Quelques Exemples de Programmation Réseau

Philippe Langevin

Octobre 2007.

Motivation

L'objectif de ces notes est de présenter la communication interprocessus sur un réseau en utilisant les outils *socket BSD*. Nous implanterons *à la main* une architecture *client/serveur* pour réaliser des grandes boucles :

```
for( i = 0; i < n; i++ )
    calcul( i );
```

Typiquement, si $n = 2^{40}$ et si le temps de calcul pour chaque itération est de l'ordre de 2^{20} cycles, sur une machine usuelle le temps d'exécution de cette boucle vaut :

$$2^{40} \times 2^{20} \times 2^{-31}\,\text{sec} = 6213\,\text{jours}$$

En *distribuant* les calculs sur 256 processeurs, le temps de réalisation de l'*expérience numérique* ne dépassera pas 4 jours !

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.
- Le chapitre VII du BSD-Handbook developer. socket bsd

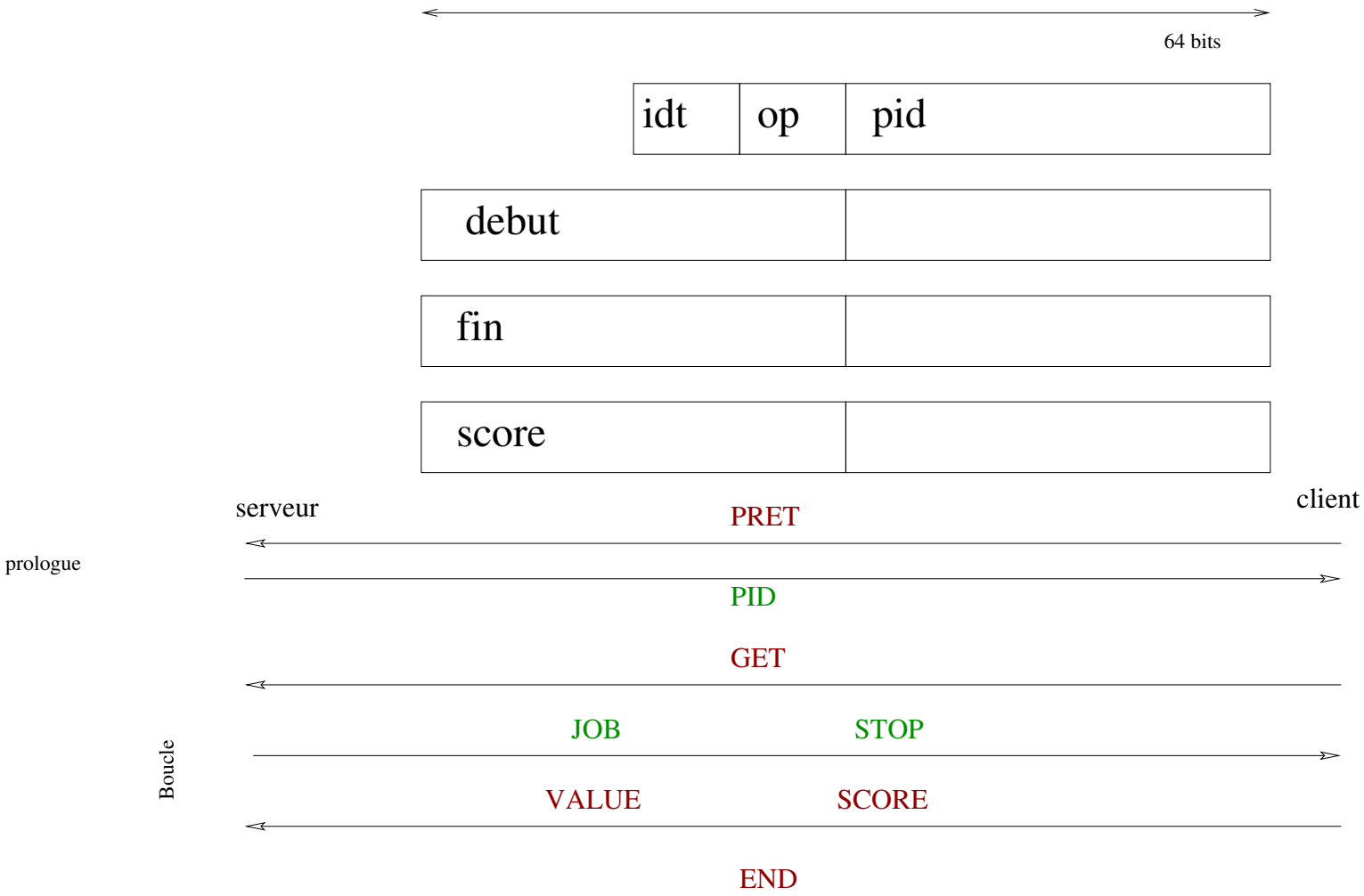
Connexion

Les sockets (*prises*) s'appuient sur les deux principaux protocoles de transports TCP/IP :

- UDP: application orientée sans connexion.
- TCP: application orientée connexion.

Le protocole UDP est particulièrement bien adapté à notre but, échange de petits paquets d'information entre les processus clients et un processus serveur.

