l’Architecture générale et les Protocoles

ISMAYILOVA Maryam, MAMMADOV Ali

SERVEUR D’ACCESS

Le rôle principal de notre projet est joué par le serveur d'accès, qui reçoit les clients et les serveurs de données. Nous avons deux sockets pour la communication avec les clients et les nœuds. Premier socket que nous utilisons est entre accès -- client. Et le second est pour les données -- accès.
Pour chaque socket, nous utilisons une adresse IP différente et un numéro de port différent. Après nous commençons à la communication. Il y a "select", "FD_SET" pour savoir de quel socket on va recevoir un message par "recvfrom". Pour la communication, nous utilisons une structure de message qui comprend le type de message, le tampon avec des données, la taille du tampon. Le type de message est très varié:

```
#define INIT 1 //initialisation
#define ECRIRE 2 //requête de client: écrire
#define LIRE 3 //requête de client:lire
#define FIN 4 // pour finir la communication
#define CONFAIL 5 // Erreur de connexion
#define CONEST 6 // connexion réussie
#define SUCCESS 7 //réussie
#define SUPR 8 // requête de client: suppression
#define NOPERMISSION 9 // pas de Droits
#define SIZE 512 // taille de tampon
#define ERREUR -1
#define GETDATA 10 // Pour serveur données, il attend des données
#define SENDDATA 11// Pour serveur données, il peut envoyer les données
```

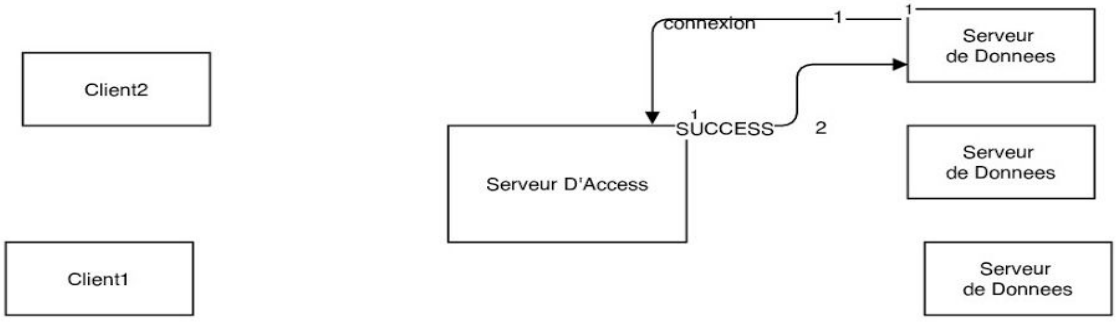
Après avoir reçu un message, nous vérifions de quelle prise il provient.

En cas de serveur de données. Pour les serveurs de données, nous avons une structure qui consiste en une adresse de noeud, type et champ. En plus, on a créé tableau de cette structure, dans lequel on garde des noeuds.

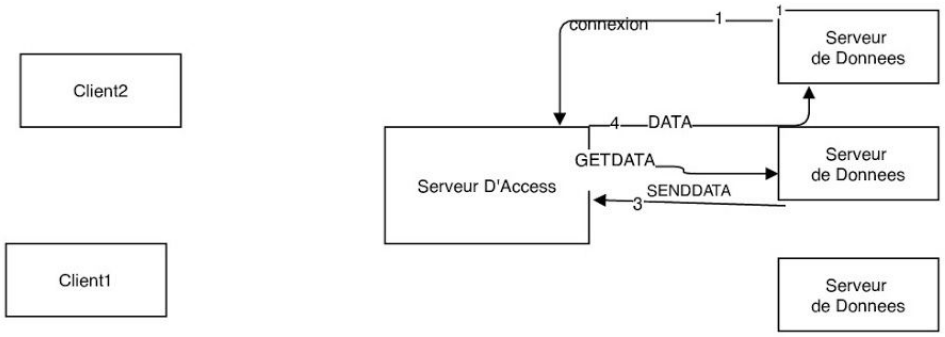
```
struct serveurDeDonnees{
```

```
    struct sockaddr_in addr; // adress de node
    int type; // GETDATA - attend des donnees || SETDATA - peut envoyer
    char champ[128]; // age||nom||taille...
};
```

