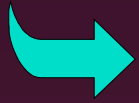




Programmation réseau

cours @urec.cnrs.fr

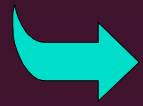


- 1997 : *Bernard Tuy, Jean-Paul Gautier*

Plan

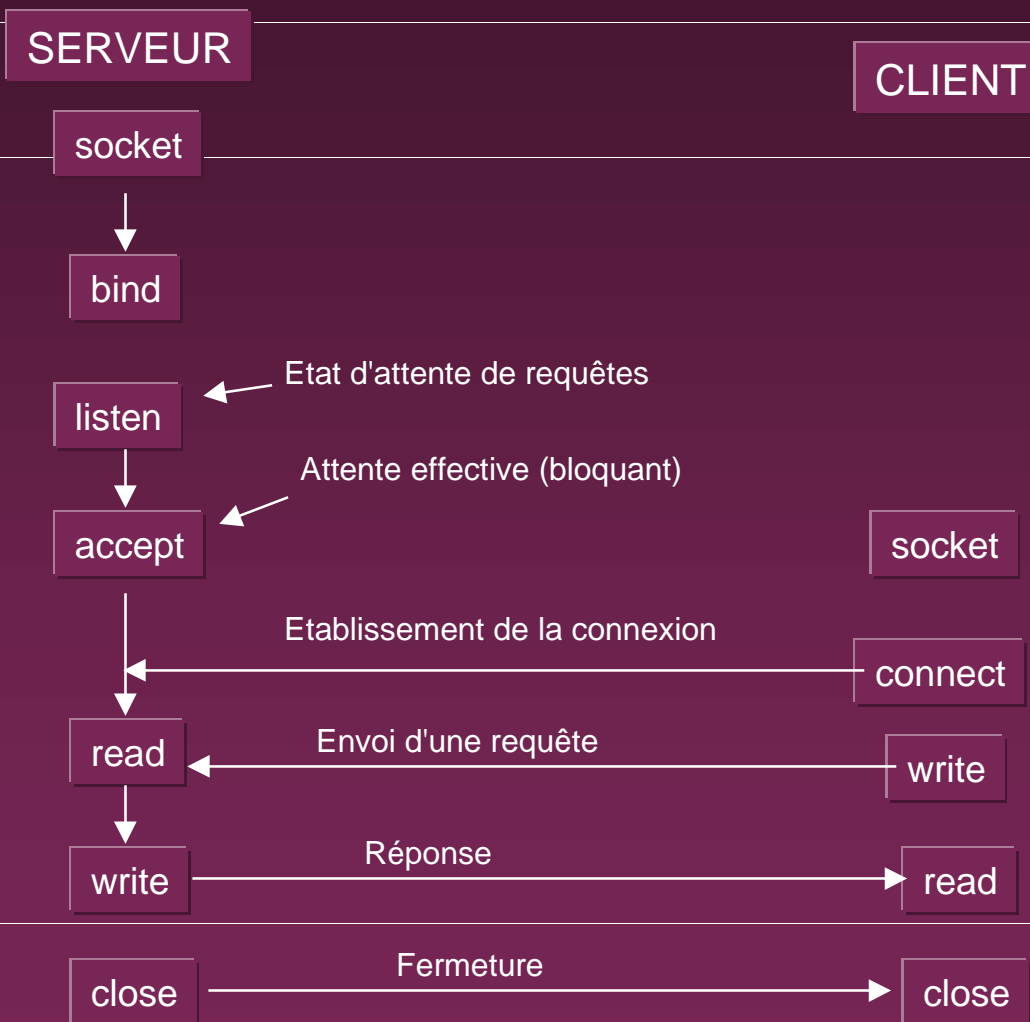
- Mode client - serveur
- Généralités sur les différents modes de communications
 - » Socket
 - » Stream
 - » RPC
- Les Sockets

source : Réseaux locaux et Internet (Laurent Toutain)[HERMES]