Protocole BigLoop



Architectures

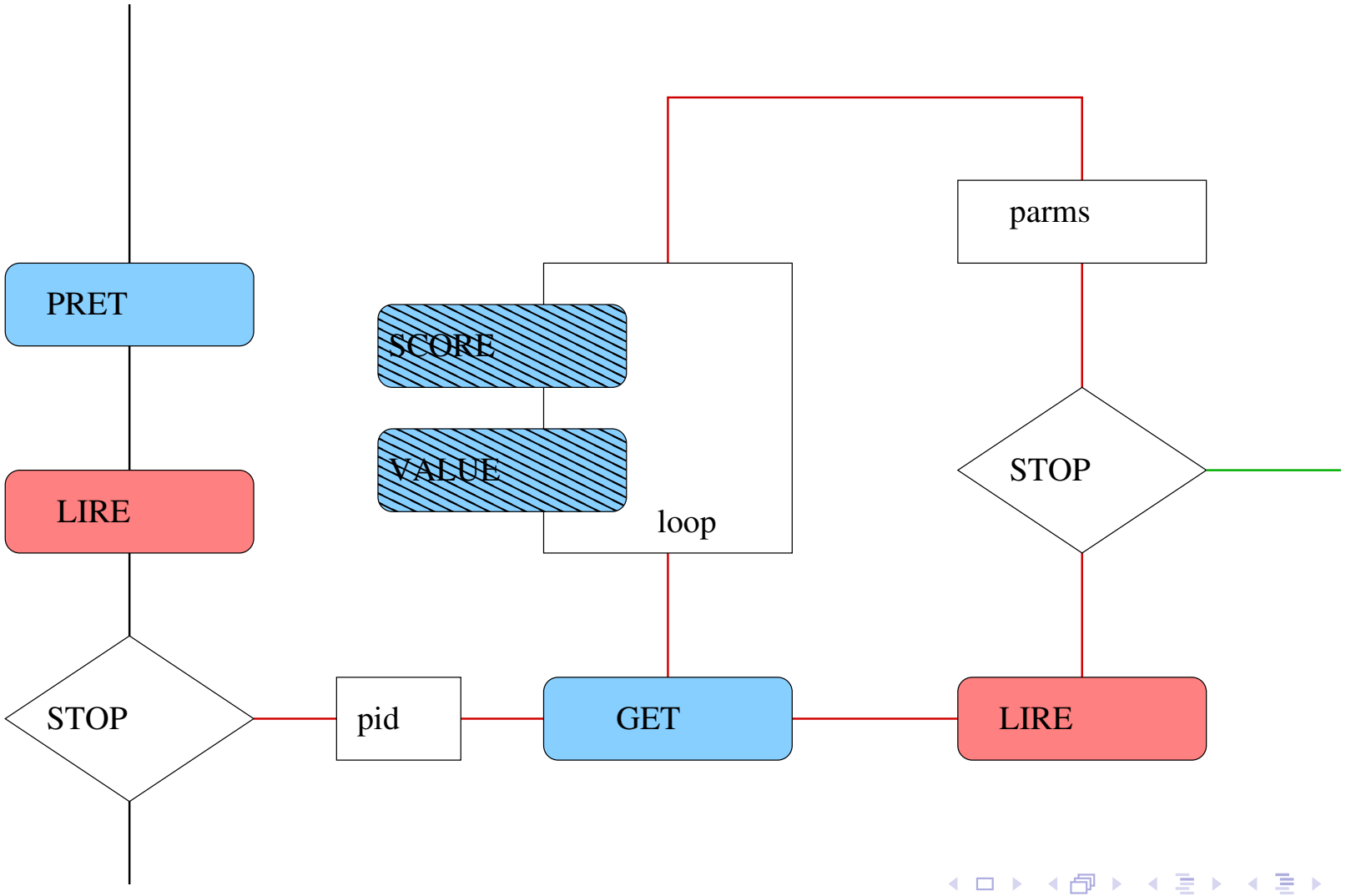
```
32 model name          : Intel(R) Xeon(R) CPU 5150 @ 2.66GHz
12          : Itanium 2          @ 1.50GHz gitane
4 model name : Intel(R) Xeon(TM) CPU 3.00GHz     tcan
                              arcadia
32 model name          : Intel(R) Core(TM)2 Duo CPU    E6750
32 model name          : Intel(R) Core(TM)2 Duo CPU    E7400
1 model name : Intel(R) Pentium(R) 4 CPU 1500MHz
8 model name : Intel(R) Pentium(R) 4 CPU 2.00GHz
6 model name : Intel(R) Pentium(R) 4 CPU 2.40GHz
11 model name          : Intel(R) Pentium(R) 4 CPU 2.66GHz
8 model name : Intel(R) Pentium(R) 4 CPU 3.00GHz
4 model name : Intel(R) Xeon(R) CPU 5110 @ 1.60GHz waveste
30 model name          : Pentium(R) Dual-Core CPU    E5200 @
```

Structure

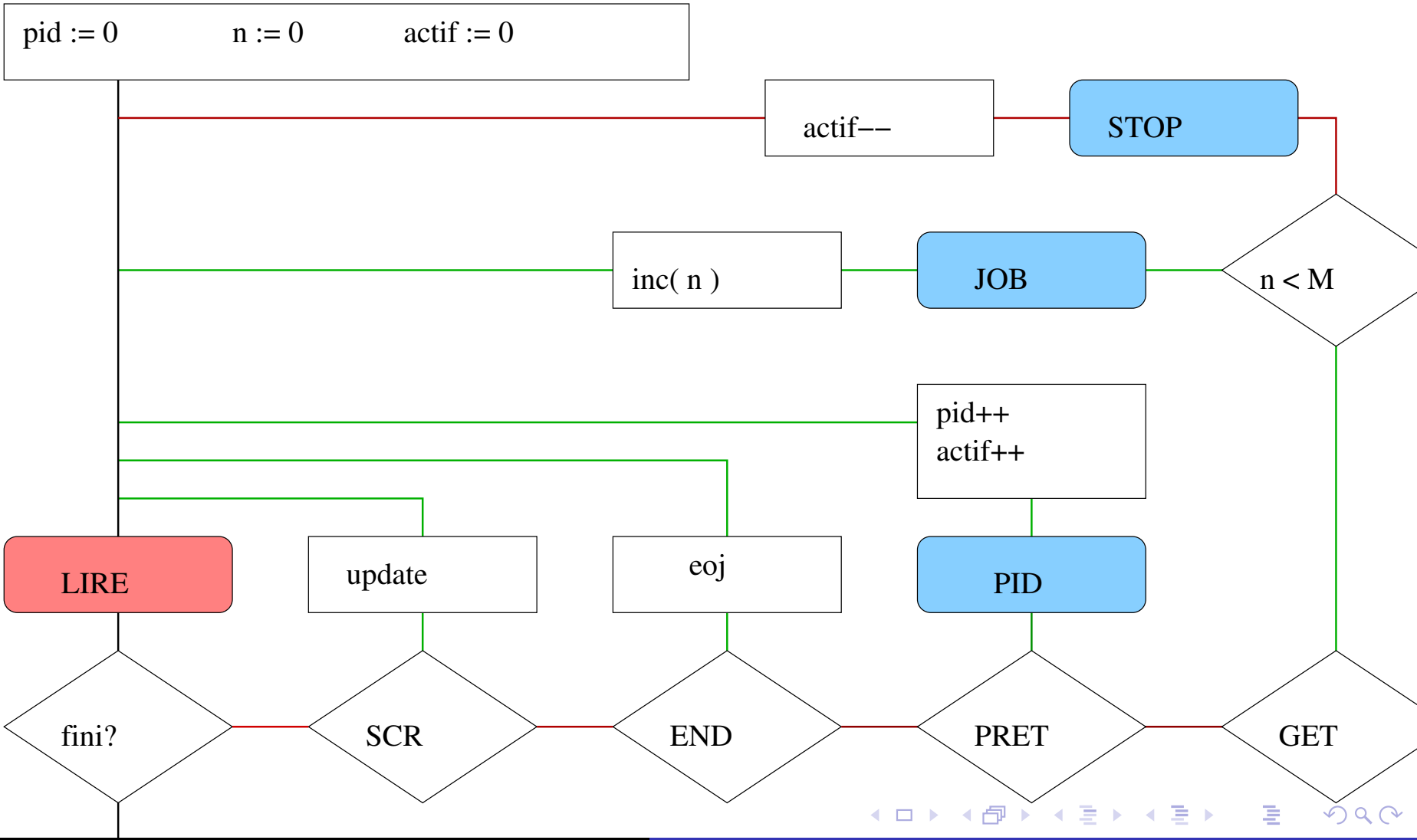
```
struct msg {
  unsigned char idt;
  unsigned char op;
  unsigned int pid;
  ullong deb;
  ullong fin;
  long long int scr;
} __attribute__ ((__packed__));

typedef struct msg ticket;
```

Boucle du client



Boucle du serveur



Application : Nombre Parfait

Un nombre entier z est dit *parfait* quand il est égal à la somme de ses diviseurs stricts

$$z = \sum_{z \neq d | z} d$$

En lançant,

- `server.exe -i1 -p31415 -f1 -L30 -S25`
- `client.exe -i1 -p31415 -a adresse-serveur`

sur la grappe de machines, on détermine les nombres parfaits inférieurs à 2^{30} en quelques minutes.

client.c: Parfait

```
int parfait( ullong z , ullong *cpt)
{ ullong d;
  ullong sum = 1;
  for( d = 2; d*d <=z; d++ )
    if ( z % d == 0 ) {
        *cpt = *cpt + 1;
         sum+= d;
         if ( d*d != z ) s+= z/d;
         if ( sum > z ) return 0;
    }
  return ( sum == z );
}
```