Dès que nous recevons un message du nœud (en fait, le premier message est son champ), il est ajouté au tableaux avec adresse, champ et type appropriés. La fonction **checkSDs** vérifie s'il existe dans le tableau un nœud ayant le même champ. Si la réponse est Oui, le type **GETDATA** signifie que serveur doit obtenir les données d'un autre nœud. Sinon, il renverra le type **SENDDATA**, ce qui signifie que vous(le serveur de données) avez des données, pouvez recevoir les requêtes et envoyer les données à un autre noeud. Dans le cas de **SENDDATA**, nous envoyons au serveur **SUCCESS** pour attendre quelconque demande.Mais dans le cas de **GETDATA**, nous transmettons. Le programme trouve le serveur avec le même champ qui est déjà connecté et qui peut envoyer des données. Puit avoir pris les données de l'expéditeur et envoyé au nœud actuel. Lorsque le nouveau serveur reçoit toutes les données, son type est changé en **SENDDATA** et ce qui signifie qu'il devient également un serveur pouvant envoyer les données et recevoir les demandes. Maintenant, la connexion entre le serveur d'accès et les serveurs de données est établie, il attend les requêtes des clients.

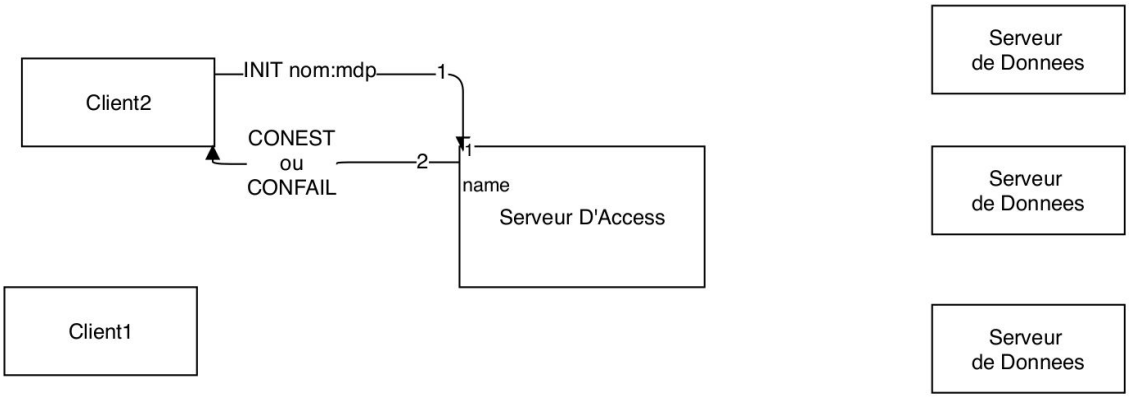


Au dessus c'est un exemple de connexion de serveur de données à serveur. Car il n'y a aucun autre serveur déjà connecté sur le même champs le serveur d'accès l'envoie message de SUCCESS.



Sinon, si le serveur n'est pas le première de servir un champ, serveur d'accès va faire le transfert des données de l'autre serveur qui les contient.

Quand nous recevons une demande du client. Tout d'abord, il doit s'initialiser, il faut envoyer un identifiant et un mot de passe pour avoir accès aux données. En suite, nous vérifions si ce client existe dans notre base. Si oui,on retrouve ses champs sur lesquels il a des droits et ce client sera ajouté à la liste des clients couramment connectés sur serveur.



Chaque client a son identifiant, numéro de champs sur lequel il a accès et ses champs.

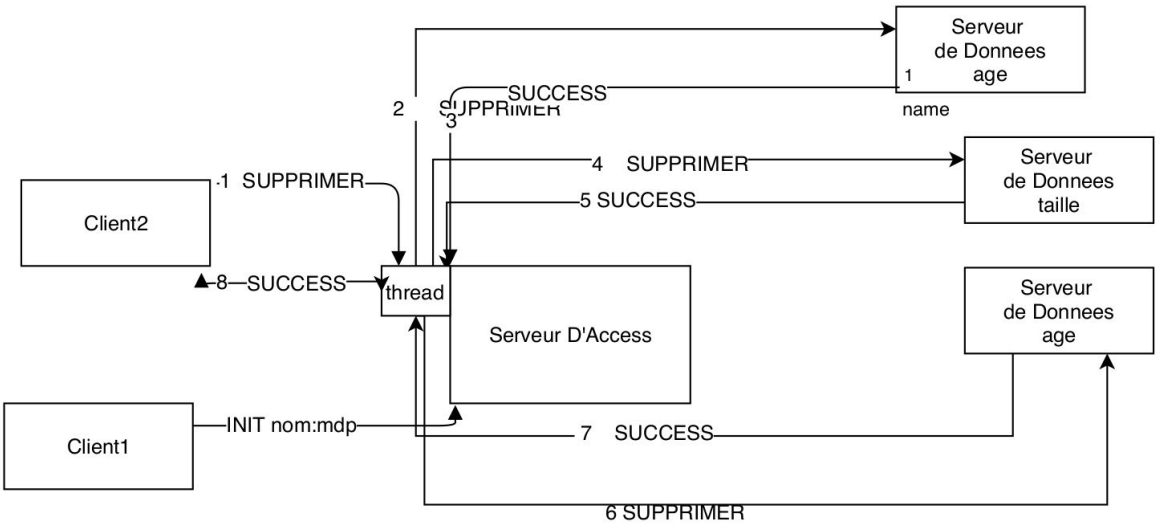
```
struct clientConnection{
   char *login;
   int id;//en fait c'est le numero de port de l'utilisateur
   struct sockaddr_in adr;
   int nbCh;
   char **champs;
};
```