Socket

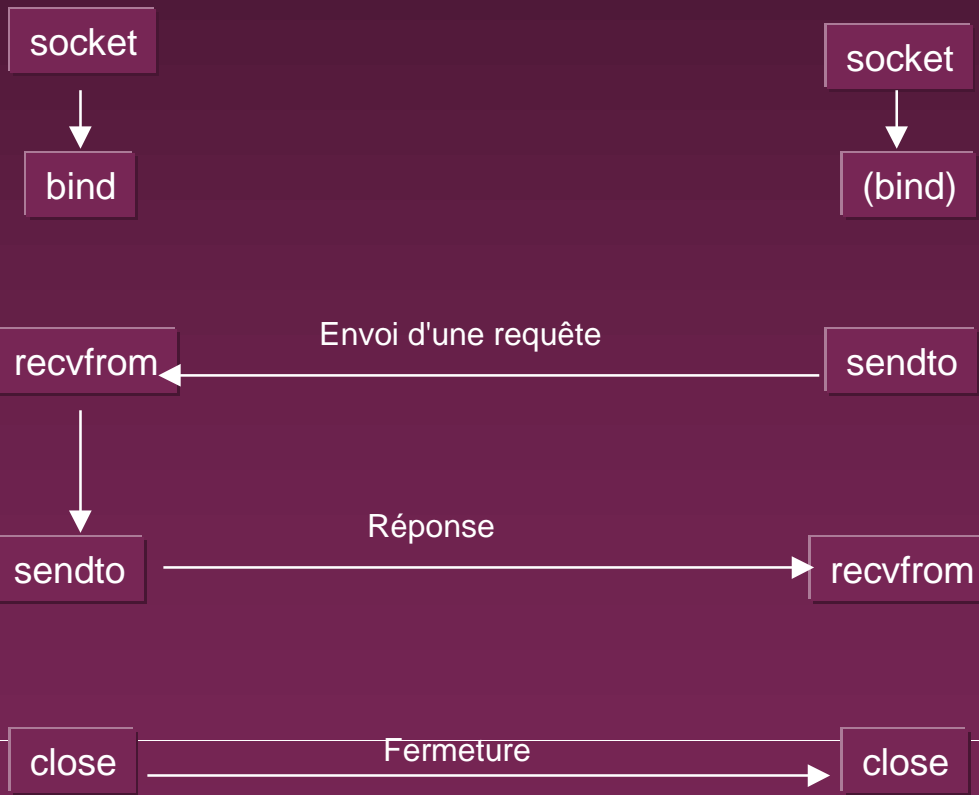
- Mécanisme d'interface de programmation
 - » permet aux programmes d'échanger des données
 - » n'implique pas forcément une communication par le réseau
- Avec les protocoles UDP et TCP, une connexion est entièrement définie sur chaque machine par :
 - » le type de protocole (UDP ou TCP)
 - » l'adresse IP
 - » le numéro de port associé au processus
 - serveur : port local sur lequel les connexions sont attendues
 - client : allocation dynamique par le système

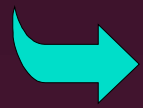




SERVEUR

CLIENT



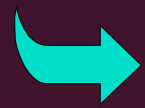


Socket

primitives Unix

● Définition d'une socket

- » Création **s = socket (domaine, type, protocole)**
 - domaine
 - » PF_UNIX : locale au système, nom de fichier dans l'arborescence
 - » PF_INET : accès au réseau en utilisant le protocole IP
 - » PF_ROUTE : passage de paramètres au noyau (tables de routages, table ARP)
 - type
 - » SOCK_STREAM : protocoles de type mode connecté (TCP)
 - » SOCK_DGRAM : protocoles de type mode datagramme (UDP)
 - » SOCK_RAW : utilisation directe des protocoles de bas niveau 3 (IP, ICMP)
 - protocole : identification du protocole utilisé. Si 0, le système déduit ce champ des 2 paramètres précédents.



