Memo minimaliste de l'API C des Sockets TCP/IP Posix

A- GENERALITES sur les sockets :

- 1/ Définition :

Une **socket** est une **porte de communication** par laquelle un processus peut émettre ou recevoir des messages. Pour utiliser ce concept, des bibliothèques ont été développées et intégrées sous UNIX BSD, puis normalisées par POSIX.

Les sockets peuvent être utilisées pour des processus qui désirent communiquer en local, ou à distance. Elles sont donc utilisées dans un domaine particulier, dit **domaine de communication**. Deux domaines de communication principaux sont définis :

- **`AF_UNIX`** : donne une portée de communication locale (i.e. sur la même machine).
- **`AF_INET`** : donne une portée de communication distante.

Nous nous intéresserons dans la suite de ce memo uniquement à ce second domaine (et ciblé IPv4).

Un **type de socket** définit le mode de communication dans le domaine : deux types principaux ont été définis pour les protocoles de transport historiques, respectivement TCP et UDP :

- **`SOCK_STREAM`** : correspond au mode connecté. Avant l'échange de données, une connexion de bout en bout est établie entre les deux processus communicants. Ce mode garantit la fiabilité de l'échange (pas de perte, pas de duplication, pas d'altération, séquencement assuré).
- **`SOCK_DGRAM`** : correspond à un mode de communication sans connexion (ou datagrammes) qui ne garantit pas la fiabilité des échanges.

Nous nous intéresserons dans la suite de ce memo uniquement au premier type.

Le tableau ci-dessous résume les configurations classiques :

Domaine	Type	Protocole
`AF_INET`	`SOCK_STREAM` `SOCK_DGRAM`	`TCP/IP` `UDP/IP`
`AF_UNIX`	`SOCK_STREAM` `SOCK_DGRAM`	

- 2/ Fichiers et structures de données :

- Pour utiliser les primitives d'exploitation des sockets `AF_INET` et des utilitaires annexes, tout programme C doit inclure les fichiers suivants :

```
<sys/types.h>
<sys/socket.h>
<arpa/inet.h>
<netdb.h>
```

- Des structures permettent de stocker les informations d'adressage nécessaires à l'utilisation des sockets. Dans le cas de `AF_INET`, ces structures sont définies dans le fichier `<netinet/in.h>`

```
struct in_addr { u_long s_addr; };
struct sockaddr_in {
      short sin_family;        /* AF_INET */
      u_short sin_port;        /* numéro de port, supérieur à 1024 */
      struct in_addr sin_addr; /* adresse(s) IPv4 */
      char sin_zero[8];        /* inutilisé */
};
```

B- UTILISATION en mode CONNECTE

Cette section s'intéresse à l'utilisation des sockets `AF_INET` de type `SOCK_STREAM`. Elle rappelle les opérations à effectuer d'un côté par un **processus client** et de l'autre un **processus serveur**.

Opérations réalisées dans l'ordre côté SERVEUR

- Créer une socket d'écoute.
- Associer une adresse socket (adresse IP et numéro de port) au service : c'est l'opération de binding.
- Mettre la socket à l'écoute de demandes d'établissement de connexion TCP entrantes.
- Accepter une connexion TCP entrante et créer une nouvelle socket dédiée à cette connexion.
- Recevoir/Lire puis Emettre/Ecrire sur la nouvelle socket.
- Fermer la nouvelle socket

Opérations réalisées dans l'ordre côté CLIENT

- Créer une socket.
- Se connecter au serveur en donnant l'adresse de la socket distante (adresse IP du serveur et numéro de port de service). Cette connexion attribue un numéro de port au client.
- Emettre/Ecrire puis Recevoir/Lire sur la socket.
- Fermer la socket.

D- PRIMITIVES d'UTILISATION des SOCKETS en mode CONNECTE

- a - création d'une socket :

```
int socket (domaine, type, protocole)
int domaine;      /* AF_INET ( ou AF_UNIX) */
int type;         /* SOCK_STREAM (ou SOCK_DGRAM) */
int protocole;    /* code du protocole à utiliser. */
                  /* 0 pour détermination automatique */
```

Cette fonction retourne un entier, qui identifiera la socket créée dans le reste du programme.
Pour plus d'informations : http://manpagesfr.free.fr/man/man2/socket.2.html

Exemple d'utilisation :

```
int sid;
...
sid = socket(AF_INET, SOCK_STREAM, 0);
if (sid == -1) {
    perror("Socket creation error");
    return EXIT_FAILURE;
}
...
```

- b - association (binding) :

```
int bind (sock, localaddr, addrlen)
int sock;                          /* descripteur de socket */
struct sockaddr_in *localaddr;     /* adresse socket locale*/
int addrlen;                       /* longueur de l'adresse */
```

Cette fonction permet d'associer une adresse locale à un descripteur de socket créé par `socket()`. Cette opération est facultative pour les clients qui utilisent les sockets en mode connecté, car l'adresse est attribuée au moment de la connexion avec le serveur si le binding n'a pas été effectué.

Cette fonction retourne un nombre négatif si l'opération s'est mal déroulée.

