Communication Interprocessus sous BSD
les sockets

Université François Rabelais de Tours
Faculté des Sciences et Techniques
Antenne Universitaire de Blois

Licence Sciences et Technologies

Mention : Informatique

2ème Année

Mohamed TAGHELIT
taghelit@univ-tours.fr

Communication Interprocessus sous BSD
les sockets

- Caractéristiques des Sockets (Type, Domaine)
- Fichiers d'entête et structures associées
- Fichiers d'administration liés au réseau et routines associées
- Création/Nommage d'une Socket
- Communication locale (modes non connecté et connecté)
- Communication distante (modes non connecté et connecté)
- Multiplexage des Entrées/Sorties

Caractéristiques des Sockets

- À une Socket est associé un **TYPE** (Qualité de transmission) :
  - **SOCK_DGRAM**
  - **SOCK_STREAM**
  - **SOCK_RAW**
  - **SOCK_RDM**
  - **SOCK_SEQPACKET**
  - …

- À une Socket est associé un **DOMAINE** (Type d'adressage) :
  - **AF_UNIX**
  - **AF_INET**
  - **AF_IPX**
  - **AF_APPLETALK**
  - …
  - …

- Associations possibles entre **TYPE** et **DOMAINE**
  - **SOCK_DGRAM + AF_UNIX**
  - **SOCK_DGRAM + AF_INET**
  - **SOCK_STREAM + AF_UNIX**
  - **SOCK_STREAM + AF_INET**
  - **SOCK_RAW + AF_INET**
  - …

Fichiers d'Entête et Structures Associées

- Les fichiers suivants (contenus dans le répertoire **/usr/include**) sont nécessaires pour toute utilisation des Sockets
  - **sys/types.h**

```
…
typedef unsigned short u_short;
typedef unsigned long u_long;
…
```

  - **sys/socket.h**

```
struct sockaddr {
   u_short sa_family;     /* address family : AF_XXX value */
   char sa_data[14];      /* up to 14 bytes of protocol-specific address */
};
```

  - **sys/un.h**

```
struct sockaddr_un {
       short sun_family;  /* AF_UNIX */
       char sun_path[108];
};
```

Fichiers d'Entête et Structures Associées

- **netinet/in.h**

```
struct in_addr
{
   u_long s_addr;                    /* 32-bit netid/hostid */
}
```

```
struct sockaddr_in
{
   short sin_family;                 /* AF_INET */
     u_short sin_port;               /* 16-bit port number */
          struct in_addr sin_addr; /* 32-bit netid/hostid */
               char sin_zero[8];   /* unused */
};
```

Fichiers d'Entête et Structures Associées

- **netdb.h**

```
struct hostent {
   char    *h_name;        /* Nom officiel de l'hote.   */
   char   **h_aliases;     /* Liste d'alias.            */
   int      h_addrtype;    /* Type d'adresse de l'hote. */
   int      h_length;      /* Longueur de l'adresse.    */
   char   **h_addr_list;   /* Liste d'adresses.         */ };
```

```
struct servent {
   char  *s_name;          /* Nom officiel du service */
   char **s_aliases;       /* Liste d'alias           */
   int    s_port;          /* Numero de port          */
   char  *s_proto;         /* Protocole utilise       */ };
```

```
struct netent {
   char        *n_name;          /* Nom officiel  du reseau */
   char       **n_aliases;       /* Liste d'alias           */
   int          n_addrtype;      /* Type d'adresse reseau   */
   unsigned long int n_net;      /* Adresse du reseau       */ };
```

```
struct protoent {
   char   *p_name;         /* Nom officiel du protocole */
   char  **p_aliases;      /* Liste d'alias             */
   int     p_proto;        /* Numero du protocole       */ };
```

Fichiers d'Administration (Réseau) et Routines Associées

- **/etc/hosts**

```
10.153.3.1   papin        s0      /* Exemple d'entrée */
```

```
struct hostent *gethostent(void);
struct hostent *gethostbyname(const char *name);
struct hostent *gethostbyaddr(const char *addr, int len, int type);
```

- **/etc/networks**

```
iup-net      10.153.3.0           /* Exemple d'entrée */
```

```
struct netent *getnetent(void)
struct netent *getnetbyname(const char *name);
struct netent *getnetbyaddr(long net, int type);
```

- **/etc/protocols**

```
ip     0      IP    # Protocole internet /* Exemple d'entrée */
tcp    6      TCP   # Protocole de contrôle de transmission
```

```
struct protoent *getprotoent(void)
struct protoent *getprotobyname(const char *name);
struct protoent *getprotobynumber(int proto);
```

- **/etc/services**

```
smtp   25/tcp       mail          /* Exemple d'entrée */
```

```
struct servent *getservent(void)
struct servent *getservbyname(const char *name, const char *proto);
struct servent *getservbyport(int port, const char *proto);
```

Autres Routines Concernant la Manipulation des Adresses

- `int bcmp (const void *s1, const void *s2, int n);`
- `void bcopy (const void *src, void *dest, int n);`
- `void bzero (void *s, int n);`
- `unsigned long int htonl(unsigned long int hostlong);`

  Convertit une valeur de 32 bits de l'ordre de la machine dans l'ordre du réseau.
- `unsigned short int htons(unsigned short int hostshort);`
- `unsigned long int ntohl(unsigned long int netlong);`

  Convertit une valeur de 32 bits de l'ordre du réseau dans l'ordre de la machine.
- `unsigned short int ntohs(unsigned short int netshort);`

❑ **Exemple**

Impression d'un numéro de port sur n'importe quelle machine, quel que soit le codage des données :

```
printf("Numéro de port %d", ntohs(sp->s_port));
```

Création d'une Socket

- Un processus peut créer à tout moment un point de communication **socket**

```
#include<sys/types.h>
#include<sys/socket.h>
      int socket(int domain, int type, int protocol);
```

Type de de la socket

Domaine de la socket

Protocole utilisé

- Valeur de retour :
  - entier positif (descripteur) en cas de succès,
  - -1 en cas d'erreur (et **errno** est modifiée en conséquence).

- Choix du protocole :
  - si **protocol = 0** → le système choisit le protocole adéquat
  - sinon,

```
struct protoent *pp;
…
pp = getprotobyname("tcp");
s = socket(AF_INET, SOCK_STREAM, pp->p_proto);
```

- À ce niveau, aucun processus d'une autre filiation ne peut atteindre la socket, il faut lui donner un nom.

Nommage d'une Socket

- Pour être accessible, une **socket** doit être nommée

```
#include<sys/types.h>
#include<sys/socket.h>
      int bind(int sockfd, const struct sockaddr *addr,
                               socklen_t addrlen);
```

Nom de la socket

Descripteur de la socket (retourné par **socket()**)

Taille du nom de la socket

- Un nom est un pointeur sur une structure de type **sockaddr_un** ou **sockaddr_in**,
  - nom dans le domaine Unix → **pathname**,
  - nom dans le domaine Internet → **n° de port** + **@IP** = **TSAP**.

- le nommage de la socket pour l'expéditeur (mode symétrique) ou pour le client (mode asymétrique) n'est pas obligatoire,

- Valeur de retour :
  - 0 en cas de succès,
  - -1 en cas d'erreur (et **errno** est modifiée en conséquence).

