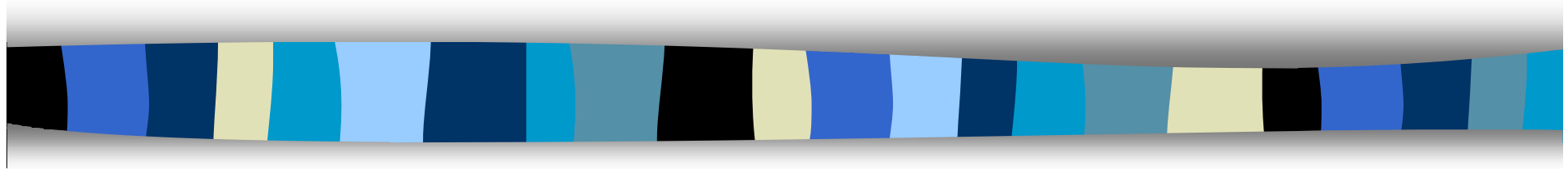


Programmation des Sockets sous Unix



Hafid Bourzoufi

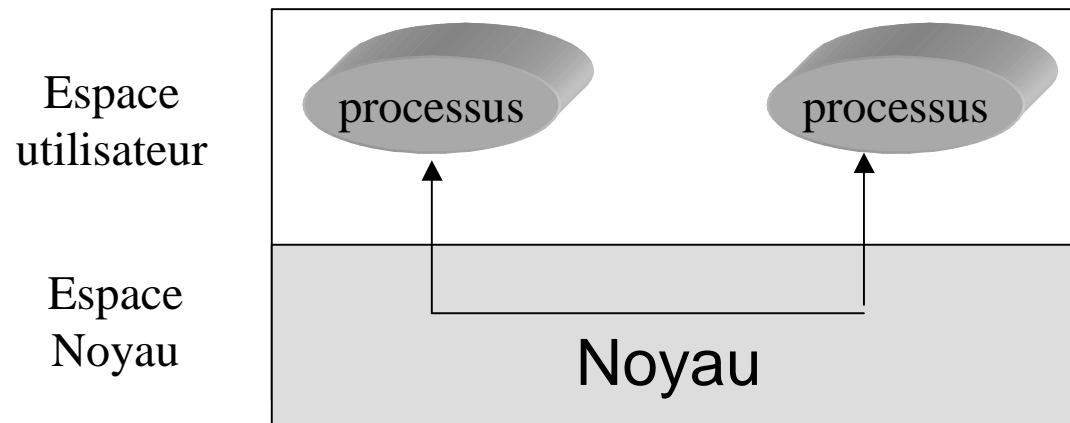
Didier Donsez

Université de Valenciennes

Institut des Sciences et Techniques de Valenciennes

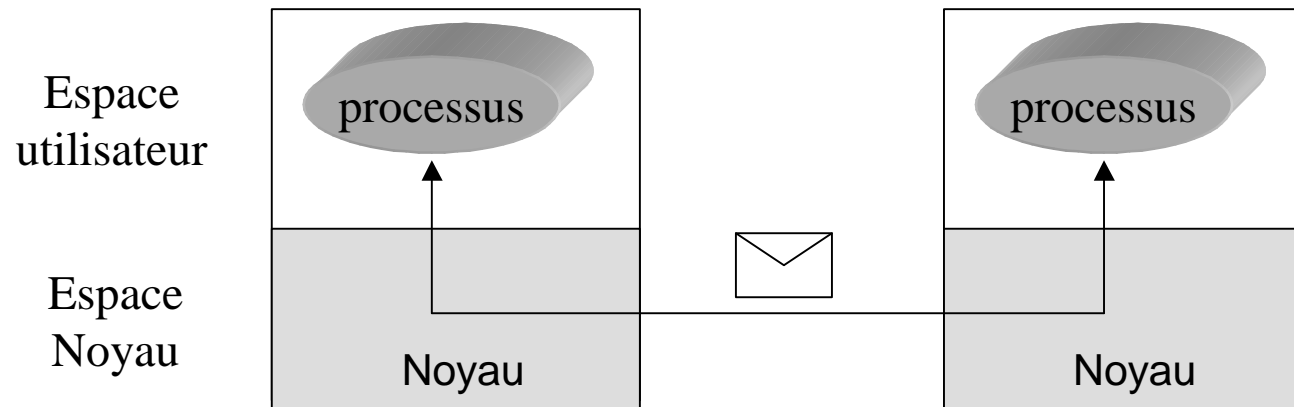
`donsez@univ-valenciennes.fr`

Les communications dans les systèmes centralisés (Rappel)



Les communications supposent
l'existence d'une mémoire partagée
Exemples : les pipes d'Unix, les IPCs système V

Les communications dans les systèmes répartis



Les communications par envoi de message

Protocole de communication

- La mise en œuvre des communications entre systèmes nécessite des protocoles de communication
- Protocole = un ensemble d'accords sur
 - la représentation des bits
 - détection de la fin d'un message
 - acheminement des message
 - représentation des nombres, des caractères
 - etc ...
 - Il faut un accord à plusieurs niveaux, depuis les détails de bas niveau de la transmission des bits jusqu'à ceux de plus haut niveau de la représentation des données

Le modèle OSI (Open System Interconnexion)

7- Couche Application

Un ensemble de services standard :
terminal virtuel, transfert de fichiers, ...