Socket

primitives Unix

» Adressage

struct sockaddr {

```

    u_char          sa_len;          /*longueur totale*/
    u_char          sa_family;      /* famille d'adresse */
    char            sa_data [14];   /* valeurs */
};

```

- dans le fichier source <sys/socket.h>

struct sockaddr {

```

    u_int8_t        sin_len ;       /*longueur totale*/
    u_int8_t        sin_family;     /* famille d'adresse */
    u_int16_t       sin_port;
    struct          in_addr sin_addr;
    int8_t          sin_zero[8];
};

```

- dans le fichier source netinet/in.h



Socket

primitives Unix

» Lien entre la socket et le protocole

error = bind (s, adr, adrlong)

- error : entier qui contient le compte-rendu de l'instruction
 - » 0 : opération correctement déroulée
 - » -1 : une erreur est survenue
- s descripteur de la socket
- adr pointeur vers la zone contenant l'adresse de la station
- adrlong logueur de la zone adr

» Ouverture d'une connexion

error = connect (s, destaddr, adrlong)

- mêmes paramètres que pour **bind**, la structure **destaddr** contient l'adresse de la machine distante ainsi que le numéro de port à atteindre



Socket

primitives Unix

» Primitives du serveur

listen (s, backlog)

- permet de créer une file d'attente pour recevoir les demandes de connexion qui n'ont pas encore été prises en compte.
 - » s descripteur de la socket
 - » backlog nombre de requêtes maximum autorisées.

snew = accept (s, clientaddr, clientaddrlen)

- blocage dans l'attente d'une connexion si **accept** est OK
- les données peuvent être lues ou écrites à travers la socket **snew**



Socket

primitives Unix

- *Adresses locales et distantes d'une socket*

- `getsockname (s, localadr, addrlen)`

- permet de connaître l'adresse locale d'une socket (celle du `bind`).
 - utile pour connaître le numéro de port attribué dynamiquement par le système.

- `getpeername (s, destadr, addrlen)`

- permet de connaître l'adresse distante d'une socket (celle du `connect`) pour les sockets en mode connecté.



Socket

primitives Unix

- *Réception de données*

`cc = read (s , buffer, taillemax)`

- `buffer` est un pointeur vers la zone de réception.
- `cc` : nombre d'octets réellement reçus.

`cc = recv (s , buffer, taillemax, drapeau)`

- `drapeau` permet de configurer la connexion
 - » `MSG_OOB` : lecture "out of band" des messages urgents
 - » `MSG_PEEK` : lecture des données sans les retirer du tampon

`cc = recvfrom (s , buffer, taillemax, drapeau , émetteur, adrlg)`

- `émetteur` contient l'adresse de l'émetteur, utilisé en mode datagramme



Socket

primitives Unix

- *Emission de données*

`write (s , buffer, longueur)`

- utilisable uniquement en mode connecté (pas d'adresse de destinataire)

`send (s , buffer, taillemax, drapeau)`

- `drapeau` permet de configurer la connexion
 - » `MSG_OOB` : écriture "out of band" des messages urgents
 - » `MSG_DONTROUTE` : déboguage .

`sendto (s , buffer, taillemax, drapeau , récepteur, adrlg)`

- `récepteur` contient l'adresse du destinataire, utilisé en mode datagramme



Socket

primitives Unix

- *Fin d'utilisation d'une socket*

`close (s)`



Socket

primitives Unix

- *Accès aux bases de données relatives aux sites*

```
struct hostent * gethostbyname ( name )
```

- interrogation sur /etc/hosts, NIS, DNS
- la structure est définie dans netdb.h

```
struct hostent * gethostbyaddr ( addr, len, type )
```

```
gethostname ( name , namelen)
```

- permet de connaître le nom de la machine locale sur laquelle s'exécute le programme.

```
getnetbyname ( name ), getnetbyaddr ( netaddr, addrtype)
```

- le réseau sur lequel on travaille

```
getprotobyname ( name ), getprotobynumber ( number)
```

