Mini-projet langage C Avancé :
Socket et Structures de donnée

Sous l'encadrement :

Pr. EL BOUHDIDI

Réalisé par :

KARIMALLAH Nouhaila

Filière : GI1

Introduction:

Dans ce projet de module Langage C Avancé, on va créer deux applications client et serveur en utilisant les sockets.

Le socket en général est un modèle permettant la communication entre un système unique et un réseau utilisant le protocole TCP/IP. Les sockets sont inclus dans la bibliothèque standard du langage,

Pour programmer cette application, je vais utiliser par la suite DEV C++ comme interface.

Bien qu'ils aient apparues pour la première fois dans les systèmes UNIX, les sockets sont des points de terminaison mis à l'écoute sur le réseau, afin de faire transiter des données logicielles. Et donc il est possible de l'utiliser sur le système d'exploitation Windows. Pourtant vous devez être conscient que les sockets ne s'utilisent pas de manière identique selon les différents systèmes d'exploitation.

Dans ce rapport là on décriera les étapes de réalisation de notre application.

- ## Initialiser Winsock :

Avant d'entrer dans les détails de l'application à créer, il faut d'abord suivre des étapes initiales et fondamentales pour la création des sockets en général.

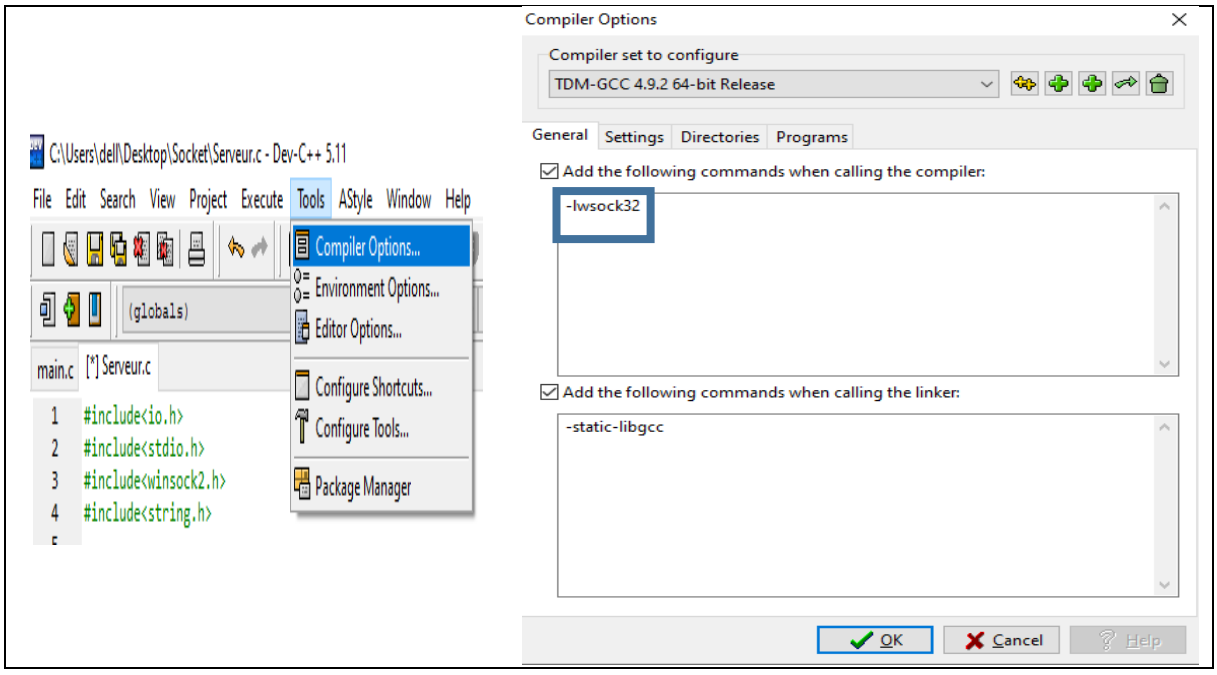
Tout d'abord, juste après avoir décidé le système d'exploitation à utiliser pour le projet on doit chercher le fichier qui contient tous les fonctions qui touche aux sockets, sur Windows le fichier « winsock2.h » c'est le fichier standard de windows, qui est disponible dans le dossier header de l'IDE, et qui contient tout ce qui concerne les sockets Windows.

Nous allons donc tout de suite l'inclure dans notre premier programme avec les autres bibliothèques du langage C usuelle telles que Stdio.h, Stdlib.h , String.h ….etc.

```
#include<stdio.h>
#include<winsock2.h>
#include<string.h>
```

Normalement dans le compilateur Visual C++on doit ajouter le fichier "ws2_32.lib" : le fichier de bibliothèque à lier au programme pour pouvoir utiliser les fonctions winsock.

