Exemples de Programmation Réseau

Philippe Langevin

Octobre 2007.

Modèle Client/Serveur

Les applications réseaux sont classées en deux catégories:

- Serveur qui attend une communication : attente d'une demande d'ouverture de communication, réception d'une requête et envoie d'une réponse.
- Client qui initie le lancement d'une communication : demande d'ouverture de connexion, d'une requête, attente de la réponse et traitement.

Ouvrage de référence : Programmation Système en Langage C sous Linux, par Christophe Blaess.

Connexion

- UDP: application orientée sans connexion, mode non connecté.
- TCP: application orientée connexion.

Adresses

Les adresses existent sous forme de chaine de caractères,

`mail.univ-tln.fr  127.0.0.1`

où encore de mots de 32 bits.

$$1 * 256^3 + 0 * 256^2 + 0 * 256^1 + 127 = 16777343$$

./whoami.exe	./whoami.exe mail.univ-tln.fr
hote :msnet.sollies.fr	hote :mail.univ-tln.fr
addr :16777343	addr :39858625
ip :127.0.0.1	ip :193.49.96.2

- complétée par un numéro de port de 32 bits.

Who am I ?

```
int main( int argc, char* argv[])
{ struct in_addr    num;
  struct hostent *infos;
  char nom[MAXNAME];
  if ( argc > 1 )
       infos = gethostbyname( argv[1] );
  else {
       gethostname(nom,MAXNAME);
       infos = gethostbyname( nom );
  }
  if ( infos ) {
     printf("\nhote :%s ", infos->h_name);
     memcpy( &num, infos -> h_addr_list[0], 4);
     printf("\naddr :%lu ", (unsigned long) num.s_addr);
     printf("\nip   :%s",  inet_ntoa( num ) );
  }
  printf("\n");
  return 0;
}
```

structures hostent & in_addr

```
struct in_addr { unsigned long int s_addr; }
#include <netdb.h>
struct hostent {
    char     *h_name;
    char    **h_aliases;
    int       h_addrtype;
    int       h_length;
    char    **h_addr_list;
 }
#define h_addr   h_addr_list[0]
```

- h_aliases: table d'alternatives au nom officiel de l'hôte, terminée par un pointeur NULL.
- h_addrtype : toujours AF_INET ou AF_INET6.
- h_length : la longueur, en octets, de l'adresse.
- h_addr_list: une table, terminée par un pointeur NULL, d'adresses réseau pour l'hôte, avec l'ordre des octets du réseau.
- h_addr: première adresse dans h_addr_list pour respecter la compatibilite ascendante

Résolution des noms

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

- La fonction `gethostbyname()` renvoie une structure de type `hostent` pour l'hôte name. La chaîne name est soit un nom d'hôte, soit une adresse IPv4 en notation pointée standard, soit une adresse IPv6 avec la notation points-virgules et points. Si name est une adresse IPv4 ou IPv6, aucune recherche supplémentaire n'a lieu et `gethostbyname()` copie simplement la chaîne name dans le champ h_name et le champ équivalent `struct in_addr` dans le champ `h_addr_list[0]` de la structure hostent renvoyée.

Résolution inverse

```
#include <sys/types.h> #include <sys/socket.h>
extern int h_errno;
struct hostent *gethostbyaddr(const void *addr,
        int len, int type);
```

- La fonction `gethostbyaddr()` renvoie une structure du type hostent pour l'hôte d'adresse addr. Cette adresse est de longueur len et du type donné. Les types d'adresse valides sont AF_INET et AF_INET6. L'argument adresse de l'hôte est un pointeur vers une structure de type dépendant du type de l'adresse, par exemple `struct in_addr *` (probablement obtenu via un appel `inet_addr()`) pour une adresse de type AF_INET.
- Retour: Les fonctions `gethostbyname()` et `gethostbyaddr()` renvoient un pointeur sur la structure hostent , ou bien un pointeur NULL si une erreur se produit, auquel cas `h_errno` contient le code d'erreur.

Manipulation des adresses

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

struct in_addr {
                unsigned long int s_addr;
            }

in_addr_t inet_addr (const char * cp);
char * inet_ntoa (struct in_addr in);
```

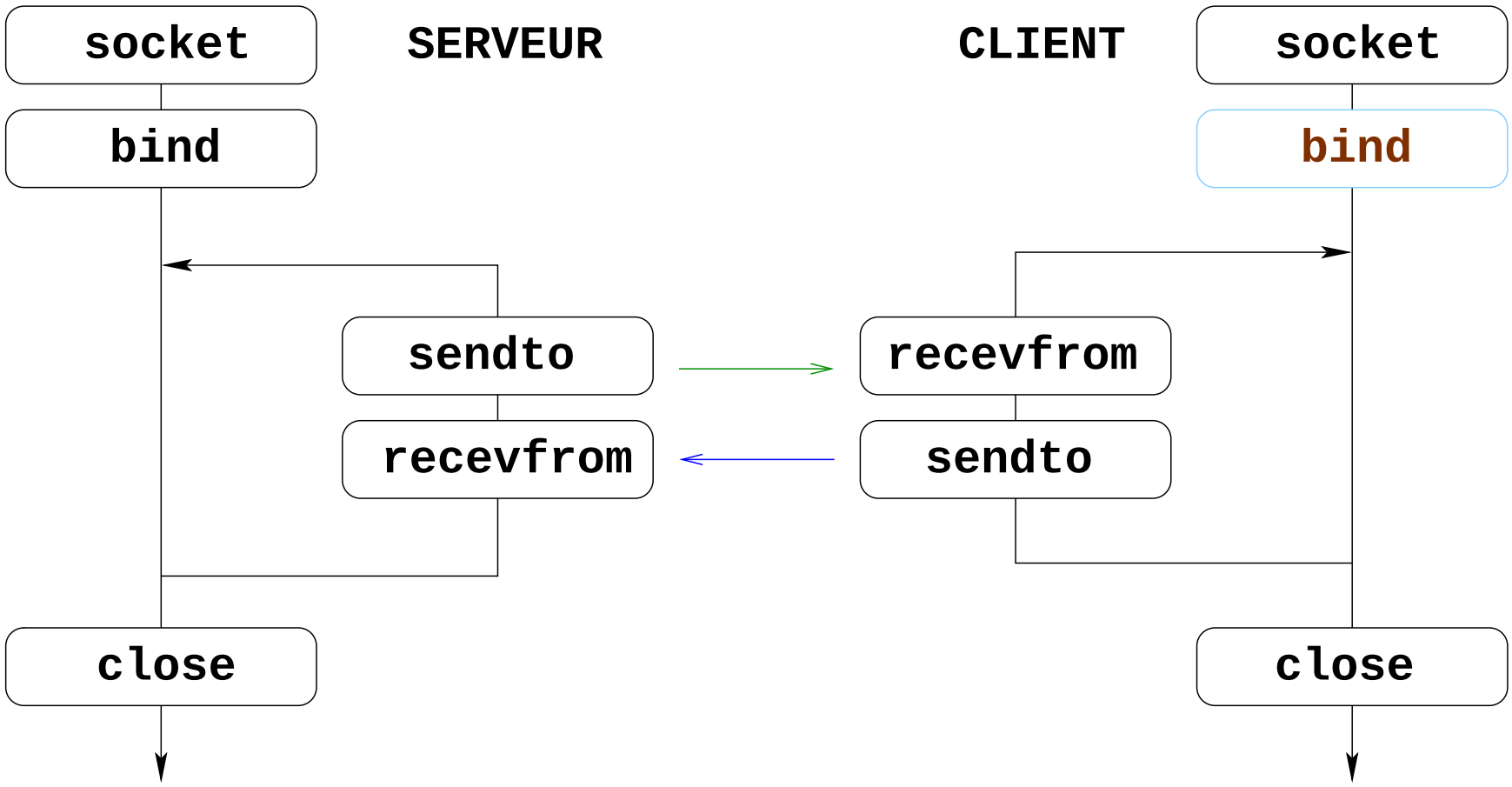
- La fonction `inet_ntoa()` convertit l'adresse Internet de l'hôte in donnée dans l'ordre des octets du réseau en une chaîne de caractères dans la notation avec nombres et points. La chaine est renvoyée dans un buffer
- La fonction `inet_addr()` convertit l'adresse Internet de l'hôte cp depuis la notation standard avec nombres et points en une donnée binaire dans l'ordre des octets du réseau.

Manipulation des adresses