6- Couche Représentation

Codage des informations sous une forme
standard

5- Couche Session

Gestion complète d'une session de
communications entre 2 utilisateurs.

4- Couche Transport

Transport de l'information entre deux
processus (Multiplexage)

3- Couche Réseau

Transport de l'information entre deux
points du réseau (Routage)

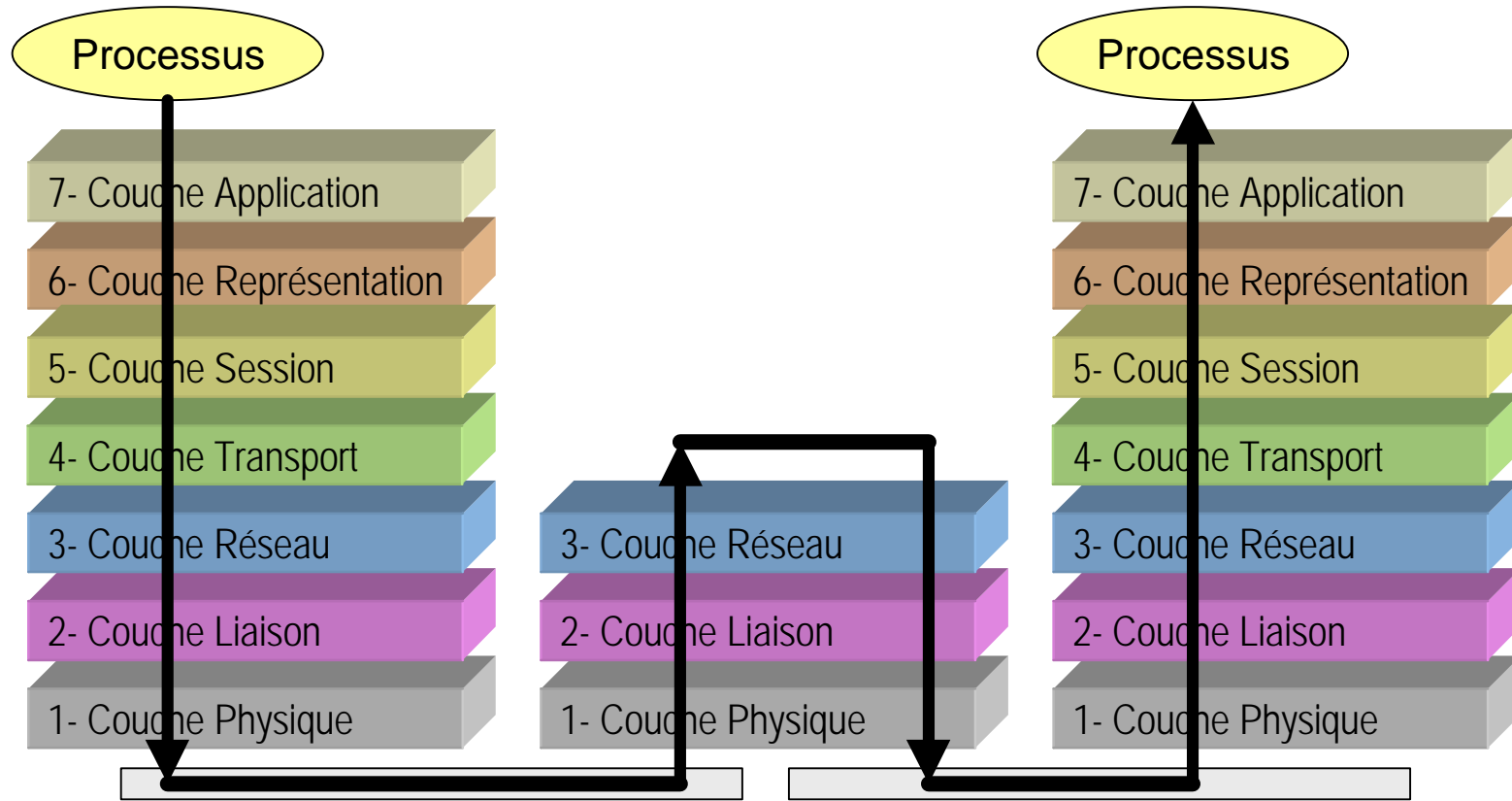
2- Couche Liaison

Transport, sans erreur entre deux points,
des blocs de données (trames)

