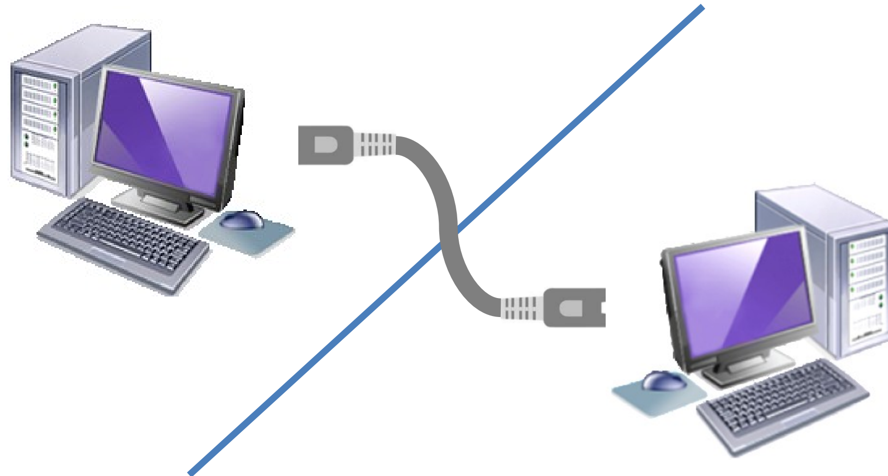
Communication réseau : Sockets



CNAM

Plan

- Contexte réseau
  - Concepts importants
  - Protocoles de transport
  - Architectures réseaux
- Communication via sockets
  - Types de Sockets
  - API Sockets
  - Exemples Client/Serveur
  - Sockets avancés

Contexte réseau

Communication réseau

- Communiquer entre 2 machines nécessite
  - Référencer la machine de destination
  - Indiquer l'application réceptrice des données
  - Echanger des informations avec le même langage

Identification réseau

- Chaque machine a une identification unique sur un réseau (e.g., internet)
  - Adresse IP : série de 4 nombres entre 0 et 255
    - 163.215.82.55 (notation pointée)
  - Plusieurs adresses par machine (fonction destination)
    - Adresse interne (localhost ou boucle locale) : 127.0.0.1
    - Adresse sur le réseau local : 192.168.0.x
    - Adresse Internet (adresse du routeur vers extérieur) : 85.79.27.105
  - Adresses découpées en deux parties
    - Adresse du réseau et adresse de la machine (notation CIDR indiquant nb bits partie réseau)
- Obtenir son adresse
  - Locale : *ipconfig* (sous windows) / *ifconfig* (Linux/Mac OS)
  - Internet : par exemple *www.whatismyip.com*

Identification réseau

- Adresses classiques des machines utilisent IPv4
  - Adresse IP sur 32 bits : 4 nombres entre 0 et 255
    - 163.215.82.55 (notation pointée)
  - Milliards d'adresses possibles
- Problème : le nombre d'adresses avec IPv4 n'est plus suffisant !
  - Nécessité d'avoir une adresse par personne
- Migration de plus en plus vers IPv6
  - Adresse IP sur 128 bits : caractères en hexa, paire octets séparés par « : »
    - 2005:0db8:c9d2:aee5:73e3:924a:a5ae:9238
  - *Nb adresses supérieur au nb d'étoiles de l'univers*

Ports réseau

- Plusieurs processus/services destinataires sur même machine
  - Données simultanément pour navigateur / client mail / jeux …
- Besoin d'indiquer le processus destinataire des informations
  - Utilisation **numéro de port** sur 16 bits (entier 1 à 65 536)
    - Ex : 21 pour FTP, 80 pour HTTP …
  - Protocoles/services fréquemment utilisés ont numéro de port dédié (indiqué dans */etc/services* sous Linux)
  - Utiliser un numéro de port libre (***supérieur à 1024***)
  - Rôle firewall : bloquer les ports non utilisés

Protocole réseau

- Protocole de communication
  - Ensemble de règles communes à 2 machines leur permettant d'échanger de l'information
- Certains nombre de protocoles existant
  - Haut niveau : manipulation spécifiée des données, documentation disponible (FTP, SMTP ...)
  - Bas niveau : manipulation des données à définir (octet par octet), difficile mais grande liberté (TCP/IP, UDP/IP)
- **Tous protocoles haut niveau utilisent soit TCP/IP ou UDP/IP**

Couches réseaux

Modèle OSI

Application

Présentation

Session

Transport

Réseau

Liaison

Physique

Modèle Internet

FTP

SNMP | NFS | DNS | Telnet | SMTP | X11

UDP | TCP

ICMP | IP | Routage | ARP | RARP

Technologies (Ethernet, X25, LS, ...)

