Support de cours du Module M1
Communication par sockets

Conventions d'écriture :
- `E` désigne l'expéditeur d'un message
- `D` désigne le destinataire d'un message

1 La communication par datagrammes

Les sockets utilisées sont du type `SOCK_DGRAM` quel que soit leur domaine. Les caractéristiques d'une telle communication sont :
- lorsqu'un message est envoyé de E vers D, E n'a aucune information sur l'arrivée du message
- les limites du message sont préservées

1.1 Principe général

Un processus souhaitant communiquer avec un autre doit :

1. avoir un point de communication local (socket E)
2. connaître l'adresse du point de communication de son interlocuteur (adresse réseau de D)

Aucune garantie n'est donnée à E sur le fait que D a bien rattaché une socket à l'adresse de son point de communication.

Comme la communication s'effectue avec le protocole UDP, un message peut être comparé à une lettre. Il comporte :
- le contenu du message
- l'identité de l'expéditeur
- l'adresse du destinataire

1.2 L'envoi de messages

Il est réalisé par la primitive

```
ssize_t
sendto( int sock,                          /* Descripteur socket E */
        const void *msg,                   /* message a envoyer */
        size_t len,                        /* longueur du message */
        int option,                        /* =0 pour le type SOCK_DGRAM */
        const struct sockaddr *p_dest,     /* @ socket D */
        socklen_t lgdest );                /* Longeur de l'@ de la socket I
```

La valeur de retour est :

- le nombre de caractères envoyés si réussite
- -1 en cas d'échec

Les erreurs détectées sont **locales** :

- descripteur de socket non valide
- adresse message non valide
- message trop long pour le protocole UDP ($< 2K$)...

$\rightarrow$Ceci veut dire en particulier que si D n'a pas attaché son adresse à une socket, le message est perdu et E ne détecte pas d'erreur.

On a également la possibilité de faire une succession d'envois de messages avec la primitive `sendmsg`. L'intérêt d'utiliser cette primitive est que l'on réalise plusieurs envois de messages avec un seul appel système, ce qui améliore les performances.

```
ssize_t sendmsg( int sock,                      /* Descripteur socket E */
                 const struct msghdr *msg,      /* Tableau de messages */
                 int option );                  /* =0 pour SOCK_DGRAM */

struct msghdr
{
  void          *msg_name;        /* Adresse optionnelle */
  socklen_t      msg_namelen;     /* Taille de l'adresse */
  struct iovec  *msg_iov;         /* Tableau de messages */
  int            msg_iovlen;      /* Nombre d'elements dans msg_iov */
  caddr_t        msg_accrights;   /* Non utilise pour les sockets */
  int            msg_accrightslen; /* Non utilise pour les sockets */
};

typedef struct iovec
{
  void    *iov_base; /* adresse du message  */
  size_t   iov_len;  /* Longueur du message */
} iovec_t;
```

Remarque : il n'est pas nécessaire qu'une socket soit nommée (attachée à une adresse) pour pour envoyer un message avec `sendto`. Le système réalise automatiquement son attachement au cours du premier envoi.

1.3 La réception de messages

Elle est réalisée par la primitive :

```
ssize_t
recvfrom( int sock ,                    /* Descripteur socket D */
          void *msg,                    /* Buffer recuperation du message rec
          size_t  len,                  /* Taille allouee pour le buffer */
          int  option,                  /* 0 ou MSG_PEEK */
          struct sockaddr *p_exp,       /* Pour recuperer l'@ de E */
          socklen_t *p_lgexp            /* Appel : taille allouee a p_exp */
          );                              /* Retour : taille du resultat */
```

La valeur de retour est :

- le nombre de caractères reçus si réussite
- -1 en cas d'échec

L'appel de cette primitive est par défaut bloquant s'il n'y a pas de message à extraire : comme pour les verrous bloquants sur les fichiers, le processus D demandant une réception de message est mis en sommeil jusqu'à ce qu'un message arrive sur la socket de D.
Après l'appel de la fonction, le paramètre `p_exp` contient l'adresse du point de communication de E : le processus D a donc la possibilité de répondre au processus E par la primitive `sendto` (les sockets permettent une communication de type duplex).

Exemple

```
struct sockaddr_in adr_E ;
int sock_D, n , lg_adr_E, lg_reponse ;
char message[1024], reponse[1024] ;
    . . .
    . . .
lg_adr_E = sizeof(struct sockaddr_in);
n = recvfrom( sock_D, message, 1024, 0, &adr_E, &lg_adr_E);
     . . .
     . . .
sendto( sock_D, reponse, 1024, 0, &adr_E, lg_adr_E);
```

L'option `MSG_PEEK` permet de lire le message sans l'extraire de la socket.

La primitive `recvmsg` permet de réaliser une série de réceptions de messages :

```
ssize_t recvmsg( int sock,              /* descripteur socket D */
                 struct msghdr *msg,    /* liste des messages recus */
                 int options );
```

1.4 Exemple

Cet exemple met en relation un processus *serveur* (un processus ¡¡démon¿¿) et un processus *client*. Les deux processus communiquent par datagrammes dans le domaine INTERNET avec le protocole UDP. Le *serveur* offre un service attaché au n^o de port 2222. Ce service consiste a fournir les informations dont dispose la machine du serveur sur un utilisateur dont le nom est donné par un processus *client*.