```
#include ...

int inet_aton (const char *cp, struct in_addr *inp);
in_addr_t inet_network (const char * cp);
```

- `inet_aton()` convertit l'adresse Internet de l'hôte cp depuis la nota- tion standard avec nombres et points en une donnée binaire, et la stocke dans la structure pointée par inp. `inet_aton` renvoie une valeur non nulle si l'adresse est valide, et zéro sinon.
- La fonction `inet_network()` extrait la partie réseau de l'adresse cp fournie dans la notation avec nombres et points, et renvoie cette valeur dans l'ordre des octets de l'hôte. Si l'adresse est invalide, -1 est renvoyé.

Mode non connecté



Créer un point de communication.

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

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

Création d'un point de communication, et renvoie un descripteur.

- domain indique un domaine de communication, l'intérieur duquel s'établira le dialogue PF-INET pour IPv4 et PF-INET6 pour IPv6.
- type fixe la sémantique du dialogue : SOCK-STREAM, SOCK-DGRAM
- protocol numéro de protocol.
- Quand une session se termine, on referme la socket avec close.
- retour un descripteur référenant la socket créée en cas de réussite. En cas d'échec -1 est renvoyé, et `errno` contient le code d'erreur.

Assigner un nom à une socket.

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

int bind(int sockfd, struct sockaddr *my_addr,
    socklen_t addrlen);
```

Fournit à la socket sockfd, l'adresse locale my_addr. my_addr est longue de addrlen octets. Traditionnellement cette opération est appelée "assignation d'un nom à une socket".

- La plage de port 1–511 est réservée au root.
- Plage des ports libres: 5001–65535
- Attention à l'ordre des octets.
- retour :renvoie 0 s'il réussit, ou -1 s'il échoue, auquel cas errno con- tient le code d'erreur.

Conversion des entiers

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

ulong    htonl (ulong) ;
ushort   htons (ushort) ;
ulong    ntohl (ulong) ;
ushort   ntohs (ushort) ;
```

- La fonction `ntohl()` convertit un entier non-signé netlong depuis l'ordre des octets du réseau vers celui de l'hôte.
- `ntohl()` fait le contraire.
- Pour affecter le numéro de port 13 (service "daytime") au champ `sin_port` d'une structure de type `sockaddr_in` :

```
saddr.sin_port = htons(13) ;
```

Recevoir un message

```
#include <sys/types.h>      #include <sys/socket.h>
int recvfrom(int s, void *buf, int len, uint flags,
    struct sock_addr *from, socklen_t *fromlen);
```

L'appel-système `recvfrom` est utilisé pour recevoir un message depuis une socket s (socket orientée connexion ou non).

- L'adresse de la source du messages est insérée dans from. L'argument fromlen est un paramètre résultat, initialisé à la taille du buffer from, et modifié en retour pour indiquer la taille réelle de l'adresse enregistrée.
- La réception se mettent en attente, à moins que la socket soit non blo- quante auquel cas la valeur -1 est renvoyée, et `errno` est positionnée à EAGAIN. Les fonctions de réception renvoient nor- malement les données disponibles sans attendre d'avoir reu le nombre exact réclamé.
- La routine renvoie le nombre d'octets lus si elle réussit. Si un message est trop long pour tenir dans le buffer, les octets supplémentaires peuvent être abandonnés.

Serveur en mode non connecté

```
    char bfr[1024], *ptr;
    struct sockaddr_in serv, client;
    struct sockaddr    aux;
    sd = socket( AF_INET, SOCK_DGRAM, 0);
    memset( &serv, 0 , sizeof(serv) );
    serv.sin_family = AF_INET;
    serv.sin_port = htons( 31415 );
    memcpy( &serv.sin_addr, IPno, 4);
    bind( sd, (struct sockaddr *) &serv, sizeof(serv));
    while ( 1 ) {
        nb = recvfrom( sd, &bfr, 1024 , 0, &aux , &taille)
        memcpy( &client, &aux, sizeof(aux) );
        ptr = inet_ntoa( client.sin_addr )  ;
        printf("\n%d octets <-(%s): ", nb, ptr);
        for( i = 0; i < nb; i++)
          if ( isprint( bfr[i] ) ) printf("%c", bfr[i] );
    }
    close(sd);
    return 0;
}
```

Structure sockaddr &sockaddr_in