En suite, avec le nouveau type de données qui a un client, un numéro de port et une adresse IP le programme crée un nouveau thread pour chaque client et un nouveau type de données est envoyé comme argument de la fonction thread. Le nouveau port de connexion dédié au client sera envoyé à lui, et sera utilisé par le thread.

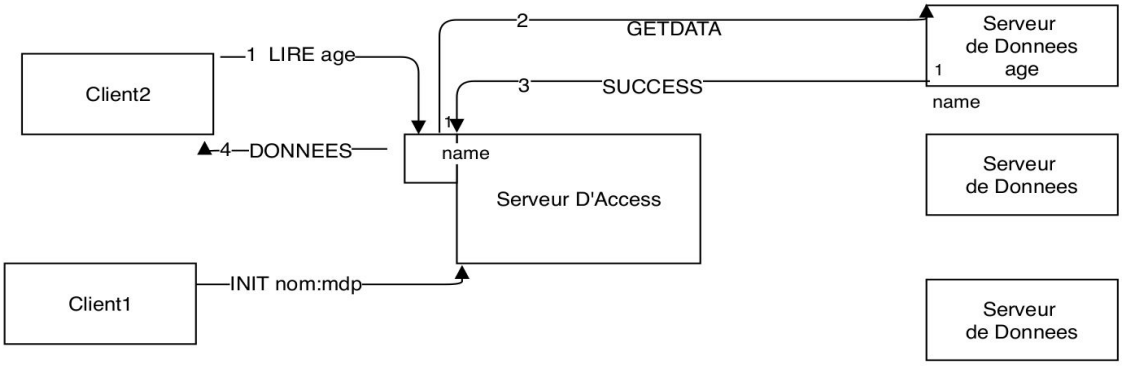
```
struct connectiondata{
   struct clientConnection client;
   int sockIP;
   short sockPort;
};
```

La fonction thread a également deux sockets. Il est en charge de toutes les demandes des clients. Il peut recevoir 4 types de demandes: LIRE, ECRIRE, SUPRIMMER, FIN.
ECRIRE: le client envoie le nom du champ du message et la valeur qu'il veut modifier l'ancien ou tout simplement écrire (exemple: age: 20), la fonction de thread le reçoit. Et envoyez ce message et id de client à tous les nœuds qui ont le même nom de champ que celui indiqué dans le message. Le reste du travail sera effectué par le nœud et il retournera le succès quand il aura fini. Le thread informera le client des changements. Pour savoir s'il y a un problème, nous utilisons la fonction **SELECT**. Si aucun serveur ne répond, il sera supprimé de la liste des serveurs.

SUPRIMMER: Même idée avec ECRIRE mais à ce moment-là, le message de suppression et l'identifiant du client seront envoyés au noeud.



LIRE: il recevra le message des clients sur le champ que le client veut lire, avec la fonction droitsauxChamps on vérifie à quels champs ce client a les droits. Après il enverra la requête de LIRE aux nœuds responsables des champs auxquels le client a droit. Le reste du travail sera effectué par les nœuds. chaque nœud renverra des données au serveur d'accès à un moment donné. Le serveur d'accès enverra ces données au client qui a fait la demande.



FIN: ce serveur de cas trouvera ce client dans la liste des clients et le supprimera de là. Et enverra un message au client. Enfin, dès que le client recevra ce message, il se terminera.

CLIENT

Le client prend son numéro de port auprès de l'utilisateur et crée un socket avec ce port pour la communication avec le serveur d'accès.