1.4.1 Le serveur

Le serveur

1. crée un point de communication pour son service et attache une adresse à ce point.
2. effectue une boucle infinie dans laquelle
   - attend une requête d'un client
   - traite la requête
   - met la réponse dans le format attendu par le client
   - envoie la réponse

```
/*
 * Serveur dans le domaine INTERNET sur le port numero 2222
 * Description du Service :
 *   - Donnee : nom d'un utilisateur
 *   - Resultat : donne les informations sur cet utilisateur
 */

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/ioctl.h>
#include <netinet/in.h>
#include <pwd.h>
#include <unistd.h>
#include <netdb.h>
#include <strings.h>

#define LONGUEUR_MESSAGE 20
#define LONGUEUR_REPONSE 256
```

```
#define LONGUEUR_NOM_MACHINE 256
#define PORT 2222

int
main( int argc , char * argv[] )
{
     /* Variables Client */
     struct sockaddr_in adr_C ;
     int lg_adr_C ;
     char message_C[LONGUEUR_MESSAGE] ;

     /* Variable Serveur */
     struct sockaddr_in adr_S;
     int sock_S ;
     typedef struct rep
     {
          char type ;
          char info[LONGUEUR_REPONSE] ;
     } rep_t ;
     rep_t reponse_S ;
     char nom_machine_S[LONGUEUR_NOM_MACHINE];
     struct hostent  *infos_machine_S ;

     /* Variables de travail */
     struct passwd *getpwnam(), *p ;
     ssize_t nb_cars ;

     /*-----*/

     /* Creation point d'entree */
     if( (sock_S = socket(AF_INET, SOCK_DGRAM, 0 )) == -1 )
     {
          perror("Pb socket");
          exit(-1);
     }

    /* Initialisation de l'adresse */
     adr_S.sin_port = PORT ;
     adr_S.sin_family = AF_INET ;
     gethostname(nom_machine_S,LONGUEUR_NOM_MACHINE);
     infos_machine_S = gethostbyname(nom_machine_S);
     bcopy( infos_machine_S->h_addr , &adr_S.sin_addr , infos_machine_S->h_length ) ;
     bzero( adr_S.sin_zero , sizeof(adr_S.sin_zero) ) ;

     /* Attachement du point d'entree a l'adresse */
     if( bind(sock_S,(struct  sockaddr *)&adr_S,sizeof(adr_S)) == -1 )
     {
          perror("Pb bind");
          exit(-3);
```

```
    }

    while(1)
    {
        /* Initialisations pour un client */
        lg_adr_C = sizeof(adr_C);
        bzero(message_C,LONGUEUR_MESSAGE);
        bzero((char *)&reponse_S , sizeof(reponse_S));

        /* Attente d'un message d'un client */
        nb_cars = recvfrom( sock_S, (void *)message_C, LONGUEUR_MESSAGE, 0,
                            (struct sockaddr *)&adr_C, (socklen_t *)&lg_adr_C);
        /* Fin d'attente */

        /* Consultation du fichier /etc/passwd */
        if(( p=getpwnam(message_C)) == NULL)
        {
            reponse_S.type = '1' ;
        }
        else
        {
            reponse_S.type = '2' ;
            sprintf( reponse_S.info , "%ld_%ld_%s_%s\n",
                     p->pw_uid, p->pw_gid, p->pw_gecos, p->pw_dir ) ;
        }

        /* Envoi de la reponse */
        nb_cars = sendto( sock_S, (void *)&reponse_S, sizeof(reponse_S), 0,
                          (struct sockaddr *)&adr_C, (socklen_t)lg_adr_C);
    }
}
```

1.4.2 Le client

Le client

1. crée une socket locale
2. construit l'adresse du serveur
3. envoie sa requête
4. attend le résultat
5. exploite le résultat

Dans le code du serveur on peut voir l'attachement du point d'entrée (la socket) à une adresse. Cet attachement est dans ce cas optionnel puisque le système l'aurait effectué automatiquement à l'envoi de la première (et unique dans cet exemple) requête.

/*

```
 * Client dans le domaine INTERNET du service de numero de port 2222
 * service = donne les infos realtives a un utilisateur sur une machine particuliere
 * Parametres : <nom de la machine> <nom de l'utilisateur>
 */

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/ioctl.h>
#include <netinet/in.h>
#include <pwd.h>
#include <unistd.h>
#include <netdb.h>
#include <strings.h>

#define LONGUEUR_REPONSE 256
#define LONGUEUR_NOM_MACHINE 256
#define PORT_S 2222
#define PORT_C 2223

int
main( int argc , char * argv[] )
{
     /* Variables Client */
     int sock_C ;
     struct sockaddr_in adr_C ;
     char nom_machine_C[LONGUEUR_NOM_MACHINE];
     struct hostent  *infos_machine_C ;

     /* Variable Serveur */
     struct sockaddr_in adr_S;
     int lg_adr_S ;
     char nom_machine_S[LONGUEUR_NOM_MACHINE];
     struct hostent  *infos_machine_S ;
     typedef struct rep
     {
          char type ;
          char info[LONGUEUR_REPONSE] ;
     } rep_t ;
     rep_t reponse_S ;

     /* Variables de travail */
     ssize_t nb_cars ;
     char msgerr[126] ;
     char pw_prenom[LONGUEUR_REPONSE],
          pw_nom[LONGUEUR_REPONSE],
          pw_dir[LONGUEUR_REPONSE];
     int uid , gid ;
```

```
/*-----*/

if( argc != 3 )
{
     fprintf( stderr , "Usage : %s <machine> <utilisateur\n", argv[0] );
     exit(-1);
}

strcpy( nom_machine_S , argv[1] );

/* Creation point d'entree Client */
if( (sock_C = socket(AF_INET, SOCK_DGRAM, 0 )) == -1 )
{
     perror("Pb socket");
     exit(-1);
}

/* l'Attachement de la socket Client n'est ici pas necessaire */
adr_C.sin_family = AF_INET ;
adr_C.sin_port = htons(PORT_C) ;
gethostname(nom_machine_C,LONGUEUR_NOM_MACHINE);
infos_machine_C = gethostbyname(nom_machine_C);
bcopy( infos_machine_C->h_addr , &adr_C.sin_addr , infos_machine_C->h_length ) ;
bzero( adr_C.sin_zero , sizeof(adr_C.sin_zero) ) ;

if( bind(sock_C,(struct  sockaddr *)&adr_C,sizeof(adr_C)) == -1 )
{
     perror("Pb bind");
     exit(-3);
}

/* Preparation de l'adresse du Serveur */
adr_S.sin_family = AF_INET ;
adr_S.sin_port = htons(PORT_S) ;
if( (infos_machine_S = gethostbyname(nom_machine_S)) == NULL )
{
     sprintf( msgerr , "Pb sur le nom machine %s", argv[1]);
     perror(msgerr);
     exit(-2);
}
bcopy( infos_machine_S->h_addr , &adr_S.sin_addr , infos_machine_S->h_length ) ;
bzero( adr_S.sin_zero , sizeof(adr_S.sin_zero) ) ;

/* Envoi du message = nom d'un utilisateur */
nb_cars = sendto( sock_C, (void *)argv[2], strlen(argv[2]), 0,
                  (struct sockaddr *)&adr_S, (socklen_t)sizeof(adr_S));

/* Attente de la reponse = infos de structure <msg_t> */
lg_adr_S = sizeof(adr_S) ;
```

```
    nb_cars = recvfrom( sock_C, (void *)&reponse_S, sizeof(reponse_S), 0,
                        (struct sockaddr *)&adr_S, (socklen_t *)&lg_adr_S );

    /* Exploitation de la reponse */
    if( reponse_S.type == '1' )
    {
        fprintf( stderr, "%s : utilisateur inconnu sur la machine %s\n",
                 argv[2] , argv[1] );
        exit(-3);
    }

    printf( "Utilisateur %s sur machine %s\n\n", argv[2], argv[1] );
    sscanf(reponse_S.info, "%d %d %s %s %s", &uid , &gid , pw_prenom, pw_nom, pw_dir
    printf( "   uid=%d   gid=%d\n", uid , gid );
    printf( "   %s %s %s\n", pw_prenom, pw_nom, pw_dir );

    exit(0);
}
```