- source : client.c

client.c: main

```
int main( int argc, char*argv[])
{ ullong deb, fin, wf = 0;
  if ( ! bigloopargs( argc, argv ) )
       exit(1);
  initbigloop();
  if ( registration( ) )
    while ( newjob( ) ){
      wf = 0;
      for( s = current.deb ; s < current.fin; s++ )
        if ( parfait( s, &wf) ) sendvalue( s );
      sendend( wf );
    }
  return 0;
}
```

server.c: main (1/3)

```
int main( int argc, char*argv[])
{
  double perf;
  ullong s, f;
  long long score = 0;
  if ( ! bigloopargs( argc, argv ) )
       exit(1);
  initbigloop();
  initserver();
  bigloopparms();
  initproc();
  s = FIRST;
```

server.c: main (2/3)

```
while ( s < LAST || actif ){
  if ( getticketfromclient( &current ) ){
    pticket( current );  fflush(stdout);
    switch ( current.op ) {
      case READY :
        openproc(); sendpid( proc ); proc++; break;
       case GET   :
        if ( s < LAST ) {
          f = s + STEP;  if ( f > LAST) f = LAST;
          initjob(s, f );
          sendjob(s, f, score );
          s += STEP;
        } else { sendstop( ); actif--; }    break;
      case VALUE :
        savevalue(  ); break;
      case END :
        savejob(  );  break;
```

server.c: main (3/3)

```
runtime = difftime( time(NULL), initial );
printf("\nrunning time   : %d", runtime );
printf("\ncpu      time   : %d", cputime );
printf("\njob count       : %d", jobcount);
printf("\nprocessus       : %d", proc );
printf("\nwork factor     : %Ld", work );
perf = work; perf /= cputime;
printf("\n%E operations per seconde", perf);
report();
printf("\nwaiting for zombies...");
fflush(stdout);
while ( getticketfromzombie( &current ) ) {
  pticket( current );
  sendstop( );
}
printf("\neoj\n");
stopserver();
```

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
hote :msnet.sollies.fr
addr :16777343
ip :127.0.0.1
```

```
./whoami.exe mail.univ-tln.fr
hote :mail.univ-tln.fr
addr :39858625
ip :193.49.96.2
```

- Pour une communication socket l'adresse IP doit complétée par un numéro de *port* de 16 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");
```

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

Boutisme réseau

Le boutisme réseau est *big indian* comme sur les architectures motorola 68000, sparc et système IBM/370, mais ce n'est pas forcément pas le point de vue de votre processeur !

$$1 + 2 * 256 + 3 * 256^2 + 3 * 256^3$$

```
memory  byte order: 1 2 3 4
network byte order: 4 3 2 1
```

Pour paramétrer les fonctionnalités du réseau, il convient d'utiliser le bon boutisme !

Indianness test

```c
#include <stdio.h>
#include <arpa/inet.h>


int main(int argc, char *argv[])
{ int   i, z;
  char *ptr;
  z = 1 + (2<<8) + (3<<16) + (4<<24);
  ptr = (char*) &z;
  printf("\nmemory  byte order:");
  for( i = 0; i < 4; i++ , ptr++)
    printf(" %c", *ptr + '0');
  z = htonl( z );
  ptr = (char*) &z;
  printf("\nnetwork byte order:");
  for( i = 0; i < 4; i++ , ptr++)
    printf(" %c", *ptr + '0');
  return 0;
```

Conversion des entiers

```
#include        <arpa/inet.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) ;
```

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

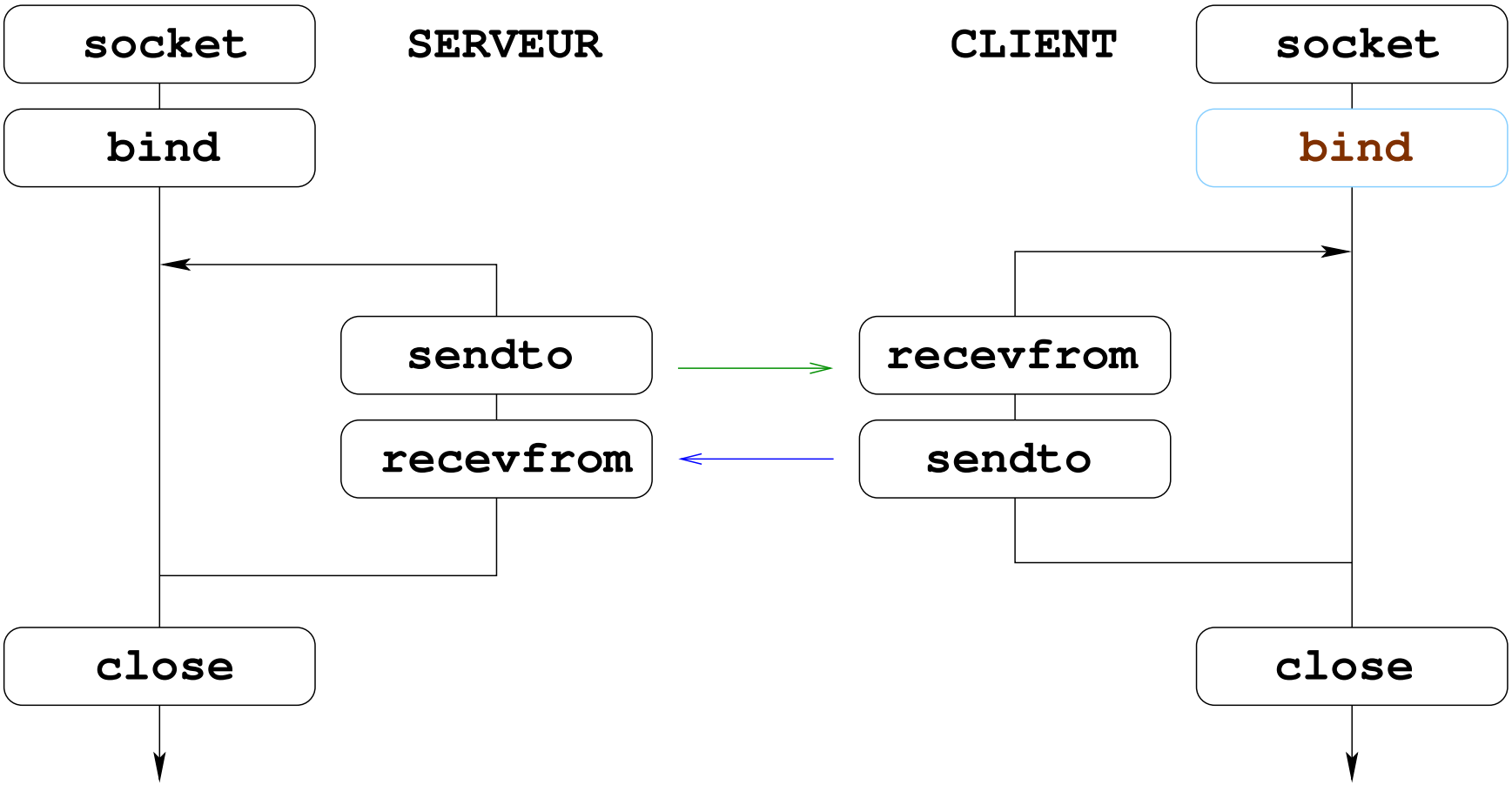
Manipulation des adresses

```
#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 notation 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 contient le code d'erreur.

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 normalement les données disponibles sans attendre d'avoir reçu 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

Serveur en mode non connecté

