Master Informatique **M1** - Spécialité SAR
UE POSIX

SOCKETS

I – Introduction aux sockets
II – Mode non connecté
III – Mode connecté
IV – Concepts avancés

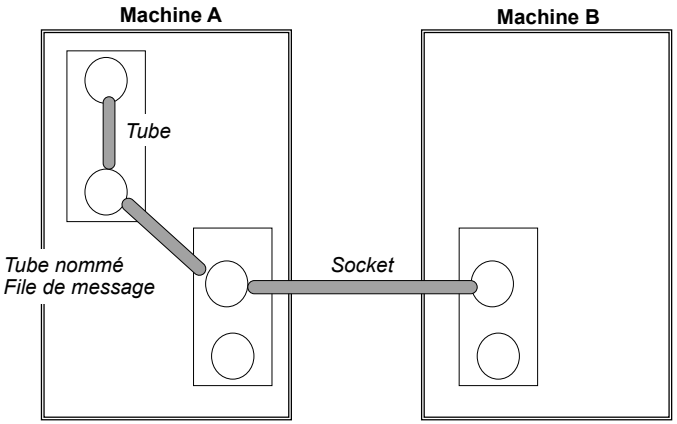
Olivier Marin

olivier.marin@lip6.fr

Communication sous UNIX

- Communication intra-tâche (entre threads)
  - Partage de variables globales, tas
- Communication inter-tâches : locale (même machine)
  - Disque (Tube, Fichier)
  - Mémoire (Segment partagé, Sémaphore, File de messages)
- Communication inter-tâches : distante
  - Socket (TCP / UDP / IP)
  - Appel de procedure à distance (RPC : Remote Procedure Call)
  - Appel de méthode à distance (Corba, Java-RMI)
- Outils de haut niveau
  - Systèmes de fichiers répartis (NFS, RFS, AFS, Netware)
  - Bases de données réparties (NIS)

Communication sous UNIX



Communications Distantes : Qualité de Service

Type de service
- Connecté (fonctionnement similaire à un tube),
- Non connecté (possibilité d'inversion de paquets)

Fiabilité
- Pas de perte de données,
- Utilisation d'un acquittement,
- Augmentation de la charge du réseau et des délais de transmission

Exemples
- Connexion fiable : Session ssh d'ordinateur à ordinateur
- Connexion non fiable : Voix numérisée
- Sans connexion / non fiable : Diffusion électronique de prospectus

Communications Distantes : Modèle de référence ISO

- Physique, Liaison **(ARPANET, SATNET, LAN)**
- Réseau
  - Internet Protocol (IP)
- Transport
  - Transmission Control Protocol (TCP), User Datagram Protocol (UDP),
  - Internet Control Message Protocol (ICMP),
  - Internet Group Management Protocol (IGMP)
- Session
  - (Remote Procedure Call) RPC
- Présentation
  - (eXternal Data Representation) XDR
- Application (telnet, ftp, smtp, dns, nfs)

Communications Distantes : Notion de Socket

Extension de la notion de tube nommé

4.2 BSD (1983)

Socket : Point extrême de communication

- Bidirectionnelle
- Associée à un nom (unique)
- Appartient à un domaine
- Possède un type

Exemples en ligne sur le site du cours

Types de Socket

SOCK_STREAM

- Mode connecté
- avec contrôle de flux
- bidirectionnel
- protocole TCP (IPPROTO_TCP)

SOCK_DGRAM

- Mode non connecté
- transmission par paquet
- sans contrôle de flux
- bidirectionnel
- protocole UDP (IPPROTO_UDP) ou IGMP (IPPROTO_IGMP)

Types de Socket

SOCK_SEQPACKET

- Mode non connecté garantissant l'intégrité
- transmission par paquet
- bidirectionnel
- protocole UDP (IPPROTO_UDP) ou IGMP (IPPROTO_IGMP)