Mais dans mon cas sur DEV C++, j'ai dû faire une autre procédure.



Puisque j'avais un problème de liaison pour les fonctions de Winsock2. J'ai résolu le problème en ajoutant une commande -lwsock32 en suivant les étapes suivantes :

1. **Tools**.
2. **compiler options**.
3. choisir **general**.
4. Puis on click **add the following commands when calling the compilers**.

5. Et on ajoute la commande **–lwsock32.**

Ensuite une procédure nécessaire doit être suivie afin d'initialiser Winsock.

```
int main(int argc , char *argv[])
{
    WSADATA wsa;


//  Initialisation de Winsock...

    if (WSAStartup(MAKEWORD(2,2),&wsa) != 0)
    {
        printf("Erreur d'initialisation Code : %d",WSAGetLastError());
        return 1;
    }
```

La fonction **WSAStartup** est utilisée pour démarrer ou initialiser la bibliothèque winsock. Elle accepte deux paramètres ; la première est la version que nous voulons charger et la seconde est une structure WSADATA qui contiendra des informations supplémentaires après le chargement de winsock.

Si une erreur se produit, la fonction WSAStartup renvoie une valeur différente de zéro et **WSAGetLastError** peut être utilisé pour obtenir plus d'informations sur l'erreur qui s'est produite.

Coté client :

L'étape qui suit l'initialisation de Winsock dans l'application client, consiste à créer un socket.

- ## Création de socket :

La fonction **socket()** permet de créer un socket. La fonction **socket()** crée une socket et renvoie un descripteur de socket qui peut être utilisé dans d'autres commandes réseau. Le code ci-dessus va créer un socket de :

- Famille d'adresses : AF_INET (il s'agit de la version IP 4)
- Type : SOCK_STREAM, cela signifie protocole TCP orienté connexion [SOCK_DGRAM indique le protocole UDP]
- Protocole : 0 (Dans le cas de la suite TCP/IP il n'est pas utile, on le mettra ainsi toujours à 0)

```
// Créer un socket

    if((s = socket(AF_INET , SOCK_STREAM , 0 )) == INVALID_SOCKET)
     {
         printf("Could not create socket : %d" , WSAGetLastError());
     }
```

- Se connecter à un serveur :

Pour se connecter à un serveur nous avons besoin d'une adresse IP et d'un numéro de port.

Tout d'abord, nous devons créer une structure **sockaddr_in** avec les valeurs appropriées.

```
//  créer une structure sockaddr_in / address du Socket server

    struct sockaddr_in server;

    server.sin_addr.s_addr = inet_addr("127.0.0.1");
    server.sin_family = AF_INET;
    server.sin_port = htons( 23 );
```

Le sockaddr_in a un membre appelé sin_addr de type in_addr qui a un s_addr qui n'est rien d'autre qu'un long. Il contient l'adresse IP au format long.

On utilise ainsi la fonction **inet_addr** pour convertir une adresse IP en un format long.

Le membre **sin_family** précise la famille d'adresse et avec le membre **sin_port** on précise notre port de connexion.

La dernière chose à faire pour assurer la connexion c'est la fonction de connexion **connect().** Il a besoin d'un socket et d'une structure **sockaddr** pour se connecter.

```
//Se connecter au server
if (connect(s , (struct sockaddr *)&server , sizeof(server)) < 0)
{
    puts("connect error");
    return 1;
}
```

Coté serveur :

Les serveurs effectuent essentiellement les opérations suivantes :

- Création du socket serveur :

Pour la création du socket serveur on suit la même procédure de celui du client, en utilisant toujours la fonction **socket().**

```
//  Créer a socket

   SOCKET s;


    if((s = socket(AF_INET , SOCK_STREAM , 0 )) == INVALID_SOCKET)
    {
        printf("erreur creation Socket : %d" , WSAGetLastError());
    }

    puts("Le socket 208 est maintenant ouverte en mode TCP/IP");


// Prearer la structure sockaddr_in

    server.sin_family = AF_INET;
    server.sin_addr.s_addr = INADDR_ANY;
    server.sin_port = htons( 23 ); // port de connexion
```

- Lier à une adresse :

La fonction **bind()** peut être utilisée pour lier un socket à une adresse et un port particuliers. Elle a besoin d'une structure **sockaddr_in** qui spécifie l'adresse locale à travers laquelle le programme doit communiquer, et la taille du champ adresse locale.

```
// Bind

    if( bind(s ,(struct sockaddr *)&server , sizeof(server)) == SOCKET_ERROR)
    {
        printf("Erreur Bind()  : %d" , WSAGetLastError());
    }

    puts("------>Listage du port 23...");
```

Maintenant que la liaison est terminée, il est temps que le socket écoute les connexions. Nous lions un socket à une adresse IP particulière et à un certain numéro de port. En faisant cela, nous nous assurons que toutes les données entrantes qui sont dirigées vers ce numéro de port sont reçues par cette application.