```
struct sockaddr {
 unsigned short sa_family;
 char           sa_data[14];
};
struct sockaddr_in {
  short int          sin_family;
  unsigned short int sin_port;
  struct in_addr     sin_addr;
  unsigned char      sin_zero[8];
};
```

- `sa_family` : AF_UNIX et AF_INET sont les plus courantes.
- `sa_data` : addresse de destination et numéro de port, nom de fichier dans le cas AF_UNIX.
- La structure générique `sockaddr` est plutôt le difficile à manier, `sockaddr_in` spécifique au socket “internet”.
- taille identique, transtypage.

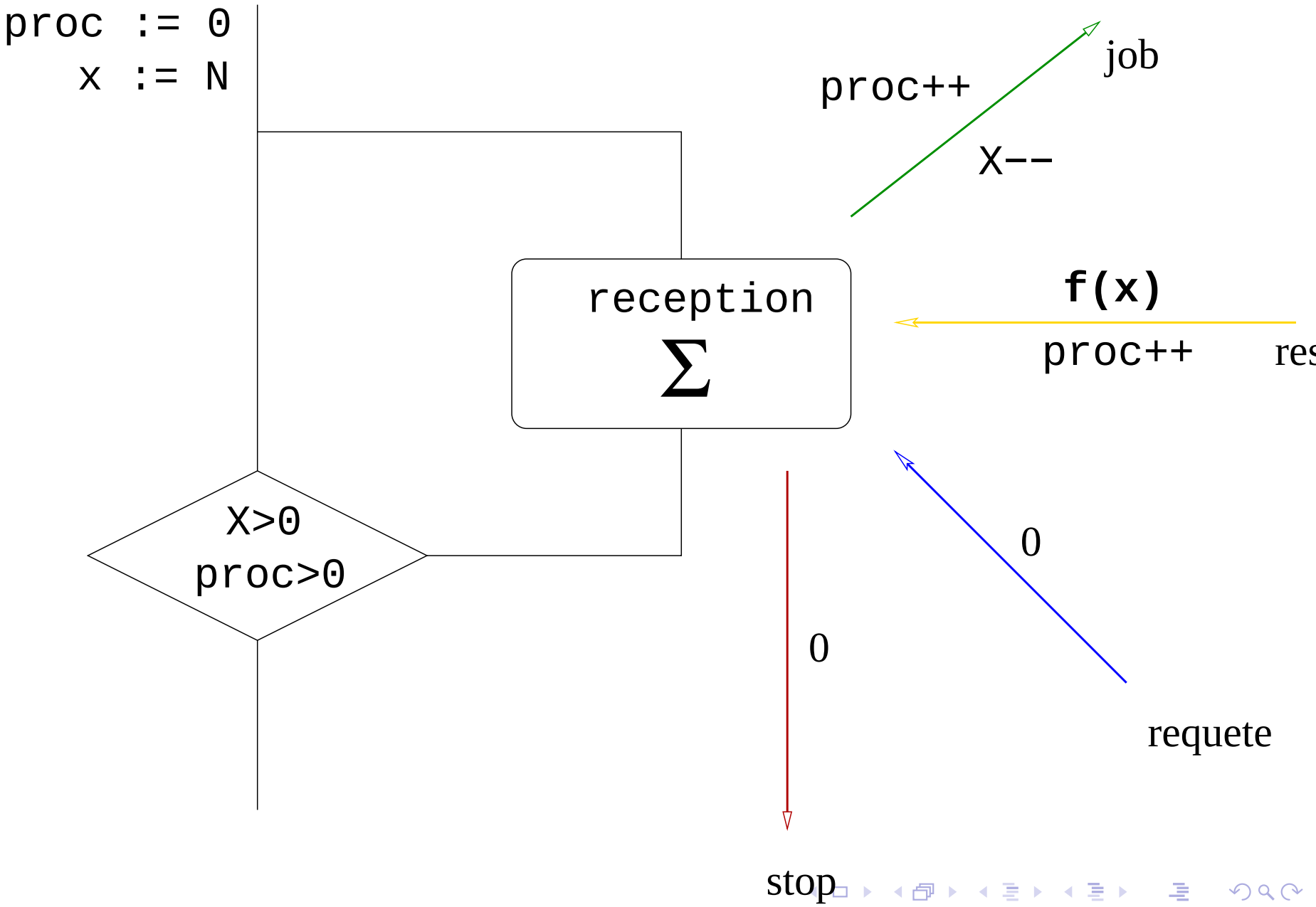
netstat & client dig

```
[pl@ou812] ./udpserveur.exe &
           [2] 4078

[pl@ou812] netstat -ap | grep 31415
           udp 0 0 ou812.univ-tln.fr:31415 *:*
           16257/sdcserv.exe

[pl@ou812] ssh maitinfo1
[pl@maiti] dig HelloWorld @10.2.73.86 -p31415
           28 octets <- 10.9.185.217xHelloWorld
           28 octets <- 10.9.185.217xHelloWorld

; <<>> DiG 9.3.2 <<>> HelloWorld @10.2.73.86 -p31415
; (1 server found)
;; global options: printcmd
;; connection timed out; no servers could be reached
```

client telnet & nmap

```
[pl@ou812] telnet 10.2.73.86  31415
Trying 10.2.73.86...
telnet: connect to address 10.2.73.86: Connection refuse
telnet: Unable to connect to remote host: Connection ref

[pl@ou812] su
[ root   ] nmap -sU -p31414-31416 10.2.73.86

Starting Nmap 4.03

           0 octets <- 10.2.73.86

Interesting ports on (10.2.73.86):
PORT      STATE          SERVICE
31414/udp closed         unknown
31415/udp open|filtered  unknown
31416/udp closed         unknown

Nmap finished: 1 IP address (1 host up)
```

Un client/serveur



option : sdcserveur

```
        switch ( opt ) {
        case 'p' : PORT = atoi(optarg);    break;
        case 's' : SERV = optarg;          break;
        case 'n' : NMAX = atoi(optarg);    break;
        case 'h' : printf("\nusage %s: -p port -n max", ar
                   printf("\nusage %s: -s serveur\n", argv
        default  : return 0;
        }
    };
    return 1;
}
```

main : sdcserveur

```
int main(int argc, char *argv[])
{ int  sd, res = 0;
  struct sockaddr_in serv;

  if ( ! argok( argc, argv )  ) return 1;

  sd = socket( AF_INET, SOCK_DGRAM, 0);

  serv = sockaddrinlocal( PORT );

  bind( sd, (struct sockaddr *) &serv, sizeof(serv));

  res = sdcnet( sd, NMAX );
  printf("\nSomme %d premiers carres : %d", NMAX, res);
  close(sd);
  return 0;
}
```

sockaddrinlocal:socktools.c

```
struct sockaddr_in sockaddrinlocal( int port)
{ struct sockaddr_in res;
  memset( &res, 0 , sizeof( res ) );
  res.sin_family = AF_INET;
  res.sin_port = htons( port );
  res.sin_addr.s_addr  = htonl(INADDR_ANY);
  return res;
}
```

sdcnet : sdcserveur

```
int sdcnet( int sd, int n )
{ int  res = 0, val, nbproc = 0;
  struct sockaddr client;
  while ( n || nbproc  ) {
      val = recvpaquetfrom( sd, &client );
      printfrom( &client );
      if ( val ) {
        printf("-> %d", val);
        res += val;
        nbproc--;
      } else {
        sendpaquetto( sd, &client, n );
        if ( n ) { nbproc++; n--;}
      }
      printf("\n%d actifs, reste=%d sdc=%d", nbproc, n,
  }
  sendpaquetto( sd, &client, 0);
  return res;
}
```

paquet : socktools.c

```
    nb = sendto( sd, &val, 4 , 0, client, TAILLE);
    if ( nb < 0 ) erreur("sendto");
}
int recvpaquetfrom( int sd,  struct sockaddr *client )
{ int val, nb;
    nb = recvfrom( sd, &val, 4 , 0 , client , &TAILLE);
    if ( nb < 0 ) erreur("recvfrom");
    return val;
}
int nobloqpaquetfrom( int sd,  struct sockaddr *client )
{ int  val, nb;
    nb = recvfrom( sd,&val,4,MSG_DONTWAIT, client,&TAILLE);
    if ( nb < 0 ) { val = -1;
      if ( errno != EAGAIN ) erreur("nb recvfrom");
    }
    return val;
}


int argok( int argc, char* argv[])
```

Envoyer un message

```
int sendto(int s, const void *buf, size_t len, int flags
    const struct sockaddr *to, socklen_t tolen);
```

Les appels systèmes `send()`, `sendto()`, et `sendmsg()` permettent de transmettre un message à destination d'une autre socket. Si `sendto()` est utilisée sur une socket en mode connexion (SOCK_STREAM, SOCK_SEQPACKET), les paramètres to et tolen sont ignorés. Autrement, l'adresse de la cible est fournie par to, tolen spécifiant sa taille.

- Retour : S'ils réussissent, ces appels systèmes renvoient le nombre de caractères émis. S'ils échouent, ils renvoient -1 et `errno` contient le code d'erreur.

main:sdclient