Communication en Mode Non Connecté

- En mode non connecté (symétrique), la **socket** doit être de type **SOCK_DGRAM**
- Émission de données

```
#include<sys/types.h>
#include<sys/socket.h>
    int sendto(int sockfd, const void *buf, size_t len, int flags,
        const struct sockaddr *dest_addr, socklen_t addrlen);
```

Descripteur de la socket locale (sockfd) ; Tampon et taille du message à transmettre (buf, len) ; Options (flags) ; Adresse (nom) de la socket cible (dest_addr) ; Taille de l'adresse (nom) de la socket cible (addrlen)

  - Valeur de retour :
    - le nombre de caractères émis en cas de succès,
    - -1 en cas d'erreur (et **errno** est modifiée en conséquence).

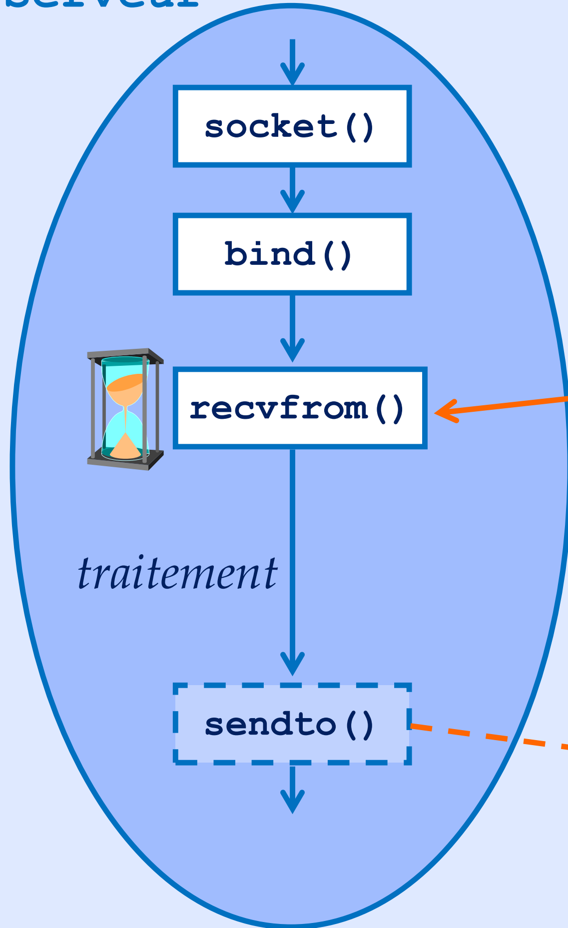
- Réception de données

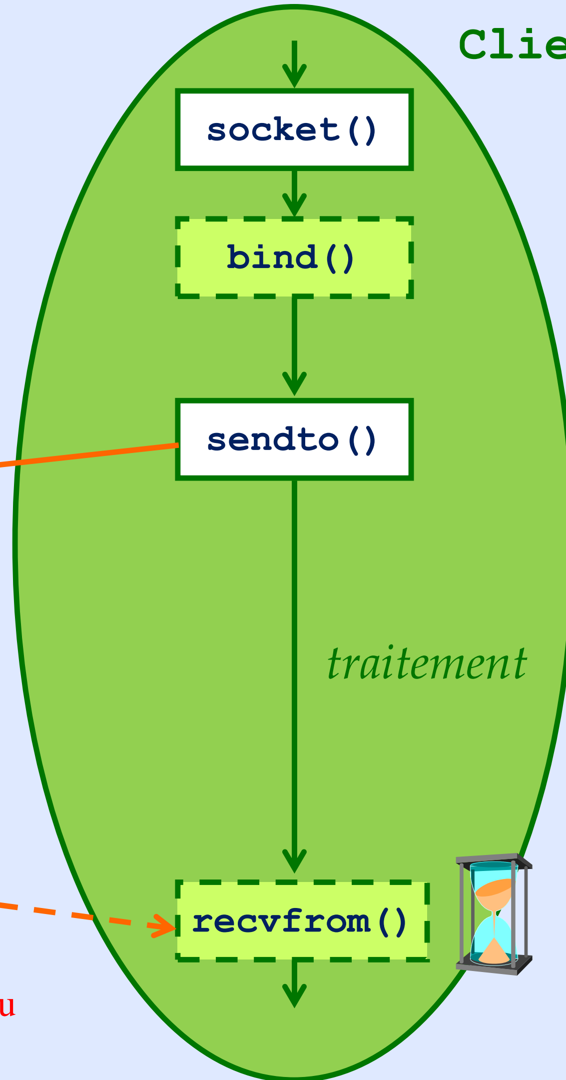
```
    int recvfrom(int sockfd, void *buf, size_t len, int flags,
        struct sockaddr *src_addr, socklen_t *addrlen);
```

Descripteur de la socket locale (sockfd) ; Tampon et taille du message à recevoir (buf, len) ; Options (flags) ; Adresse (nom) de la socket émettrice (src_addr) ; Taille de l'adresse (nom) la socket émettrice (addrlen)

  - Valeur de retour :
    - le nombre de caractères reçus en cas de succès,
    - -1 en cas d'erreur (et **errno** est modifiée en conséquence).

Communication en Mode Non Connecté "Canevas"





requête

réponse

possible si la socket du Client a été nommée (domaine AF_UNIX)

Communication en Mode Non Connecté Dans le Domaine AF_UNIX

❑ Programme **serv_unix.c** (processus récepteur)

```c
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <stdio.h>
#include <errno.h>
#include <signal.h>
#include <string.h>
#include <stdlib.h>
```

```c
void interrupt(int signo) {
    unlink("serv_sock"); exit(0);
}

void main() {
    int ret; int sd; char msg[128];

    struct sockaddr_un locale, source, cible;
    int fromlen = sizeof(source);

    signal(SIGINT, interrupt);

    locale.sun_family = AF_UNIX;
    strcpy(locale.sun_path, "serv_sock");

    if ( (sd = socket(AF_UNIX, SOCK_DGRAM, 0)) != -1)
        printf("socket = %d\t", sd);
    else ("Erreur socket");

    if ( (ret = bind(sd, (const struct sockaddr *)&locale, sizeof(locale))) != -1)
        printf("bind = %d\n", ret);
    else perror("Erreur bind");

for(;;) {
    if( (ret=recvfrom(sd, msg, sizeof(msg), 0, (struct sockaddr *)&source, &fromlen)) != -1 ){
        msg[ret] = '\0'; fprintf(stdout, "\trecvfrom = %d; msg = %s\n", ret, msg); }
    else perror("Erreur recvfrom");
cible = source;
    if((ret=sendto(sd,msg,strlen(msg),0,(const struct sockaddr *)&cible,sizeof(cible))) != -1)
        fprintf(stdout, "\t\tsendto = %d; msg = %s\n", ret, msg);
    else perror("Erreur sendto");
}
}
```

- s'exécute sur réception du signal **SIGINT**, et supprime le nom de la socket locale
- capte le signal **SIGINT**
- initialise la structure **locale**
- crée la socket locale
- nomme la socket locale
- réception d'un message
- émission d'une réponse (message reçu)