- ## Ecouter les connexions entrantes :

La fonction **listen()** permet de mettre le socket en écoute. On ajoute simplement la ligne suivante après la liaison.

```
//Pret à se connecter
listen(s , 3);
```

Elle a besoin de deux paramètres le socket représente le socket précédemment ouvert et le nombre maximal de connexions pouvant être mises en attente

Maintenant vient la partie principale de l'acceptation de nouvelles connexions.

- ## Accepter les connexions :

La fonction **accept()** permet la connexion en acceptant un appel.

```
//Attendre la connexion
puts("------>Patientez pendant que le client se connecte sur le port 23...");

c = sizeof(struct sockaddr_in);

char *login,*passwd;
int l,p;


while(( new_socket = accept(s , (struct sockaddr *)&client, &c)) != INVALID_SOCKET )
{
    puts("Un client se connecte avec le socket :) ");
```

La fonction **accept()** retourne un identificateur du socket de réponse. Si une erreur intervient la fonction **accept()** retourne la valeur INVALID_SOCKET.

Après l'établissement de la connexion, le serveur et le client sont maintenant prêts à se communiquer.

- ## Send/ Recv :

La fonction **send()** envoit simplement des données du serveur au client et vice versa. Il a besoin du descripteur de socket, des données à envoyer et de sa taille. La fonction **send()** renvoie le nombre d'octets effectivement envoyés.

```
send(new_socket , "\n Donner le nom du contact:" , 100 , 0);
```

Maintenant que nous avons envoyé des données, il est temps de recevoir une réponse du serveur.

La fonction **recv()** permet de lire dans un socket en mode connecté (TCP). Elle renvoie le nombre d'octets lus. De plus cette fonction bloque le processus jusqu'à ce qu'elle reçoive des données.

```
t = recv(new_socket , c.nom , 30 , 0);
c.nom[t] = '\0';
puts(c.nom);
```

- Fermer la connexion :

Finalement nous terminerons par la fonction ***closesocket*** **()** qui permet de fermer un socket.

Aussi, il ne faut pas oublier de mettre fin à l'utilisation de Winsock avec la fonction **WSACleanup()** .

```
closesocket(s);
WSACleanup();
```

Communication des données :

Après avoir connecté les deux sockets Client-Serveur on passe à analyser et élaborer le code de communication entre eux.

La première étape à faire c'est la connexion et l'authentification.

Le serveur qui a accès au document « comptes.txt » doit demander au client le login et le mot de passe pour savoir s'il est un administrateur ou bien un utilisateur, et ceci après vérification des données saisies dans le document « comptes ».

Dans ce cas-là, nous sommes sensé de réaliser deux codes coté client et coté serveur afin d'établir cet échange de données.

On crée d'abord une structure Compte :

```
typedef struct {
    char login[30];
    int mdp;
    char profil[30];
}compte;
```

Ensuite on crée deux fonctions Identification **(Socket s)** pour faire l'authentification :

Serveur

```
char *Identification(SOCKET s){

        send(s , "********************BIENVENUE**************************\n\n

        compte c;
        int trouve=0;
        int passwd;
        char *login,CH[2];
        int m ,i, j;
        char *fichier="comptes.txt";
        FILE *f=fopen(fichier,"r");


        login=(char*)malloc(sizeof(char)*100);

        do{

        if((m = recv(s , login , 100 , 0)) == SOCKET_ERROR)
        { puts("Erreur finction recv() ");
        }

        login[m] = '\0';
        printf("authentification#%s",login);


        i=0;

        for(j=0;j<2;j++){
           while (login[i] != '#' && login[i])
            {
             strcpy(CH[j],login+i);
            }
             puts(CH[j]);
        }


         i=0;

         if(f!=NULL)

         while(!feof(f))
         {
             fscanf(f,"%s %d %s \n",c.login,&c.mdp,c.profil);
             if(strcmp(c.login,CH[0])==0 && c.mdp==CH[1])
            { trouve=1; break;}
         }
         fclose(f);

         if(trouve==1){
         printf("Trouve !!!");
         send (s, c.profil, 50,0 );
         return &c.profil;
         }else {
         send(s,"0",100,0);
          i++;}

         if(i=2){
             send(s,"\n Vous avez dépacer le nombre de tentative \n",50,0);
             closesocket(s);
         }
        }while(i<3);

        return NULL;
}
```