```
int main(int argc, char *argv[])
{ int    sd;
  struct sockaddr_in serv;

  if ( ! argok(argc, argv ) ) return 1;

  sd = socket( AF_INET, SOCK_DGRAM, 0);
  if ( sd < 0 ) erreur("socket");

  serv = sockaddrinbyname( SERV , PORT );


  calcul( sd , (struct sockaddr*) &serv);

  close(sd);

  return 0;
}
```

sockaddrinbyname:socktools.c

```
struct sockaddr_in sockaddrinbyname(char *nom, int port)
{ struct sockaddr_in res;
  struct hostent* infos;
  infos = gethostbyname( nom );
  memset( &res, 0 , sizeof( res ) );
  res.sin_family = AF_INET;
  res.sin_port = htons( port );
  memcpy(&res.sin_addr, infos -> h_addr_list[0], 4);
  return res;
}

struct sockaddr_in sockaddrinlocal( int port)
{ struct sockaddr_in res;
  memset( &res, 0 , sizeof( res ) );
  res.sin_family = AF_INET;
  res.sin_port = htons( port );
  res.sin_addr.s_addr  = htonl(INADDR_ANY);
  return res;
}
```

calcul:sdclient

```
void calcul( int sd, struct sockaddr *serv)
{ int  val;
  while ( 1 ) {
      val = 0;
      sendpaquetto( sd, serv, val);
      val = recvpaquetfrom( sd, serv );
      if ( ! val ) return;
      val = val * val;
      sleep( random() % 10);
      sendpaquetto( sd, serv , val );
  }
}
```

Grappe de machines

```
hal9000.univ-tln.fr          grim 1
gitane.univ-tln.fr        zanotti 1
```

Synchronisation des codes

```
grappe=$1
rm     ko.grp
touch ko.grp
while read host login max
do
   ping -c1 -w2 $host >/dev/null
   ALIVE=$?
   if [ "$ALIVE" == "0" ]
      then
          rsync  -av *.c --delete $login@$host:SOCKET/
          rsync  -av *.h --delete $login@$host:SOCKET/
          rsync  -av Makefile --delete $login@$host:SOCKE
          ssh  $login@$host "cd SOCKET;make -f Makefile"
     else
         echo "$host $login $max" >> ko.grp
   fi
done < $grappe
echo "ko are :"
cat ko.grp
```

Lancement

```
grappe=grappe.grp
port=31415
serv=ou812.univ-tln.fr
max=5
./install.sh    $grappe

ps aux | grep $1
rm nohup.out
while read host login max
do
   ping -c1 -w2 $host >/dev/null
   ALIVE=$?
   if [ "$ALIVE" == "0" ]
    then
     while [ $max != 0 ]
      do let max=$max-1
           ssh  $login@$host "nice SOCKET/$2 -s $serv -p $
< /dev/null &
      done
```

Output