1.5 Les "pseudo-connexions"

Ce mécanisme permet d'alléger l'écriture des primitives d'envoi et de réception de messages. Ils s'appliquent dans le cas d'une socket du type `SOCK_DGRAM` qui ne communique qu'avec une seule autre.

Utilisée avec des sockets de type `SOCK_DGRAM`, la primitive `connect` précise que toutes les communications s'effectueront avec un point de communication précis.

```
int
connect( int socket,                    /* descripteur de la socket local
         const struct sockaddr *p_adr, /* pointeur sur adresse
                                          associee a la socket distante
         socklen_t lg_adr);             /* Longueur de l'adresse */
```

Il est possible d'utiliser les primitives d'entrée/sortie usuelles `read/write` car l'adresse de destination ou d'origine est implicite. Il existe aussi deux primitives spécifiques `recv` et `send` qui sont des interfaces "allégées" des primitives `recvfrom` et `sendto`.

```
  ssize_t send( int socket,         /* descripteur socket locale */
                const void *msg,    /* message a envoyer */
                size_t lg_msg,      /* longueur du message */
                int option);        /* = 0 */
```

et

```
  ssize_t recv( int socket,         /* descripteur socket locale */
                void *msg,          /* message a recevoir */
```

```
size_t lg_msg,      /* longueur du message */
int options);       /* = 0 ou MSG_PEEK */
```

2 La communication en mode connecté

Les sockets utilisées sont du type `SOCK_STREAM`. C'est le mode de communication des applications standard dans

- le domaine Internet : telnet, ftp
- le domaine UNIX : rlogin

Les communications sont plus fiables au prix d'un accroissement de leur volume, mais ce type de socket permet l'envoi et la réception de messages urgents.

2.1 Principe général

Pour utiliser ce mode il faut établir une connexion (ou un "circuit virtuel") entre les peux points de communication. Contrairement à la communication par datagrammes, il n'y a pas symétrie entre le code de `E` et celui de `D`. Le processus `E` demande au processus `D` s'il accepte une communication avec lui. `E` se trouve alors en position de *`client`* et `D` en position de *`serveur`*.
Les caractéristiques d'une telle communication sont :

- la fiabilité : `E` sait (rapidement) si `D` accepte de communiquer avec lui
- le flot d'information : l'information circule en continu, il n'y a plus de limite de message ; `D` n'a pas *a priori* la possibilité de récupérer la structure du message.

La communication s'effectue avec le protocole TCP.

2.2 Du côté du serveur

2.2.1 Présentation

Son rôle est passif dans l'établissement de la communication :

1. il avise le système auquel il appartient qu'il est prêt à recevoir des communications
2. il attend des connexions de clients : pour cela il dispose d'une **socket d'écoute** liée au n^o de port TCP correspondant au n^o de service que le serveur donne.

Quand une demande de communication parvient au système, une **socket de service** est créée et connectée à celle du client qui demande la connexion.
Le serveur peut alors déléguer le travail pour ce client à un processus fils (créé

par un `fork`) et reprendre sa veille sur la socket d'écoute. Ainsi, le processus serveur peut s'occuper de plusieurs clients ¡¡en parallèle¿¿.