Client

```
void Identification(SOCKET new_socket){

        char message[2000],reponse[100];
        int m = recv(new_socket , message , 2000 , 0);
        message[m] = '\0';

        compte c;
        int size;

        if((size = recv(new_socket , message , 2000 , 0)) == SOCKET_ERROR)
            puts("recv failed");


        message[size] = '\0';   // Ajouter '\0' pour marquer la chaine
        puts(message);

        do{

        printf("\n    S'il vous palait saisir votre login:");
        scanf("%s",c.login);
        printf("\n    S'il vous plait saisir votre password:");
        scanf("%s",c.mdp);

        strcat(strcat(c.login, "#"),c.mdp);
        send(new_socket , c.login , 100 , 0);

        size = recv(new_socket , reponse , 2000 , 0);
        reponse[size]='\0';


       }while(strcmp(reponse,"admin")!=0 || strcmp(reponse,"invite")!=0);
}
```

On envoie le message de bienveillance, puis dans la partie client on demande login et le mot de passe. Je concatène les deux données saisies en les séparant par # puis on les envois au Serveur qui par la suite va séparer le login du mot de passe, ouvre le fichier « comptes.txt », lit les données et vérifie l'existence de l'utilisateur ainsi que son profil.

Si le login n'existe pas ou le mot de passe et incorrect on renvoie un message au client en lui demandant d'essayer encore une fois, sinon on envoie le profil. Sauf que le client n'a pas plus de 3 tentatives et si la dernière tentative et incorrect le socket du client va se fermer.

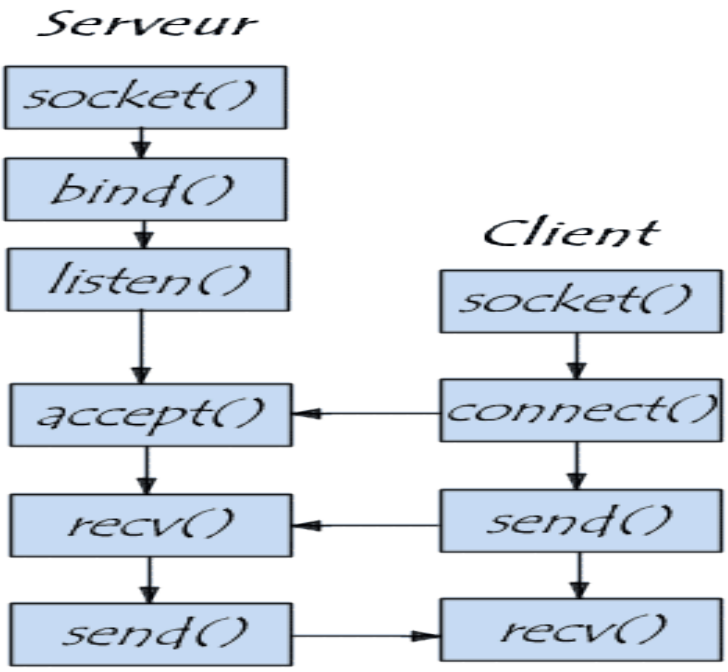
Après l'authentification on passe au menu. Le menu doit être personnalisé selon le profil du client :

Serveur :

```
void menuAdmin(SOCKET s){

    int c;
    char *choix;

    do{
    send(s,"\n          ----------------------------VOUS ETES ADMINISTRATEUR-

    c = recv(s , choix , 1 , 0);
    choix[c] = '\0';

    switch (*choix){

        case '1':
            Ajouter_Contact(s);
            break;
        case'2':
            Recherche_Contact(s);
            break;
        case'3':
            Supprimer_Contact(s);
            break;
        case'4':
            Modifier_Contact(s);
            break;
        case'5':
            Afficher_Tous(s);
            break;
        case'6':
            break;
        default:
            send(s,"Choix invalid essayez encore une fois",50,0);
            break;
    }
    }while(choix!='6');
```

```
void menuInvite(SOCKET s){

    int c;
    char *choix;

    do{
    send(s,"\n          ----------------------------VOUS ETES UTILISATEUR--

    c = recv(s , choix , 1 , 0);
    choix[c] = '\0';

    switch (*choix){
        case '1':
            Ajouter_Contact(s);
            break;
        case'2':
            Afficher_Tous(s);
            break;
        case'3':
            break;
        default:
            send(s,"Choix invalid essayez encore une fois",50,0);
            break;
    }
    }while(choix!='3');
}
```

Le serveur envoie le menu au client, puis il va attendre la réception de la réponse du client. Et puisque la communication connectée en mode TCP/IP insiste le suivi du schéma suivant :



On doit créer une fonction **menu()** similaire à celle du serveur pour recevoir et envoyer les données. Pour que à chaque fois on aura une fonction **send()** dans le coté serveur on aura ainsi une autre **recv()** chez le code du client et vice versa.

C'est le principe de la communication TCP/IP.