```
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);
```

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 refused
telnet: Unable to connect to remote host: Connection refused

[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
```

bigloop.c: socket server

```
void initsocketserver( void )
{ memset( & SERVEUR, 0 , sizeof( SERVEUR ) );
  SERVEUR.sin_family = AF_INET;
  SERVEUR.sin_port = htons( PORT );
  SERVEUR.sin_addr = ADDRSERV;
}
void initserver( void )
{
  sock_server  = socket( AF_INET, SOCK_DGRAM, 0);
  if ( bind( sock_server, (struct sockaddr *) &SERVEUR, sizeo
      perror("intit socket");
      exit(1);
    }
}
```

bigloop.c: recevoir un ticket

```
int getticketfromclient( ticket *p )
{ int nb;
  uint len = sizeof( struct sockaddr_in );
  nb = recvfrom( sock_server, p, sizeof( ticket )  , 0, (stru
  if ( nb <= 0 ) {
    perror("getticketfromclient");
    return 0;
  }
  if ( p->idt != IDENT ){
    pticket( *p );
    printf("\nbad ident (%d) :  ignored", p->idt );
    return 0;
  }
  return 1;
}
```

bigloop.c: envoyer un ticket

```
int sndticketoserver( ticket p )
{
  int nb;
  socklen_t len = sizeof( SERVEUR );
  int sock = socket( AF_INET, SOCK_DGRAM, 0);
  nb = sendto( sock, &p, sizeof( ticket ) , 0, (struct sockad
  close(sock);
  if ( nb < 0 ) {
    perror("sendticketoserver");
    return 0;
  }
  return 1;
}
```

bigloop.c: lecture non bloquante

```
int getticketfromzombie( ticket *p )
{ int retry = 2;
  int nb;
  uint len = sizeof( struct sockaddr_in );
  while ( retry-- ){
    nb = recvfrom( sock_server, p, sizeof( ticket )   , MSG_DO
    if ( nb <= 0 ) {
      if ( errno != EAGAIN )
        perror("zombie");
      else sleep( 10 );
    }
  }
  return ( nb > 0);
}
```

Grappe de machines

```
maitinfo1.univ-tln.fr  langevin
wavester.univ-tln.fr   langevin
gitane.univ-tln.fr     grim
tcan.univ-tln.fr       langevin
```

Hosts file

```
# Do not remove the following line, or various programs
# that require network functionality will fail.
127.0.0.1               localhost.localdomain localhost ou812
::1             localhost6.localdomain6 localhost6
10.2.81.2 wavester.univ-tln.fr  wavester
10.2.81.42 gitane.univ-tln.fr  gitane
10.9.185.217 maitinfo1.univ-tln.fr  maitinfo1
10.9.185.218 maitinfo2.univ-tln.fr  maitinfo2
10.9.185.219 maitinfo3.univ-tln.fr  maitinfo3
10.9.185.220 maitinfo4.univ-tln.fr  maitinfo4
10.9.185.221 maitinfo5.univ-tln.fr  maitinfo5
10.9.185.222 maitinfo6.univ-tln.fr  maitinfo6
10.9.185.223 maitinfo7.univ-tln.fr  maitinfo7
10.9.185.224 maitinfo8.univ-tln.fr  maitinfo8
10.9.185.225 maitinfo9.univ-tln.fr  maitinfo9
10.9.185.226 maitinfo10.univ-tln.fr  maitinfo10
10.9.185.227 maitinfo11.univ-tln.fr  maitinfo11
10.9.185.228 maitinfo12.univ-tln.fr  maitinfo12
```

Synchronisation des codes

```
GRAPPE=master.grp ; DIR=$(basename $PWD)
rm -f ko-$$.grp
touch  ko-$$.grp
while read host login max ligne
do
 if ping -c1 -w2 $host >/dev/null
   then
     rsync -av bigloop.conf  --delete $login@$host:$DIR/
     rsync -av *.c       --delete $login@$host:$DIR/
     rsync -av makefile    --delete $login@$host:$DIR/
     rsync -av runwavester.sh --delete $login@$host:$DIR/
     ssh  $login@$host "cd $DIR;make" < /dev/null
  else
    echo "$host $login $max" >> ko-$$.grp
 fi
done < $GRAPPE
mv ko-$$.grp sync.grp
```

Lancement du serveur

```
SERVEUR=./server.exe
CLIENT=./client.exe
DIR=$( basename $PWD )
echo "running bigloop on $DIR"
echo "starting server"
rm -f  nohup.out
nohup  $SERVEUR &
echo  "server started"
```

Lancement des clients

```
GRAPPE=$1; DIR=$( basename $PWD )
SERVEUR=./server.exe ;CLIENT=./client.exe ;
while read host login max ligne
do
  if ping -c1  -w5 $host  > /dev/null
   then
    while [ $max != 0 ]
        do
         echo -e "starting client $max on $host..."
         let max=$max-1
      if [ $host = "wavester" ]
       then
       ssh  $login@$host "cd $DIR; qsub runwavester.sh" < /de
       else
       ssh  $login@$host "cd $DIR; nice $CLIENT" < /dev/null
      fi
 done
```

Surveiller la marmite

```
grappe=$1
cmd=$2
rm -f ko-$$.grp
touch ko-$$.grp
cat $grappe | while read machine compte max ligne
do

   ping -c1 -w2 $machine >/dev/null
   ALIVE=$?
   if [ "$ALIVE" != "0" ]
         then
          echo $machine:unreachable
          echo "$machine $login $max" >> ko-$$.grp
         else
          echo "$machine"
          ssh -l $compte $machine "who" < /dev/null
          ssh -l $compte $machine "ps aux | grep $2 | grep -v
```

Log des échanges

```
using server : 10.2.73.86-31415
identificator: 1
usually it takes a while...
RDY 1:0 0 0 0 :  10.9.185.217:57821
GET 1:0 0 0 0 :  10.9.185.217:29834
VAL 1:0 0 1048576 1 :  10.9.185.217:61577
VAL 1:0 0 1048576 6 :  10.9.185.217:17370
VAL 1:0 0 1048576 28 :  10.9.185.217:11232
VAL 1:0 0 1048576 496 :  10.9.185.217:50142
VAL 1:0 0 1048576 8128 :  10.9.185.217:34753
RDY 1:0 0 0 0 :  10.9.185.217:40351
GET 1:1 0 0 0 :  10.9.185.217:34484
RDY 1:0 0 0 0 :  10.9.185.218:62373
GET 1:2 0 0 0 :  10.9.185.218:39652
RDY 1:0 0 0 0 :  10.9.185.218:49046
GET 1:3 0 0 0 :  10.9.185.218:45738
RDY 1:0 0 0 0 :  10.9.185.219:14539
GET 1:4 0 0 0 :  10.9.185.219:12506
```

Trace des communications

```
[root@ou812 ~]# tcpdump -n -i any udp port 31415 -t
tcpdump: WARNING: Promiscuous mode not supported on the "any"
tcpdump: verbose output suppressed, use -v or -vv for full p
listening on any, link-type LINUX_SLL (Linux cooked), capture
IP 10.9.185.217.45403 > 10.2.73.86.31415: UDP, length 30
IP 10.2.73.86.31415 > 10.9.185.217.45403: UDP, length 30
IP 10.9.185.217.42778 > 10.2.73.86.31415: UDP, length 30
IP 10.2.73.86.31415 > 10.9.185.217.42778: UDP, length 30
IP 10.9.185.217.48978 > 10.2.73.86.31415: UDP, length 30
IP 10.9.185.217.34333 > 10.2.73.86.31415: UDP, length 30
IP 10.9.185.217.44033 > 10.2.73.86.31415: UDP, length 30
IP 10.9.185.217.51550 > 10.2.73.86.31415: UDP, length 30
IP 10.9.185.217.55578 > 10.2.73.86.31415: UDP, length 30
IP 10.9.185.217.37268 > 10.2.73.86.31415: UDP, length 30
IP 10.2.73.86.31415 > 10.9.185.217.37268: UDP, length 30
IP 10.9.185.217.38770 > 10.2.73.86.31415: UDP, length 30
IP 10.2.73.86.31415 > 10.9.185.217.38770: UDP, length 30
IP 10.9.185.220.57911 > 10.2.73.86.31415: UDP, length 30
```