2.2.2 Attente des connexions

Elle est réalisée par la primitive `listen` qui signale au système que le serveur est prêt à recevoir des connexions. Elle accepte comme paramètre un descripteur de socket de type `SOCK_STREAM`.
Si la socket n'a été préalablement liée à un port, le système procède à cette liaison. Dans ce cas il faudra faire appel à la primitive `getsockname` pour connaître le n^o de port attribué à cette socket.

```
int listen( int sock,   /* Descripteur de la socket d'ecoute */
            int nb  ); /* nombre max de demandes de connexions */
                       /* pendantes */
```

Le second paramètre définit la taille de la file d'attente des demandes de connexions établies du côté du client mais non encore effectuées du côté du serveur. Cette file est limitée à une certaine valeur par le système (en général 5).

```
int getsockname( int sock,              /* Descripteur socket    */
                 struct sockaddr *adr, /* Adresse de la socket  */
                 socklen_t *lg_adr);   /* Longueur de l'adresse */
```

2.2.3 Prise en compte des connexions

La primitive `accept` permet d'extraire une demande de connexion pendante de la file d'attente de `listen`. Une liaison est effectuée entre

- la socket du client
- une nouvelle socket dont le descripteur est renvoyé par la primitive `accept`

```
int accept( int sock,                  /* Socket ecoute */
            struct sockaddr *adr_C,    /* @ socket client */
            socklen_t *lg_adr_C );     /* longueur @ client */
```

La socket de service est attachée à un nouveau port. La primitive `accept` est **bloquante** : dans le cas où il n'y a aucune connexion pendante dans la file d'attente, le processus serveur est bloqué jusqu'à ce qu'il y en ait une.

2.2.4 Sous-traitance du traitement d'une connexion

Après acceptation d'une connexion, le serveur a deux possibilités :

- ou bien se charger lui-même du travail pour le client : mais cela signifie que les connexions suivantes ne seront prises en compte que lorsque ce travail aura été fait.
- ou bien déléguer ce travail à un processus fils. Le code d'un tel serveur pourrait correspondre au squelette de programme suivant :

```
extern void service();
in sock_ecoute , sock_service;
struct sockaddr_in adr;
int lg_adr ;
        .
        .
sock_ecoute = socket(AF_INET,SOCK_STREAM,0);
        .
        .
listen(sock_ecoute,5);
while(1)
{
  lg_adr = sizeof(adr);
  sock_service=accept(sock_ecoute,&adr,&lg_adr);
  if(fork() == 0)
  {
    /* Le processus de service n'utilise pas la socket d'ecoute */
    close(sock_ecoute);
    /* Realisation du service */
    service();
    exit(0);
  }
  /* Le processus pere n'utilise pas la socket de service */
  close(sock_service);
}
```

Ce schéma pose cependant un problème dans la mesure où il provoque l'accumulation de processus inutiles : chaque fois qu'un processus se termine, il devient zombie. Sans précaution, la table des processus finit par se remplir. Il faut donc éliminer les processus de service lorsqu'ils se terminent. La réalisation de cette action dépend du système d'exploitation sur lequel est exécuté le serveur. Par exemple, sous UNIX souche SYSTEM V, il suffit que le processeur ignore le signal `SIGCLD`.

```
signal(SIGCLD,SIG_IGN);
```

2.3 Du côté du client

2.3.1 Présentation

Le client est le processus qui prend l'initiative de la demande de connexion à un serveur. Cette demande est réalisée avec la primitive `connect` qui prend une autre sémantique par rapport aux pseudo-connexions : ici elle demande l'établissement d'une connexion qui sera connue aux deux extrémités. Le client est aussi averti de la réussite ou de l'échec de la tentative de connexion.

2.3.2 Création du circuit virtuel

C'est le résultat de la primitive `connect`. Ce circuit permet des échanges bidirectionnels.

```
int
connect( int socket,                        /* descripteur de la socket local
         const struct sockaddr *p_adr,      /* adresse socket distante */
         socklen_t lg_adr);                 /* Longueur adresse distante */
```

La connexion est établie si les conditions suivantes sont réalisées

a) les paramètres sont ¡¡localement¿¿ corrects

b) la socket distante est en mode `SOCK_STREAM`

c) l'adresse de la socket distante n'est pas déjà utilisée dans une autre connexion

d) la file d'attente des connexions pendantes n'est pas pleine

En cas de réussite la socket locale est connectée avec une nouvelle socket et la connexion est pendante jusqu'à ce que le serveur en prennent connaissance avec la primitive `accept`. Le client peut cependant commencer à écrire (ou lire) sur la socket.
Par défaut la primitive `connect` est bloquante. Donc si la condition **d)** n'est pas remplie, le processus client est bloqué. La demande de connexion est répétée un certain temps et si au bout de ce laps de temps la connexion n'a pu être établie, le processus est réveillé (retour de `connect` $= -1$ et `errno` $=$ `ETIMEDOUT`).

2.4 Dialogue Client/Serveur

2.4.1 Introduction

Une fois établie la connexion entre le serveur et un client au travers de deux sockets, les deux processus peuvent échanger des flots d'informations.

Contrairement aux datagrammes, les limites des messages ne sont pas sauvegardées : le résultat d'une opération de lecture peut provenir d'informations issues de plusieurs opérations d'écriture. Réciproquement, l'écriture d'une volumineuse information peut être découpée en n paquets récupérables par n opérations de lecture. La seule garantie du protocole transport TCP utilisé avec le protocole réseau IP est que les fragments d'un même message arrivent dans le bon ordre.
$\Rightarrow$ la synchronisation des émissions et réceptions n'est pas assurée par le mécanisme.

