

# RE41 TP3: Aide sockets UNIX

---

Christophe DUMEZ <christophe.dumez@utbm.fr>

## Création d'un socket

Extrait de « man socket » :

```
#include <sys/types.h>
#include <sys/socket.h>

int socket(int domain, int type, int protocol);
```

exemple pour un socket TCP en mode « flux de données » :

```
int sock = socket(AF_INET, SOCK_STREAM, 0);
```

## Connexion au serveur

Extrait de « man connect » :

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

int connect(int sockfd, const struct sockaddr *serv_addr,
            socklen_t addrlen);
```

Extrait de « man gethostbyname » :

```
#include <netdb.h>

struct hostent *gethostbyname(const char *name);
```

Exemple :

```
struct sockaddr_in sin = { 0 };
struct hostent *hostinfo = gethostbyname("localhost");
sin.sin_addr = *(struct in_addr *) hostinfo->h_addr;
sin.sin_port = htons(PORT);
sin.sin_family = AF_INET;
connect(sock, (struct sockaddr *) &sin, sizeof(struct sockaddr));
```

## **Lecture sur un socket**

Extrait de « man recv » :

```
#include <sys/types.h>
#include <sys/socket.h>

ssize_t recv(int s, void *buf, size_t len, int flags);
```

exemple :

```
#define BUF_SIZE 1024
char buffer[BUF_SIZE];
/* Effacer la chaîne de caractères */
memset(&buffer[0], 0, BUF_SIZE);
/* Lecture sur le socket */
int nb = recv(sock, buffer, BUF_SIZE, 0);
```

## **Ecriture sur un socket**

Extrait de « man send » :

```
#include <sys/types.h>
#include <sys/socket.h>

ssize_t send(int s, const void *buf, size_t len, int flags);
```

Exemple :

```
#define BUF_SIZE 1024
char buffer[BUF_SIZE] = "HELLO\n";
send(sock, buffer, strlen(buffer), 0);
```

## **Fermeture d'un socket**

Extrait de « man close » :

```
#include <unistd.h>

int close(int fd);
```

## Sélection d'un file descriptor

Dans le cadre de ce TP, le client peut recevoir du texte à partir de deux descripteurs de fichiers :

1. `stdin` (`#define STDIN 0`): correspondant au texte écrit par l'utilisateur au clavier
2. `sock`: le socket de communication avec le socket

Une solution serait de traiter ces deux descripteurs en parallèle, dans des processus différents. Cependant, la fonction `SELECT` rend cette tâche inutile en effectuant la mise en attente sur plusieurs descripteurs de fichiers.

Extrait de « `man select` » :

```
#include <sys/select.h>
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>

int select(int nfd, fd_set *readfds, fd_set *writefds,
          fd_set *exceptfds, struct timeval *timeout);

void FD_CLR(int fd, fd_set *set);
int FD_ISSET(int fd, fd_set *set);
void FD_SET(int fd, fd_set *set);
void FD_ZERO(fd_set *set);
```

### Paramètre de `select()`

1. **`nfd`** correspond au numéro du plus grand descripteur de fichier parmi les 3 ensembles (`readfds`, `writefds`, `exceptfds`)
2. **`readfds`** correspond à l'ensemble surveillé pour vérifier si des caractères deviennent disponibles en lecture
3. **`writefds`** correspond à l'ensemble surveillé pour vérifier si des caractères deviennent disponibles en écriture (« `(fd_set *) 0` » pour ignorer)
4. **`exceptfds`** correspond à l'ensemble surveillé pour l'occurrence de conditions exceptionnelles (« `(fd_set *) 0` » pour ignorer)
5. **`timeout`** est une limite supérieure au temps passé dans `select()` avant son retour (« `NULL` » pour indéfiniment)

### Fonctions de manipulation d'ensembles

1. **`FD_CLR()`** supprime un descripteur d'un ensemble
2. **`FD_ISSET()`** teste l'appartenance d'un descripteur à un ensemble
3. **`FD_SET()`** ajoute un descripteur à un ensemble
4. **`FD_ZERO()`** vide un ensemble

### Fonctionnement

```
TANT QUE 1
    % Nécessaire d'initialiser les ensembles à chaque itération de la boucle car
    % select() modifie les ensembles afin de ne laisser uniquement ceux qui ont
    % causé son retour
```

```

INITIALISER_ENSEMBLES(readfds, writefds, exceptfds)
entier nb = select( readfds, writefds, exceptfds, NULL)
Si nb > 0 alors
    % Utiliser FD_ISSET() pour tester l'appartenance d'un descripteur de
    % de fichier à un ensemble et ainsi savoir s'il est prêts
    % Puis, lire ou écriture le descripteur de fichier
    Si FD_ISSET(sock, readfds) Alors
        read(sock, buffer, BUF_SIZE)
    FinSi
    Si FD_ISSET(STDIN, readfds) Alors
        read(STDIN, buffer, BUF_SIZE)
    FinSi
    [...]
FinSi
FinTantQue

```

## Annexe

### Création d'un serveur

Extrait de « man bind » :

```

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);

```

Après création d'une socket, il s'agit de la lier à un point de communication défini par une adresse et un port, c'est le rôle de la fonction *bind()*.

Extrait de « man listen » :

```

#include <sys/types.h>
#include <sys/socket.h>

int listen(int sockfd, int backlog);

```

La fonction *listen()* marque la socket comme passive, ce qui signifie qu'elle sera utilisé pour accepter des connexions entrantes.

*backlog* correspond au nombre maximal de connexions dans la file d'attente.

Extrait de « man accept » :

```

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

```

```
int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
```

La fonction `accept()` est employé sur les sockets en mode connecté. Elle extrait la première connexion en attente sur le socket `sockfd`. Elle crée un nouveau socket pour cette connexion et le retourne.

S'il n'y a pas de connexion dans la file, `accept()` met le programme en attente jusqu'à une éventuelle connexion.

Exemple :

```
struct sockaddr_in sin = { 0 };
sin.sin_addr.s_addr = htonl(INADDR_ANY);
sin.sin_port = htons(PORT);
sin.sin_family = AF_INET;
/* Attachement de la socket */
bind(socket, (struct sockaddr *) &sin, sizeof(struct sockaddr));
listen(socket,10);
struct sockaddr_in saddr = { 0 };
socklen_t saddr_size = sizeof(saddr);
/* Attente de connexion */
int client_sock = accept(sock, (struct sockaddr *)&saddr, &saddr_size);
printf("Client IP: %s\n", inet_ntoa(saddr.sin_addr));
```