```
from 10.2.73.127 in child -> 25
from 10.2.81.42 in child -> 9
from 10.2.81.42 in child -> 4
from 10.2.81.42 in child -> 1
from 10.2.73.127 in child -> 25
from 10.2.81.42 in child -> 9
from 10.2.81.42 in child -> 4
from 10.2.81.42 in child -> 1
from 10.2.73.127 in child -> 25
from 10.2.81.42 in child -> 9
from 10.2.81.42 in child -> 4
from 10.9.185.217 in child -> 1
```

Capture de trames tcpdump

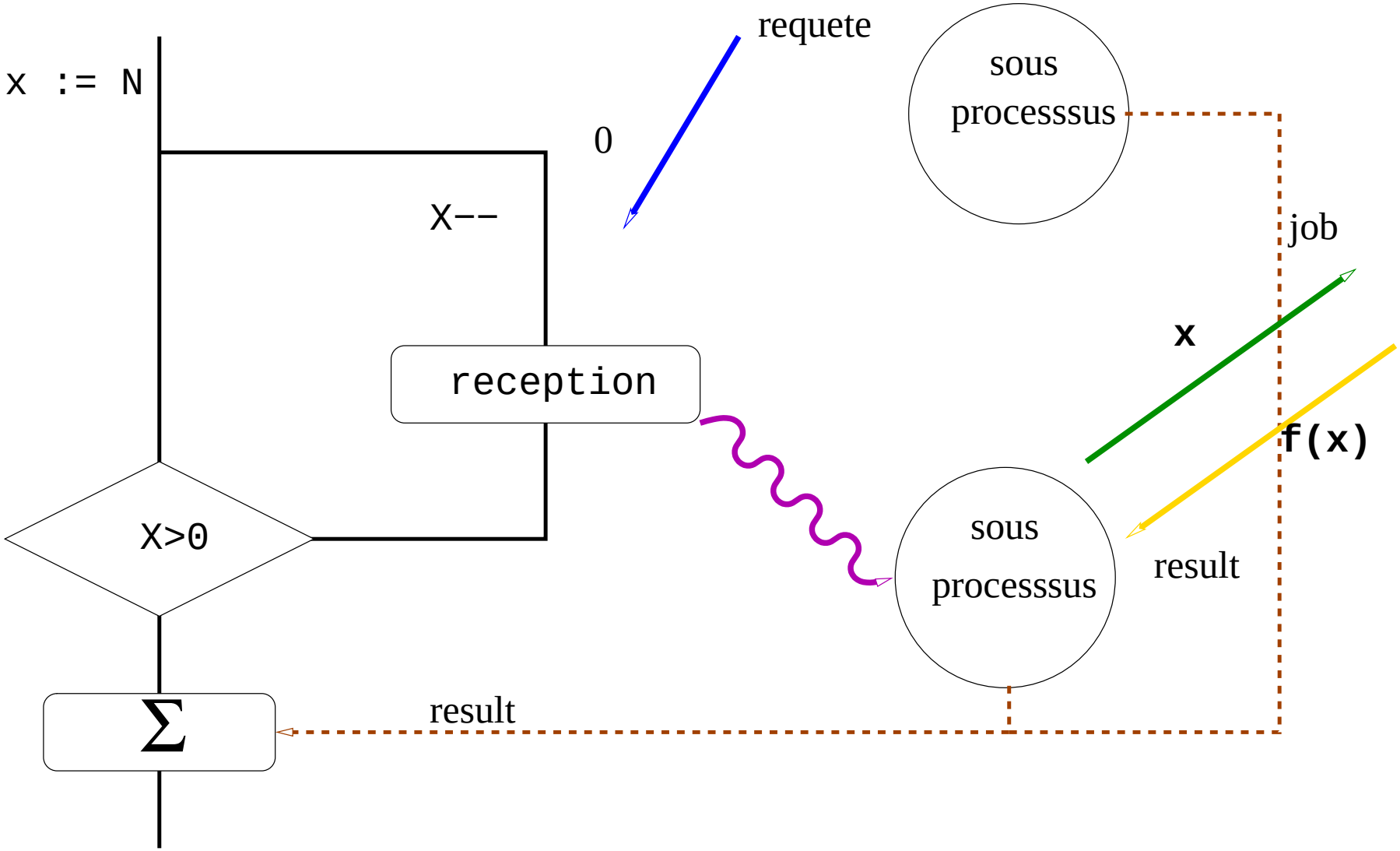
```
[root@ou812 ~]# tcpdump -n port 3000
tcpdump: verbose output suppressed, use -v or -vv for fu
listening on eth0, link-type EN10MB (Ethernet), capture
10:48:48.IP 10.2.73.127.32794 > 10.2.73.86.3000: UDP, le
10:48:48.IP 10.2.73.86.3000 > 10.2.73.127.32794: UDP, le
10:48:48.IP 10.2.73.85.32774 > 10.2.73.86.3000: UDP, len
10:48:48.IP 10.2.73.86.3000 > 10.2.73.85.32774: UDP, len
10:48:48.IP 10.2.81.42.32773 > 10.2.73.86.3000: UDP, len
10:48:48.IP 10.2.73.86.3000 > 10.2.81.42.32773: UDP, len
10:48:49.IP 10.9.185.217.33327 > 10.2.73.86.3000: UDP, l
10:48:49.IP 10.2.73.86.3000 > 10.9.185.217.33327: UDP, l
10:48:49.IP 10.2.81.42.32774 > 10.2.73.86.3000: UDP, len
10:48:49.IP 10.2.73.86.3000 > 10.2.81.42.32774: UDP, len
10:48:49.IP 10.9.185.217.33328 > 10.2.73.86.3000: UDP, l
10:48:49.IP 10.2.73.86.3000 > 10.9.185.217.33328: UDP, l
10:48:49.IP 10.9.185.201.32816 > 10.2.73.86.3000: UDP, l
10:48:49.IP 10.2.73.86.3000 > 10.9.185.201.32816: UDP, l
10:48:49.IP 10.2.81.42.32775 > 10.2.73.86.3000: UDP, len
10:48:49.IP 10.2.73.86.3000 > 10.2.81.42.32775: UDP, len
10:48:51.IP 10.2.73.127.32794 > 10.2.73.86.3000: UDP, le
```

Gestion détachée

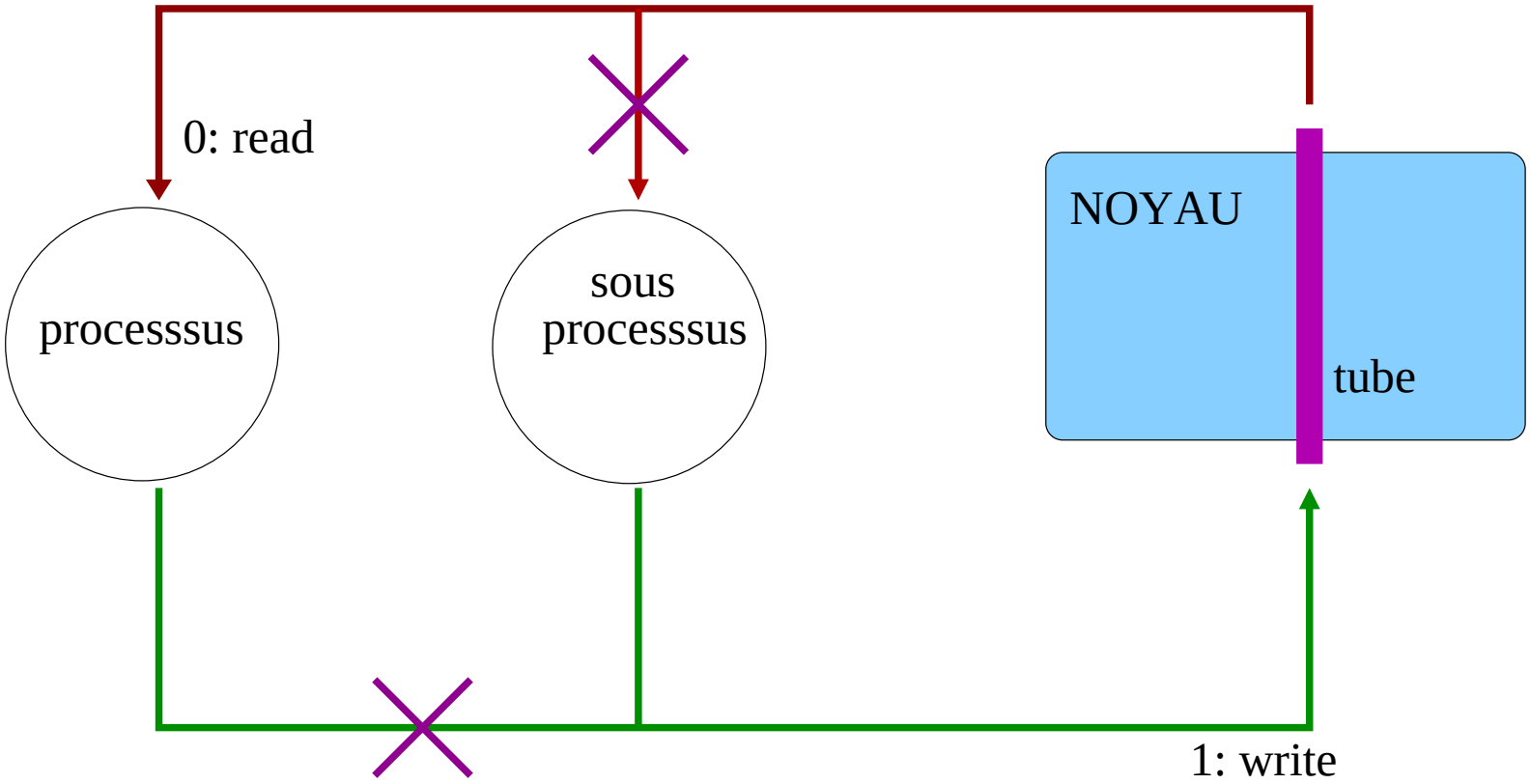
Les clients lancent leur requête sur le serveur qui détache un processus pour la gestion du client.

- communication interprocessus par socket
- communication interprocessus par tube
- mémoire partagée.

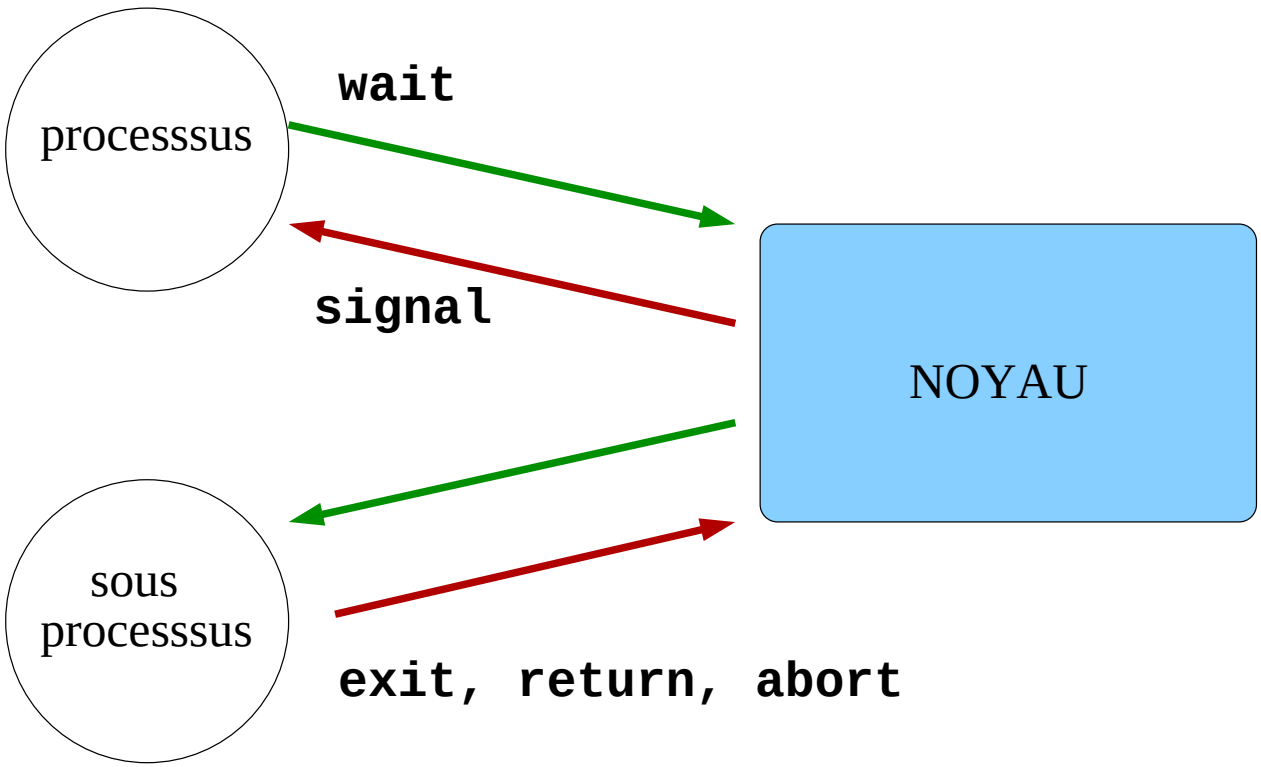
Communication interprocessus par tube



Tube de communication, pipe



Signal SIGCHILD, zombie



Nouveau client