2.4.2 Émission des messages

L'écriture sur une socket est réalisable par la primitive standard :

```
ssize_t write( int socket,         /* descripteur socket locale */
               const void *msg,    /* message a envoyer */
               size_t lg_msg );    /* longueur du message */
```

Mais on n'a pas la possibilité d'utiliser tous les mécanismes offerts par les protocoles particuliers (pour TCP par exemple, la possibilité d'envoyer des messages urgents). Une primitive plus spécifique est :

```
ssize_t send( int socket,        /* socket locale       */
              const void *msg,   /* message a envoyer   */
              size_t lg_msg,     /* longueur du message */
              int option );      /* 0 ou MSG_OOB */
```

L'écriture d'une socket est bloquante : le processus est mis en sommeil si
- le tampon de réception de la socket `D` est plein **et**
- le tampon d'émission de la socket de `E` est plein

2.4.3 Réception des messages

Il est possible d'utiliser la primitive standard :

```
ssize_t read( int socket,       /* Descripteur socket locale */
              void *msg,        /* Message a recevoir        */
              size_t lg_msg ); /* Longueur du message       */
```

et la primitive `recv` qui permet l'extraction de messages urgents.

```
ssize_t recv( int socket,     /* socket locale */
              void *msg,      /* message a recevoir */
              size_t lg_msg,  /* Nombre max de caracteres a lire */
              int option );   /* 0   : standard
                               * MSG_PEEK : consultation sans extraction
                               * MSG_OOB : extraction d'info urgente
                               */
```

Ces deux primitives sont bloquantes. C'est à dire que :

- si aucun caractère n'est parvenu à la socket **et**
- si la connexion est toujours établie

alors le processus est bloqué. Dès que **au moins** un caractère arrive sur la socket, le processus est réveillé.

2.4.4 Coupure de la connexion

La primitive :

```
int shutdown( int socket, /* descripteur socket locale */
              int sens);  /* 0,1 ou 2 */
```

permet à un processus de spécifier qu'il ne souhaite :

- plus recevoir (sens = 0)
- plus émettre (sens = 1)
- ni recevoir ni émettre (sens = 2)

sur la socket locale

2.4.5 Exemple

Code du serveur

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/ioctl.h>
#include <netinet/in.h>
#include <pwd.h>
#include <unistd.h>
#include <netdb.h>
#include <strings.h>
#include <signal.h>

#define LONGUEUR_NOM_UTILISATEUR 32
#define LONGUEUR_REPONSE 256
#define LONGUEUR_INFOS 128
#define LONGUEUR_NOM_MACHINE 256
#define PORT 2222

/* Variable Globale pour handler */
int Fin_Serveur = 0 ;

/* Handler pour SIGINT */
static
void
sigint( int sig )
{
```

```
        printf( "Recu signal %d : Arret du serveur\n" , sig );
        Fin_Serveur = 1 ;
}

/* Description du service */
static
int
service( int sock_service_S ,
         struct sockaddr_in adr_C,
         int lg_adr_C )
{
        char question[LONGUEUR_NOM_UTILISATEUR] ;
        char reponse[LONGUEUR_REPONSE] ;
        char infos[LONGUEUR_INFOS];
        struct passwd *getpwnam(), *p ;
        ssize_t nb_cars = (ssize_t)0 ;

        /*-----*/

        /* Attente du message d'un client */
        nb_cars = recv( sock_service_S ,
                        (void *)question ,
                        LONGUEUR_NOM_UTILISATEUR,
                        0 );
        /* Fin d'attente */

        /* Consultation du fichier /etc/passwd */
        strcpy( reponse , "" );
        if(( p=getpwnam(question)) == NULL)
        {
                strcat( reponse , "1" ) ;
        }
        else
        {
                strcat( reponse , "2" ) ;
                sprintf( infos , " %ld %ld %s %s\n",
                        p->pw_uid, p->pw_gid, p->pw_gecos, p->pw_dir ) ;
                strcat( reponse , infos );
        }

        /* Envoi de la reponse */
        nb_cars = send( sock_service_S ,
                        (void *)&reponse ,
                        strlen(reponse),
                        0);
        return(0);
}

int
```

```
main( int argc , char * argv[] )
{
     /* Variables Client */
     struct sockaddr_in adr_C ;
     int lg_adr_C ;

     /* Variables Serveur */
     struct sockaddr_in adr_S;
     int sock_ecoute_S ;
     int sock_service_S ;
     char nom_machine_S[LONGUEUR_NOM_MACHINE];
     struct hostent  *infos_machine_S ;

     /*-----*/

     signal( SIGINT , sigint ) ;

     /* Pour eviter que les processus fils ne deviennent zombies */
     signal( SIGCLD , SIG_IGN );

     /* Creation socket d'ecoute */
     if( (sock_ecoute_S = socket(AF_INET, SOCK_STREAM, 0 )) == -1 )
     {
          perror("Pb_socket_ecoute");
          exit(-1);
     }

     /* Initialisation de l'adresse du serveur */
     adr_S.sin_port = PORT ;
     adr_S.sin_family = AF_INET ;
     gethostname(nom_machine_S,LONGUEUR_NOM_MACHINE);
     infos_machine_S = gethostbyname(nom_machine_S);
     bcopy( infos_machine_S->h_addr , &adr_S.sin_addr , infos_machine_S->h_length ) ;
     bzero( adr_S.sin_zero , sizeof(adr_S.sin_zero) ) ;

     /* Attachement de la socket d'ecoute a l'adresse */
     if( bind(sock_ecoute_S,(struct  sockaddr *)&adr_S,sizeof(adr_S)) == -1 )
     {
          perror("Pb_bind");
          exit(-2);
     }

    /* Initialisation file des connexions pendantes */
     if( listen( sock_ecoute_S , 5 ) == -1 )
     {
          perror("Pb_listen");
          exit(-3);
     }
```

```
        while(!Fin_Serveur)
        {
             /* Initialisations pour un client */
             lg_adr_C = sizeof(adr_C);

             /* Attente demande connexion d'un client */
             sock_service_S = accept( sock_ecoute_S,
                                      (struct sockaddr *)&adr_C,
                                      &lg_adr_C);

             /* Sous-traitance du service */
             switch(fork())
             {
                case -1 :  /* Erreur  */
                     perror("Pb_fork");
                     exit(-4);

                case 0 :  /* Processus fils  */
                     {
                          int noerr = 0 ;

                          /* Le processus de service n'utilise pas la socket d'ecoute */
                          close(sock_ecoute_S);

                          /* Realisation du service */
                          if((noerr = service( sock_service_S ,
                                               adr_C,
                                               lg_adr_C )))
                               exit(noerr);