SOCK_RAW

- Accès direct avec la couche IP,
- réservé au super-utilisateur,
- protocole ICMP (IPPROTO_ICMP),
- Définition de nouveaux protocoles (IPPROTO_RAW)

Choix du protocole

TCP (Transport Control Protocol)
=> Pas de perte, pas de déséquencement, flux
=> Coûteux (connexion, transfert)
=> Type = STREAM

UDP (User Datagram Protocol)
=> Perte, déséquencement, taille limitée
=> Performant
=> Type = DGRAM

Notion de client/serveur

Utilisation des Sockets

- Création (client / serveur)
- Nommage (serveur)
- Communication en mode connecté (TCP)
- Communication en mode non connecté (UDP)

Utilisation des Sockets : Création

Socket associée à un descripteur de fichier

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

domaine
AF_UNIX / *AF_LOCAL* / *AF_FILE* (local uniquement)
AF_INET / *AF_INET6*
AF_UNSPEC
type
SOCK_STREAM, SOCK_DGRAM, …
protocole
0 => choix automatique, (IPPROTO_TCP, IPPROTO_UDP)
Retourne le descripteur de fichier ou –1 en cas d'erreur

```
eg. int s = socket (AF_INET, SOCK_STREAM, 0);
```

Utilisation des Sockets : Nommage

Objectif : Associer un nom à une socket

ie. identifier un serveur de manière unique

Domaine unix

nom = nom de fichier

Domaine internet (inet)

nom = <numéro de port, adresse IP>

```
int bind(int sock, struct sockaddr *nom, int lg_nom);
```

sock	numéro de socket (retourné par `socket()`)
nom	nom de la socket (dépendant du domaine)
lg_nom	longueur du nom (`sizeof(struct sockaddr)`)

Nommage dans le domaine Unix (local)

Communication via le système de fichiers (sur partition locale uniquement)

```
#include <sys/un.h>
struct sockaddr_un {
  short int sun_family;     /* AF_UNIX (AF_LOCAL) */
  char  sun_path[104];      /* Chemin du fichier */
};
```


Nommage dans le domaine Unix Exemple

```
/* fichier "bindlocal.c" */
#define _XOPEN_SOURCE 700
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>

int main(int argc, const char **argv)
{
     int sock;
     struct sockaddr_un addr;
     memset(&addr,'\0', sizeof(struct sockaddr_un));
     addr.sun_family = AF_UNIX;
     strcpy(addr.sun_path, "./MySock");
     if ((sock = socket(AF_UNIX, SOCK_STREAM, 0)) == -1)
                    {perror("Erreur creation socket");exit(1);}
     if (bind(sock, (struct sockaddr *)&addr, strlen(addr.sa_data)) == -1)
                    {perror("Erreur au nommage");exit(2);}
     return (0);
}
```


Nommage dans le domaine Internet

```
#include <netinet/in.h>
struct sockaddr_in {
      short sin_family;              domaine de communication (AF_INET)
      u_short sin_port;              numéro du port
      struct in_addr sin_addr;       adresse IP
};

struct in_addr {
      u_long s_addr;
};
```

in_addr.s_addr = INADDR_ANY => machine locale

Nommage dans le domaine Internet
Numéro IP

Entier sur 32 bits (4 octets)

Classe,
identification de réseau,
identification d'ordinateur

5 classes d'adresse (IPv4)

classe A → 0 + id_res sur 7 bits + id_ord sur 24 bits
126 réseaux jusqu'à 16 millions d'ordinateurs
classe B → 10 + id_res sur 14 bits + id_ord sur 16 bits
16382 réseaux jusqu'à 65536 d'ordinateurs
classe C → 110 + id_res sur 21 bits + id_ord sur 8 bits
2 millions de réseaux jusqu'à 254 d'ordinateurs
classe D → 1110 + adresse multi-destinataires sur 28 bits
classe E → 11110 + réservé pour usage ultérieur

Délivré par les agences accréditées par l'ICANN

Nommage : formattage d'adresse

Adresses codées en format « réseau »