Client

```c
void menuAdmin(SOCKET s){
    int c;
    char *menu=(char*)malloc(sizeof(char)*2000);
    char *choix=(char*)malloc(sizeof(char)*10);
    char *erreur=(char*)malloc(sizeof(char)*50);
    do{
    c = recv(s , menu , 2000 , 0);
    menu[c] = '\0';
    puts(menu);

    scanf("%s",choix);
    send(s,choix,4,0);
    switch (*choix){
        case '1':
            Ajouter_Contact(s);
            break;
        case'2':
            Recherche_Contact(s);
            break;
        case'3':
            Supprimer_Modifier(s);
            break;
        case'4':
            Supprimer_Modifier(s);
            break;
        case'5':
            Afficher_Tous(s);
            break;
        case'6':
            break;
        default:
            c = recv(s , erreur , 2000 , 0);
            erreur[c] = '\0';
            puts(erreur);
            break;
    }
    }while(*choix !='6');
```

```c
void menuInvite(SOCKET s){

    int c;
    char *menu=(char*)malloc(sizeof(char)*2000);
    char *choix=(char*)malloc(sizeof(char)*10);
    char *erreur=(char*)malloc(sizeof(char)*50);

    do{

    c = recv(s , menu , 2000 , 0);
    menu[c] = '\0';
    puts(menu);

    scanf("%s",choix);
    send(s,choix,4,0);

    switch (*choix){
        case '1':
            Ajouter_Contact(s);
            break;
        case'2':
            Afficher_Tous(s);
            break;
        case'3':
            break;
        default:
            c = recv(s , menu , 2000 , 0);
            menu[c] = '\0';
            puts(menu);
            break;
    }
    }while(*choix!='3');
}
```

A partir du menu on voit qu'on a annoncé plusieurs choix dans le menu afin de traiter les données du contact dont l'adresse stocké dans l'espace du serveur.

On crée ainsi les différentes fonctions nécessaires pour traiter les demandes du client.

Ajouter un contact :

Avant de saisir la fonction on doit tout d'abord déclarer la structure de nos données ici, on a choisi les deux structures suivantes :

```
typedef struct {
    char rue[30];
    char ville[30];
    char pays[30];
}adresse;


typedef struct {
    char nom[30];
    char prenom[30];
    char GSM[30];
    char email[50];
    adresse *adr;
}contact;
```

Ensuite on créer la fonction qui nous permet de saisir un contact et retourne les valeurs enregistrées sous forme d'une structure.

Serveur

```
contact SaisirServeur(SOCKET s){

    contact c;
    int t;
    c.adr=(adresse*)malloc(sizeof(adresse));

    send(s , "\n Donner le nom du contact:" , 100 , 0);
    t = recv(s , c.nom , 30 , 0);
    c.nom[t] = '\0';
    puts(c.nom);

    send(s , "\n Donner le prenom du contact:" , 100 , 0);
    t = recv(s , c.prenom , 30 , 0);
    c.prenom[t] = '\0';
    puts(c.prenom);

    send(s , "\n Donner le GSM:" , 100 , 0);
    t = recv(s , c.GSM , 30 , 0);
    c.GSM[t] = '\0';
    puts(c.GSM);

    send(s , "\n Donner l'email':" , 100 , 0);
    t = recv(s , c.email , 50 , 0);
    c.email[t] = '\0';
    puts(c.email);

    send(s , "\n Saisi l'adresse:\n\n veuillez entrer le nom de la rue :" , 100 , 0);
    t = recv(s , c.adr->rue , 30 , 0);
    c.adr->rue[t] = '\0';
    puts(c.adr->rue);

        send(s , "\n veuillez entrer la ville du contact :" , 100 , 0);
        t = recv(s , c.adr->ville , 30 , 0);
        c.adr->ville[t] = '\0';
        puts(c.adr->ville);

        send(s , "\n veuillez entrer la pays du contact :" , 100 , 0);
        t = recv(s , c.adr->pays , 30 , 0);
        c.adr->pays[t] = '\0';
        puts(c.adr->pays);

        return c;
    }
```

Client

```
contact SaisirClient(SOCKET s){

    contact c;
    int t;
    char nom[30],prenom[30],GSM[30],email[50],rue[30],ville[30],pays[30];

    c.adr=(adresse*)malloc(sizeof(adresse));

    t = recv(s , nom , 30 , 0);
    nom[t] = '\0';
    scanf("%s",c.nom);
    send(s , c.nom , 30 , 0);


    t = recv(s , prenom , 30 , 0);
    prenom[t] = '\0';
    scanf("%s",c.prenom);
    send(s , c.prenom , 30 , 0);


    t = recv(s , GSM , 30 , 0);
    GSM[t] = '\0';
    scanf("%s",c.GSM);
    send(s , c.GSM , 30 , 0);

    t = recv(s , email , 30 , 0);
    email[t] = '\0';
    scanf("%s",c.email);
    send(s , c.email , 30 , 0);


    t = recv(s , rue , 30 , 0);
    rue[t] = '\0';
    scanf("%s",c.adr->rue);
    send(s , c.adr->rue , 30 , 0);

        t = recv(s , ville , 30 , 0);
        ville[t] = '\0';
        scanf("%s",c.adr->ville);
        send(s , c.adr->ville , 30 , 0);

        t = recv(s , pays , 30 , 0);
        pays[t] = '\0';
        scanf("%s",c.adr->pays);
        send(s , c.adr->pays , 30 , 0);

        return c;
    }
```