1- Couche Physique

Transport de l'information comme une
suite de bits

Communication sous OSI



Routage entre plusieurs supports de communication

Internet Protocol : Généralités

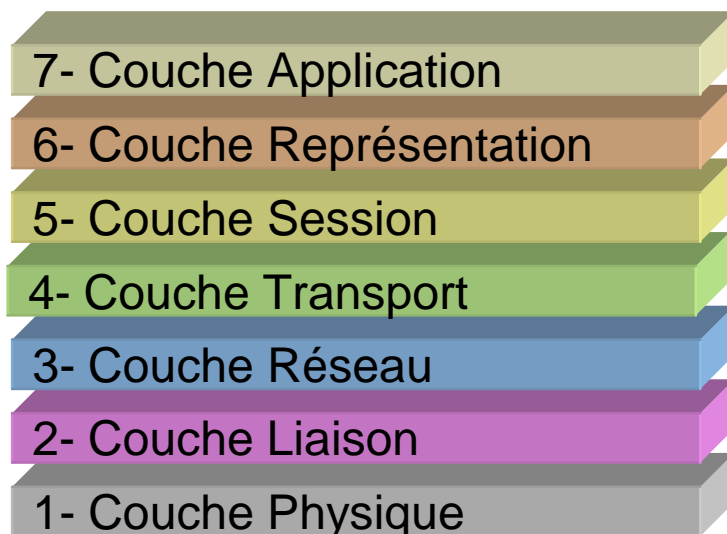
■ Un peu d 'histoire

- Mis au point par l 'agence DARPA
- Le réseau concerné à l 'origine était ARPANET devenu INTERNET
- Unix BSD 4.x premier a avoir intégré IP

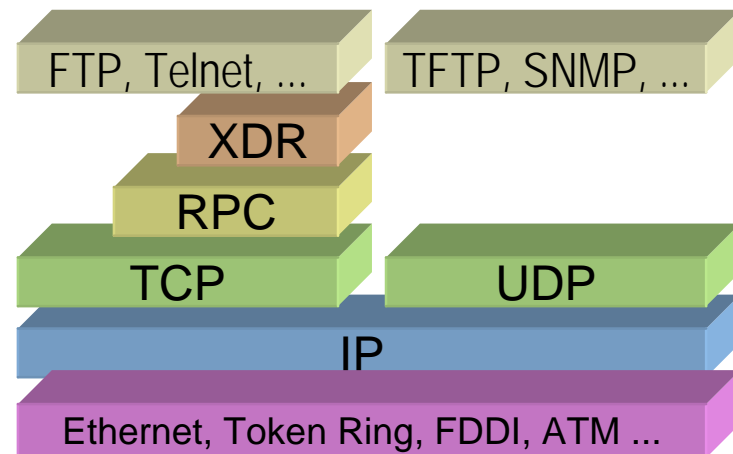
■ Ensemble de protocoles des couches 3 à 7

- permettant l 'interconnexion de différents types de machines et de réseaux

Le modèle OSI



Internet



Les Sockets

- Les sockets = API (Application Program Interface)
 - interface entre les programmes d'applications et les couches réseaux
 - le terme Socket désigne aussi un canal de communication par lequel un processus peut envoyer ou recevoir des données
- L'API Socket s'approche de l'API Fichier d'Unix
 - Descripteur de socket dans la table des descripteurs du processus
 - Primitives read(), write(), close(), ioctl(), fcntl(), select()
 - Signaux SIGIO, SIGPIPE ...
 - Les descripteurs peuvent être partagés par les descendants de son créateur
- Un peu d'histoire
 - Proposé par le système Unix BSD
 - implémentation WinSock de Trumpet pour Windows 3.x

Création d 'un Socket

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

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

```
int socket(int domaine, int type , int protocole);
```

- crée un socket et retourne son numéro de descripteur dans la table des descripteurs du processus
- -1 si erreur (errno est positionné)
- cependant le socket n 'est pas lié (*bounded*) à un socket distant
 - ce qui est nécessaire pour le mode connecté
 - ce qui n 'est pas nécessaire pour le mode non connecté

Création d 'un socket

■ Domaine d 'adresse:

- Internet `AF_INET` , Unix `AF_UNIX`
- mais aussi `AF_OSI` , `AF_NS` , `AF_SNA` , `AF_CCITT` , `AF_APPLETALK` ,
...

■ Type

- au niveau bas du protocole (exemple : datagramme IP) `SOCK_RAW`
- en mode non connecté `SOCK_DGRAM` , `SOCK_RDM` ,
`SOCK_SEGPACKET`
- en mode connecté `SOCK_STREAM`

■ Protocole :

- par défaut à 0, le système choisit le protocole
- `IPPROTO_UDP` pour UDP avec les sockets de type `SOCK_DGRAM` ,
`SOCK_RDM` , `SOCK_SEGPACKET`
- `IPPROTO_TCP` pour TCP avec une socket de type `SOCK_STREAM`
- il existe d 'autres protocoles
- dans le domaine `AF_UNIX`, il n 'y a pas de protocole

Création d'un Socket

- Coté noyau Unix

Table des
Descripteurs
(un par processus)

0	●	→
1	●	→
2	●	→
3	●	→
4	Non alloué	
5	●	→
...		

Vers
des descripteurs
de fichiers

Structure d'un Descripteur
(partagé entre plusieurs
processus)

PF_INET	Family
SOCK_STREAM	Service
	Local IP Address
	Local Port
	Remote IP Address
	Remote Port
	...

Attachement d 'un socket à une adresse

- Après sa création, un socket n'est connu que du processus qui l 'a crée (et de ses descendants).
- Il doit être désigné par une adresse pour pouvoir être contacté de l 'extérieur (autres processus locaux ou distants).