Couche Application

Couche Hôte à Hôte

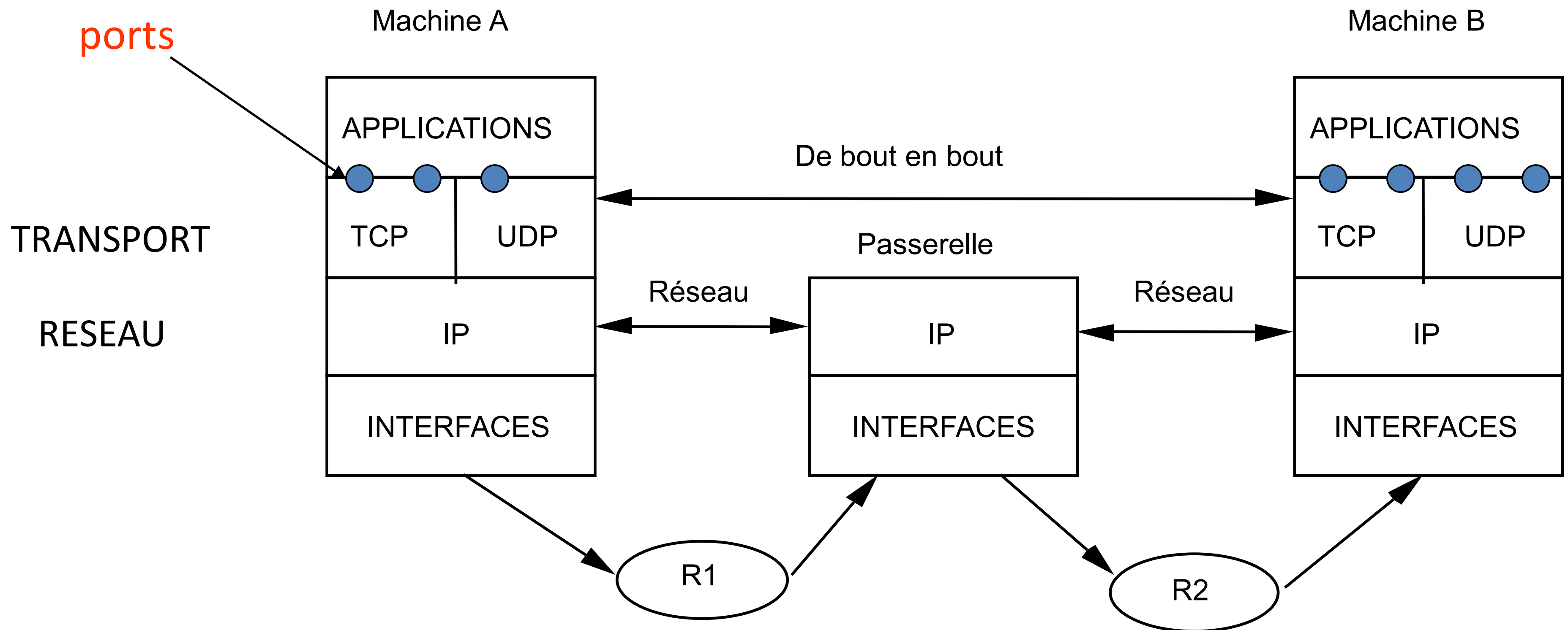
Couche Internet

Couche accès réseau

Quelques principes

- Echanges d'informations réalisés par morceaux
  - Niveau TCP/UDP : Envoie/réception de paquets de données
  - Niveau physique : paquets découpés en trames
  - Taille dépend de la couche MAC utilisée
- Deux méthodes d'échanges d'informations
  - TCP (Transmission Control Protocol) :
    - Etablie une connexion entre les 2 machines (**mode connecté**)
    - Système de contrôle des paquets, renvoie des paquets perdus
    - Echange plus coûteux et lent -> ajout d'informations de contrôle
  - UDP (User Datagram Protocol) :
    - Echange de données sans connexion (**mode non-connecté**)
    - Aucun contrôle des paquets (possibilité de perte ou désordonné)
    - Transmission des informations plus rapide (peu d'infos de contrôle)