Tout en respectant le principe de la communication TCP/IP on voit que les deux fonction sont antisymétrique, chaque fonction **send()** dans le code serveur a comme correspondante une fonction **recv().**

Cette fonction va être utile pour la fonction Ajouter un Contact, en effet la fonction Ajouter coté serveur va au premier lieu ouvrir le fichier « contacts.txt » ou les données des contacts sont stockées en mode écriture puis on demande au client de saisir les coordonnées du contact pour qu'on puisse les enregistrées dans le fichier du serveur enfin on ferme le fichier.

Serveur :

```
void Ajouter_Contact(SOCKET s){
    char *nom_fich="contacts.txt";
    FILE *f=fopen(nom_fich,"a");
    contact c = SaisirServeur(s);
    if(f!=NULL){
        fprintf(f,"%s %s %ld %s %s %s %s \n",c.nom,c.prenom,c.GSM,c.email,c.adr->rue,c.adr->ville,c.adr->pays);
    }
    fclose(f);
    send(s , "\n Ajout avec succes:" , 100 , 0);
}
```

Client :

```
void Ajouter_Contact(SOCKET new_socket){

    char *msg;
    int size;

    msg=(char*)malloc(sizeof(char)*100);

    contact c = SaisirClient(new_socket);

    size=recv(new_socket , msg , 100 , 0);
    msg[size]='\0';
    puts(msg);
}
```

Afficher Contact (s) :

Pour le serveur j'ai utilisé deux fonctions une pour concaténer les valeurs de chaque contact afin que le résultat sera comme suit : **nom#prénom#GSM#email#..... ,** Et une autre pour afficher tous les contact. En paramètre la première accepte le contact à afficher puis le socket client pour utiliser la fonction **Send()** et la deuxième fonctions prend comme argument le socket client uniquement.

```
void Afficher_Contact(contact c, SOCKET s){

 strcat(strcat(strcat(strcat(strcat(strcat(strcat(strcat(strcat(strcat(c.nom,"#"),c.prenom),"#"),c.email),"#"),c.GSM),"

 send(s,c.nom,2000,0);

}

void Afficher_Tous(SOCKET s){

 FILE *f=fopen("contacts.txt","r");
 contact c;
   c.adr=(adresse*)malloc(sizeof(adresse));

    if(f!=NULL){
        while(!feof(f))
        {
            fscanf(f,"%s %s %ld %s %s %s %s \n",c.nom,c.prenom,c.GSM,c.email,c.adr->rue,c.adr->ville,c.adr->pays);
            Afficher_Contact(c,s);
        }

    fclose(f);
}
}
```

Pour le client une seule fonction est suffisante pour recevoir le résultat uniquement.

```
void Afficher_Tous(SOCKET new_socket){

    char *msg;
    int size;

    while(recv(new_socket , msg , 2000 , 0)!=0)
    {
        size=recv(new_socket , msg , 2000 , 0);
        msg[size]='\0';
        puts(msg);
    }
}
```

Recherche contact :

Pour chercher un contact dans le fichier « contacts.txt », le serveur demande au client le nom de contact à chercher, il reçoit son nom puis ouvre le fichier et effectue la requête. Si le contact existe le Client va recevoir les données du contact affichées à l'aide de la fonction **Afficher_Contact()** sinon il reçoit un message qui précise que le contact n'existe pas.

Serveur :

```
void Recherche_Contact(SOCKET s){

FILE *f=fopen("contacts","r");
contact c;
char *nom=(char*)malloc(sizeof(char)*30);
int t;

c.adr==(adresse*)malloc(sizeof(adresse));

send(s , "\n Donner le nom du contact:" , 100 , 0);
   t = recv(s , nom , 30 , 0);
   nom[t] = '\0';
   printf("Recherche Contact: %s",nom);


int trouve=0;
   if(f!=NULL){

      while(!feof(f))
      {
         fscanf(f,"%s %s %ld %s %s %s %s \n",c.nom,c.prenom,c.GSM,c.email,c.adr->rue,c.adr->ville,c.adr->pays);
       if(strcmp(c.nom,nom))
        { trouve=1; break;}

      }
      fclose(f);
      if(trouve==0)
      send(s," \n Contact introuvable \n ",50,0);
      else {
      Afficher_Contact(c,s);
        }

}
}
```