■ Primitive bind()

`int bind(int descsock, struct sockaddr *ptlocsockaddr,int locsockaddrlen)`

- associe l 'adresse locale ptlocsockaddr au socket descsock
- locsockaddrlen est la taille de l 'adresse *ptlocsockaddr

Adressage des sockets

- Adresse de la domaine Unix `AF_UNIX`
 - = une entrée dans le système de fichiers (local ou par NFS)
- Adresse du domaine Internet `AF_INET`
 - = <adresse IP de la machine , numéro de port>
 - Sous Unix, les numéros de ports < 1024 sont réservés au SU
 - Exemple de numéros réservés (ypcat services)
 - » ftp : 21/tcp
 - » telnet : 23/tcp
 - La liste des services se trouvent dans le fichier `/etc/services`
 - La liste des exécutable servant ces services via `inetd` et dans `/etc/inetd`

Adressage des sockets

- **Format générique des adresses de sockets**

```
struct sockaddr {
    short sa_family;      /* domaine AF_UNIX, AF_INET */
    char sa_data[14];    /* adresse */
}
```

- **Type associé aux adresses dans le domaine Unix**

```
#include <sys.un.h>
struct sockaddr_un {
    short sun_family;    /* domaine AF_UNIX */
    char sun_path[128]; /* chemin */
}
```

- **Type associé aux adresses dans le domaine Internet**

```
#include <netinet/in.h>
struct in_addr { u_long s_addr; };
struct sockaddr_in {
    short          sin_family;      /* domaine AF_INET */
    u_short       sin_port;        /* port de la socket */
    struct in_addr sin_addr;      /* N° IP de la machine format réseau */
    char          sin_zero[8];    /* 8 caractères nuls de bourrage */
};
```

Adressage des sockets

Fabrication d 'adresse AF_INET

- Dépend de son contexte d 'utilisation
 - associer l 'adresse fabriquée au socket local pour qu'il soit accessible de l extérieur
 - fabriquer l 'adresse d 'une socket éloigné

Adressage des sockets

Fabrication d 'adresse AF_INET

- Fabrication d 'une adresse destiné à être attachée localement (par bind)

```
void localaddress( struct sockaddr_in *s_loc) {  
    s_loc->sin_family = AF_INET;  
    s_loc->sin_port = 0; /* le port sera alloué dynamiquement */  
    s_loc->sin_addr.s_addr = INADDR_ANY; /* adresse jocker */  
}
```

- Fabrication d 'une adresse éloignée

```
void remoteaddress(struct sockaddr_in *s_rem, const char *rhost, int port) {  
    struct hostent *h;  
    s_rem->sin_family = AF_INET;  
    s_rem->sin_port = port;  
    if((h=gethostbyname(rhost))==0) {fprintf(stderr, "%s : machine inconnue ",rhost); }  
    bcopy((char *)h->h_addr,(char *)s_rem->sin_addr, h->h_length);  
}
```


La résolution DNS

■ Obtenir un adresse IP à partir d 'un nom DNS

```
#include <sys/types.h><sys/socket.h><netinet/in.h><arpa/inet.h><netdb.h>
struct hostent *gethostbyname(char *hostname)
```

- retournent un pointeur sur une structure du type suivant :

```
struct hostent {
    char *h_name;          /* nom canonique de la machine */
    char **h_aliases;     /* tableau des autres noms d 'alias */
    int h_addrtype;       /* type d 'adresse (AF_INET pour Internet) */
    int h_length;         /* la longueur de l 'adresse (4 en IPv4) */
    char **h_addr_list;   /* tableau des adresses de type struct in_addr */
};
```

■ Autres fonctions de consultation

int gethostname(char *name, size_t lg) *retourne l 'adresse IP dans la zone name*

long gethostid() *retourne l 'adresse IP de la machine locale*

struct hostent *gethostbyaddr(char *paddr, int longaddr, int typeaddr)

struct hostent *gethostent(char *hostname) *parcourt le fichier /etc/hosts*

La résolution DNS

■ Fonction de conversion

```
#include <sys/types.h><sys/socket.h><netinet/in.h><arpa/inet.h><netdb.h>
char *inet_ntoa(const struct in_addr in) convertit la structure en adresse lisible
unsigned long inet_addr(const char *cp) donne la forme condensée d 'un adresse
```