- le protocole utilisé

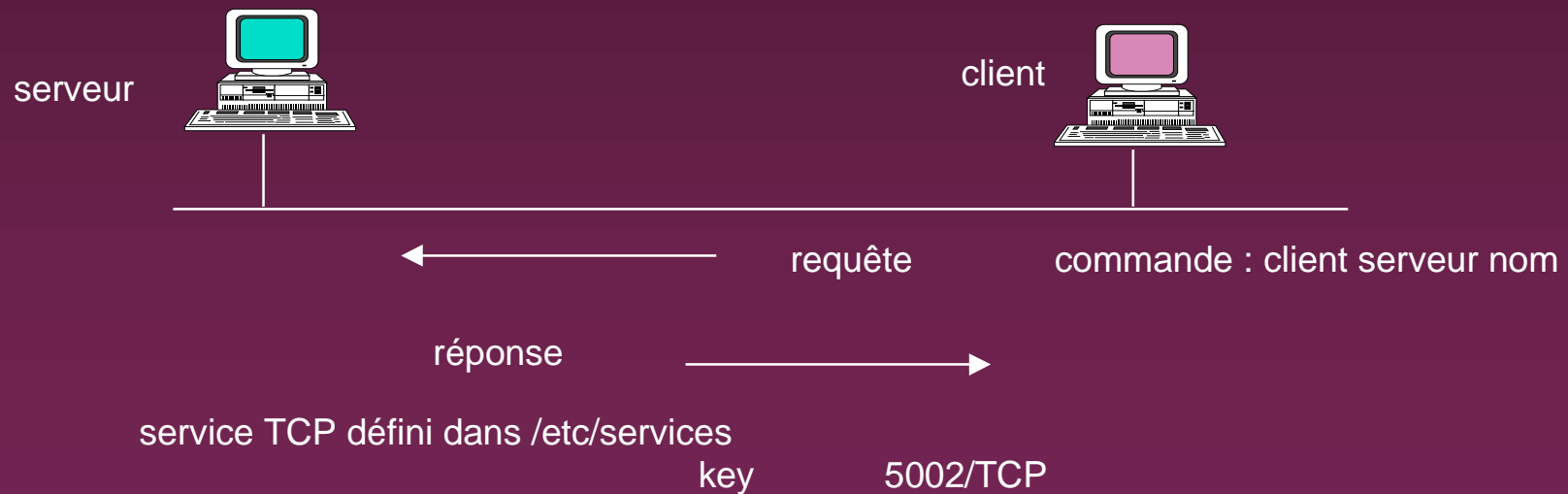
```
getservbyname ( name , proto), getservbyport ( port, proto)
```

- le service utilisé

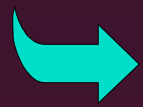
Socket

exemple

- Programme permettant de savoir si un utilisateur est connu sur une station.



source : Internetworking with TCP/IP (Douglas COMER)



Sockets

exemple client

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#define BUFSIZE 200
#define NORMAL 0
```

```
main (argc,argv)
```

```
int argc;
```

```
char **argv;
```

```
{
```

```
int s;
```

```
int len;
```

```
struct sockaddr_in sa;
```

```
struct hostent *hp;
```

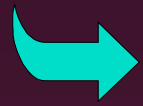
```
struct servent *sp;
```

descripteur de socket

structure adresse Internet

structure service de nom

structure service Internet



Socket

exemple client

```
char *myname;
```

pointeur sur le nom du programme

```
char buf[BUFSIZE];
```

```
char *host,*user;
```

pointeurs sur le serveur et l'utilisateur

```
myname = argv[0];
```

test des arguments

```
if (argc != 3) {
```

```
    fprintf (stderr, "Usage : %s serveur user\n",myname);
```

```
    exit (1);
```

```
}
```

```
user = argv[2];
```

```
host = argv[1];
```

```
if ((hp = gethostbyname(host)) == NULL) {
```

```
    fprintf (stderr, "%s : %s serveur inconnu\n",myname,host);
```

```
    exit (1);
```

```
}
```

voir si le serveur existe,
structure hp remplie



Socket

exemple client

```
bcopy ((char *)hp->h_addr,(char *)&sa.sin_addr,hp->h_length);  
sa.sin_family = hp->h_addrtype;
```

*copie de l'adresse du
serveur et du type
d'adresse dans sa*

```
if ((sp = getservbyname("key","tcp")) == NULL) {  
    fprintf (stderr, "%s : Pas de service QUI sur ce système \n",myname);  
    exit (1);  
}
```

*voir si le service est
autorisé sur le client,
rempli la structure sp*

```
sa.sin_port = sp->s_port;
```

numéro du service

```
if ((s = socket (hp->h_addrtype, SOCK_STREAM,0)) < 0 ) {  
    perror ("socket");  
    exit (1);  
}
```

*création de la socket
client, allocation par
le système*

```
fprintf ( stdout, "Service %d demandé à %s\n", sa.sin_port,host );
```

```
fprintf ( stdout, "Type d'adresse %d ; descripteur de socket\n",  
sa.sin_family,s);
```



Socket

exemple client