Client :

```
void Recherche_Contact(SOCKET s){

 contact c;
 char *nom=(char*)malloc(sizeof(char)*30);
 char *cont=(char*)malloc(sizeof(char)*30);
 char *resultat=(char*)malloc(sizeof(char)*30);
 int t;

    t = recv(s , nom , 30 , 0);
    nom[t] = '\0';
    puts(nom);
    scanf("%s",cont);
    send(s,cont,30,0);


    t = recv(s , resultat , 30 , 0);
    resultat[t] = '\0';
    puts(resultat);

}
```

Supprimer / Modifier un contact :

Pour supprimer un contact, le client d'abord précise le contact à supprimer ensuite on ouvre le fichier qui contient les données et on crée un nouveau fichier temporaire tel que à chaque fois on lit un contact du fichier « contacts.txt » on l'insère dans le nouveau fichier si ce dernier et différent du contact à supprimer. Quand on arrive au contact voulu on passe au suivant sans l'insérer puis on supprime l'ancien fichier et renomme le nouveau on lui donnant le même nom de l'ancien fichier des données.

Serveur :

```
void Supprimer_Contact(SOCKET s){
 FILE *fin=fopen("contacts.txt","r");
 char *nom_res="temp.txt";
 FILE *fout=fopen(s,"w");
 contact c;
 c.adr = (adresse*)malloc(sizeof(adresse));
 char *nom = (char*)malloc(sizeof(char)*30);
 int t;

    send(s , "\n Donner le nom du contact:" , 100 , 0);
    t = recv(s , nom , 30 , 0);
    nom[t] = '\0';
    scanf("Recherche Contact: %s",nom);


    if(fin!=NULL&&fout!=NULL){

        while(!feof(fin))
        {
            fscanf(fin,"%s %s %ld %s %s %s %s \n",c.nom,c.prenom,c.GSM,c.email,c.adr->rue,c.adr->ville,c.adr->pays);
          if(strcmp(c.nom,nom)!=0)
            fprintf(fout,"%s %s %ld %s %s %s %s \n",c.nom,c.prenom,c.GSM,c.email,c.adr->rue,c.adr->ville,c.adr->pays);
           if(strcmp(c.nom,nom)==0)
           send(s," Contact Supprime ",30,0);

        }
        fclose(fin);
        fclose(fout);
        remove("contacts.txt");
        rename(nom_res,"contact.txt");

          }
}
```

De même, pour la modification on suit la même procédure de la suppression, la différence sera au niveau du contact modifié. Cette fois ci au lieu de lui dépasser on demande au client d'enregistrer les modifications nécessaires en utilisant la fonction **saisir_Contact**() et on les ajoute au nouveau fichier créé.

Serveur :

```
void Modifier_Contact(SOCKET s){

 FILE *fin=fopen("contacts.txt","r");
 char *nom_res="temp.txt";
 FILE *fout=fopen(s,"w");
 contact c;
 char *nom = (char*)malloc(sizeof(char)*30);
  int t;
 c.adr = (adresse*)malloc(sizeof(adresse));


    send(s , "\n Donner le nom du contact:" , 100 , 0);
    t = recv(s , nom , 30 , 0);
    nom[t] = '\0';
    printf("Recherche Contact: %s",nom);


    if(fin!=NULL&&fout!=NULL){

        while(!feof(fin))
        {
            fscanf(fin,"%s %s %ld %s %s %s %s \n",c.nom,c.prenom,c.GSM,c.email,c.adr->rue,c.adr->ville,c.adr->pays);
          if(strcmp(c.nom,nom)!=0)
            fprintf(fout,"%s %s %ld %s %s %s %s \n",c.nom,c.prenom,c.GSM,c.email,c.adr->rue,c.adr->ville,c.adr->pays);
           if(strcmp(c.nom,nom)==0)
            {c=SaisirServeur(s);
             fprintf(fout,"%s %s %ld %s %s %s %s \n",c.nom,c.prenom,c.GSM,c.email,c.adr->rue,c.adr->ville,c.adr->pays);
             send(s , "\n contact modifie :" , 100 , 0);
            }

        }
        fclose(fin);
        fclose(fout);
        remove("contacts.txt");
        rename(nom_res,"contact.txt");

          }
```

Pour le coté client, tout simplement on doit répondre à chaque fonction **send()** ou **recv()** venant du serveur, et dans le cas de la suppression et la modification on aura le code suivant :

