Les sockets datagrammmpes

Les différents types de sockets :

Un petit rappel sur les différents types de sockets existants:

- Les sockets datagramme **SOCK_DGRAM** (fonctionnement de boite aux lettres) :
    - échanges de message individuels
    - pas de connexion
    - basé sur le protocole **UDP** : pas de garantie de reception ni d'ordre.
- Les sockets stream **SOCK_STREAM** (fonctionnement comme un appel téléphonique) :
    - échange de flux de données
    - nécessite une connexion
    - basé sur le protocol **TCP** : assure la reception dans le bon ordre.

Communication via les sockets datagramme :

Client	**Serveur**
Créer la socket	**Créer la socket**
sclient = socket(...)	sserveur = socket(...)
	Attacher la socket à une adresse
	bind(sserveur, ...)
Communiquer	**Communiquer**
sendto(sclient, ...)	recvfrom(sserveur, ...)
recvfrom(sclient, ...)	sendto(sserveur, ...)
Fermer la socket	**Fermer la socket**
close(sclient)	close(sserveur)

Communication via les sockets stream :

Serveur (appelé)	**Client (appelant)**
Créer les sockets	
secoute = socket(...)	sclient = socket(...)
Attacher la socket à une adresse	
bind(secoute,...)	
Mettre la socket en mode écoute	

Serveur (appelé)	**Client (appelant)**
listen(secoute,...)	
Accepter une demande de connexion	**Demander une connexion**
sservice = accept(secoute,...)	connect(sclient,...)
Communiquer	
read(sservice,...)	write(sclient,...)
write(sservice,...)	read(sclient,...)
Fermer la connexion dans un ou deux sens	
shutdown(sservice,...)	shutdown(sclient,...)
Fermer la connexion et la socket associée	
close(sservice)	close(sclient)
Fermer la socket d'écoute	
close(secoute)	

Recevoir des données d'une socket datagramme

```
int recvfrom(
    int sockfd,                 /* descripteur de la socket */
    void *tampon,               /* zone de réception */
    size_t nbOct,               /* taille max du message */
    int drapeaux,               /* défaut : 0 */
    struct sockaddr *adresse,   /* pointeur vers l'adresse */
    socklen_t *longueur_adresse  /* pointeur vers la longueur */
);
```

Renvoie le nombre d'octets lus (un entier) ou *-1* en cas d'erreur.

Si *nbOct* < la taille du message reçu : le reste du message est perdu.

L'appel à la primitive *recvfrom()* est bloquant, il attendra jusqu'a la reception de nouvaux messages.

Envoyer des données datagramme

```
int sendto(
    int sockfd,                 /* descripteur de la socket */
    void *tampon,               /* adresse du message */
    size_t nbOct,               /* taille du message */
    int drapeaux,               /* défaut : 0 */
    struct sockaddr *adresse,   /* pointeur vers l'adresse */
    socklen_t *longueur_adresse  /* pointeur vers la longueur */
);
```

Renvoie le nombre d'octet envoyés ou -1 en cas d'échec.

L'appel est bloquant, tant qu'il y a quelque chose dans le tampon, la primitive attends que le tampon se décharge.

Un exemple de serveur

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <unistd.h>

#define PORT 12345
#define BUFFER_SIZE 1024

int main() {
    int sserveur = socket(AF_INET, SOCK_DGRAM, 0);

    struct sockaddr_in saddr = {0}, caddr = {0};
    saddr.sin_family = AF_INET;             // domaine IPv4
    saddr.sin_port = htons(PORT);           // port en format réseau
    saddr.sin_addr.s_addr = htonl(INADDR_ANY);  // toute adresse

    bind(sserveur, (struct sockaddr*) &saddr, sizeof(saddr));

    while (1) {
        // traiter les requêtes entrantes
        int caddrlen = sizeof(caddr);
        char requete[BUFFER_SIZE];
        int nbLus = recvfrom(sserveur, &requete, sizeof(requete), 0,
(struct sockaddr*) &caddr, &caddrlen);

        if (nbLus == 0) continue;           // un datagramme vide

        // traiter la requête
        char reponse[BUFFER_SIZE] = "Reponse du serveur";
        sendto(sserveur, &reponse, sizeof(reponse), 0, (struct sockaddr*)
&caddr, caddrlen);
    }

    close(sserveur);
    return 0;
}
```

Voici les étapes principales :

- Le serveur crée un socket UDP.

- Il attache ce socket à un port et attend des requêtes de clients.
- Lorsque le serveur reçoit un message d'un client, il traite la requête et envoie une réponse au client.