Socket

Séquence de connexion côté serveur

1. Création d'un socket
   `int s1 = socket(AF_INIT,...);`
   La variable `s1` est de même nature qu'un descripteur de fichier.
2. Liaison du socket avec l'adresse du réseau de la machine et le port du serveur.
   `bind(s1, <struc>);`
   La structure est composée d'une multitude d'information.
   Seul les paramètre d'adresse et de port nous intéresse dans notre cas.
   L'accès à l'adresse et port passe par des variable spécifique qui devrons prendre comme valeur. L'adresse de la machine défini dans un constante définie : `INADDR_ANY` et les port doit être rentrer avec une forme de type network : `htons(<port>)`
3. Se mettre à l'écoute des demandes de connexion.
   `listen(s1, <size>);`
   La variable `size` modélise la taille de la file d'attente de connexion.
   La fonction d'écoute est non bloquante.
4. Accepter la connexion
   `accept(s1, <struct>, <size_struct>);`
   La structure peut être aussi remplacé par l'adresse IP du client.
   Fonction bloquante, en attente d'une connexion
   Débloqué dès l'arrivée d'une connexion.
   retourne 1 socket à utilliser pour dialoguer sur la connexion.

Résumé :

```
void init_socket() {
        int s1 = socket(...);
        bind(s1, ...);
        listen(s1, ...);
        return s1;
}

int main() {
        int s1 = init_socket();
        int s2 = accept(s1, ...);
        return EXIT_SUCCESS;
}
```

Séquence de connexion côté client

1. Création d'un socket
   `int s = socket(...);`
2. Se connecter à l'application du serveur.
   `connect(s, <struct>, <size_struct>)`
   La structure contient l'adresse IP du serveur (32bits) et son port (16bits).
   Puisque il nous faut une adresse IP alors il faut gérer la redirection d'adresse (DNS)
   Retourne -1 si échec et 0 si Ok.

> **Info**
>
> La fonction `resolv(<nom_domaine>, <port>);` permet de faire de la résolution d'adresse en retournant une structure réseau qui peut être directement envoyé à la fonction `connect()`.

Envoie et réception de données (côté serveur et client)

Le serveur à initialiser sont socket. Il a accepté un socket client.
Côté client, sont socket a été initialisé et connecté au serveur.
Les données sont bidirectionnel dès lors que la connexion est établie entre les deux.
Pour écrire : `write(<socket>,...);`
Pour lire : `read(<socket>,...);`
Pour fermer la connexion : `close(<socket>);`

Comportement de la lecture

La fonction `read(...);` est bloquante si pas de données envoyées.
Les données lues sont extraites du flux de données arrivant.

> **Remarque**
>
> Dans le TP3, nous utiliserons des fonction plus haut niveau pour facilité la communication.

Fils d'attente des connexions (côté serveur)

La taille est fixé sur `listen()`.
La demande de connexion du client est mise en attente, si le serveur est occupé avec un autre client. C'est à dire pas sur `accept`.

> **Warning**
>
> Lorsque le client est mise en attente celui-ci n'est pas avertie. La fonction `connect` ne renvoie pas d'erreur. De plus, les fonction `write` sont acceptés cependant il seront

dans une fils d'attente. L'avantage est que les données ne sont pas perdu. Si le serveur débloque le client il peut retrouver toute les données envoyer. (la taille de la fils d'attente es données reste limité).

Gestion des erreurs

Pour tout les appels système, retourne -1 si echec.
Exemple :

```
int fd;
fd = open("fifo", ...);
if( fd == -1) {
        erreur_IO("ouverture fifo");
}
```