Communication en Mode Non Connecté Dans le Domaine AF_UNIX

❑ Programme `cli_unix.c` (processus émetteur)

```
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <stdio.h>
#include <errno.h>
#include <string.h>
```

```
void main() {
    int ret; int sd; char msg[128];
    struct sockaddr_un source, cible; int fromlen = sizeof(source);

    cible.sun_family = AF_UNIX;
    strcpy(cible.sun_path, "serv_sock");

    if ( (sd = socket(AF_UNIX, SOCK_DGRAM, 0)) != -1)
        printf("socket = %d\n", sd);
    else perror("Erreur socket");

for(;;) {
    printf("Entrer message : "); scanf("%s", msg);

    if((ret=sendto(sd,msg,strlen(msg),0,(const struct sockaddr*)&cible,sizeof(cible))) != -1)
        fprintf(stdout, "\tsendto = %d; msg = %s\n", ret, msg);
    else perror("Erreur sendto");

    if ((ret=recvfrom(sd,msg,sizeof(msg),0,(struct sockaddr *)&source,&fromlen)) != -1 ){
        msg[ret] = '\0'; fprintf(stdout, "\t\trecvfrom = %d; msg = %s\n", ret, msg); }
    else perror("Erreur recvfrom");
} }
```

- initialise la structure **cible**
- crée la socket locale
- invite l'utilisateur à introduire un message (texte) qui sera stocké dans **msg**
- émission du message
- réception d'un message (réponse)

❑ Exécution de `cli_unix.c`

```
[student]$ ./cli_unix
socket = 3
Entrer message : bonjour
    sendto = 7; msg = bonjour

^C
[student]$
```

❑ Exécution de `serv_unix.c`

```
[student]$ ./serv_unix
socket = 3  bind = 0
    recvfrom = 7; msg = bonjour
Erreur sendto: Invalid argument
^C
[student]$
```

Communication en Mode Non Connecté Dans le Domaine AF_UNIX

❑ Programme **cli2_unix.c** (processus émetteur modifié – nommage socket locale)

```
void main() {
    int ret; int sd; char msg[128];
    struct sockaddr_un locale, source, cible; int fromlen = sizeof(source);

    locale.sun_family = AF_UNIX;
    strcpy(locale.sun_path, "cli_sock");

    cible.sun_family = AF_UNIX;
    strcpy(cible.sun_path, "serv_sock");

    if ((sd = socket(AF_UNIX, SOCK_DGRAM, 0)) != -1)
        printf("socket = %d\t", sd);
    else perror("Erreur socket");

    if (( ret = bind(sd, (const struct sockaddr *)&locale, sizeof(locale))) != -1)
        printf("bind = %d\n", ret);
    else perror("Erreur bind");

for (;;) {
    printf("Entrer message : "); scanf("%s", msg);

    if ((ret=sendto(sd,msg,strlen(msg),0,(const struct sockaddr*)&cible,sizeof(cible))) != -1)
        fprintf(stdout, "\tsendto = %d; msg = %s\n", ret, msg);
    else perror("Erreur sendto");

    if ((ret=recvfrom(sd,msg,sizeof(msg),0,(struct sockaddr *)&source,&fromlen)) != -1) {
        msg[ret] = '\0'; fprintf(stdout, "\t\trecvfrom = %d; msg = %s\n", ret, msg); }
    else perror("Erreur recvfrom");
}
}
```

initialise la structure **locale**

```
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <stdio.h>
#include <errno.h>
#include <string.h>
```

nommage de la socket locale

Communication en Mode Non Connecté Dans le Domaine AF_UNIX

❑ Exécution de **cli2_unix.c**

```
[student]$ gcc cli2_unix.c -o cli2_unix
[student]$ ./cli2_unix
socket = 3  bind = 0
Entrer message : bonjour
    sendto = 7; msg = bonjour
        recvfrom = 7; msg = bonjour
Entrer message : au revoir
    sendto = 2; msg = au
        recvfrom = 2; msg = au
Entrer message :    sendto = 6; msg = revoir
        recvfrom = 6; msg = revoir
Entrer message : ^C       ← interruption de cli2_unix
[student] ./cli2_unix     ← relance de cli2_unix
Erreur bind: Address already in use
socket = 3  Entrer message : ^C
[student]$ rm cli_sock    ← suppression du nom de la socket locale
/bin/rm : supprimer socket « cli_sock » ? y
[student]$ ./cli2_unix    ← relance de cli2_unix
socket = 3  bind = 0
Entrer message : merci
    sendto = 5; msg = merci
        recvfrom = 5; msg = merci
Entrer message : ^C
[student]
```

❑ Exécution de **serv_unix.c**

```
[student]$ gcc serv_unix.c -o serv_unix
[student]$ ./serv_unix
socket = 3  bind = 0

    recvfrom = 7; msg = bonjour
        sendto = 7; msg = bonjour
    recvfrom = 2; msg = au
        sendto = 2; msg = au
    recvfrom = 6; msg = revoir
        sendto = 6; msg = revoir
```

serv_unix toujours en cours d'exécution et en attente de réception de message

```
    recvfrom = 5; msg = merci
        sendto = 5; msg = merci

^C
[student]
```

Communication en Mode Non Connecté Dans le Domaine AF_INET

❑ Programme **serv_inet.c** (processus récepteur)

```c
void main() {
     int ret; int sd; char msg[128]; char name[128];

     struct sockaddr_in locale, source, cible; struct hostent *hp;
     int fromlen = sizeof(source);

     if (gethostname(name, sizeof(name)) != 0) perror("Erreur de gethostname");
     if ((hp = gethostbyname(name)) == NULL) perror("Erreur gethostbyname");

     bzero((char *) &locale, sizeof(locale));
     bcopy(hp->h_addr, (char *) &locale.sin_addr, hp->h_length);
     locale.sin_family = hp->h_addrtype;
     locale.sin_port = htons(2000);

     if ((sd = socket(AF_INET, SOCK_DGRAM, 0)) != -1)
          printf("socket = %d\t", sd);
     else perror("Erreur socket");

     if ((ret = bind(sd, (const struct sockaddr *) &locale, sizeof(locale))) != -1)
          printf("bind = %d\n", ret);
     else perror("Erreur bind");

for (;;) {
     if ((ret = recvfrom(sd, msg, sizeof(msg), 0, (struct sockaddr *) &source, &fromlen)) != -1)
          msg[ret] = '\0'; fprintf(stdout, "\trecvfrom = %d; msg = %s\n",
     else perror("Erreur recvfrom");
cible = source;
     if ((ret=sendto(sd,msg,strlen(msg),0,(const struct sockaddr *)&cible,sizeof(cible))) != -1)
          fprintf(stdout, "\t\tsendto = %d; msg = %s\n", ret, msg);
     else perror("Erreur sendto");
}
}
```

récupère dans **name** le nom de la machine locale

récupère la structure **hostent** de la machine locale

initialise la structure **locale**

crée la socket locale

nomme la socket locale

réception d'un message

émission d'une réponse (message reçu)

```c
#include <sys/types.h>
#include <sys/socket.h>
#include <stdio.h>
#include <errno.h>
#include <netdb.h>
#include <netinet/in.h>
#include <string.h>
```