                          exit(noerr);
                     }

                default :
                     /* Le processus pere n'a pas besoin de la socket de service */
                     close(sock_service_S);
                     break ;
             }

        } /* Attente d'un autre client en parallele */

        exit(0);
}
```

Code du client

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

```c
#include <netinet/in.h>
#include <pwd.h>
#include <unistd.h>
#include <netdb.h>
#include <strings.h>

#define LONGUEUR_REPONSE 256
#define LONGUEUR_ITEM 64
#define LONGUEUR_NOM_MACHINE 256
#define PORT_S 2222

int
main( int argc , char * argv[] )
{
     /* Variable Client */
     int sock_C ;

     /* Variables Serveur */
     struct sockaddr_in adr_S;
     int lg_adr_S ;
     char nom_machine_S[LONGUEUR_NOM_MACHINE];
     struct hostent  *infos_machine_S ;

     /* Variables de travail */
     ssize_t nb_cars ;
     char msgerr[126] ;
     char reponse[LONGUEUR_REPONSE] ;
     char pw_prenom[LONGUEUR_ITEM],
          pw_nom[LONGUEUR_ITEM],
          pw_dir[LONGUEUR_ITEM];
     int uid , gid ;
     int type ;

     /*-----*/

     if( argc != 3 )
     {
          fprintf( stderr , "Usage : %s <machine> <utilisateur\n", argv[0] );
          exit(-1);
     }
     strcpy( nom_machine_S , argv[1] );

     /* Creation point d'entree Client */
     if( (sock_C = socket(AF_INET, SOCK_STREAM, 0 )) == -1 )
     {
          perror("Pb socket");
          exit(-1);
     }
```

```
/* Preparation de l'adresse du Serveur */
adr_S.sin_family = AF_INET ;
adr_S.sin_port = htons(PORT_S) ;
if( (infos_machine_S = gethostbyname(nom_machine_S)) == NULL )
{
     sprintf( msgerr , "Pb sur le nom machine %s", argv[1]);
     perror(msgerr);
     exit(-2);
}
bcopy( infos_machine_S->h_addr , &adr_S.sin_addr , infos_machine_S->h_length ) ;
bzero( adr_S.sin_zero , sizeof(adr_S.sin_zero) ) ;

/* Demande de connexion */
lg_adr_S = sizeof(adr_S) ;
if( connect( sock_C,
             (struct sockaddr *)&adr_S,
             lg_adr_S ) == -1 )
{
     perror("Pb connect");
     exit(-3);
}

/* Envoi du message = nom d'un utilisateur */
nb_cars = send( sock_C,
                (void *)argv[2],
                strlen(argv[2]),
                0);

/* Attente de la reponse = type  */
/* Exemple consultation sans extraction */
bzero( reponse , LONGUEUR_REPONSE );
nb_cars = recv( sock_C,
                (void *)reponse,
                sizeof(char),
                MSG_PEEK );
sscanf( reponse , "%d" , &type );

/* Test du type */
if( type == 1 )
{
     fprintf( stderr, "%s : utilisateur inconnu sur la machine %s\n",
              argv[2] , argv[1] );
     exit(-3);
}

/* Attente de la reponse = infos  */
bzero( reponse , LONGUEUR_REPONSE );
nb_cars = recv( sock_C,
                (void *)reponse,
```

```
                        LONGUEUR_REPONSE,
                        0);

    /* Deconnexion */
    shutdown(sock_C,2);

    /* Exploitation de la reponse */
    printf( "Utilisateur %s sur machine %s\n\n", argv[2], argv[1] );
    sscanf(reponse , "%d %d %d %s %s %s", &type , &uid , &gid , pw_prenom, pw_nom, p
    printf( "   uid=%d   gid=%d\n", uid , gid );
    printf( "   %s %s %s\n", pw_prenom, pw_nom, pw_dir );

    exit(0);
}
```

2.5 Les messages urgents

2.5.1 Introduction

Avec les sockets de type `SOCK_STREAM` l'information est un flot continu de caractères.

flot de caractères = abcdefg**h**ijklmno...

Dans cet exemple, le caractère "h" ne peut pas être lu avant tous ceux qui le précèdent ("abcdefg").
Pour prendre en compte immédiatement certains caractères, le protocole TCP prévoit la possibilité de transmettre au moins un caractère urgent. C'est un mécanisme de transport qui notifie au processus `D` l'arrivée de caractères pour qu'il puisse les prendre en compte immédiatement. La réalisation UNIX correspond à la notion de caractères de type `OOB` (*Out Of Band*) dont l'arrivée peut être notifiée au processus `D` par le signal `SIGURG`.

2.5.2 Émission des OOB

Il faut donc
- une socket locale de type `SOCK_STREAM`
- être dans le domaine INTERNET (pour utiliser le protocole TCP)
- utiliser la primitive `send` avec l'option `MSG_OOB`

Dans l'implantation UNIX, on ne peut envoyer qu'un seul caractère urgent (le dernier caractère du message envoyé par `send`). Exemple :

```
send( socket , ''abcd'' , 4 , MSG_OOB );
```

- les caractères "abc" sont ordinaires et accessibles dans le flot normal
- le caractère "d" est urgent et accessible à l'autre extrémité de la connexion hors du flot normal

2.5.3 Réception des OOB

Le système repère un caractère urgent par socket, grâce à une marque logique.

La lecture d'un caractère OOB est réalisée par `recv` avec l'option `MSG_OOB`
- il n'est pas nécessaire d'être sur la marque repérant le caractère OOB pour pouvoir le lire
- la lecture sans l'option `MSG_OOB` bute contre la marque. Par exemple si le *flot* = abcd**e**fghi avec "e" caractère urgent alors après l'appel à

```
recv(sock,ch,6,0) ;
```

la chaîne `ch` aura comme valeur "abcd" ;

La recherche de la marque est réalisée au moyen de la primitive de contrôle des entrées/sorties `ioctl` avec la commande `SIOCATMARK` qui teste si la position courante est celle du caractère urgent.