Fonctions de conversion

```
u_short  htons(u_short) /* host to network short */
```
=> Conversion du numéro de port
```
u_long  htonl(u_long) /* host to network long */
```
=> Conversion de l'adresse IP
```
u_short  ntohs(u_short) /* network to host short */
u_long  ntohl(u_long) /* network to host long  */
```

Adresse <=> Chaîne de caractères « a.b.c.d »

```
char  *inet_ntoa(struct in_addr adr)/ *adr à chaine (affichage)*/
u_long inet_addr(char *chaine)   /* chaine à adr */
```


Nommage : correspondance nom/adresse

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

int getaddrinfo(const char *hostname, const char *servname,
              const struct addrinfo *hints, struct addrinfo **res);
```

Recherche les informations dans /etc/hosts ou pages jaune (NIS) ou serveur de noms (DNS) - Ordre défini dans /etc/nsswitch.conf

```
struct addrinfo {
            int ai_flags;           /* input flags */
            int ai_family;          /* protocol family for socket */
            int ai_socktype;        /* socket type */
            int ai_protocol;        /* protocol for socket */
            socklen_t ai_addrlen;   /* length of socket-address */
            struct sockaddr *ai_addr; /* socket-address for socket */
            char *ai_canonname;     /* canonical name for service location */
            struct addrinfo *ai_next; /* pointer to next in list */
    };
```


Correspondance nom/adresse :
Exemple Nom => IP

name2ip.c

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

int main(int argc, const char **argv)
{
  struct addrinfo *result;

  if (getaddrinfo(argv[1], 0, 0, &result) != 0) {
      perror("getaddrinfo");
      exit(EXIT_FAILURE);
  }
  printf("address - %s\n",
      inet_ntoa(((struct sockaddr_in*)rp->ai_addr)->sin_addr));
  freeaddrinfo(result);      /* No longer needed */
  return (0);
}
```

Correspondance nom/adresse : Exemple IP => Nom

ip2name.c

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <arpa/inet.h>
#include <netdb.h>

int main(int argc, const char **argv)
{
    struct sockaddr_in sin;
    char host[64];

    memset((void*)&sin, 0, sizeof(sin));
    sin.sin_addr.s_addr = inet_addr(argv[1]);
    sin.sin_family = AF_INET;

    if (getnameinfo((struct sockaddr*)&sin, sizeof(sin),
                    host, sizeof(host), 0, 0, 0) != 0) {
        perror("getnameinfo");
        exit(EXIT_FAILURE);
    }

    printf("Name : %s\n", host);
    return (0);
}
```


Nommage dans le domaine Internet Numéro de port

Entier sur 16 bits

Fichier /etc/services

Numéros de port utilisés par les services systèmes,
3 champs (type de service, numéro de port, protocole utilisé),
FTP (port 21), SSH (port 22), TELNET (port 23), KERBEROS (port 88)

Appel système `getservbyname(char *name, char *protocole)`

Retourne un pointeur sur un struct servent,
Recherche les informations dans /etc/services

```
struct  servent {
        char *s_name;      /* official name of service */
        char **s_aliases;  /* alias list */
        int s_port;        /* port service resides at */
        char *s_proto;     /* protocol to use */
};
```


Sockets non connectées : communication

```
int recvfrom(int sock, char* buffer, int tbuf, int flag,
             struct sockaddr *addsrc, socklen_t *taille)
```

Lecture d'un buffer addressé à la socket sock,
Adresse de l'émetteur retournée dans addsrc