Communication en Mode Non Connecté Dans le Domaine AF_INET

- Programme **cli_inet.c** (processus émetteur)

```
void main(int argc, char **argv) {
    int ret; int sd; char msg[128];

    sockaddr_in source, cible; struct hostent *hp;
    int fromlen = sizeof(source);

    if ((hp = gethostbyname(argv[1])) == NULL) perror("Erreur gethostbyname");

    bzero((char *) &cible, sizeof(cible));
    bcopy(hp->h_addr, (char *) &cible.sin_addr, hp->h_length);
    cible.sin_family = hp->h_addrtype;
    cible.sin_port = htons(2000);

    if ((sd = socket(AF_INET, SOCK_DGRAM, 0)) != -1)
        printf("socket = %d\n", sd);
    else perror("Erreur socket");

for (;;) {
    fprintf(stdout, "Entrer message : "); scanf("%s", msg);

    if ((ret=sendto(sd,msg,strlen(msg),0,(const struct sockaddr *)&cible,sizeof(cible))) != -1)
        fprintf(stdout, "\tsendto = %d; msg = %s\n", ret, msg);
    else perror("Erreur sendto");

    if ((ret = recvfrom(sd, msg, sizeof(msg), 0, (struct sockaddr *) &source, &fromlen)) != -1){
        msg[ret] = '\0'; fprintf(stdout, "\t\trecvfrom = %d; msg = %s\n", ret, msg); }
    else perror("Erreur recvfrom");
}
}
```

récupère la structure **hostent** de la machine distante dont le nom est passé en paramètre de ligne de commande (**argv[1]**).

initialise la structure **cible**

crée la socket locale

invite l'utilisateur à introduire un message (texte) qui sera stocké dans **msg**

émission du message

réception d'un message (réponse)

```
#include <sys/types.h>
#include <sys/socket.h>
#include <stdio.h>
#include <errno.h>
#include <netdb.h>
#include <netinet/in.h>
#include <string.h>
```

Communication en Mode Non Connecté Dans le Domaine AF_INET

- Exécution de **cli_inet.c**

```
[student]$ gcc cli_inet.c -o cli_inet
[student]$ ./cli_inet machine_serveur
socket = 3
Entrer message : bonjour
    sendto = 7; msg = bonjour
        recvfrom = 7; msg = bonjour
Entrer message : au revoir
    sendto = 2; msg = au
        recvfrom = 2; msg = au
Entrer message :    sendto = 6; msg = revoir
        recvfrom = 6; msg = revoir
Entrer message : merci
    sendto = 5; msg = merci
        recvfrom = 5; msg = merci
Entrer message : ^C
[student]]$ ./cli_inet machine_serveur
socket = 3
Entrer message : encore
    sendto = 6; msg = encore
        recvfrom = 6; msg = encore
Entrer message : ^C
[student]$
```

- Exécution de **serv_inet.c**

```
[student]$ gcc serv_inet.c -o serv_inet
[student]$ ./serv_inet
socket = 3  bind = 0

    recvfrom = 7; msg = bonjour
        sendto = 7; msg = bonjour
    recvfrom = 2; msg = au
        sendto = 2; msg = au
    recvfrom = 6; msg = revoir
        sendto = 6; msg = revoir
    recvfrom = 5; msg = merci
        sendto = 5; msg = merci

    recvfrom = 6; msg = encore
        sendto = 6; msg = encore
^C
[student]$
```

Établissement d'Une Connexion

- En mode connecté (asymétrique – client/serveur), la **socket** doit être de type **SOCK_STREAM**
- Établissement d'une connexion (côté serveur)
  - Attente de demandes de connexions

```
#include<sys/socket.h>
   int listen(int sockfd, int backlog);
```

longueur maximale de la file des connexions en attente pour **sockfd**

Descripteur de la socket locale

  - Notifie que la socket, référencée par **sockfd**, sera utilisée pour accepter (en utilisant **accept()**) les demandes de connexions entrantes. La socket est dite en l'état d'écoute.
  - **backlog** définit la longueur maximale pour la file des demandes de connexions en attente.
  - Valeur de retour : 0 en cas de succès et -1 en cas d'erreur.

  - Acceptation de demandes de connexions

paramètre-résultat qui sera rempli au retour avec l'adresse de l'entité se connectant

```
#include<sys/socket.h>
   int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
```

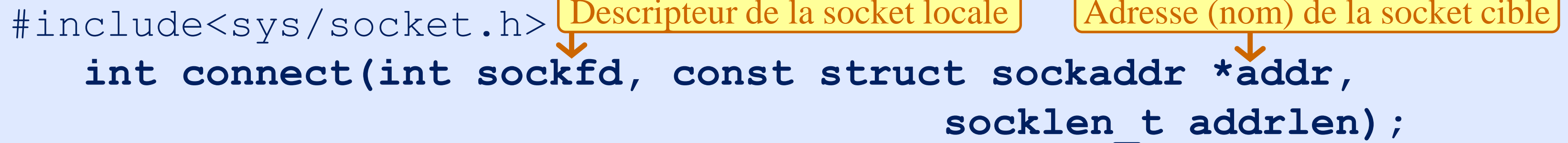
Descripteur de la socket locale à l'écoute

paramètre-résultat initialisé à la taille de **addr** et renseigné au retour à la taille réelle

  - Extrait la première demande de connexion de la file d'attente de la socket **sockfd**.
  - Crée une nouvelle socket et lui alloue un nouveau descripteur qui sera retourné.
  - La nouvelle socket n'est pas en l'état d'écoute et la socket originale n'est pas modifiée.
  - Valeur de retour : entier positif ou nul (descripteur) en cas de réussite et -1 en cas d'erreur.

Établissement d'Une Connexion

- Établissement d'une connexion (côté client)
  - Demande de connexion




```
#include<sys/socket.h>
   int connect(int sockfd, const struct sockaddr *addr,
                                      socklen_t addrlen);
```

- Tente de connecter la socket associée au descripteur **`sockfd`** à une autre socket dont l'adresse est indiquée par **`addr`**.
- Valeur de retour :
  - 0 en cas de succès,
  - -1 en cas d'erreur (et **`errno`** est modifiée en conséquence).

Communication en Mode Non Connecté

- En mode connecté (asymétrique), la **socket** doit être de type **SOCK_STREAM**
- Émission de données

```
#include<sys/types.h>
#include<sys/socket.h>
   ssize_t send(int sockfd, const void *buf, size_t len, int flags);
```

Descripteur de la socket locale (sockfd) ; Tampon du message à transmettre (buf) ; Taille du message à transmettre (len) ; Options (flags)

  - Ne peut être utilisée qu'avec les sockets connectées (le destinataire est connu).
  - Équivalent à **sendto(sockfd, buf, len, flags, NULL, 0);**
  - Valeur de retour :
    - le nombre de caractères émis en cas de succès,
    - -1 en cas d'erreur (et **errno** est modifiée en conséquence).