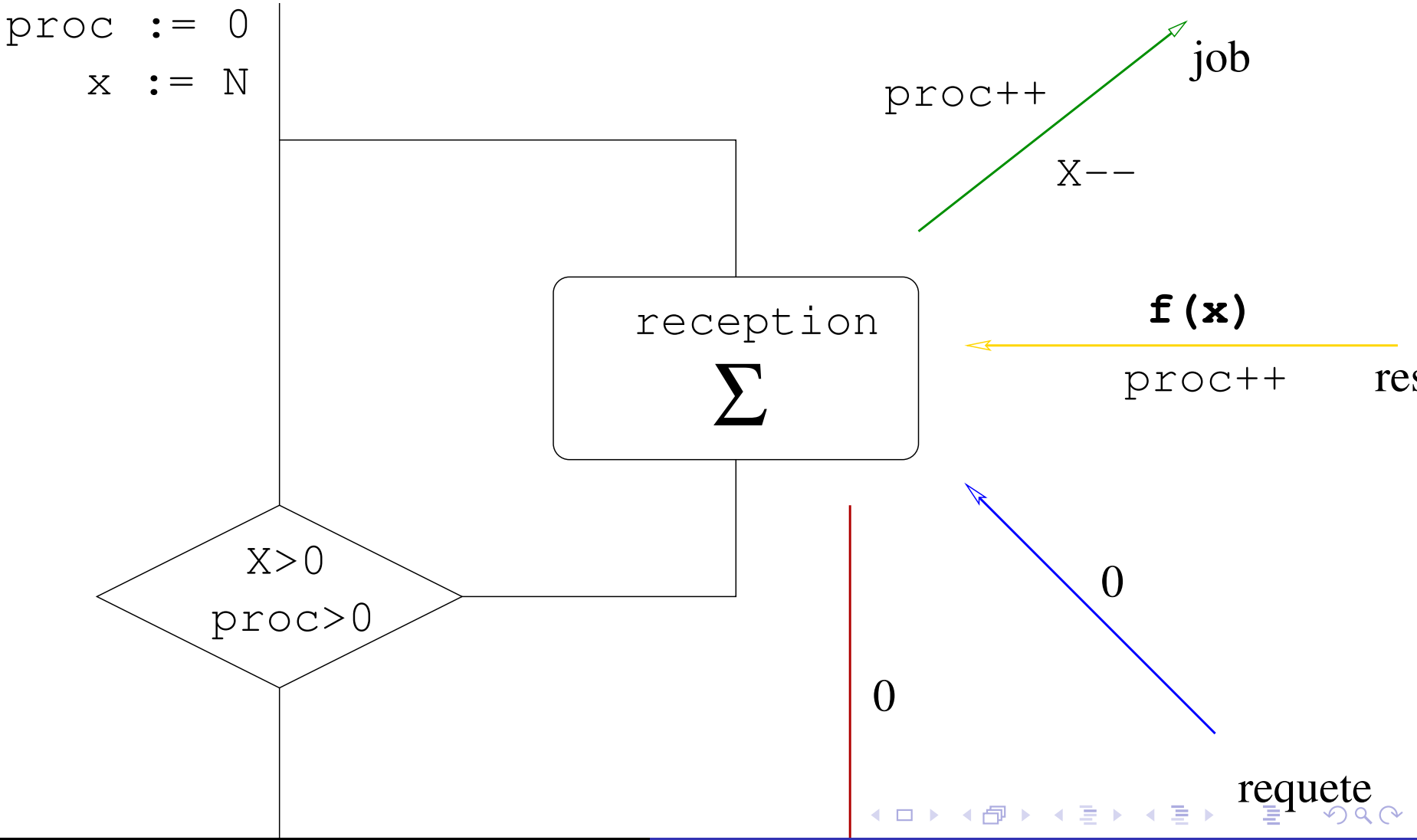
Valeurs collectées

```
value=1
value=6
value=28
value=496
value=8128
value=33550336
```

Facteur de travail

```
running time   : 2939
cpu      time  : 302689
job count      : 1024
processors     : 147
work factor    : 5075000390 -> 1.676639E+04 w/sec
#performance              host      job
28901.938      10.9.185.217        11  maitinfo1.univ-tln.fr
28775.199      10.9.185.218        11  maitinfo2.univ-tln.fr
28664.283      10.9.185.218        11  maitinfo2.univ-tln.fr
28639.389      10.9.185.217        11  maitinfo1.univ-tln.fr
28550.486      10.9.185.219        11  maitinfo3.univ-tln.fr
28543.299      10.9.185.219        11  maitinfo3.univ-tln.fr
28526.621      10.9.185.220        11  maitinfo4.univ-tln.fr
28399.588      10.9.185.220        11  maitinfo4.univ-tln.fr
28335.178      10.9.185.221        11  maitinfo5.univ-tln.fr
28324.965      10.9.185.221        11  maitinfo5.univ-tln.fr
28175.230      10.9.185.225        11  maitinfo9.univ-tln.fr
28130.512      10.9.185.225        11  maitinfo9.univ-tln.fr
```

Mini client/serveur



option : sdcserveur

```
int argok( int argc, char* argv[])
{ int opt;
  char * optliste = "p:n:s:h";
  while (( opt = getopt( argc, argv, optliste )) >= 0 ) {
      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", argv[0]
                 printf("\nusage %s: -s serveur\n", argv[0]);
      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, res);
  }
  sendpaquetto( sd, &client, 0);
```

paquet : socktools.c

```
void sendpaquetto( int sd,  struct sockaddr *client, int n)
{ int val = n;
  int nb;
  nb = sendto( sd, &val, 4 , 0, client, TAILLE);
  if ( nb < 0 ) erreur("sendto");
}
```

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;
}
```

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 );
  }
}
```