```
int rep ;
ioctl(sock,SIOCATMARK,&rep);
```

$rep = 1 \Leftrightarrow$ la marque est atteinte.
Pour obtenir un caractère OOB il faut vider tous les caractères précédant

la marque, par exemple par le bout de code suivant :

```
/* Vidage des caracteres precedant le OOB */
ioctl(socket_recepteur,SIOCATMARK,&reponse);
while( reponse != 1 )
{
  recv(socket_recepteur,buffer,LONGUEUR_MESSAGE,0);
  ioctl(socket_recepteur,SIOCATMARK,&reponse);
}
/* Recuperation du caractere OOB */
recv(socket_recepteur,buffer,1,MSG_OOB);
```

La prise en compte d'un caractère OOB est réalisé par le code du "vidage" quand on reçoit le signal `SIGURG`. Il faut donc :
- mettre le code du "vidage" en *handler* du signal `SIGURG`
- dire au système d'envoyer le signal `SIGURG` au processus récepteur quand un caractère OOB arrive sur sa socket. Ceci est réalisé par la primitive `fcntl` avec l'opération `F_SETOWN` :

```
fcntl( sock,F_SETOWN, getpid());
```

2.5.4 Exemple

Code d'un émetteur d'un caractère urgent

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/ioctl.h>
#include <netinet/in.h>
#include <pwd.h>
#include <unistd.h>
#include <netdb.h>
#include <strings.h>

#define LONGUEUR_MESSAGE 256
#define LONGUEUR_NOM_MACHINE 256
#define PORT_R 2222

/* Code de l'emtteur */
int
main( int argc , char * argv[] )
{
      /* Variable Emetteur */
      int sock_E ;

      /* Variables Serveur */
      struct sockaddr_in adr_R;
      int lg_adr_R ;
      char nom_machine_R[LONGUEUR_NOM_MACHINE];
      struct hostent  *infos_machine_R ;

      /* Variables de travail */
      ssize_t nb_cars ;
      char msgerr[126] ;
      char message[LONGUEUR_MESSAGE] ;
      int nb_messages = 0 ;
      int i = 0 ;

      /*-----*/

      if( argc != 3 )
      {
            fprintf( stderr , "Usage : %s <machine> <nb_messages>\n", argv[0] );
            exit(-1);
      }
      strcpy( nom_machine_R , argv[1] );
      sscanf( argv[2] , "%d" , &nb_messages );

      /* Creation point d'entree Emetteur */
      if( (sock_E = socket(AF_INET, SOCK_STREAM, 0 )) == -1 )
```

```
{
      perror("Pb socket");
      exit(-1);
}

/* Preparation de l'adresse du Recepteur/Serveur */
adr_R.sin_family = AF_INET ;
adr_R.sin_port = htons(PORT_R) ;
if( (infos_machine_R = gethostbyname(nom_machine_R)) == NULL )
{
      sprintf( msgerr , "Pb sur le nom machine %s", argv[1]);
      perror(msgerr);
      exit(-2);
}
bcopy( infos_machine_R->h_addr , &adr_R.sin_addr , infos_machine_R->h_length ) ;
bzero( adr_R.sin_zero , sizeof(adr_R.sin_zero) ) ;

/* Demande de connexion */
lg_adr_R = sizeof(adr_R) ;
if( connect( sock_E,
             (struct sockaddr *)&adr_R,
             lg_adr_R ) == -1 )
{
      perror("Pb connect");
      exit(-3);
}

for(i=0;i<nb_messages;i++)
{
      sprintf( message , "abcde%cfghij" , '0'+(i%10) );
      printf(" Emetteur : message envoye = %s\n", message);
      /* Envoi du message */
      if((nb_cars = send( sock_E,
                          (void *)message,
                          strlen(message),
                          0))== -1 )
      {
            perror("Pb send");
            exit(-4);
      }
}

/* Envoi du message de fin */
sprintf( message , "abcdexfghij");
printf(" Emetteur : message envoye = %s\n", message);
if((nb_cars = send( sock_E,
                    (void *)message,
                    6,
                    MSG_OOB))== -1 )
```

```
    {
        perror("Pb send");
        exit(-4);
    }

    /* Deconnexion */
    shutdown(sock_E,2);

    exit(0);
}
```

Code d'un récepteur d'un caractère urgent

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/sockio.h>
#include <sys/ioctl.h>
#include <netinet/in.h>
#include <pwd.h>
#include <unistd.h>
#include <netdb.h>
#include <strings.h>
#include <signal.h>
#include <fcntl.h>

#define LONGUEUR_MESSAGE 256
#define LONGUEUR_NOM_MACHINE 256
#define PORT 2222

/* Variables globales (handlers) */
int Fin_Recepteur = 0 ;
int sock_service_R ;

/* Handler SIGURG */
static
void
traite_oob( char oob )
{
    switch(oob)
    {
       case 'x' :
            Fin_Recepteur = 1 ;
            break;
       default :
            printf("\t caractere urgent inconnu : [%c]\n", oob );
            break;
    }
}
```

```
static
void
sigurg( int sig )
{
     int reponse = 0 ;
     char buffer[LONGUEUR_MESSAGE] ;

     /*-----*/
     signal( SIGURG , sigurg ) ;

     printf( "Recepteur : recu signal SIGURG (%d) : caractere urgent\n", sig );

     /* Vidage des caracteres precedant le OOB */
     while( reponse != 1 )
     {
          recv(sock_service_R,buffer,LONGUEUR_MESSAGE,0);
          ioctl(sock_service_R,SIOCATMARK,&reponse);
     }
     /* Recuperation du caractere OOB */
     recv(sock_service_R,buffer,1,MSG_OOB);