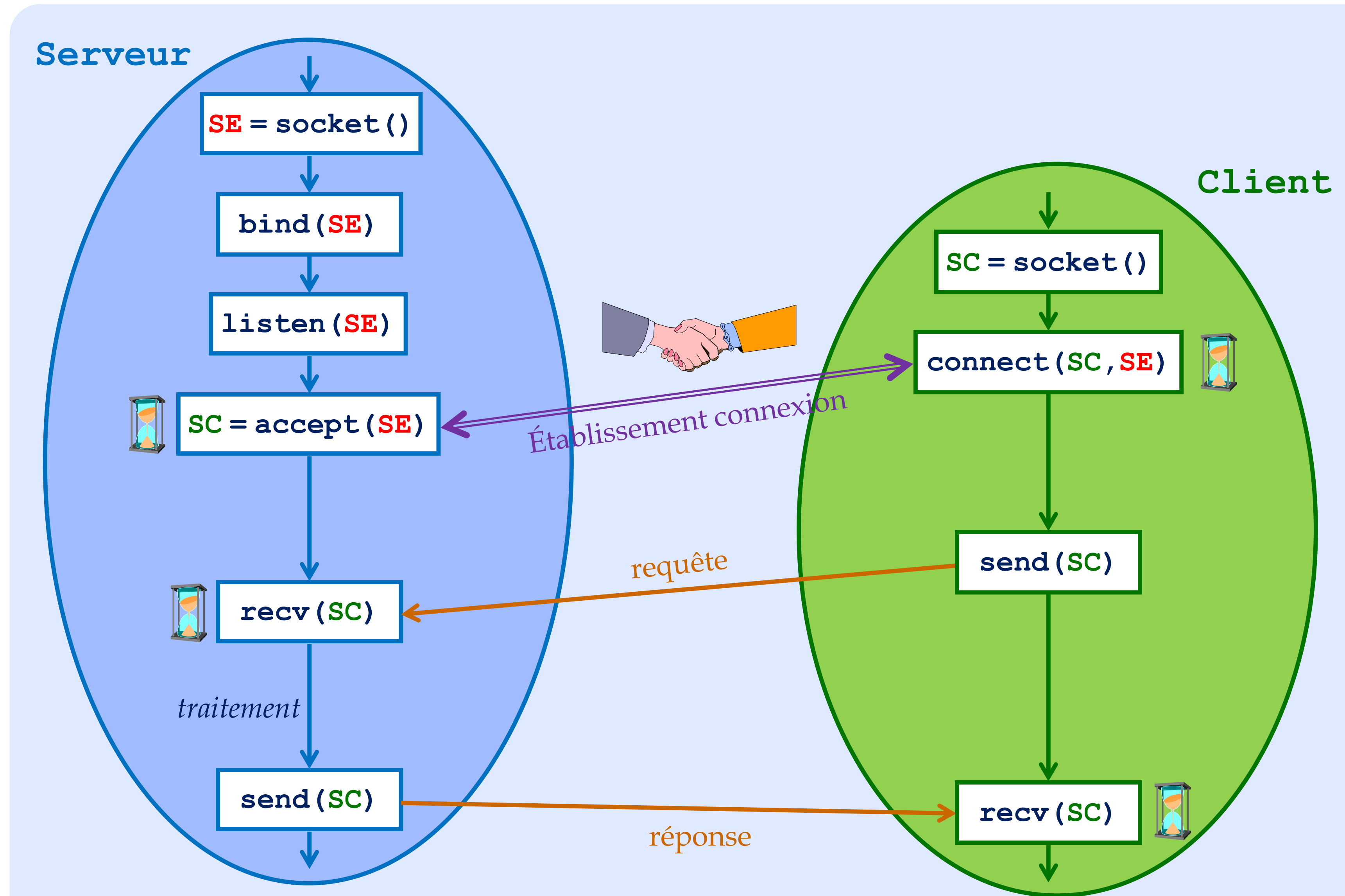
- Réception de données

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

Descripteur de la socket locale (sockfd) ; Tampon et taille du message à recevoir (buf, len) ; Options (flags)

  - Équivalent à **recvfrom(sockfd, buf, len, flags, NULL, 0);**
  - Valeur de retour :
    - le nombre de caractères reçus en cas de succès,
    - -1 en cas d'erreur (et **errno** est modifiée en conséquence).

Communication en Mode Connecté "Canevas"



Communication en Mode Connecté Dans le Domaine AF_UNIX

❑ Programme **serv_unix_conn.c** (serveur)

```
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <signal.h>
```

```
void interrupt(int signo) {
     unlink("serv_sock"); exit(0); }

void main() {
     int ret, sd, newsd; char msg[128];

     struct sockaddr_un locale, source; int fromlen = sizeof(source);
     signal(SIGINT, interrupt);
     locale.sun_family = AF_UNIX; strcpy(locale.sun_path, "serv_sock");
     if ((sd = socket(AF_UNIX, SOCK_STREAM, 0)) != -1) printf("socket = %d\t", sd);
     else { ("Erreur socket"); exit(-1); }
     if ((ret = bind(sd, (const struct sockaddr *) &locale, sizeof(locale))) != -1)
          printf("bind = %d\n", ret);
     else { perror("Erreur bind"); exit(-1); }
     if (listen(sd, 5) == -1) { perror("Erreur listen"); exit(-1); }
for(;;) {
     if ((newsd = accept(sd, (struct sockaddr *) &source, &fromlen)) == -1) {
          perror("Erreur accept"); exit(-1);
     }
     for (;;) {
          if ((ret = recv(newsd, msg, sizeof(msg), 0)) != 0 ) {
               msg[ret] = '\0'; fprintf(stdout, "\trecv = %d; msg = %s\n", ret, msg); }
          else { printf("Connexion coupée !\n"); break; }
     strcat(msg, " traitée !");
          if ((ret = send(newsd, msg, strlen(msg), 0)) != -1)
               fprintf(stdout, "\t\tsend = %d; msg = %s\n", ret, msg);
          else perror("Erreur send");
     }
close(newsd);
}
}
```

- interrupt : s'exécute sur réception du signal **SIGINT**, et supprime le nom de la socket locale
- signal : capte le signal **SIGINT**
- initialise la structure **locale**
- socket : crée la socket locale
- bind : nomme la socket locale
- listen : notifie que la socket locale est en état d'écoute
- for(;;) : boucle infinie sur les connexions
- accept : acceptation de connexion, sinon attente de demande
- for (;;) : boucle infinie sur les réceptions/émissions (d'une connexion)
- recv : réception d'un message
- strcat : ajout du texte " traitée !" à la fin du message reçu (**msg**)
- send : émission d'une réponse (message reçu + texte " traitée !")
- close : fermeture du descripteur de la socket de communication

Communication en Mode Connecté Dans le Domaine AF_UNIX

❑ Programme **cli_unix_conn.c** (client)

```
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
```

```
main() {
    int ret; int sd; char msg[128]; struct sockaddr_un locale, cible;
    cible.sun_family = AF_UNIX; strcpy(cible.sun_path, "serv_sock");
    if ((sd = socket(AF_UNIX, SOCK_STREAM, 0)) != -1) printf("socket = %d\n", sd);
    else { ("Erreur socket"); exit(-1); }
    if (connect(sd, (const struct sockaddr *) &cible, sizeof(cible)) == -1) {
        perror("Erreur connect"); exit(-1); }
for (;;) {
    printf("Entrer message : "); scanf("%s", msg);
    if ((ret = send(sd, msg, strlen(msg), 0)) != -1)
        fprintf(stdout, "\tsend = %d; msg = %s\n", ret, msg);
    else perror("Erreur send");
    if ((ret = recv(sd, msg, sizeof(msg), 0)) != -1) {
        msg[ret] = '\0'; fprintf(stdout, "\t\trecv = %d; msg = %s\n", ret, msg); }
    else error("Erreur recv");
}}
```

initialise la structure **cible** (distante)

crée la socket locale

demande la connexion de sa socket locale à la socket cible (distante)

invite l'utilisateur à introduire un message (texte) qui sera stocké dans **msg**