■ Exemple

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
int main(int argc, char** argv[]) {
    struct hostent *res; struct in_addr * addr;
    res = gethostbyname(argv[1]);
    addr = (struct in_addr *) res->h_addr_list[0];
    printf("l 'adresse principale IP est %s\n", inet_ntoa(*addr));
    for(int i=1; addr=res->h_addr_list[i] ; i++) printf("l 'autre adresse IP est %s\n", inet_ntoa(*addr));
}
```

Utilisation des sockets en mode connecté `SOCK_STREAM`

■ Etablissement asymétrique d'un circuit virtuel

- Côté Client (demandeur de la connexion)

: le socket est dit actif

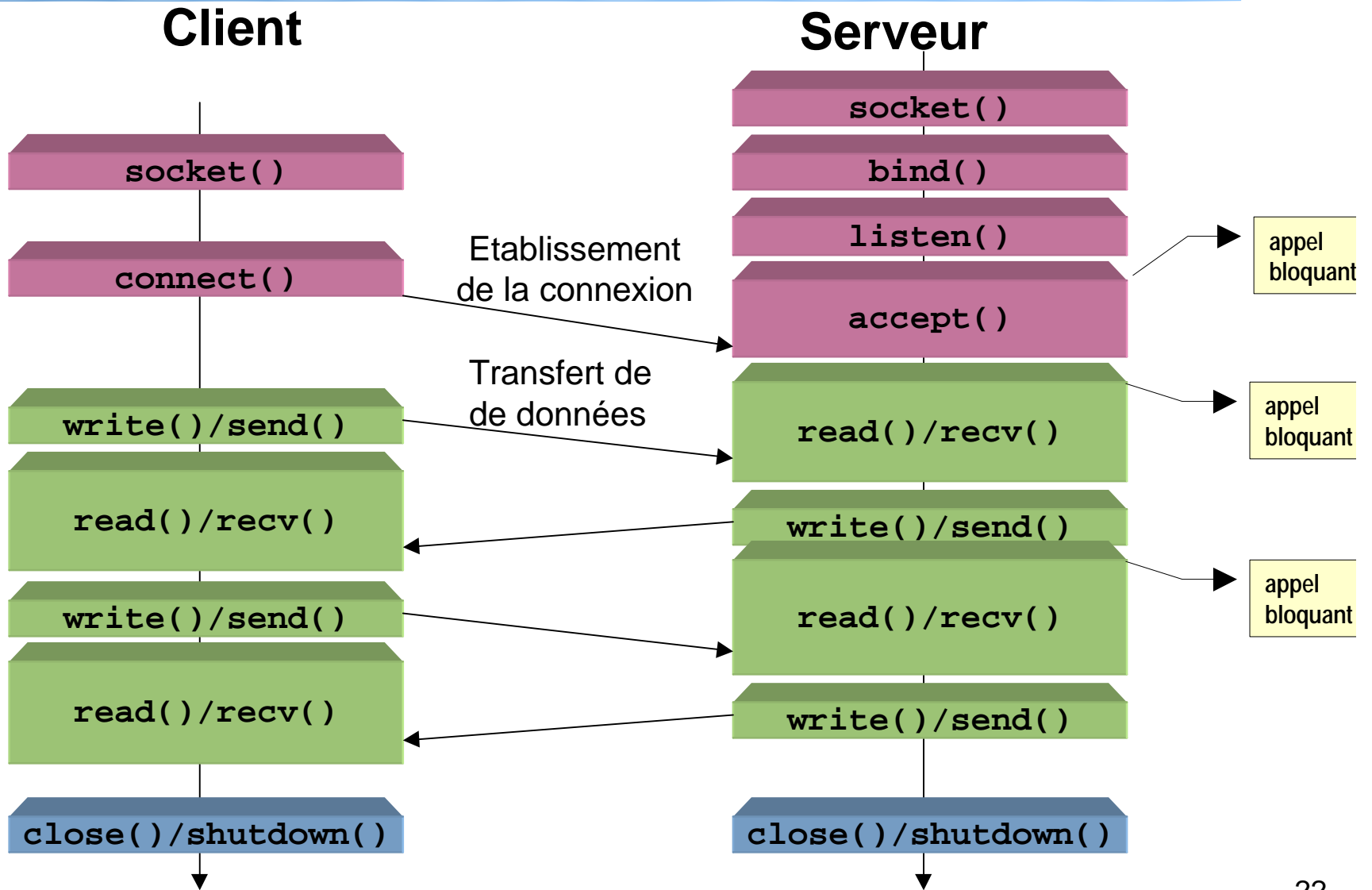
- crée un socket `socket()`
- se connecte à une `<adresse,port>` `connect()`
- lit et écrit dans le socket `read(),recv();write(),send()`
- ferme le socket `close()`

- Côté Serveur (en attente de connexion)

: le socket est dit passif

- crée un socket `socket()`
- associe une adresse au socket `bind()`
- se met à l'écoute des connexions entrantes `listen()`
- accepte une connexion entrante `accept()`
- lit et écrit sur le socket `read(),recv();write(),send()`
- ferme le socket `close()`

Utilisation des sockets en mode connecté SOCK_STREAM



Primitives spécifique au mode connecté

■ Coté Client

- `connect()` : Demande d'une pseudo-connexion vers un socket distant

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

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