```
void calcul( int sd, struct sockaddr *serv)
{ int  val, retry ;
  struct sockaddr slave;
  while ( 1 ) {
      sendpaquetto( sd, serv, 0);
      retry = 2;
      do {
          val = nobloqpaquetfrom( sd, &slave);
          if ( val < 0) {
            printf("\nPas de reponse %d %s", retry, name)
            sleep(1);
          }
          retry--;
      } while (( val < 0 ) && retry );
      if ( val <= 0 ) return;
      printf("\njob = %d %s", val, name);
      val = val * val;  sleep(5);
      sendpaquetto( sd, &slave, val);
  }
}
```

Code du serveur fork

```
int sdcnet( int sd, int n )
{ int  somme = 0, val, nb, status;
  struct sockaddr  aux;
  int pid, tube[2];
  if ( pipe( tube ) < 0) erreur("pipe");
  while (  n  ) {
    val = recvpaquetfrom( sd, &aux );
    pid = fork();
    if ( pid == 0 ) {
          close( tube[0] );
          tache( tube[1], &aux, n);
          close( tube[1] ); close(sd); exit( 0 );
        };
     n--;
    }
   while ( ( pid = wait( &status ) ) >= 0 ) {
    fprintf(stdout, "\nsignal du fils pid = %d", pid);
    nb = read( tube[0], &val, 4 );
    if ( nb != 4 )  erreur("read");
    somme += val;
```

Tache du fork serveur

```
int tache( int dt, struct sockaddr *client, int n )
{ int sd, res, nb;
  sd = socket( AF_INET, SOCK_DGRAM, 0);
  sendpaquetto( sd,  client, n);
  res = recvpaquetfrom( sd, client );
  fprintf(stdout, "\nfils %d  <- %d", getpid(), res);
  printfrom( client );
  nb = write( dt, &res, 4 );
  if ( nb != 4 ) erreur("write");
  close(sd);
  return res;
```

Output

```
[drmichko@msnet SOCKET]$ ./sdcservforktube.exe -p 31415

fils 4051  <- 25 ( 192.168.0.30 )
fils 4052  <- 16 ( 192.168.0.30 )
fils 4054  <- 9 ( 192.168.0.30 )
fils 4056  <- 4 ( 192.168.0.30 )
signal du fils pid = 4051
signal du fils pid = 4052
signal du fils pid = 4054
signal du fils pid = 4056
fils 4057  <- 1 ( 192.168.0.30 )
signal du fils pid = 4057
Somme 5 premiers carres : 55
```

[root@ou812]tcpdump -n host ou812 and udp

```
16:56:57 IP 10.2.73.127.32806 > 10.2.73.86.31415: UDP, l
16:56:57 IP 10.2.73.86.33074 > 10.2.73.127.32806: UDP, l
16:56:57 IP 10.2.81.42.32917 > 10.2.73.86.31415: UDP, le
16:56:57 IP 10.2.73.86.33075 > 10.2.81.42.32917: UDP, le
16:57:03 IP 10.2.73.127.32806 > 10.2.73.86.33074: UDP, l
16:57:03 IP 10.2.73.127.32806 > 10.2.73.86.31415: UDP, l
16:57:03 IP 10.2.73.86.33076 > 10.2.73.127.32806: UDP, l
16:57:03 IP 10.2.81.42.32917 > 10.2.73.86.33075: UDP, le
16:57:03 IP 10.2.81.42.32917 > 10.2.73.86.31415: UDP, le
16:57:03 IP 10.2.73.86.33077 > 10.2.81.42.32917: UDP, le
16:57:09 IP 10.2.73.127.32806 > 10.2.73.86.33076: UDP, l
16:57:09 IP 10.2.73.127.32806 > 10.2.73.86.31415: UDP, l
16:57:09 IP 10.2.73.86.33078 > 10.2.73.127.32806: UDP, l
16:57:09 IP 10.2.81.42.32917 > 10.2.73.86.33077: UDP, le
16:57:09 IP 10.2.81.42.32917 > 10.2.73.86.31415: UDP, le
16:57:14 IP 10.2.73.86.33080 > 10.1.65.1.domain:  56074+
cache.univ-tln.fr IP(35)
16:57:14 IP 10.1.65.1.domain > 10.2.73.86.33080:  56074*
16:57:14 IP 10.2.73.86.33080 > 10.1.65.1.domain:  12385+
cache.univ-tln.fr.univ-tln.fr IP(47)
```

tcpdump -n -X -t host ou812 and udp and not port 53