émission du message

réception d'un message (réponse)

❑ Exécution de **serv_unix_conn.c**

```
[student]$ ./serv_unix_conn
socket = 3    bind = 0

    recv = 4; msg = cmd1
        send = 15; msg = cmd1 traitée !
Connexion coupée !

    recv = 4; msg = cmd2
        send = 15; msg = cmd2 traitée !
Connexion coupée !
```

❑ Exécution de **cli_unix_conn.c**

```
[student]$ ./cli_unix_conn
socket = 3
Entrer message : cmd1
    send = 4; msg = cmd1
        recv = 15; msg = cmd1 traitée !
Entrer message : ^C
[student]$ ./cli_unix_conn
socket = 3
Entrer message : cmd2
    send = 4; msg = cmd2
        recv = 15; msg = cmd2 traitée !
Entrer message : ^C
[student]$
```

Communication en Mode Connecté Dans le Domaine AF_INET

```c
#include <sys/types.h>  #include <errno.h>
#include <sys/socket.h> #include <netdb.h>
#include <stdio.h>      #include <netinet/in.h>
#include <stdlib.h>     #include <string.h>
```

❑ Programme **serv_inet_conn.c** (serveur)

```c
void main() {
    int ret; int sd, newsd; char msg[128]; char name[128];
    struct sockaddr_in locale, source; struct hostent *hp; int fromlen = sizeof(source);

    if (gethostname(name, sizeof(name)) != 0) { perror("Erreur  gethostname"); exit(-1); }
    if ((hp = gethostbyname(name)) == NULL) { perror("Erreur  gethostbyname"); exit(-1); }

bzero((char *)&locale,sizeof(locale)); bcopy(hp->h_addr,(char *)&locale.sin_addr,hp->h_length);
locale.sin_family = hp->h_addrtype; locale.sin_port = ntohs(2000);      // initialise la structure locale

    if ((sd = socket(AF_INET, SOCK_STREAM, 0)) == -1) { perror("Erreur  socket"); exit(-1); }
    if (bind(sd, (const struct sockaddr *) &locale, sizeof(locale)) == -1) {
        perror("Erreur  bind"); exit(-1); }                  // crée et nomme la socket locale

if (listen(sd, 5) != 0) { perror("Erreur  listen"); exit(-1); }  // met la socket locale en état d'écoute

for (;;) {      ← boucle infinie sur les connexions
    if ((newsd = accept(sd, (struct sockaddr *) &source, &fromlen)) == -1) {
        perror("Erreur accept"); exit(-1); }    // acceptation de connexion, sinon attente de demande

    for (;;) {      ← boucle infinie sur les réceptions/émissions (d'une connexion)
        if ((ret = recv(newsd, msg, sizeof(msg), 0)) != 0) {
            msg[ret] = '\0'; fprintf(stdout, "\trecv = %d; msg = %s\n", ret, msg); }
        else { printf("Connexion coupée !\n"); break; }        réception d'un message
        strcat(msg, " traitée !");   ← ajout du texte " traitée !" à la fin du message reçu (msg)
        if ((ret = send(newsd, msg, strlen(msg), MSG_NOSIGNAL)) != -1)
            fprintf(stdout, "\t\tsend = %d; msg = %s\n", ret, msg);
        else perror("Erreur send");   émission d'une réponse (message reçu + texte " traitée !")
    }
close(newsd);   ← fermeture du descripteur de la socket de communication
}
}
```

Communication en Mode Connecté Dans le Domaine AF_INET

```
#include <sys/types.h>  #include <errno.h>
#include <sys/socket.h> #include <netdb.h>
#include <stdio.h>      #include <netinet/in.h>
#include <stdlib.h>     #include <string.h>
```

❑ Programme **cli_inet_conn.c** (client)

```
void main(int argc, char **argv) {
    int ret; int sd; char msg[128]; struct sockaddr_in locale, cible; struct hostent *hp;
    if ((hp = gethostbyname(argv[1])) == NULL) { perror("Erreur  gethostbyname"); exit(-1); }
bzero((char *) &cible, sizeof(cible)); bcopy(hp->h_addr, (char *) &cible.sin_addr, hp->h_length);
cible.sin_family = hp->h_addrtype; cible.sin_port = htons(2000);   // initialise la structure cible (distante)
    if ((sd = socket(AF_INET, SOCK_STREAM, 0)) == -1) { perror("Erreur socket"); exit(-1); }
    if (connect(sd, (const struct sockaddr *) &cible, sizeof(cible)) == -1) {
        perror("Erreur connect"); exit(-1); }
for (;;) {
    printf("Entrer message : "); scanf("%s", msg);

    if ((ret = send(sd, msg, strlen(msg), 0)) != -1)
        fprintf(stdout, "\tsend = %d; msg = %s\n", ret, msg);
    else perror("Erreur send");
                                                          émission du message
    if ((ret = recv(sd, msg, sizeof(msg), 0)) != -1) {
        msg[ret] = '\0'; fprintf(stdout, "\t\trecv = %d; msg = %s\n", ret, msg); }
    else error("Erreur recv");
                                                          réception d'un message (réponse)
}}
```