Remarque :

Avant d'appeler cette primitive, il est nécessaire d'initialiser les champs de la structure `sockaddr_in` à zéro, puis de donner les valeurs correctes des champs suivants :

- le domaine de communication,
- le numéro de port (soit fixé par programme, soit 0),
- l'adresse IP d'une interface de la machine, ou bien la constante `INADDR_ANY` qui permet de lier à la socket l'ensemble des interfaces de l'équipement.

Exemple d'utilisation :

```
struct sockaddr_in sk_addr;
...
memset(&sk_addr, 0, sizeof(sk_addr));
sk_addr.sin_family = AF_INET;
sk_addr.sin_port = htons(PORT_NUMBER);
/* htons: host to net byte order (short int) */
sk_addr.sin_addr.s_addr = htonl(INADDR_ANY); /* any interface */

if (bind(sid, (struct sockaddr*) &sk_addr, sizeof(sk_addr)) == -1) {
  perror("Bind error");
  close(sid);
  return EXIT_FAILURE;
}
```

- c - demande de connexion d'un client à un serveur :

```
int connect (sock, servaddr, addrlen)
int sock;                    /* descripteur de socket */
struct sockaddr_in *servaddr; /* adresse du serveur distant*/
int addrlen;                 /* longueur de l'adresse */
```

Cette fonction retourne un entier négatif si la connexion TCP avec le processus distant n'a pu être réalisée.

Remarque :

Avant d'appeler cette primitive, il est nécessaire d'initialiser les champs de la structure `sockaddr_in` à zéro, puis de fixer les valeurs correctes pour les champs suivants :

- le domaine de communication,
- le numéro de port (celui associé à la socket du serveur distant),
- l'adresse de la machine distante sur laquelle s'exécute le processus serveur (la fonction `inet_pton` permet de convertir une chaîne de caractères représentant une adresse IPv4 en notation décimale pointée en une structure d'adresse réseau).

Exemple d'utilisation :

```
struct sockaddr_in dst_serv_addr;
...
memset(&dst_serv_adr, 0, sizeof(dst_serv_adr));
dst_serv_adr.sin_family = AF_INET;
dst_serv_adr.sin_port = htons(SERVER_PORT);
/* htons: host to net byte order (short int) */
inet_pton(AF_INET, server_ipv4, &(dst_serv_adr.sin_addr));
...
if (connect(sid, (struct sockaddr*) &dst_serv_adr,
            sizeof(dst_serv_adr)) == -1) {
      perror("Connection error");
      close(sid);
      return EXIT_FAILURE;
}
```

- d - mise d'un serveur à l'état d'écoute :

```
int listen (sock, qlen)
int sock;   /* descripteur de socket */
int qlen;   /* nombre maximum de demandes de connexion TCP  */
            /* qui peuvent être mises en attente */
```

Cette fonction indique que le serveur est prêt à recevoir au maximum `qlen` demandes de connexion simultanément.

Exemple d'utilisation :

```
if (listen(sid, 15) == -1) {
      perror("Listen error");
      close(sid);
      return EXIT_FAILURE;
}
```

- e - acceptation de connexion par un serveur :

```
int accept (sock, addrdist, addrlen)
int sock;   /* descripteur de socket locale d'écoute */
struct sockaddr_in *addrdist; /* adresse du socket distant se connectant*/
int *addrlen;/* longueur de l'adresse */
```

Cette fonction, bloquante, est utilisée par le serveur pour créer une socket dédiée à une nouvelle connexion et retourner le descripteur de cette socket spécifique, descripteur qui sera utilisé pour l'échange de données avec le client distant désormais connecté. A noter que la structure pointée par `addrdist` sera remplie par l'adresse du correspondant se connectant.

Exemple d'utilisation :

```
int newsock;
int sid;
struct sockaddr_in cli_adr;
socklen_t cli_adr_len = sizeof(cli_adr);
...
newsock = accept(sid, &cli_adr, &cli_adr_len);
if (newsock < 0) {
      perror("accept failed");
      close(sid);
      exit(EXIT_FAILURE);
  }
```

- f - fermeture de connexion :

```
int close (sock)
int sock;         /* descripteur de socket */
```

Le noyau essaie d'envoyer les données non émises avant de sortir du close().

Exemple d'utilisation :

```
      int sid;
      ...
      close(sid);
```

- g - envoi de messages en mode connecté :

```
int send (sock, mess, longmess, flags)
int sock;          /* descripteur de socket */
char * mess;       /* message à expédier */
int longmess;      /* nombre d'octets du message */
inf flags;         /* à 0 en utilisation classique */
```

Cette primitive permet d'envoyer sur la socket `sock` un message `mess` de `longmess` octets. Elle retourne le nombre d'octets envoyés, -1 en cas d'échec.

- h - réception de messages en mode connecté :

```
int recv (sock, mess, longmess, 0))
int sock;          /* descripteur de socket */
char * mess;       /* adresse de stockage du message reçu */
int longmess;      /* nombre d'octets à prélever*/
inf flags;         /* à 0 en utilisation classique */
```

Cette primitive permet de retirer depuis la socket `sock` un message `mess` de `longmess` octets. Elle retourne le nombre d'octets reçus, -1 en cas d'échec.