     /* Traitement du caractere urgent */
     traite_oob(*buffer);
}

/* Code du Recepteur */
int
main( int argc , char * argv[] )
{
     /* Variables Emetteur */
     struct sockaddr_in adr_E ;
     int lg_adr_E ;

     /* Variables Recepteur */
     struct sockaddr_in adr_R;
     int sock_ecoute_R ;
     char nom_machine_R[LONGUEUR_NOM_MACHINE];
     struct hostent  *infos_machine_R ;

     /* Variables de travail */
     ssize_t nb_cars = (ssize_t)0 ;
     char message[LONGUEUR_MESSAGE] ;

     /*-----*/

     /* Creation socket d'ecoute */
     if( (sock_ecoute_R = socket(AF_INET, SOCK_STREAM, 0 )) == -1 )
     {
```

```
        perror("Pb socket ecoute");
        exit(-1);
}

/* Initialisation de l'adresse du recepteur */
adr_R.sin_port = PORT ;
adr_R.sin_family = AF_INET ;
gethostname(nom_machine_R,LONGUEUR_NOM_MACHINE);
infos_machine_R = gethostbyname(nom_machine_R);
bcopy( infos_machine_R->h_addr , &adr_R.sin_addr , infos_machine_R->h_length ) ;
bzero( adr_R.sin_zero , sizeof(adr_R.sin_zero) ) ;

/* Attachement de la socket d'ecoute a l'adresse */
if( bind(sock_ecoute_R,(struct  sockaddr *)&adr_R,sizeof(adr_R)) == -1 )
{
        perror("Pb bind");
        exit(-2);
}

/* Initialisation file des connexions pendantes */
if( listen( sock_ecoute_R , 5 ) == -1 )
{
        perror("Pb listen");
        exit(-3);
}

/* Initialisations pour un emetteur */
lg_adr_E = sizeof(adr_E);

/* Attente demande connexion d'un emetteur */
sock_service_R = accept( sock_ecoute_R,
                         (struct sockaddr *)&adr_E,
                        &lg_adr_E);

 /* Reception des caracteres urgents */
signal( SIGURG , sigurg ) ;
fcntl( sock_service_R , F_SETOWN , getpid() );

while(!Fin_Recepteur)
{
        /* Attente du message de l'emetteur */
        bzero( message , LONGUEUR_MESSAGE );
        nb_cars = recv( sock_service_R,
                        (void *)message,
                        LONGUEUR_MESSAGE,
                        0 );

        /* Fin d'attente */
        if( !Fin_Recepteur )
```

```
                printf("␣Recepteur␣:␣message␣recu␣=␣%s\n", message );
        }

        /* Deconnexion d'avec l'emetteur */
        close(sock_service_R);

        exit(0);
}
```

Sortie du côté de l'émetteur

```
prompt> emetteur_oob machine_hote 100

 Emetteur : message envoye = abcde0fghij
 Emetteur : message envoye = abcde1fghij
 Emetteur : message envoye = abcde2fghij
 Emetteur : message envoye = abcde3fghij
 Emetteur : message envoye = abcde4fghij
 Emetteur : message envoye = abcde5fghij
    |          |      |          |
             (100 messages)
    |          |      |          |
 Emetteur : message envoye = abcde9fghij
 Emetteur : message envoye = abcdexfghij
```

Sortie possible du côté du récepteur

```
prompt> recepteur_oob

 Recepteur : message recu = abcde0fghij
 Recepteur : message recu = abcde1fghij
 Recepteur : message recu = abcde2fghijabcde3fghijabcde4fghijabcde5fghijabcde6fghi
jabcde7fghijabcde8fghijabcde9fghijabcde0fghijabcde1fghijabcde2fghijabcde3fghijabcd
e4fghijabcde5fghijabcde6fghijabcde7fghijabcde8fghijabcde9fghijabcde0fghijabcde1fgh
ijabcde2fghijabcde3fghijabcde4fghijabc
 Recepteur : message recu = de5fghijabcde6fghijabcde7fghijabcde8fghijabcde9fghijab
cde0fghijabcde1fghijabcde2fghijabcde3fghijabcde4fghijabcde5fghijabcde6fghijabcde7f
ghijabcde8fghijabcde9fghijabcde0fghijabcde1fghijabcde2fghijabcde3fghijabcde4fghija
bcde5fghijabcde6fghijabcde7fghijabcde8
 Recepteur : message recu = fghijabcde9fghijabcde0fghijabcde1fghijabcde2fghijabcde
3fghijabcde4fghijabcde5fghijabcde6fghijabcde7fghijabcde8fghijabcde9fghijabcde0fghi
jabcde1fghijabcde2fghijabcde3fghijabcde4fghijabcde5fghijabcde6fghijabcde7fghijabcd
e8fghijabcde9fghijabcde0fghijabcde1fgh
 Recepteur : message recu = ijabcde2fghijabcde3fghijabcde4fghijabcde5fghijabcde6fg
hijabcde7fghijabcde8fghijabcde9fghijabcde0fghijabcde1fghijabcde2fghijabcde3fghijab
cde4fghijabcde5fghijabcde6fghijabcde7fghijabcde8fghijabcde9fghijabcde0fghijabcde1f
ghijabcde2fghijabcde3fghijabcde4fghija
 Recepteur : message recu = bcde5fghijabcde6fghijabcde7fghijabcde8fghij
Recepteur : recu signal SIGURG (21) : caractere urgent
```

Constatations

- Le caractère urgent "x" est envoyé après une série de messages, mais il est pris en compte par le récepteur en dehors du flot normal des caractères, bien avant l'arrivée de tous les messages.
- on peut se rendre compte de l'aspect *flot d'informations*, sans sauvegarde des limites de messages qui ne sont pas les mêmes lors de l'émission et lors de la réception.