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

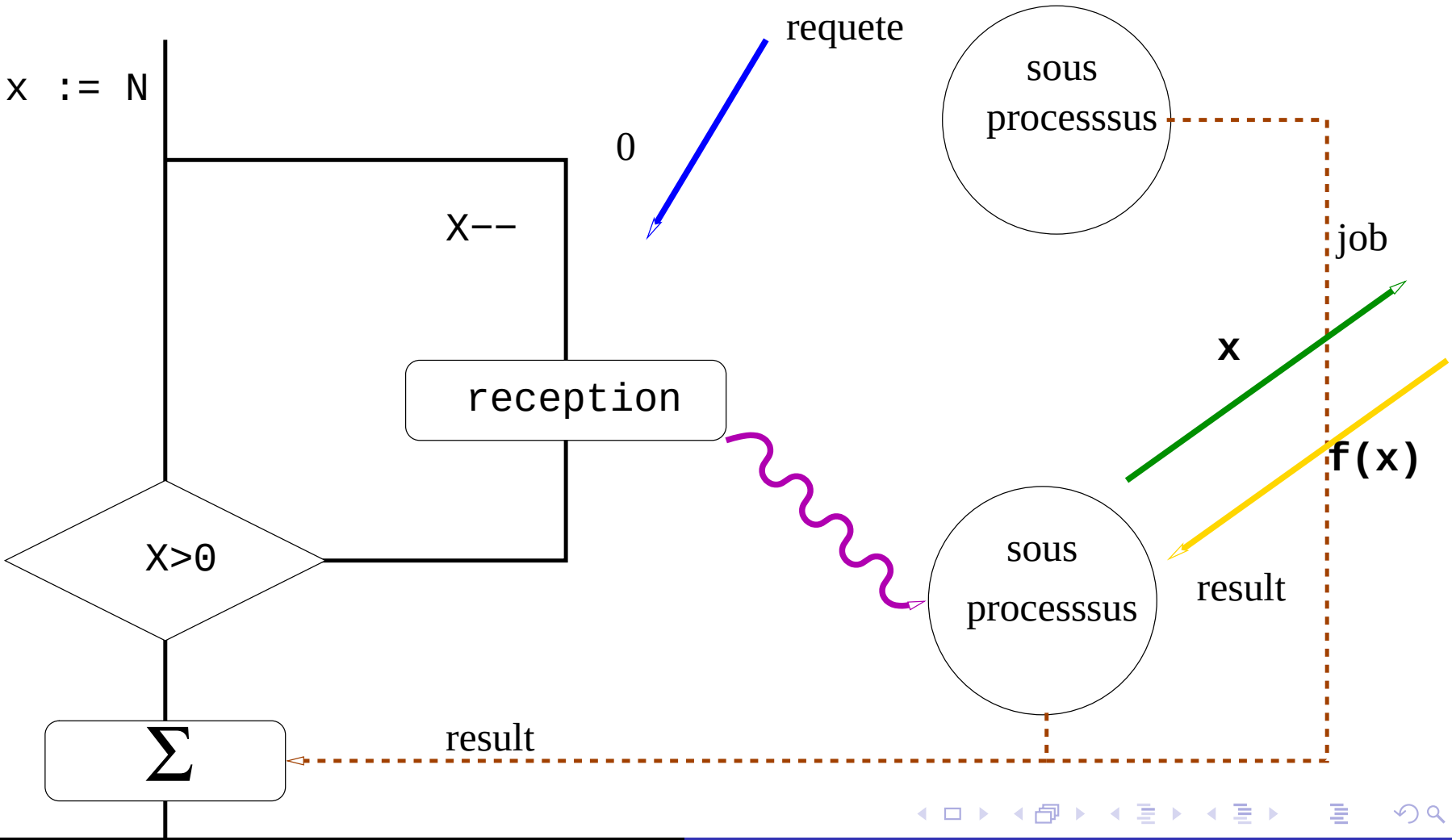
```
[root@ou812 ~]# tcpdump -n port 3000
tcpdump: verbose output suppressed, use -v or -vv for full p
listening on eth0, link-type EN10MB (Ethernet), capture size
10:48:48.IP 10.2.73.127.32794 > 10.2.73.86.3000: UDP, length
10:48:48.IP 10.2.73.86.3000 > 10.2.73.127.32794: UDP, length
10:48:48.IP 10.2.73.85.32774 > 10.2.73.86.3000: UDP, length 2
10:48:48.IP 10.2.73.86.3000 > 10.2.73.85.32774: UDP, length 2
10:48:48.IP 10.2.81.42.32773 > 10.2.73.86.3000: UDP, length 2
10:48:48.IP 10.2.73.86.3000 > 10.2.81.42.32773: UDP, length 2
10:48:49.IP 10.9.185.217.33327 > 10.2.73.86.3000: UDP, length
10:48:49.IP 10.2.73.86.3000 > 10.9.185.217.33327: UDP, length
10:48:49.IP 10.2.81.42.32774 > 10.2.73.86.3000: UDP, length 2
10:48:49.IP 10.2.73.86.3000 > 10.2.81.42.32774: UDP, length 2
10:48:49.IP 10.9.185.217.33328 > 10.2.73.86.3000: UDP, length
10:48:49.IP 10.2.73.86.3000 > 10.9.185.217.33328: UDP, length
10:48:49.IP 10.9.185.201.32816 > 10.2.73.86.3000: UDP, length
10:48:49.IP 10.2.73.86.3000 > 10.9.185.201.32816: UDP, length
10:48:49.IP 10.2.81.42.32775 > 10.2.73.86.3000: UDP, length 2
```

Gestion détachée

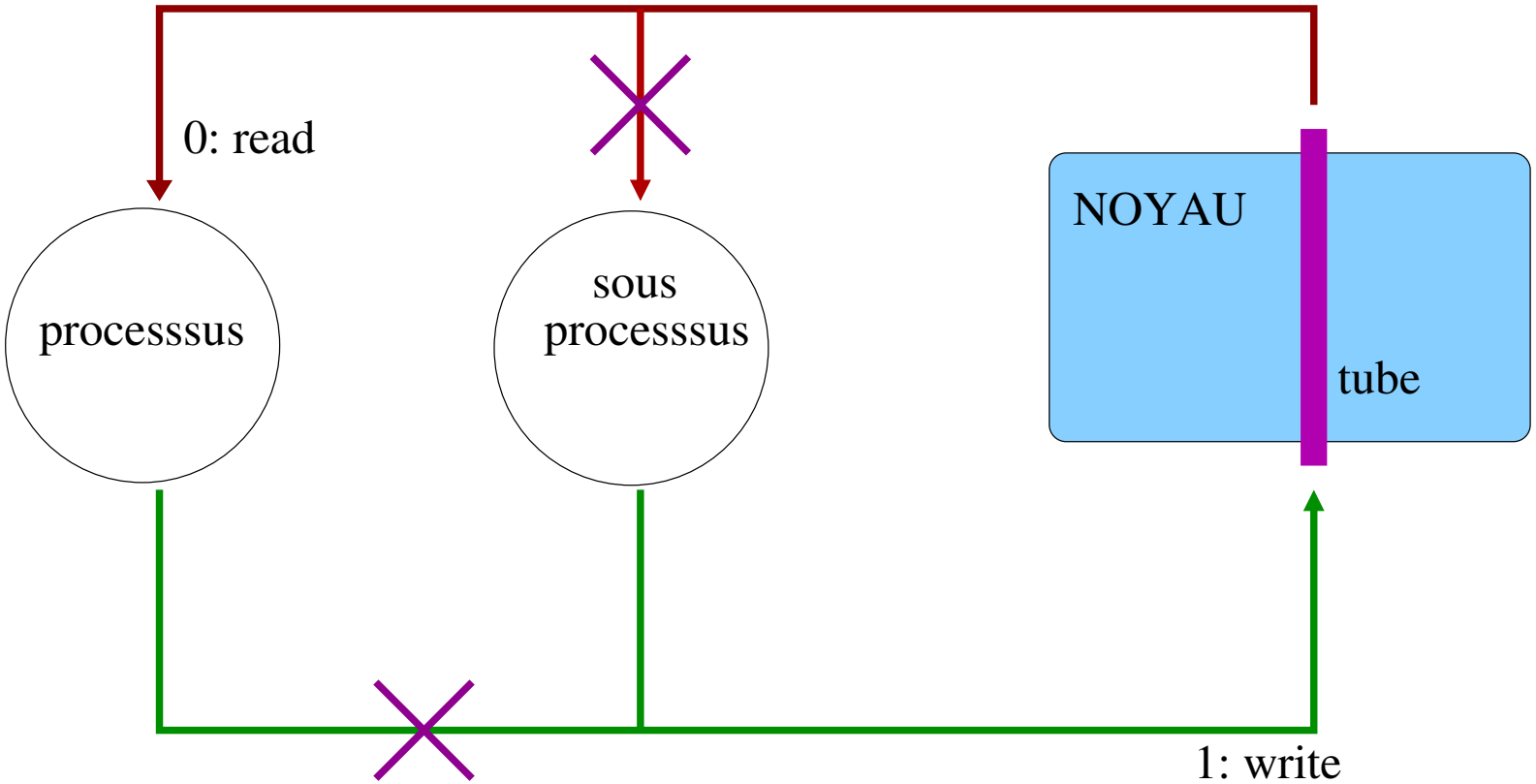
Les clients lancent leur requête sur le serveur qui détache un processus pour la gestion du client.