```
if ( connect (s,&sa,sizeof(sa)) < 0 ) {  
    perror ("connect");  
    exit (1)  
}
```

*connexion au serveur, infos dans la
structure adresse Internet*

```
send ( s, user, strlen(user)+1, NORMAL);
```

envoi de la requête

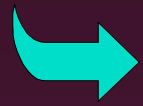
```
recv ( s, buf, BUFSIZE, NORMAL );
```

lecture de la réponse

```
fprintf ( stdout, "Réponse : %s _n", buf);  
close ( s );  
exit (0);  
}
```

écriture de la réponse

fermeture de la connexion



Sockets

exemple serveur

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <pwd.h>
#define BUFSIZE 200
#define NORMAL 0
#define BACKLOG 1
#define MAXHOSTNAME 32
#define MAXREQ 3

main (argc,argv)
int argc;
char **argv;

void quiet();
```

taille de la file d'attente initialisée par listen

nombre de requêtes traitées par ce programme, il s'arrête dès que la limite est atteinte



Socket

exemple serveur

<pre>int s, t;</pre>	<i>descripteur de socket</i>
<pre>int i, count_req;</pre>	
<pre>struct sockaddr_in sa;</pre>	<i>structure adresse Internet</i>
<pre>struct sockaddr_in isa;</pre>	
<pre>struct hostent *hp;</pre>	<i>structure service de nom</i>
<pre>struct servent *sp;</pre>	<i>structure service Internet</i>
<pre>char localhost[MAXHOSTNAME+1];</pre>	
<pre>if ((sp = getservbyname("key", "tcp")) == NULL) {</pre>	<i>voir si service existe</i>
<pre> fprintf(stderr, "Pas de service QUI sur ce système \n");</pre>	
<pre> exit(1);</pre>	
<pre>}</pre>	
<pre>sa.sin_port = sp->s_port;</pre>	<i>récupération du numéro de port serveur dans la structure sa</i>



Socket

exemple serveur

```
gethostname (localhost, MAXHOSTNAME);
hp = gethostbyname (localhost);
bcopy(hp->h_addr,(char *)&sa.sin_addr,hp->h_length);
sa.sin_family = hp->h_addrtype;
if ((s = socket (hp->h_addrtype, SOCK_STREAM,0)) < 0 ) {
    perror ("Serveur : problème création socket");
    exit (1);
}
if ( bind ( s, &sa, sizeof(sa)) < 0 ) {
    fprintf (stderr, "Serveur : problème création lien");
    exit (1);
}
listen (s, BACKLOG);
fprintf ( stdout, "Service %d sur %s en attente\n",sa.sin_port,localhost);
fprintf (stdout, "    type d'adresse : %d _n",sa.sin_family);
```

*infos sur le serveur dans la structure **hp***

*rempli la structure adresse **sa***

allocation d'un descripteur de socket

lien socket avec adresse IP et port

attente de demande de connexion



Socket

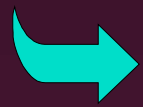
exemple serveur

```
for (count_req = 0; count_req <=MAXREQ ; count_req++) {  
    t = accept ( s, &isa, &i );  
    fprintf (stdout, "Requête %d\n", count_req);  
  
    quiest (t);  
  
    close (t);  
}  
close ( s );  
fprintf (stdout,"fin du service pour %s\n",localhost);  
exit (0);  
}
```

*attend une requête sur la primitive
accept*

traitement de la requête

*fermeture de la socket réservée au
traitement de cette requête*



Socket

exemple serveur

```
void quiest (sock)
```

procédure de traitement de la requête

```
int sock;
```

```
{
```

```
struct passwd *p;
```

```
char buf[BUFSIZE];
```

```
int i;
```

```
if (( i = recv( sock, buf, BUFSIZE, NORMAL)) <= 0)
```

réception de la demande

```
return;
```

```
if (( p = getpwnam (buf)) == NULL)
```

recherche dans le fichier /etc/passwd

```
strcpy ( buf, "Utilisateur inconnu sur le serveur\n");
```

```
else
```

réponse

```
sprintf ( buf, "%s : %s _n",p->pw_name, p->pw_gecos);
```

```
send (sock, buf, strlen (buf), NORMAL);
```

envoi de la réponse

```
}
```