```
int connect(int descsocket, struct sockaddr* ptsockaddr, int lensockaddr);
```

- *obligatoire en mode connecté (SOCK_STREAM)*
- *primitive bloquante*
- le primitive enregistre l'adresse distante dans le descripteur de socket
- Remarque:

tout appel à `connect()` annule l'appel précédent

si `ptsockaddr==NULL`, le socket n'est plus associé à un autre

Primitives spécifique au mode connecté

■ Coté Serveur

- `listen()` : Ouverture d'un service

`int listen(int descsocket, int dc);`

- permet de déclarer un service ouvert auprès du système local
- `dc` indique le nombre maximum de demandes de connexion mises en attente

- `accept()` : Acceptation de connexion

`int accept(int s, struct sockaddr *ptremotesaddr, int * ptlenremotesaddr)`

- primitive normalement bloquante (asynchronisme avec `select()`)
- permet de prendre connaissance d'une nouvelle connexion
 - une connexion est prise dans la liste des demandes pendantes
- retourne un descripteur d'un socket de service dédié à cette connexion
- `ptremotesaddr` contient l'adresse du client

Primitives d'émission/réception

■ Emission

- `ssize_t write (int descsock, void* buffer, size_t length)`
 - transmet les données du *buffer* sur le socket (connecté) *descsocket*.
 - bloquant jusqu'au transfert
- `ssize_t send (int descsock, void* msg, size_t length, int):`
 - idem mais permet d'utiliser des options de transfert out-of-band signaling, debugging, etc.

■ Réception

- `ssize_t read (int descsock, void* buffer, size_t length)`
 - lire les données reçues dans le *buffer* sur le socket (connecté) *descsocket* pour une longueur maximale *length*
 - bloquant jusqu'au transfert
- `ssize_t recv (int descsock, void* msg, size_t length, int):`
 - idem mais permet d'utiliser des options de transfert lecture sans extraction, etc.

Primitives de fermeture

■ Fermeture - Close

```
int close(int descsocket);
```

- ferme le descripteur du socket et libère ses ressources s 'il n 'est plus partagé
- Quand il reste des données bufferisées dans un SOCK_STREAM/AF_INET, le système essaie d 'acheminer ces données. Dans ce cas, le primitive peut être rendue bloquante

■ Fermeture Partielle - Shutdown (SOCK_STREAM/AF_INET)

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

```
int shutdown(int descsocket, int sens);
```

- ferme **partiellement** le descripteur du socket *full-duplex* (*ie SOCK_STREAM*)
 - sens=0 : le socket n 'accepte plus de lecture : read/recv renvoie 0
 - sens=1 : le socket n 'accepte plus d 'envoi: write/send provoque SIGPIPE
 - sens=2 : le socket n 'accepte plus ni lecture, ni écriture/envoi

Exemple: Code C du Client

```
/* Usage : streamclient /usr/local/mysock "Hello World !" */
#include <sys/types.h>
#include <sys/socket.h>
#include <string.h>
main(int argc, char *argv[]) {
int sd; int len;
struct sockaddr_un serveraddr;

sd=socket(AF_INET, SOCK_STREAM,0);

serveraddr.sun_family=AF_UNIX;
strcpy(argv[1], serveraddr.sun_path, sizeof(argv[1]));
if(connect(sd, (sockaddr *)&serveraddr, sizeof(serveraddr)) <0) exit();

len=strlen(argv[2]); write(sd, &len, sizeof(len));
write(sd, argv[2], len);
close(sd);
}
```

Exemple: Code C du Serveur

```
/* Usage : streamserveur /usr/local/mysock */
#include <sys/types.h>
#include <sys/socket.h>
#include <string.h>
main(int argc, char *argv[]){
    int sd, ns, nb; char buf[256]; int len;
    struct sockaddr_un serveraddr, clientaddr; int clientaddrlen;
    sd=socket(AF_UNIX, SOCK_STREAM,0);

    servaddr.sun_family=AF_UNIX;
    strcpy(argv[1], serveraddr.sun_path, sizeof(argv[1]));

    bind(sd, (sockaddr *)&serveraddr, sizeof(servaddr));
    listen(sd,1);
    ns=accept(sd,(sockaddr *)&clientaddr, &clientaddrlen);

    nb=read(ns, &len, sizeof(len)); /* les erreurs ne sont pas traitées */
    nb=read(ns, buf, len); write(1,buf,nb);
    close(ns); close(sd);
}
```

Exemple: Code Perl d'un Client

```
require './socket.ph';
($them,$port) = @ARGV;
$port = 8888 unless $port; $them = 'localhost' unless $them;
$sockaddr = 'S n a4 x8';
chop($hostname = `hostname`);
($name, $aliases, $proto) = getprotobyname('tcp');
($name, $aliases, $port) = getservbyname($port, 'tcp') unless $port =~ /^^\d+$/;
($name, $aliases, $type, $len, $thisaddr) = gethostbyname($hostname);
($name, $aliases, $type, $len, $thataddr) = gethostbyname($them);
$this = pack($sockaddr, &AF_INET, 0, $thisaddr);
$that = pack($sockaddr, &AF_INET, $port, $thataddr);
socket(S, &PF_INET, &SOCK_STREAM, $proto) || die "socket: $!";
bind(S, $this) || die "bind: $!";
connect(S, $that) || die "connect: $!";
select(S); $| = 1; select(stdout);
while (<>) { print S; }
```

Exemple: Code Perl d'un Serveur