```
int sendto(int sock, char* buff, int tbuf, int flag,
             struct sockaddr *addrdst, socklen_t taille)
```

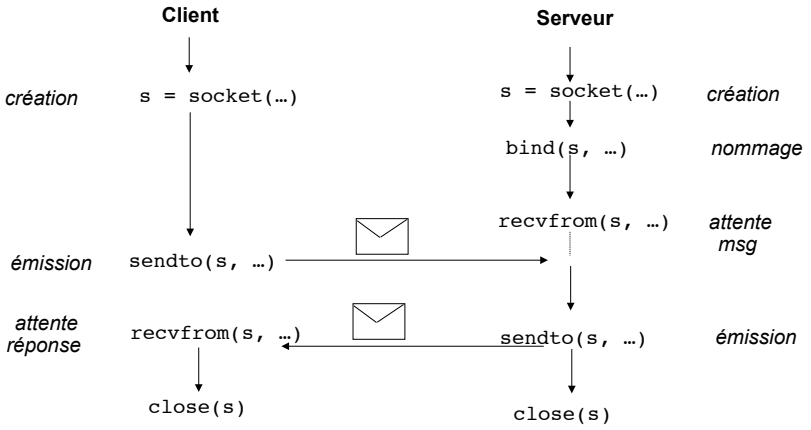
Envoi par la socket `sock` du contenu de `buff` à l'adresse `addrdst`

Taille limitée à la taille d'un paquet

Champ `flag`

`MSG_PEEK` Lecture (`recvfrom`) sans modification de la file d'attente

Sockets non connectées : un exemple



Sockets non connectées : un exemple
Partie serveur - serveurUDP.c

```
/* Includes …* /
#define PORTSERV 4567
int main(int argc, char *argv[])
{
  struct sockaddr_in sin;    /* Nom de la socket du serveur */
  struct sockaddr_in exp;    /* Nom de l'expediteur */
  char host[64];
  int sc ;
  int fromlen = sizeof(exp);
  char message[80];
  int cpt = 0;

   /* creation de la socket */
   if ((sc = socket(AF_INET,SOCK_DGRAM,0)) < 0) {
        perror("socket"); exit(1);
   }

  /* remplir le « nom » */
   memset((char *)&sin,0, sizeof(sin));
    sin.sin_addr.s_addr = htonl(INADDR_ANY);
   sin.sin_port = htons(PORTSERV);
    sin.sin_family = AF_INET;

   /* nommage */
   if (bind(sc,(struct sockaddr *)&sin,sizeof(sin)) < 0) {
        perror("bind"); exit(2);
   }
```


Sockets non connectées : un exemple
Partie serveur - serveurUDP.c (suite)

```
/*** Reception du message ***/
if ( recvfrom(sc,message,sizeof(message),0,
              (struct sockaddr *)&exp,&fromlen) == -1) {
   perror("recvfrom"); exit(2);
}
/*** Affichage de l'expediteur ***/
printf("Exp : <IP = %s,PORT = %d> \n", inet_ntoa(exp.sin_addr),
   (exp.sin_port));

/* Nom de la machine */
if (getnameinfo((struct sockaddr*)&exp, sizeof(exp),
                    host, sizeof(host), 0, 0, 0) != 0) {
   perror("getnameinfo"); exit(3);
}
printf("Machine : %s\n", host);

 /*** Traitement ***/
printf("Message  : %s \n", message);
cpt++;

/*** Envoyer la reponse ***/
if (sendto(sc,&cpt,sizeof(cpt),0,(struct sockaddr *)&exp,fromlen) == -1) {
     perror("sendto"); exit(4);
}
close(sc);
return (0);
}
```


Sockets non connectées : un exemple
Partie client - clientUDP.c

```
… /* Includes */

#define PORTSERV 4567 /* Port du serveur */

int main(int argc, char *argv[])
{
  int reponse;
  struct sockaddr_in dest;
  int sock;
  int fromlen = sizeof(dest);
  char  message[80];
  struct addrinfo *result;

  /* Le nom de la machine du serveur est passé en argument */
  if (argc != 2) {
    fprintf(stderr, "Usage : %s machine \n", argv[0]);
    exit(1);
  }

  if ((sock = socket(AF_INET,SOCK_DGRAM,0)) == -1) {
    perror("socket"); exit(1);
  }

   …
```


Sockets non connectées : un exemple
Partie client - clientUDP.c (suite)

