Opérations sur les sockets :

`int socketpair(int domain, int type, int protocol, int sv[2]) ;`
Crée une paire de sockets interconnectée, bidirectionnelle entre sv[0] et sv[1]. Tout ce qui est écrit dans sv[0] apparait dans sv[1] et inversement.

`int socket(int domain, int type, int protocol)`
Crée une socket avec les caractéristiques indiquées. `AF_INET/AF_UNIX` pour domain et `SOCK_DGRAM/SOCK_STREAM` sont les valeurs les plus communes. `0` à `protocol` indique une valeur par défaut

`int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);`
Attache une socket de descripteur `sockfd` à une adresse indiquée par les paramètres qui suivent. `addr` est de type `sockaddr` qui est un type générique, dans la pratique on utilisera un type particulier au type d'adresse (`sockaddr_in` pour `AF_INET`, `sockaddr_un` pour `AF_UNIX`, etc)

`int listen(int sockfd, int backlog);`
Marque la socket de descripteur `sockfd` comme passive, c'est à dire pouvant attendre des connexions avec `accept`. `backlog` indique la taille maximum de la file d'attente des connexions sur la socket. À utiliser sur des sockets avec la valeur `protocol` à `SOCK_STREAM`.

`int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);`
Extrait la première demande de connexion de la file de connexion d'une socket passive de descripteur `sockfd` et renvoie une nouvelle socket connectée. `addrlen` est un paramètre **valeur/résultat** qui doit être intitialisé à la longueur du type d'adresse utilisée à l'appel de `accept()`. Au retour de `accept()` Les paramètres `addr` et `addrlen` contiennent l'adresse du nouveau client connecté et `addrlen` sa longueur. À utiliser sur des sockets avec la valeur `protocol` à `SOCK_STREAM`

`int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);`
Connecte la socket de descripteur `sockfd` à l'adresse spécifié par les champs `addr` et `addrlen`

Formats d'adresses AF_UNIX, AF_INET et AF_INET6

```
struct sockaddr_un {
sa_family_t
sun_family;
char
sun_path[UNIX_PATH_M
AX];
};
```

Adresse de type `AF_UNIX`. Le champs `sun_family` vaut toujours `AF_UNIX`. Le champs `sun_path` est un chemin d'accès.

La **longueur** de l'adresse à utiliser pour `connect()`, `accept()` et `bind()` est de `strlen(sun_path) + sizeof(sun_family)`

Pour plus de détails voir page man unix(7)

```
struct sockaddr_in {
sa_family_t
sin_family;
in_port_t sin_port;
struct in_addr
sin_addr;
};
```

Adresse de type `AF_INET`. Le champs `sin_family` vaut toujours `AF_INET`.

Le champs `sin_port` est un numéro de port sur 16 bits dans l'ordre réseau, c'est à dire **big endian**. On le renseigne ou on le lit avec les primitives `htons(3)` et `ntohs(3)`

Le champs `sin_addr` est une adresse IP sur 32 bits dans l'ordre réseaux, c'est à dire **big endian** manipulable avec les primitives `inet_addr()`, `inet_ntoa()`, etc (voir man `inet_addr(3)`)

La **longueur** de l'adresse à utiliser pour `connect()`, `accept()` et `bind()` est de `sizeof(struct sockaddr_in)`. Pour attacher sur toutes les adresses IP de la machine, il faut initialiser le champs in_addr de la structure `sockaddr_in` a `htonl(INADDR_ANY)`

Pour plus de détails voir page man ip(7)

```
struct sockaddr_in6
{              sa_fam
ily_t   sin6_family;
in_port_t
sin6_port;
uint32_t
sin6_flowinfo;
struct in6_addr
sin6_addr;
uint32_t
sin6_scope_id;
};
```

Similaire à la structure `sockaddr_in` mais pour les sockets de `domain AF_INET6`

Pour plus de détails voir page man ipv6(7)