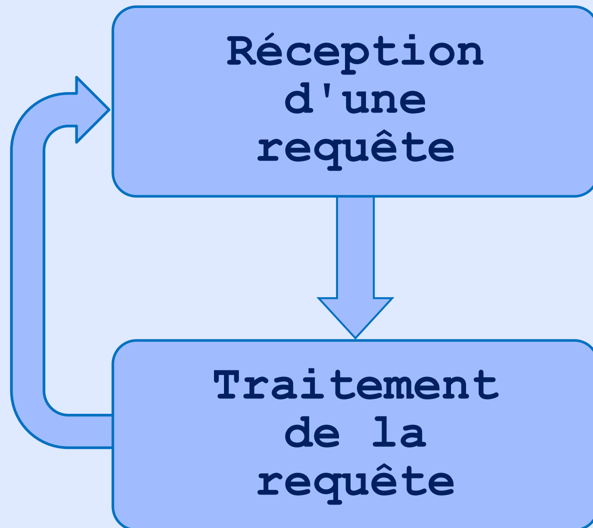
❑ Exécution de **serv_inet_conn.c**

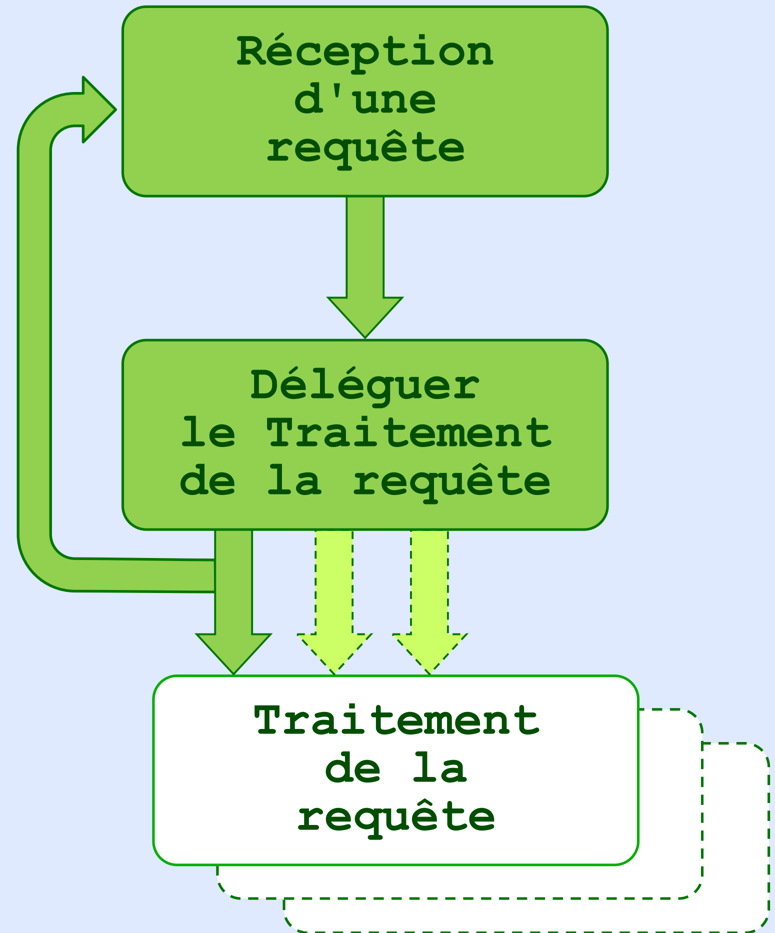
```
[student]$ ./serv_inet_conn

    recv = 4; msg = cmd1
        send = 15; msg = cmd1 traitée !
Connexion coupée !

    recv = 4; msg = cmd2
        send = 15; msg = cmd2 traitée !
Connexion coupée !
```

❑ Exécution de **cli_inet_conn.c**

```
[student]$ ./cli_inet_conn machine_serveur
Entrer message : cmd1
    send = 4; msg = cmd1
        recv = 15; msg = cmd1 traitée !
Entrer message : ^C
[student]$ ./cli_inet_conn machine_serveur
Entrer message : cmd2
    send = 4; msg = cmd2
        recv = 15; msg = cmd2 traitée !
Entrer message : ^C
[student]$
```

Serveur : Séquentiel vs Concurrent





Exemple de Serveur Séquentiel

❑ Programme **serv_seq.c** (serveur séquentiel)

```
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <stdio.h>
#include <errno.h>
#include <signal.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
```

```
void interrupt(int signo) {
     unlink("serv_sock"); exit(0);
}

void main() {
     int ret; int sd; char msg[128]; char pid[16]; char req[16];

     struct sockaddr_un locale, source, cible; int fromlen = sizeof(source);

     signal(SIGINT, interrupt);

     locale.sun_family = AF_UNIX; strcpy(locale.sun_path, "serv_sock");

     if ((sd = socket(AF_UNIX, SOCK_DGRAM, 0)) != -1) printf("socket = %d\t", sd);
     else ("Erreur socket");

     if ((ret = bind(sd, (const struct sockaddr *) &locale, sizeof(locale))) != -1)
         printf("bind = %d\n", ret);
     else perror("Erreur bind");

for (;;) {
     if ((ret = recvfrom(sd, msg, sizeof(msg), 0, (struct sockaddr *) &source, &fromlen)) == -1)
         perror("Erreur recvfrom");

     sscanf(msg, "%s%s", pid, req);

     printf("Début traitement requête %s de %s\n", req, pid);
     sleep(5);
     printf("\tFin traitement requête %s de %s\n", req, pid);
}
}
```

← lit depuis **msg** le **pid** du processus émetteur et le numéro de sa requête

Exemple de Serveur Séquentiel

❑ Programme `client.c` (serveur séquentiel)

```
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <stdio.h>
#include <errno.h>
#include <string.h>
#include <unistd.h>
```

```
void main() {
    int ret; int i; int sd; char msg[128];

    struct sockaddr_un source, cible; int fromlen = sizeof(source);

    cible.sun_family = AF_UNIX; strcpy(cible.sun_path, "serv_sock");

    if ((sd = socket(AF_UNIX, SOCK_DGRAM, 0)) != -1) printf("socket = %d\n", sd);
    else perror("Erreur socket");

for (i = 0; i < 2; i++) {
    sprintf(msg, "%d\t%d", getpid(), i);

    if ((ret=sendto(sd,msg,strlen(msg),0,(const struct sockaddr *)&cible,sizeof(cible))) == -1)
        perror("Erreur sendto");
}
}
```

← écrit dans **`msg`** le **`pid`** du processus et le numéro de sa requête

❑ Exécution de `client.c`

```
[student]$ gcc client.c -o client
[student]$ ./client&
[2] 12190
socket = 3
[student]$ ./client&
[3] 12191
[2]   Exit 7                  ./client
socket = 3
[student]$
[3]+  Exit 7                  ./client
[student]$
```

❑ Exécution de `serv_seq.c`

```
[student]$ gcc serv_seq.c -o serv_seq
[student]$ ./serv_seq
socket = 3   bind = 0
Début traitement requête 0 de 12190
    Fin traitement requête 0 de 12190
Début traitement requête 1 de 12190
    Fin traitement requête 1 de 12190
Début traitement requête 0 de 12191
    Fin traitement requête 0 de 12191
Début traitement requête 1 de 12191
    Fin traitement requête 1 de 12191
```

exécution séquentielle

Exemple de Serveur Concurrent

```
#include <sys/types.h>  #include <signal.h>
#include <sys/socket.h> #include <string.h>
#include <sys/un.h>     #include <stdlib.h>
#include <stdio.h>      #include <unistd.h>
#include <errno.h>
```

❑ Programme **serv_conc.c** (serveur concurrent)

```
void interrupt(int signo) { unlink("serv_sock"); exit(0);}

void main() {
    int ret; int sd; char msg[128]; char pid[16]; char req[16];
    struct sockaddr_un locale, source, cible; int fromlen = sizeof(source);
    signal(SIGINT, interrupt);
    locale.sun_family = AF_UNIX; strcpy(locale.sun_path, "serv_sock");
if ((sd = socket(AF_UNIX,SOCK_DGRAM, 0)) == -1) perror ("Erreur socket");
if ((ret=bind(sd,(const struct sockaddr *)&locale,sizeof(locale))) == -1) perror("Erreur bind");

for (;;) {
    if ((ret = recvfrom(sd, msg, sizeof(msg), 0, (struct sockaddr *) &source, &fromlen)) == -1)
        perror("Erreur recvfrom");
    else {   if (fork() == 0) {
                sscanf(msg, "%s%s", pid, req);
                printf("Début traitement requête %s de %s par %d\n", req, pid, getpid());
                sleep(5); ← simule le temps de traitement de la requête
                printf("\tFin traitement requête %s de %s par %d\n", req, pid, getpid());
            }
}}}
```

❑ Exécution de **client.c**

```
[student]$ ./client&
[2] 12347
socket = 3
[student]$ ./client&
[3] 12350
[2]   Exit 7                 ./client
socket = 3
[student]$
[3]+  Exit 7                 ./client
[student]$
[student]$
```

❑ Exécution de **serv_conc.c**

```
[student]$ gcc serv_conc.c -o serv_conc
[student]$ ./serv_conc
Début traitement requête 0 de 12347 par 12348
Début traitement requête 1 de 12347 par 12349
Début traitement requête 0 de 12350 par 12351
Début traitement requête 1 de 12350 par 12352
    Fin traitement requête 0 de 12347 par 12348
    Fin traitement requête 1 de 12347 par 12349
    Fin traitement requête 0 de 12350 par 12351
    Fin traitement requête 1 de 12350 par 12352
```

exécution concurrentielle

Fin de Communication

- Un processus peut à tout moment terminer toute ou une partie d'une connexion

```
#include<sys/socket.h>
      int shutdown(int sockfd, int how);
```

Descripteur de la socket locale

how permet de définir la fin de connexion selon les valeurs suivantes :

SHUT_RD (0) : l'utilisateur ne veut plus lire de données (réception désactivée),
SHUT_WR (1) : l'utilisateur ne veut plus écrire de données (émission désactivée),
SHUT_RDWR (2) : l'utilisateur ne veut plus lire ni écrire de données (émission et réception désactivées).