Client :

```
void Supprimer(SOCKET s){

 contact c;
 c.adr=(adresse*)malloc(sizeof(adresse));
 char *nom= (char*)malloc(sizeof(char)*30);
 char *cont =(char*)malloc(sizeof(char)*30);
 char *resultat=(char*)malloc(sizeof(char)*100);
 int t;

    t = recv(s , nom , 30 , 0);
    nom[t] = '\0';
    puts(nom);

    scanf("%s",cont);
    send(s,cont,30,0);


    t = recv(s , resultat , 2000 , 0);
    resultat[t] = '\0';
    puts(resultat);

}
```

```
void Modifier(SOCKET s){
    contact c;
    c.adr=(adresse*)malloc(sizeof(adresse));
    char *nom= (char*)malloc(sizeof(char)*30);
    char *cont =(char*)malloc(sizeof(char)*30);
    char *resultat=(char*)malloc(sizeof(char)*100);
    int t;

    t = recv(s , nom , 30 , 0);
    nom[t] = '\0';
    puts(nom);

    scanf("%s",cont);
    send(s,cont,30,0);

    Ajouter_Contact(s);
}
```

Fonction Main :

Enfin, la dernière partie du projet sera tout simplement l'élaboration de la fonction **main().** Dans la fonction il suffit d'initialiser les fonction du socket de la bibliothèque **winsock2.h**, puis on fait appel à la fonction login puis tout en vérifiant le profil de l'utilisateur, on affiche le menu convenable.

- **MAIN CLIENT :**

```
/****************************************************************************** Main

int main(int argc , char *argv[])
{
    WSADATA wsa;
    SOCKET s;
    struct sockaddr_in server;
    char server_reply[2000],*profil;
    int recv_size;


// Initialisation Winsock...

    if (WSAStartup(MAKEWORD(2,2),&wsa) != 0)
    {
        printf("Failed. Error Code : %d",WSAGetLastError());
        return 1;
    }


// Créer un socket

   if((s = socket(AF_INET , SOCK_STREAM , 0 )) == INVALID_SOCKET)
    {
        printf("Could not create socket : %d" , WSAGetLastError());
    }


//  créer la structure sockaddr_in  / address du socket serveur

    server.sin_addr.s_addr = inet_addr("127.0.0.1");
    server.sin_family = AF_INET;
    server.sin_port = htons( 23 );
```

```
    //Connecter au serveur

    if (connect(s , (struct sockaddr *)&server , sizeof(server)) < 0)
    {
        puts("connect error");
        return 1;
    }


    profil=(char*)malloc(sizeof(char)*30);
    *profil = Identification(s);


    if(strcmp(profil,"admin")==0)
       menuAdmin(s);
    else
       menuInvite(s);

    closesocket(s); //fermer le socket
    WSACleanup();

    return 0;
}
```

- MAIN SERVEUR :

```
int main(int argc , char *argv[])
{
    WSADATA wsa;
    SOCKET serv_socket , client_socket;
    struct sockaddr_in server , client;
    int c;
    char *profil;

//  Initialisation de Winsock...

    if (WSAStartup(MAKEWORD(2,2),&wsa) != 0)
    {
        printf("Erreur d'initialisation Code : %d",WSAGetLastError());
        return 1;
    }

//  Creér a socket


    if((serv_socket = socket(AF_INET , SOCK_STREAM , 0 )) == INVALID_SOCKET)
    {
        printf("erreur creation Socket : %d" , WSAGetLastError());
    }

    puts("Le socket 208 est maintenant ouverte en mode TCP/IP");


// Preparer la structure sockaddr_in

    server.sin_family = AF_INET;
    server.sin_addr.s_addr = INADDR_ANY;
    server.sin_port = htons( 23 );       // port de connexion
```

```
    // Bind

    if( bind(serv_socket ,(struct sockaddr *)&server , sizeof(server)) == SOCKET_ERROR)
    {
        printf("Erreur Bind()  : %d" , WSAGetLastError());
    }

    puts("------>Listage du port 23...");


    //Pret à se connecter avec maximum 3 machines
    listen(serv_socket , 3);


    //Attendre la connexion
    puts("------>Patientez pendant que le client se connecte sur le port 23...");

    c = sizeof(struct sockaddr_in);



    while(( client_socket = accept(serv_socket , (struct sockaddr *)&client, &c)) != INVALID_SOCKET )
    {
        puts("Un client se connecte avec le socket :) ");

        //Communication avec client

        profil= (char*)malloc(sizeof(char)*100);
        profil = Identification(client_socket);


        if(strcmp(*profil,"admin")==0){
            menuAdmin(client_socket);
        }

        if(strcmp(*profil,"utilisateur")==0){
            menuInvite(client_socket);
        }
    }


    if (client_socket == INVALID_SOCKET)
    {
        printf("Erreur accept() code : %d" , WSAGetLastError());
        return 1;
    }


    closesocket(serv_socket);
    WSACleanup();


    return 0;
}
```