```
require './socket.ph';
($port) = @ARGV; $port = 8888 unless $port;
$sockaddr = 'S n a4 x8';
($name, $aliases, $proto) = getprotobyname('tcp');
($name, $aliases, $port) = getservbyname($port, 'tcp') unless $port =~ /^d+$/;
$this = pack($sockaddr, &AF_INET, $port, "\0\0\0\0");
select(NS); $| = 1; select(stdout);
socket(S, &PF_INET, &SOCK_STREAM, $proto) || die "socket: $!";
bind(S, $this) || die "bind: $!";
listen(S, 5) || die "listen: $!";
select(S); $| = 1; select(stdout);
for (;;) { print "Listening again\n";
    ($addr = accept(NS,S)) || die $!; print "accept ok\n";
    ($af,$port,$inetaddr) = unpack($sockaddr,$addr);
    @inetaddr = unpack('C4',$inetaddr); print "$af $port @inetaddr\n";
    while (<NS>) { print; print NS; }}}
```

Utilisation des sockets en mode non connecté SOCK_DGRAM

■ Fonctionnement

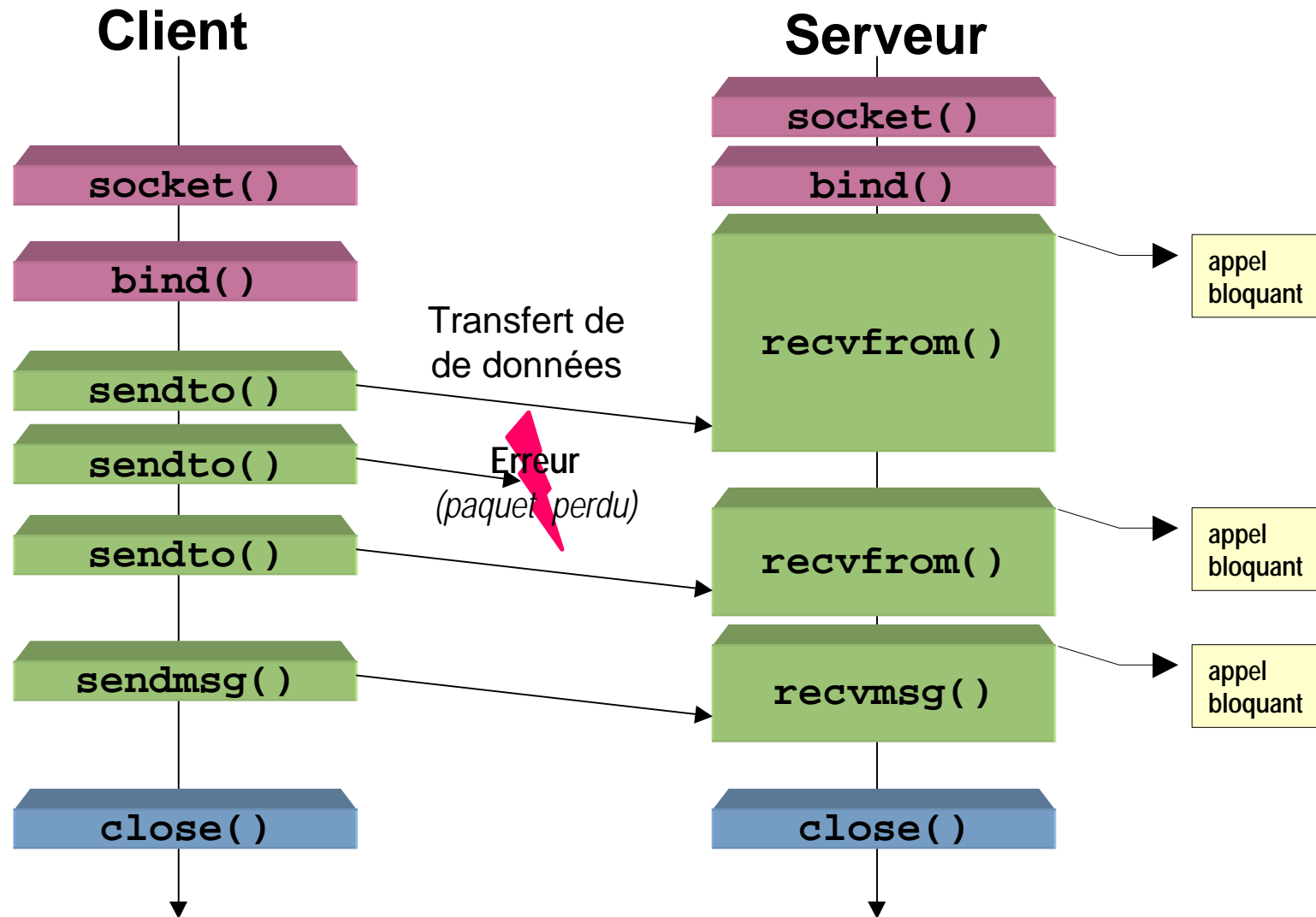
- Côté Emetteur

- crée un socket `socket()`
- associe une adresse au socket `bind()`
- envoi d'un message dans le socket `sendto()/sendmsg()`
- libère le socket `close()`

- Côté Récepteur

- crée un socket `socket()`
- associe une adresse au socket `bind()`
- écoute et réception d'un message `recvfrom()/recvmsg()`
- libère le socket `close()`

Utilisation des sockets en mode non connecté SOCK_DGRAM



Primitives d'émission/réception

■ Emission

- `int sendto(int descsock, void* buffer, size_t length, int flag, struct sockaddr* ptsockaddr, int lensockaddr)`
- transmet les données du *buffer* sur le socket *descsocket* vers l'adresse *ptsockaddr*
- `int sendmsg(int descsock, struct msghdr* msghdr, int flag)`
- transmet les données fragmentées décrites par *msghdr* sur le socket *descsocket* dans le domaine AF_UNIX

■ Réception

- `int recvfrom(int descsock, void* buffer, size_t length, int flag, struct sockaddr* ptsockaddr, int* ptlensockaddr)`
- reçoit les données du socket *descsocket* dans le *buffer*. L'adresse *ptsockaddr* est celui de l'expéditeur.
- `int recvmsg(int descsock, struct msghdr* msghdr, int flag)`
- reçoit du socket *descsocket* des données fragmentées décrites par *msghdr* dans le domaine AF_UNIX

Exemple: Code C de l 'Emetteur