  - Valeur de retour : 0 en cas de succès et -1 en cas d'erreur (et **errno** est modifiée en conséquence).

- Un processus peut à tout moment fermer le descripteur d'une socket

```
#include<sys/socket.h>
      int close(int fildes);
```

  - Libère le descripteur de fichier **fildes** de la **u_ofile** du processus.
  - Dans le domaine **AF_UNIX**, il faut supprimer le fichier créé.
  - Valeur de retour : 0 en cas de succès et -1 en cas d'erreur (et **errno** est modifiée en conséquence).

Multiplexage des Entrées/Sorties

- Un processus peut surveiller plusieurs descripteurs de fichier, en attendant qu'au moins l'un de ces descripteurs soit "prêt" pour une opération d'E/S.

```
#include<sys/time.h>
#include<sys/types.h>
#include<unistd.h>
int select(int nfds, fd_set *readfds, fd_set *writefds,
           fd_set *exceptfds, struct timeval *timeout);
```

- `nfds` : numéro du plus grand descripteur des 3 ensembles +1
- `readfds` : ensemble des descripteurs à surveiller pour vérifier si une lecture est possible
- `writefds` : ensemble des descripteurs à surveiller pour vérifier si une écriture est possible
- `exceptfds` : ensemble des descripteurs à surveiller pour l'occurrence de conditions exceptionnelles
- `timeout` : durée limite du temps passé dans **select()** avant son retour (si **NULL**, bloqué indéfiniment)

  - En sortie, les 3 ensembles sont modifiés pour indiquer les descripteurs qui ont changé de statut.
  - Valeur de retour : le nombre de descripteurs dans les 3 ensembles retournés, -1 en cas d'erreur

- Manipulation des ensembles de descripteurs

```
FD_ZERO(fd_set *set);              // efface l'ensemble set ( set = ∅ )
FD_CLR(int fd, fd_set *set);       // supprime le descripteur fd de l'ensemble set
FD_SET(int fd, fd_set *set);       // ajoute le descripteur fd à l'ensemble set
FD_ISSET(int fd, fd_set *set);     // vérifie si le descripteur fd est présent dans
                                   //   l'ensemble set
```

Exemple de Multiplexage des E/S

❑ Programme **serv_select.c**

```
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <stdio.h>
#include <errno.h>
#include <signal.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/time.h>
```

```
struct sockaddr_un locale[3], source;

void interrupt(int signo) {
    int i;
    for (i = 0; i < 3; i++) unlink(locale[i].sun_path);
    exit(0);
}

void main(int argc, char **argv) {
    int ret, i, j; int sd[3]; char msg[128]; fd_set readers; int fromlen = sizeof(source);
    signal(SIGINT, interrupt);
    for (i = 0; i < 3; i++) {
        locale[i].sun_family = AF_UNIX;
        sprintf(locale[i].sun_path, "%s%d", "serv_sock_", i);
        sd[i] = socket(AF_UNIX, SOCK_DGRAM, 0);
        bind(sd[i], (const struct sockaddr *) &locale[i], sizeof(locale[i]));
    }

for (;;) {
    FD_ZERO(&readers); for (i = 0; i < 3; i++) FD_SET(sd[i], &readers);
    printf("Avant select()\n");
    ret = select(128, &readers, NULL, NULL, NULL);
    printf("Après select()\n");
    for (i = 0; i < 3; i++) {
        if (FD_ISSET(sd[i], &readers)) {
            msg[recvfrom(sd[i],msg,sizeof(msg),0,(struct sockaddr *)&source,&fromlen)] = '\0';
            printf("\tRéception sur sd[%d] : %s\n", i, msg);
            FD_CLR(sd[i], &readers);
        }
    }
    sleep(10);
}
}
```

Annotations :

- `FD_ZERO(&readers); for (...) FD_SET(...)` ⟵ initialise l'ensemble **`readers`** aux 3 descripteurs de sockets locales
- `ret = select(...)` ⟵ attente de changement de statut d'au moins un des descripteurs contenus dans **`readers`**
- `if (FD_ISSET(sd[i], &readers))` ⟵ teste si le descripteur **`sd[i]`** est contenu dans **`readers`**
- `FD_CLR(sd[i], &readers);` ⟵ supprime le descripteur **`sd[i]`** de **`readers`**
- `sleep(10);` ⟵ pour permettre l'arrivée de plusieurs messages, sur différents descripteurs, avant le **`select()`**

Exemple de Multiplexage des E/S

❑ Programme `cli_select.c`

```c
#include <sys/types.h>   #include <errno.h>
#include <sys/socket.h>  #include <string.h>
#include <sys/un.h>      #include <unistd.h>
#include <stdio.h>
```

```c
void main(int argc, char **argv) {
     int ret; int i; int sd; char msg[128];
     struct sockaddr_un cible;
     cible.sun_family = AF_UNIX; strcpy(cible.sun_path, argv[1]);
     if ((sd = socket(AF_UNIX, SOCK_DGRAM, 0)) == -1) perror("Erreur socket");
     sprintf(msg, "%d", getpid());
     sendto(sd, msg, strlen(msg), 0, (const struct sockaddr *) &cible, sizeof(cible));
}
```

❑ Exécution de `serv_select.c`

```
[student]$ gcc serv_select.c -o
serv_select
[student]$ ./serv_select
Avant select()
Après select()
    Réception sur sd[0] : 15519
Avant select()
Après select()
    Réception sur sd[1] : 15520
Avant select()
Après select()
    Réception sur sd[2] : 15521
Avant select()
Après select()
    Réception sur sd[1] : 15522
    Réception sur sd[2] : 15523
Avant select()
Après select()
    Réception sur sd[2] : 15524
Avant select()
Après select()
    Réception sur sd[0] : 15525
Avant select()
```

❑ Exécution de `cli_select.c`

```
[student]$ gcc cli_select.c -o cli_select
[student]$
[student]$ ./cli_select serv_sock_0&
[1] 15519
[student]$
[student]$
[student]$ ./cli_select serv_sock_1&
[2] 15520
[1]    Exit 5                    ./cli_select serv_sock_0
[student]$
[student]$ ./cli_select serv_sock_2&
[3] 15521
[2]    Exit 5                    ./cli_select serv_sock_1
[student]$
[student]$ ./cli_select serv_sock_1 | ./cli_select serv_sock_2
[3]+   Exit 5                    ./cli_select serv_sock_2
[student]$
[student]$
[student]$ ./cli_select serv_sock_2 | ./cli_select serv_sock_0
[student]$
```

l'arrivée du 1er message a débloqué le **`select()`** avant l'arrivée du 2ème message