```
listening on eth0, link-type EN10MB (Ethernet), capture
IP 10.2.73.127.32806 > 10.2.73.86.31415: UDP, length 4
0x0000:  4500 0020 0000 4000 4011 93f4 0a02 497f  E.....
0x0010:  0a02 4956 8026 7ab7 000c 5e1f 0000 0000  ..IV.&
0x0020:  0000 0000 0000 0000 0000 0000 0000       ......
IP 10.2.73.86.33093 > 10.2.73.127.32806: UDP, length 4
0x0000:  4500 0020 0000 4000 4011 93f4 0a02 4956  E.....
0x0010:  0a02 497f 8145 8026 000c 5391 0400 0000  ..I..E
IP 10.2.81.42.32917 > 10.2.73.86.31415: UDP, length 4
0x0000:  4500 0020 0000 4000 3f11 8d49 0a02 512a  E.....
0x0010:  0a02 4956 8095 7ab7 000c 5605 0000 0000  ..IV..
0x0020:  0000 0000 0000 0000 0000 0000 0000       ......
IP 10.2.73.86.33094 > 10.2.81.42.32917: UDP, length 4
0x0000:  4500 0020 0000 4000 4011 8c49 0a02 4956  E.....
0x0010:  0a02 512a 8146 8095 000c 4c76 0300 0000  ..Q*.F
IP 10.2.73.127.32806 > 10.2.73.86.33093: UDP, length 4
0x0000:  4500 0020 0001 4000 4011 93f3 0a02 497f  E.....
0x0010:  0a02 4956 8026 8145 000c 4791 1000 0000  ..IV.&
0x0020:  0000 0000 0000 0000 0000 0000 0000       ......
IP 10.2.73.127.32806 > 10.2.73.86.31415: UDP, length 4
```

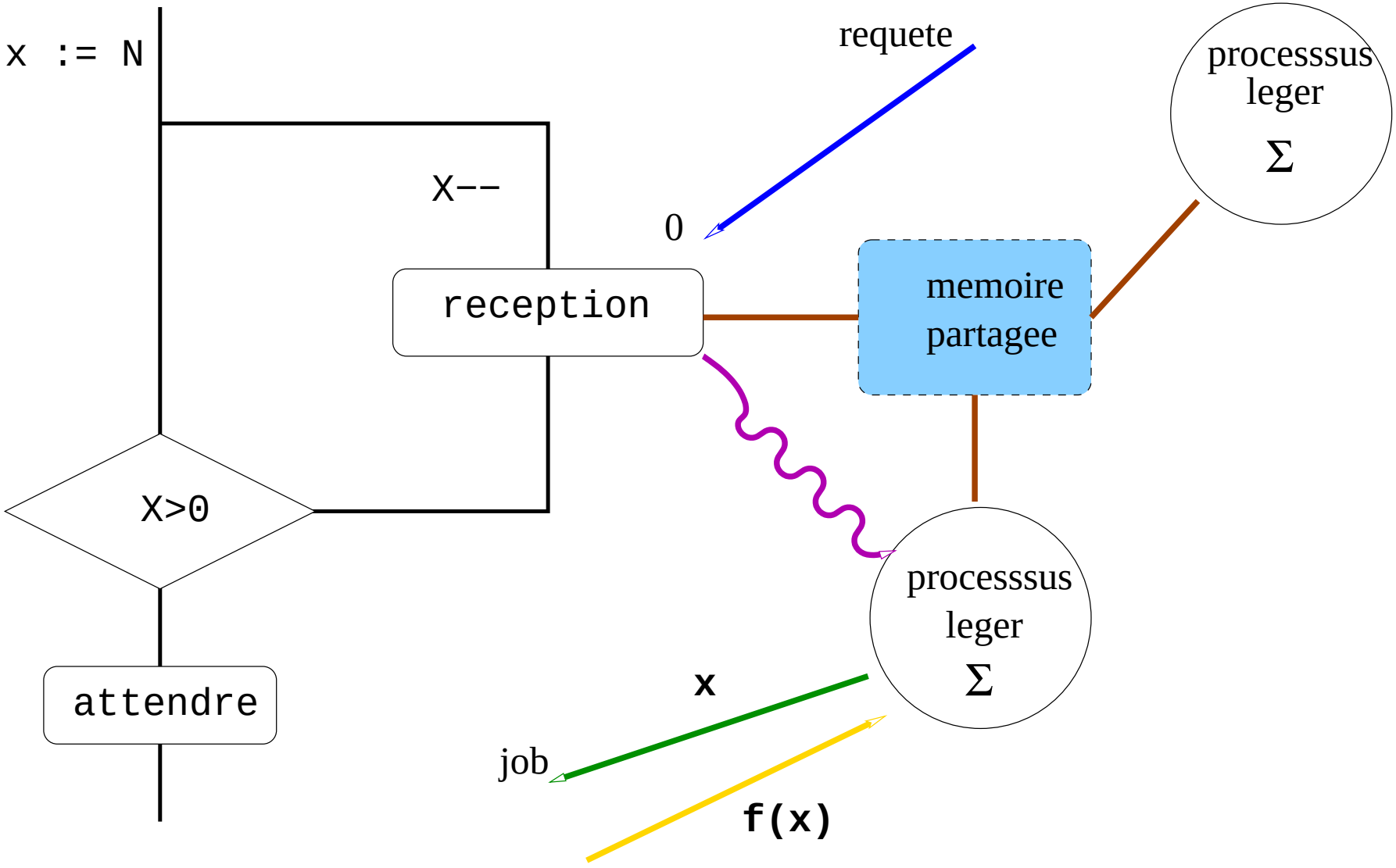
Code du serveur fork basé sur code de retour

```
int sdcnet( int sd, int n )
{ int  somme = 0,  val, pid, status;
  struct sockaddr  aux;

  while (  n  ) {
    val = recvpaquetfrom( sd, &aux );
    pid = fork();
    if ( pid == 0 ) {
          val = tache( &aux, n);
          close(sd);
          exit( val );
        };
     n--;
    }

  while ( ( pid = wait( &status ) ) > 0 ) {
   fprintf(stdout, "\nsignl du fils pid : %d", pid);
   somme += WEXITSTATUS(status);
}
```

Tache du fork serveur basé sur code de retour