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

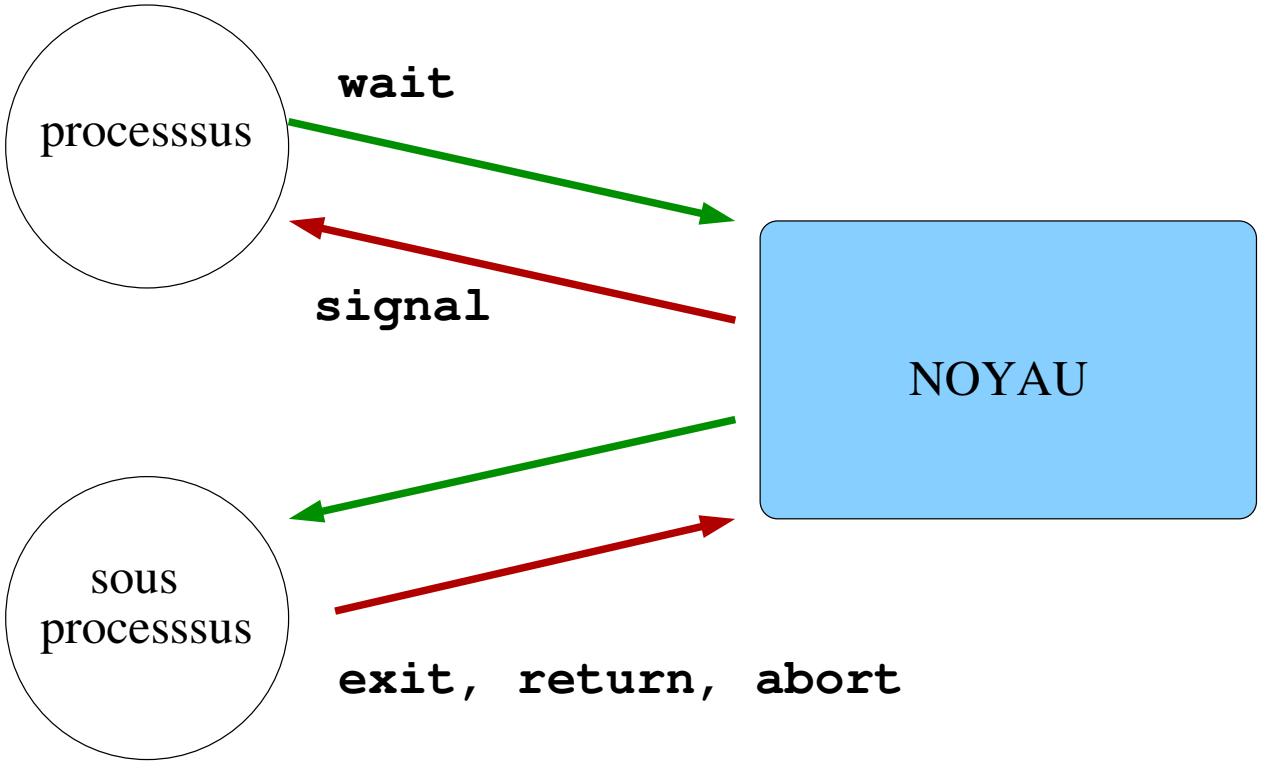
Communication interprocessus par tube



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);
```

Code du serveur fork

```
int sdcnet( int sd, int n )
{ int  somme = 0, val, nb, status, pid, tube[2];
  struct sockaddr  aux;
  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);
    if  ( 4 != (nb = read( tube[0], &val, 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 -n 5

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, length
16:56:57 IP 10.2.73.86.33074 > 10.2.73.127.32806: UDP, length
16:56:57 IP 10.2.81.42.32917 > 10.2.73.86.31415: UDP, length
16:56:57 IP 10.2.73.86.33075 > 10.2.81.42.32917: UDP, length
16:57:03 IP 10.2.73.127.32806 > 10.2.73.86.33074: UDP, length
16:57:03 IP 10.2.73.127.32806 > 10.2.73.86.31415: UDP, length
16:57:03 IP 10.2.73.86.33076 > 10.2.73.127.32806: UDP, length
16:57:03 IP 10.2.81.42.32917 > 10.2.73.86.33075: UDP, length
16:57:03 IP 10.2.81.42.32917 > 10.2.73.86.31415: UDP, length
16:57:03 IP 10.2.73.86.33077 > 10.2.81.42.32917: UDP, length
16:57:09 IP 10.2.73.127.32806 > 10.2.73.86.33076: UDP, length
16:57:09 IP 10.2.73.127.32806 > 10.2.73.86.31415: UDP, length
16:57:09 IP 10.2.73.86.33078 > 10.2.73.127.32806: UDP, length
16:57:09 IP 10.2.81.42.32917 > 10.2.73.86.33077: UDP, length
16:57:09 IP 10.2.81.42.32917 > 10.2.73.86.31415: UDP, length
16:57:14 IP 10.2.73.86.33080 > 10.1.65.1.domain:  56074+ AAAA
cache.univ-tln.fr IP(35)
16:57:14 IP 10.1.65.1.domain > 10.2.73.86.33080:  56074* 0/1/
```

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

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

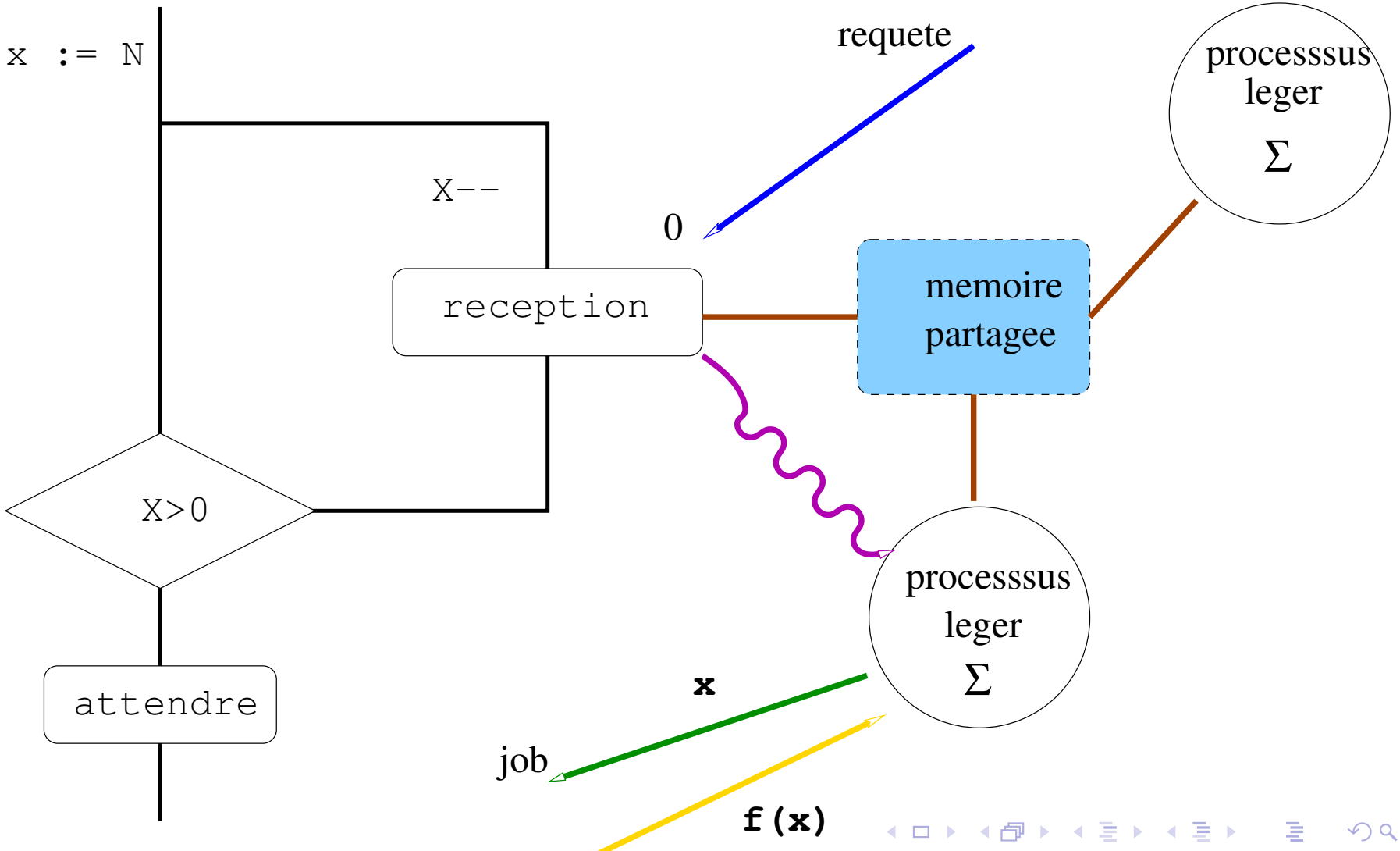
Code du serveur fork basé sur code de retour