```
  /* Remplir la structure dest */
  if (getaddrinfo(argv[1], 0, 0, &result) != 0) {
       perror("getaddrinfo"); exit(EXIT_FAILURE);}
  memset((char *)&dest,0, sizeof(dest));
  memcpy((void*)&((struct sockaddr_in*)result->ai_addr)->sin_addr,
              (void*)&dest.sin_addr, sizeof(dest));
  dest.sin_family = AF_INET;
  dest.sin_port = htons(PORTSERV);
  /* Contruire le message ...*/
   strcpy(message, "MESSAGE DU CLIENT");
  /* Envoyer le message */
  if (sendto(sock,message,strlen(message)+1,0,
              (struct sockaddr *)&dest, sizeof(dest)) == -1) {
    perror("sendto"); exit(1);
  }
  /* Recevoir la reponse */
  if (recvfrom(sock,&reponse,sizeof(reponse),0,0,&fromlen) == -1) {
    perror("recvfrom"); exit(1);
  }
  printf("Reponse : %d\n", reponse);
  close(sock);
  return(0);
}
```

Sockets connectées : introduction

Coté serveur

1. Autorisation d'un nombre maximum de connexions,
2. Attente d'une connexion,
3. Acceptation d'une connexion,
4. Entrées/sorties,
5. Fermeture de la connexion

Coté client

1. Demande de connexion
2. Entrées/sorties,
3. Fermeture de la connexion

Sockets connectées : fonctionnement

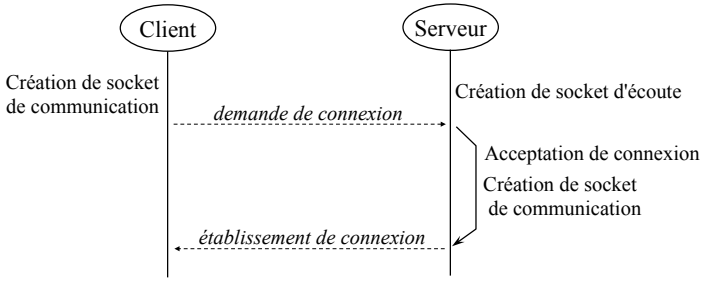
Côté serveur

1 socket pour les demandes de connexion

1 socket pour les communications

Côté client

1 socket pour communiquer



Sockets connectées
Etablissement de connexion

Côté serveur

```
int listen (int sock, int taille_file)
```

Création de la file d'attente des requêtes de connexion

Appel non bloquant

```
int accept (int sock, struct sockaddr *addrclt, socklen_t *taille)
```

Attente d'acceptation d'une connexion

Identité du client fournie dans l'adresse `addrclt`

Appel bloquant => lors de l'acceptation :

Création d'une nouvelle socket

Renvoie l'identificateur de la socket de communication

Côté client

```
int connect (int sock, struct sockaddr *addrsrv, socklen_t taille)
```

Demande d'une connexion

Appel bloquant

Sockets connectées : communication

```
int read (int sock, char *buffer, int tbuf)
int write (int sock, char *buffer, int tbuf)
```

Primitives UNIX usuelles

```
int recv (int sock, char *buffer, int tbuf, int flag)
int send (int sock, char *buffer, int tbuf, int flag)
int recvmsg (int sock, struct msghdr *msg, int flag)
int sendmsg (int sock, struct msghdr *msg, int flag)
```

Regroupement de plusieurs écritures ou lectures

Appels bloquants par défaut

`flag` :

`MSG_OOB`	données hors bande (en urgence/données de contrôle)
`MSG_PEEK`	lecture (`recv`) sans modification de la file d'attente
`MSG_WAITALL`	lecture (`recv`) reste bloquante jusqu'à réception d'au moins `tbuf` octets

Sockets connectées : déconnexion