```
int tache( struct sockaddr *client , int n )
{ int sd, res;
  sd = socket( AF_INET, SOCK_DGRAM, 0);
  sendpaquetto( sd,  client , n);
  res = recvpaquetfrom( sd, client );
  printfrom( client );
  fprintf(stdout, " dans fils %d  <- %d", getpid(), res)
  close(sd);
  return res;
}
```

Utilisation de la mémoire partagée



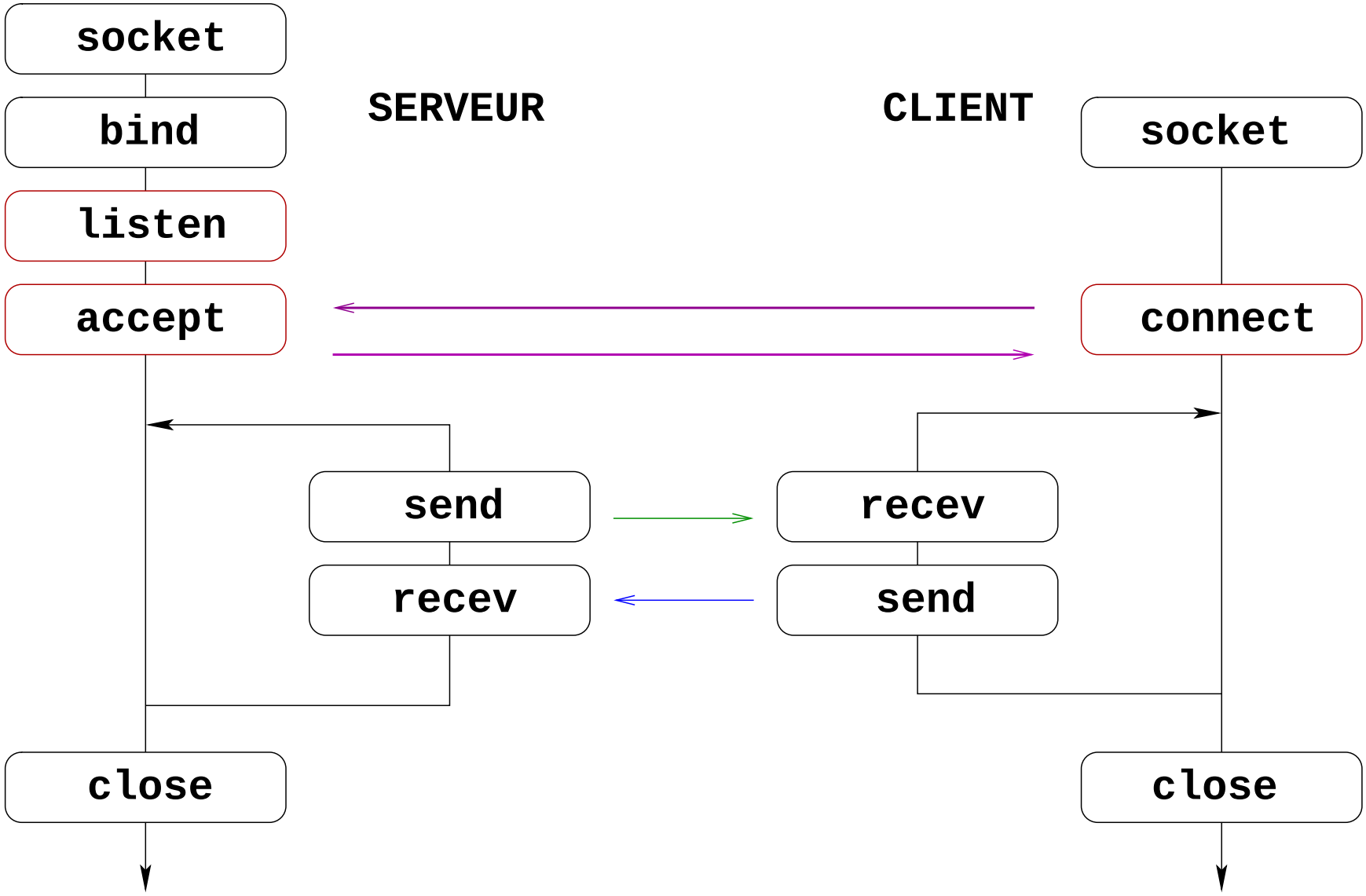
Tache du thread serveur

```
void* tache( void * parms)
{ int sd, res;
  argtask arg;
  arg = *( ( argtask *) parms );
  sd = socket( AF_INET, SOCK_DGRAM, 0);
  printfrom( &arg.client );
  printf(" thread  sub->%d", arg.val);
  fflush(stdout);
  sendpaquetto( sd,  & arg.client, arg.val);
  res = recvpaquetfrom( sd, & arg.client );
  pthread_mutex_lock( & mymutex );
  somme += res;
  nbproc--;
  pthread_mutex_unlock( & mymutex );
  pthread_exit( NULL );
}
```

Thread serveur

```
void sdcnet( int sd, int n )
{ int ret, val;
  struct task arg;
  while ( n ) {
    val = recvpaquetfrom( sd, & arg.client );
    arg.val = n;
    pthread_mutex_lock( & mymutex );
    nbproc++;
    printf("\nval=%d actif=%d", n, nbproc);
    pthread_mutex_unlock( & mymutex );
    newthread();
    ret = pthread_create( & (actifs->t) , NULL, tache, (
    if ( ret ) erreur("thread");
    n = n - 1;
  }
  while ( actifs ) {
    pthread_join( actifs->t , NULL);
    delthread();
  }
}
```

Mode connecté



Modèle de code serveur TCP

```
int master, slave;
struc sockaddr_in serv;
socklen_t longueur;

master = socket( AF_INET, SOCK_STREAM, 0);
if ( master < 0 ) erreur("socket");

serv = sockaddrinlocal( PORT );

if ( bind( master, (struct sockaddr *) &serv, sizeof(ser
     erreur("bind");

listen( master, 5);

longueur = sizeof( struct addr_in );
```

Boucle du serveur TCP

```
while ( 1 ) {
  slave = accept( master, & adresse, &longueur);
  if ( slave < 0 )
    erreur("accept");

  switch( fork () ) {
    case 0 :
      close( master );
      traitement( slave );
      exit(0);
    case -1 :
      erreur( "fork");
    default :
      close( slave );
  }

  return 0;
}
```

Modèle de code client TCP

```
int   sock;
struc sockaddr_in   serv;
socklen_t longueur;

sock = socket( AF_INET, SOCK_STREAM, 0);
if ( sock < 0 ) erreur("socket");

serv = sockaddrinbyname( PORT , SERV);

if ( connect( sock, (struct sockaddr *) &serv, sizeof(se
     erreur("connect");
while ( 1 ) {
  nb = read( sock, bfr, MAX);
  if ( nb < 0 ) erreur("read");
  traitement();
  nb = write( sock, bfr, strlen(bfr)+1);
  if ( nb < 0 ) erreur("write");
  }
```

Liens vers les sources

- socktools.h socktools.c sock.tar
- sdclient.c sdclientfork.c sdcserv.c sdcservforkexit.c sdcservforktube.c sdcservthread.c udpservmin.c whoami.c
- install.sh start.sh status.sh urls.sh
- makefile Makefile
- socket.tex urls.tex
- sdc.fig sdcfork.fig sdcproc.fig sdcthread.fig tcp.fig tube.fig udp.fig udpfork.fig zombie.fig