```
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



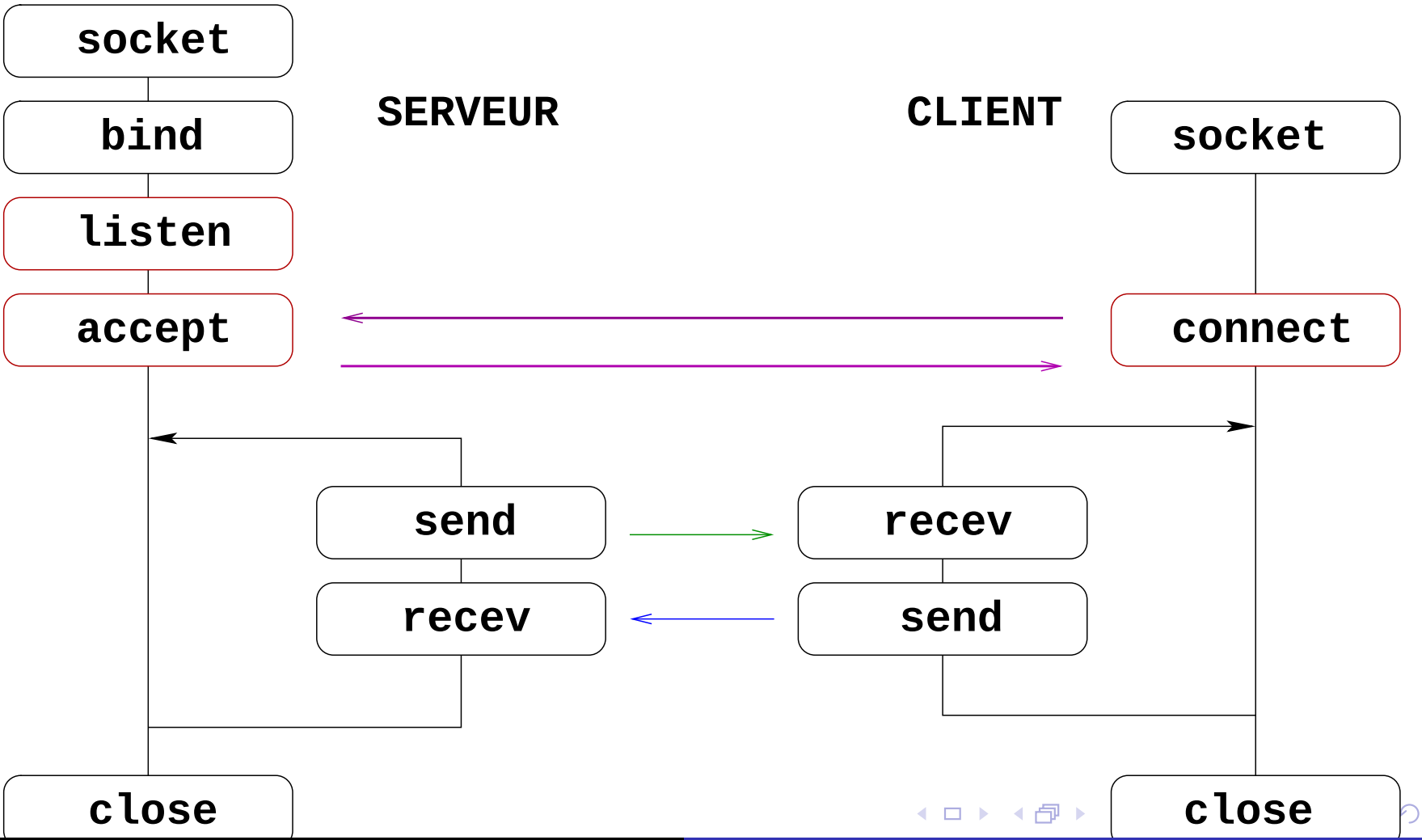
Tache du thread serveur

```
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, (void
    if ( ret ) erreur("thread");
    n = n - 1;
  }
  while ( actifs ) {
    pthread_join( actifs->t, NULL);
```

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(serv)) <
     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(serv))
     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
- listings.tex socket.tex urls.tex
- sdc.fig sdcfork.fig sdcproc.fig sdcthread.fig tcp.fig tube.fig udp.fig udpfork.fig zombie.fig

Résolveur

1. Reprendre le programme `whoami.c`.
2. Compiler, tester.
3. Faire des modifications pour tester les différents champs de la variable `infos`.

Messagerie

1. Utiliser le protocole `udp` pour écrire un système de messagerie entre deux utilisateurs sur deux hôtes.
2. Décrire un protocole de communication pour gérer les messages entre plusieurs utilisateurs : un serveur qui reçoit les messages, et les renvoie vers les clients abonnés.
3. Modifier votre code pour gérer les communications entre plusieurs utilisateurs.
4. Changer d'approche en vous basant sur la diffusion IP.

Un mini DNS

1. Trouver les RFCs qui décrivent la structure des paquets DNS.
2. Quels sont les fondateurs du DNS ?
3. Ecrire un serveur qui lit un paquet DNS, puis affiche son contenu en utilisant la description des RFCs.
4. Modifier le serveur pour donner une réponse correctement formatée.
5. Transformer votre serveur pour qu'il réponde correctement aux requêtes qui lui sont adressé.

Connection TCP

1. Ecrire un client **nc**, vérifier le fonctionnement du code client en utilisant un serveur **nc**.
2. Ecrire un serveur **nc**.
3. Vérifier le fonctionnement de votre application.