Passons maintenant aux requêtes, qui sont effectuées par la fonction gererMsg.Il prend le tampon de l'utilisateur pour l'analyser, le convertit au format de message et envoie au serveur l'accès.

```
//identificateur de type de message: INIT | SUPRIMER | LIRE | ECRIRE |
ERREUR | FIN
struct message *gererMsg( char *tampon  ){
 struct message *msg;
 msg = (struct message *)malloc( sizeof(struct message) );
 if( !memcmp(tampon,"FIN",3) ){
   msg->type = FIN;
   strcpy(msg->donnees, tampon);
   msg->nbDonnees=strlen(tampon);
 }
 else if( !memcmp(tampon,"LIRE",4) ){
   msg->type = LIRE;
   //tampon = "LIRE utilisateur:motDePasse" ==> tampon + 5 =
"utilisateur:motDePasse"
   strcpy(msg->donnees, tampon+5);
   msg->nbDonnees=strlen(tampon+5);
 }
 else if( !memcmp(tampon,"SUPPRIMER",8) ){
   msg->type = SUPR;
   strcpy(msg->donnees, tampon+9);
   msg->nbDonnees=strlen(tampon+9);
 }
 else if( !memcmp(tampon,"ECRIRE",6) ){
   msg->type = ECRIRE;
   strcpy(msg->donnees, tampon+7);
   msg->nbDonnees=strlen(tampon+7);
 }
 else if( !memcmp(tampon,"INIT",4) ){
   msg->type = INIT;
   strcpy(msg->donnees, tampon+5);
   msg->nbDonnees=strlen(tampon+5);
 }
 else{
   return NULL;
 }
 return msg;
}
```

Après chaque demande, le client recevra la réponse. Il existe quelques possibilités:
ERREUR ou FIN: cela mettra fin à l'exécution.
CONEST: la connexion est établie, l'utilisateur peut continuer.
NOPERMISSION: vous n'avez pas l'autorisation de ce (ces) champ, l'utilisateur peut essayer un autre champ
CONFAIL: Echec de la connexion, l'utilisateur doit ajouter correctement
LIRE: Le client vient d'imprimer les données.
SUCCESS: le client l'aura lorsqu'il réussira à écrire ou à supprimer.

SERVEUR DE DONNÉES

Même idéologie de la connexion avec le serveur d'accès. Chaque serveur prend le nom des champs de l'utilisateur et crée son propre fichier de données. Pour identifier son propre fichier entre plusieurs chaque serveur ajoute son numéro de processus dans le nom de fichier. Le nœud envoie son nom de champ à l'accès au serveur, après avoir reçu un message du serveur d'accès (GETDATA ou SUCCESS) s'il existe un autre nœud avec le même champ, le message sera GETDATA. Par conséquent, ce nœud devrait obtenir les données après avoir pu passer aux requêtes. S'il n'y a pas de nœud avec le même champ, il recevra SUCCESS et passera directement aux requêtes.
Maintenant, le temps des traitement des requêtes:
ECRIRE: Ici, le programme recevra l'identifiant d'utilisateur et la ligne qui seront ajoutés ou remplaceront une autre ligne dans le fichier de données. Ensuite, ces informations seront ajoutées au fichier avec la fonction **writeOrReplaceLine** (nom de fichier, user_id, data).
Le fichier de données ressemble à:

id:nom

```
1:Eric
9:Mariam
4:Ali
```

La fonction **findLine(filename, id)** lit le fichier ligne par ligne pour trouver la ligne(qui commence par ID) dont nous avons besoin.

writeOrReplaceLine() lit le fichier, vérifie s'il y a une ligne qui commence par l'identifiant de l'utilisateur. Si oui, la fonction remplacera la ligne par un. Sinon, la ligne sera ajoutée à la fin du fichier. La valeur de retour est 0 en cas de succès. Et envoie le message de réussite au serveur d'accès.
SUPPRIMER: Même idée avec l'ECRIRE pour trouver la ligne dans le fichier de données. **deleteLine.** S'il trouvera la ligne, il supprimera et retournera 0 sinon il retournera le -1. Et envoie le message de réussite/erreur au serveur d'accès.
SENDDATA et LIRE: ici nous envoyons simplement des données au serveur d'accès, à son tour il enverra des données à un autre noeud ou au client qui a fait la requête.
Pour terminer il va supprimer les fichiers, fermer les descripteurs de socket et de fichier.

Quelques fonctions qui sont utilisées:

```
char *readFile(const char *fileName);
bool authentifierUtilisateur(char *nom, char *motdepasse, int *uid,
char *fichier);
int calculerNbChamps(char *text);
char **trouverChamps(char *nom, char *motdepasse,int uid, char
*fichier, int *nbCh);
```

readfile: lire le fichier dans le tableau de caractères.
authentifierUtilisateur: ajout du nouveau client à la liste des clients
calculerNbChamps: lit le tampon et compte combien de champs y existent
trouverChamps: à partir d'un fichier comprenant des identifiants clients, des mots de passe et des champs, recherche les champs sur lesquels le client a un droit.

NB: La taille des données transmises n'est pas limitée. Le programme est adapté à la grande quantité de données.