Après chaque opération ou dans le menu principal le client obtiendra le choix de quitter le programme, il peut faire cela grâce à la fonction EXIT en tapant 'exit' s'il choisit de quitter le programme, la fenêtre du client va se fermer automatiquement et le serveur reste ouvert.

```
}else if ( (!strcmp(msg,"exit")) || (!strcmp(msg,"EXIT")) ){  
    write(sClient,msg,strlen(msg)+1);  
    exit(1);  
}
```

2. Le cas d'un utilisateur

Menu Principal d'utilisateur

Si le serveur recevoir le code d'un utilisateur il va afficher un menu personnalise qui lui permet de lister tous les contacts ou chercher un contact

Dans le terminal :

```

      |-----|
      | BIENVENUE |
      |  CHER CLIENT  |
      |-----|

>>Veuillez entrer le code d'accès:
$> 007

-----
>>-Menu Utilisateur-<<

**  1  ** pour lister les contacts selon leurs nom.
**  2  ** pour rechercher un contact.
**  S  ** pour se connecter en tant qu'administrateur.
** EXIT ** pour quitter le programme.
-----
>> Votre choix :
```

L'utilisateur n'a le droit que de rechercher un contact ou de filtrer quelques contacts par leur nom. Cependant, il n'a pas l'option de connaître tous les contacts sauf si il connaît leurs noms, et aussi n'a pas le pouvoir de modifier les contacts existant. Ses fonctions sont en commun avec l'administrateur mais ses options restent limitées par comparaison.

Conclusion:

Dans ce projet nous avons appliqué les connaissances qu'on a acquises concernant le fonctionnement de l'architecture Client/Serveur. Nous avons aussi développé nos compétences dans la programmation en langage C et découvert plusieurs informations et fonctions pratiques.

De plus, ce projet nous a aidé à être plus persévérants et créative dans la résolution des problèmes et nous a aussi rendus plus conscients de l'importance du respect des délais et le travail en équipe.