```
int shutdown(int s, int how);
```

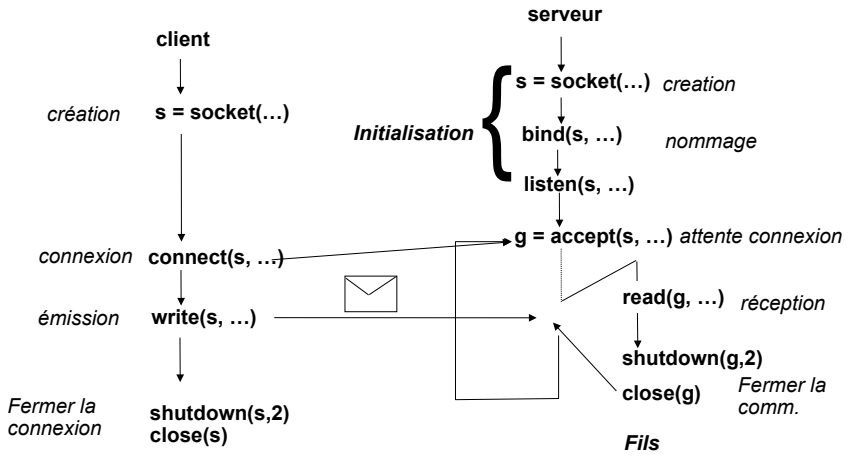
`s` descripteur de la socket

`how` mode de déconnexion

- 0 => réception désactivée
- 1 => émission désactivée
- 2 => émission et réception désactivées

`shutdown` est censé être suivi d'un `close`

Exemple de client / serveur multi-processus en mode connecté



Exemple connecté : Partie serveur

```
#define PORTSERV 7100
int main(int argc, char *argv[])
{
  struct sockaddr_in sin;  /* Nom de la socket de connexion */
  int sc ;                 /* Socket de connexion */
  int scom;                /* Socket de communication */
  struct hostent *hp;
  int fromlen = sizeof exp;
  int cpt;


  /* creation de la socket */
  if ((sc = socket(AF_INET,SOCK_STREAM,0)) < 0) {
      perror("socket"); exit(1);
  }

  memset((char *)&sin,0, sizeof(sin));
  sin.sin_addr.s_addr = htonl(INADDR_ANY);
  sin.sin_port = htons(PORTSERV);
  sin.sin_family = AF_INET;

  /* nommage */
  if (bind(sc,(struct sockaddr *)&sin,sizeof(sin)) < 0) {
      perror("bind");
      exit(1);
  }
  listen(sc, 5);
```

serveurTCP.c

Exemple connecté : Partie serveur

```
/* Boucle principale */
  for (;;) {
    if ((scom = accept(sc, (struct sockaddr *)&exp, &fromlen))== -1) {
       perror("accept"); exit(3);
    }
    /* Création d'un processus fils qui traite la requete */
    if (fork() == 0) {
        /* Processus fils */
        if (read(scom,&cpt, sizeof(cpt)) < 0) {
            perror("read"); exit(4);
        }
       /*** Traitement du message ***/
        cpt++;
        if (write(scom, &cpt, sizeof(cpt)) == -1) {
           perror("write"); exit(2);
         }
        /* Fermer la communication */
         shutdown(scom,2);
         close(scom);
        exit(0);
    }
  } /* Fin de la boucle for */
  close(sc);
  return 0;
}
```

serveurTCP.c

Exemple connecté : Partie client

```
…
#define PORTSERV 7100
#define h_addr h_addr_list[0]       /* definition du champs h_addr */
int main(int argc, char *argv[])
{
  struct sockaddr_in dest; /* Nom du serveur */
  int sock;
  int fromlen = sizeof(dest);
  int msg;
  int reponse;

   if (argc != 2) {
     fprintf(stderr, "Usage : %s machine\n", argv[0]);
     exit(1);
   }

  if ((sock = socket(AF_INET,SOCK_STREAM,0)) == -1) {
    perror("socket");
    exit(1);
  }

  /* Remplir la structure dest */

              cf. Client UDP...
```

clientTCP.c

Exemple connecté : Partie client (suite)