```
/* Usage : udpsend hostrec 1234 "Hello World !" */
#include <sys/types.h>
#include <sys/socket.h>
#include <string.h>
main(int argc, char *argv[]) {
int sd;
struct hostent hp;
struct sockaddr_in recvaddr;
sd=socket(AF_UNIX, SOCK_DGRAM,0);

hp=gethostbyname(argv[1]);
recvaddr.sin_family=AF_INET; recvaddr.sin_port=htons(atoi(argv[2]));
memcpy(& recvaddr.sin_addr.s_addr, hp->h_addr, hp->hp_length);

sendto(sd, argv[3], strlen(argv[3]),&recvaddr, sizeof(recvaddr));
close(sd);
}
```


Paramétrage des sockets AF_INET

■ plusieurs niveaux de paramétrage

- SOL_SOCKET : directement sur les sockets
- IPPROTO_IP : directement sur la couche IP
- IPPROTO_TCP : directement sur la couche TCP
- IPPROTO_UDP : directement sur la couche UDP

■ Primitives

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

```
int getsockopt(int descsocket, int level, int option, void* poptionvalue, int* ptenoptionvalue)
```

- consultation de la valeur d'une option d'un niveau pour le socket

```
int setsockopt(int descsocket, int level, int option, void* poptionvalue, int lenoptionvalue)
```

- modification de la valeur d'une option d'un niveau pour le socket

Paramétrage des sockets AF_INET

Les options

■ Niveau Socket (SOL_SOCKET)

- SO_BROADCAST : autorise la diffusion (SOCK_DGRAM/AF_INET)
- SO_DONTROUTE : court-circuite le routage standard (SOCK_STREAM/AF_INET)
- SO_KEEPALIVE : teste l'activité sur une connexion (SOCK_STREAM/AF_INET)
- SO_LINGER : temps accordé au tentative d'envoi (SOCK_STREAM/AF_INET)
- SO_OOBINLINE : les msg MSG_OOB sont placés dans le tampon de réception du socket récepteur (SOCK_STREAM/AF_INET)
- SO_REUSEADDR : réutilisation d'une adresse locale pour réaliser la connexion
- SO_TYPE : type du socket
- SO_RCVBUF : taille du buffer de réception du socket
- SO_SNDBUF : taille du buffer d'émission du socket

■ Niveau TCP (IPPROTO_TCP)

- TCP_NODELAY : force l'envoi des données sans bufferisation (SOCK_STREAM/AF_INET)
- TCP_MAXSEG : connaître la taille maximale d'un segment TCP (SOCK_STREAM/AF_INET)

Conclusion

- Les sockets sont une API bas niveau pour échanger des données entre processus distribués sur un réseau
 - les messages et flux de données ne sont pas structurés
- Solutions plus adaptés pour du Client-Serveur
 - Des outils plus adaptés (RPC, RMI, CORBA) masquent au développeur les détails de la connection et le codage des messages

Bibliographie

Architecture et Principe d 'IP

■ Généralités

- Guy Pujolle, "Les réseaux", Ed Eyrolles , 3^{ème} éd., 2000, ISBN 2-212-09119-2
 - Chapitres 12 et 16 : IP dans les grandes lignes
 - mise à jour régulière

■ Détail

- W.R. Stevens, " TCP/IP Règles et Protocoles " Volume 1,2 et 3, Ed Vuibert (Addison-Wesley pour la VA de 1994), 1998, ISBN 2-7117-8639-0
 - très détaillé, plus que complet
 - mais commence à dater
- Douglas E. Comer, « Internetworking with TCP/IP volume I », Prentice Hall
- Douglas E. Comer & David L. Stevens, « Internetworking with TCP/IP volume II (Implementation and Design Issues) », Prentice Hall
- Douglas E. Comer & David L. Stevens, « Internetworking with TCP/IP volume III (Client / Server Programming & Apps.) », Prentice Hall

■ Liens

- <http://www.yahoo.com/Computers/Software/Protocols/IP/>

Bibliographie

Programmation des Sockets

■ UNIX et C

- Jean-Marie Rifflet, « La communication sous Unix », Ed EdiScience Intl, ISBN 2-84074-106-7
 - voir les chapitres 5, 6 et 10

■ Perl

- Larry Wall, Randal Schwartz, « Programming Perl », O'Reilly (la VF existe)
- Clinton Wong, “ Programmation de clients Web avec Perl ”, Ed Oreilly, 1997, ISBN 2-84177-050-8

■ Voir le Cours « Programmation Réseau en Java »