```
  /* Etablir la connexion */
  if (connect(sock, (struct sockaddr *) &dest, sizeof(dest)) == -1)
   {
    perror("connect"); exit(1);
  }
  msg = 10;
  /* Envoyer le message (un entier ici )*/
  if (write(sock,&msg,sizeof(msg)) == -1) {
    perror("write"); exit(1);
  }

  /* Recevoir la reponse */
  if (read(sock,&reponse,sizeof(reponse)) == -1) {
    perror("recvfrom"); exit(1);
  }
  printf("Reponse : %d\n", reponse);

  /* Fermer la connexion */
  shutdown(sock,2);
  close(sock);
  return(0);
}
```

clientTCP.c

Communication connectée avancée
Attente simultanée

```
#include <sys/select.h>
int select (int max1, fd_set *lecteurs, fd_set *ecrivains,
            fd_set *exceptions, struct timeval *delai_max)
```

Attente simultanée sur trois ensembles de descripteurs
`max1` = numéro du plus grand descripteur + 1
Bloquant pendant `delai_max`
(`NULL` => bloque indéfiniment ; `delai_max` à `0` => non-bloquant)
Retourne le nb de descripteurs correspondant à une E/S (+ ens. modifiés)

`FD_ZERO (fd_set* ensemble)`
Mise à zéro de l'ensemble

`FD_SET (int fd, fd_set* ensemble)`
Ajoute un descripteur à l'ensemble

`FD_CLR (int fd, fd_set* ensemble)`
Supprime un descripteur de l'ensemble

`FD_ISSET (int fd, fd_set* ensemble)`
Teste si un descripteur est dans l'ensemble

Exemple – attente simultanée sur stdin et socket

```
…
int main(int argc, char *argv[]){
  ….   /* initialisation  idem serveurTCP.c */
  printf("Appuyer sur une <Entree> pour tuer le serveur\n");
  /* Boucle principale */
  for (;;) {
   fd_set mselect;
    /* Construire le masque du select */
     FD_ZERO(&mselect);
     FD_SET(0,&mselect);  /* stdin */
     FD_SET(sc,&mselect); /* la socket */

     if (select(sc+1, &mselect, NULL, NULL, NULL) == -1) {
       perror("select");
       exit(3);
     }
```

serveurTCP2.c

Exemple – attente simultanée sur stdin et socket

```
 /*** Un evenement a eu lieu : tester le descripteur ***/
    if (FD_ISSET(0,&mselect)) {
      /* Sur stdin */
      break; /* Sortie de la boucle */
    }
    if (FD_ISSET(sc, &mselect)) {
      /* Sur la socket de connexion */
      /* Etablir la connexion */
      if ( (scom = accept(sc, (struct sockaddr *)&exp, &fromlen))== -1) {
       perror("accept"); exit(2);
      }
       /*** Lire le message ***/
      if (read(scom,message, sizeof(message)) < 0) {
       perror("read"); exit(1);
      }
       ….
       /* Fermer la connexion */
       shutdown(scom,2);
       close(scom);
    }

  } /* Fin de la boucle */

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

serveurTCP2.c

Exemple – attente de connexion avec temporisateur

```
…
struct timeval timeout;

timeout.tv_sec = 5; /* 5 secondes */
timeout.tv_usec = 0; /* 0 micro-seconde (10E-6 sec.) */

FD_ZERO(&mselect);
FD_SET(sc,&mselect); /* la socket */

 if (select(sc+1, &mselect, NULL, NULL, &timeout) == -1) {
    perror("select");
    exit(3);
 }
```


Options d'une socket : Lecture et Ecriture

```
int getsockopt (int sock, int couche, int cmd,
                  void *val, socklen_t *taille)
```

Lecture des options
Couche de protocole (`SOL_SOCKET`, `IPPROTO_IP`, `IPPROTO_TCP`),
`cmd` utilise le champ de données `val`

`SO_TYPE`	Type de socket
`SO_RCVBUF`	Taille du buffer de réception
`SO_SNDBUF`	Taille du buffer d'émission
`SO_ERROR`	Valeur d'erreur de la socket (non connectée)

```
int setsockopt (int sock, int couche, int cmd,
                  void *val, socklen_t taille)
```

Modification des options
`cmd` utilise le champ de données `val`

`SO_BROADCAST`	Autorisation de trames broadcast
`IP_ADD_MEMBERSHIP`	Autorisation d'une requête multicast
`SO_REUSEADDR`	Autorisation de réutiliser une @ déjà affectée

Options d'une socket : Mode de diffusion

Diffusion broadcast

Envoi d'un message UDP vers tous les ordinateurs d'un sous-réseau
Mettre une adresse IP de diffusion (bits d'adresse ordinateur à 1),
Autorisation pour diffuser en mode broadcast

Diffusion multicast

Envoi d'un message UDP vers un ensemble d'ordinateurs
Abonnement à un groupe multicast,
Choisir une adresse IP de diffusion multicast
260 millions d'adresses disponibles (224.0.0.0 à 239.255.255.255)
Envoi d'une requête multicast

Exemple – serveur multi-cast (servmcast.c)

```
#define MON_ADR_DIFF "225.0.0.10" /* Adresse multi-cast */
#define PORTSERV 7200 /* Port du serveur */

int main(int argc, char *argv[])
{
  int sock;
  struct sockaddr_in sin;
  struct ip_mreq imr; /* Structure pour setsockopt */
  char message[80];

  if((sock = socket(AF_INET, SOCK_DGRAM, 0)) < 0){
    perror("socket");
    exit(1);
  }

  imr.imr_multiaddr.s_addr = inet_addr(MON_ADR_DIFF);
  imr.imr_interface.s_addr = INADDR_ANY;

  if(setsockopt(sock, IPPROTO_IP, IP_ADD_MEMBERSHIP, (char *)&imr,
                sizeof(struct ip_mreq)) == -1){
      perror("setsockopt");
      exit(2);
  }
```


Exemple – serveur multi-cast (servmcast.c)

```
  /* remplir le « nom » */
    memset((char *)&sin,0, sizeof(sin));
    sin.sin_addr.s_addr = htonl(INADDR_ANY);
    sin.sin_port = htons(PORTSERV);
    sin.sin_family = AF_INET;

    /* nommage */
    if (bind(sock,(struct sockaddr *)&sin,sizeof(sin)) < 0) {
        perror("bind");
        exit(3);
    }
    /* Reception du message */
    if (recvfrom(sock,message,sizeof(message),0,NULL,NULL) == -1) {
      perror("recvfrom");
      exit(4);
    }

    printf("Message recu :%s\n", message);
    close(sock);
    return (0);

}
```


Exemple – Client multi-cast (clientmcast.c)

```
#define MON_ADR_DIFF "225.0.0.10"
#define PORTSERV 7200

int main(int argc, char *argv[])
{
  struct sockaddr_in dest;
  int sock;
  char  message[80];

  if ((sock = socket(AF_INET,SOCK_DGRAM,0)) == -1) {
    perror("socket"); exit(1);
  }

  /* Remplir la structure dest */
  memset((char *)&dest, 0, sizeof(dest));
  dest.sin_addr.s_addr = inet_addr(MON_ADR_DIFF);
  dest.sin_family = AF_INET;
  dest.sin_port = htons(PORTSERV);

  /* Contruire le message ...*/
  strcpy(message,"MESSAGE DU CLIENT");

  /* Envoyer le message */
  if ( sendto(sock,message,strlen(message)+1,0,
                  (struct sockaddr*)&dest, sizeof(dest)) == -1) {
    perror("sendto"); exit(1);
  }
  close(sock);
  return(0);
}
```


Visualisation des sockets

Commande netstat

```
$ netstat <option>
```

`--unix`	sockets locales
`--inet`	sockets internet
`--tcp`	sockets TCP
`--udp`	sockets UDP