Quelques principes

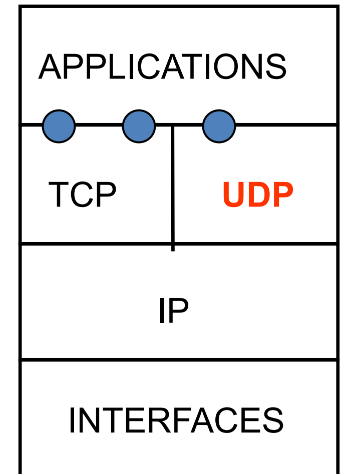


- IP : adressage de machine à machine via adresse IP
- TCP/UDP : adressage d'applications à applications
  - ➢ notion de ports (entier 16 bits)

La couche UDP (User Datagram Protocol)

- Protocole transport de bout en bout

    adressage d'application à application via les ports UDP
    Ports UDP réservés
    exemple : port 513 pour application who



- Protocole transport non fiable basé sur IP

    Datagrammes indépendants
    Perte de datagrammes, datagrammes dupliqués, pas de remise dans l'ordre d'émission

Analogie : le courrier

La couche TCP (Transport Control Protocol)

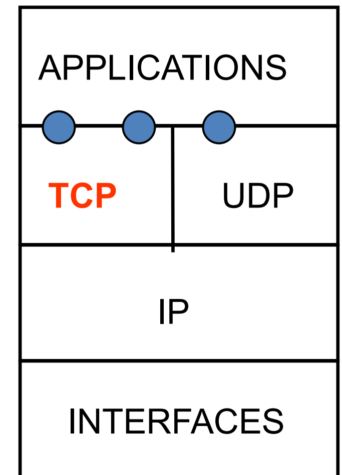
- Protocole transport de bout en bout

  adressage d'application à application via les ports TCP
  Ports TCP réservés
  exemples : port 21 application ftp

- Protocole transport fiable orienté connexion basé sur IP

  Connexion / Flux d'octets
  Pas de perte de messages, pas de duplication, remise dans l'ordre d'émission

  Analogie : le téléphone

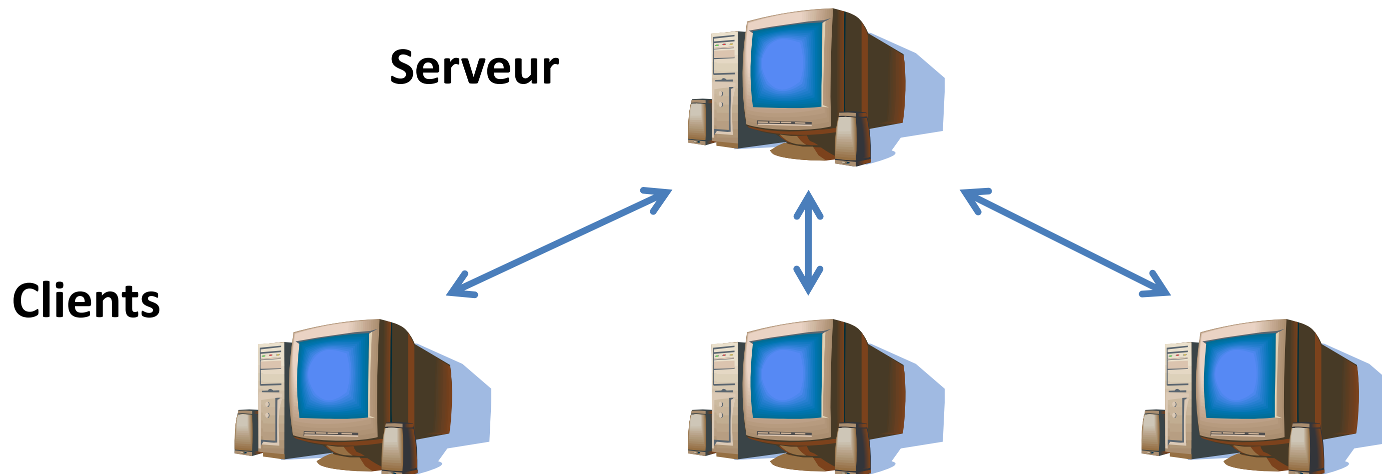


Utilisation TCP-UDP

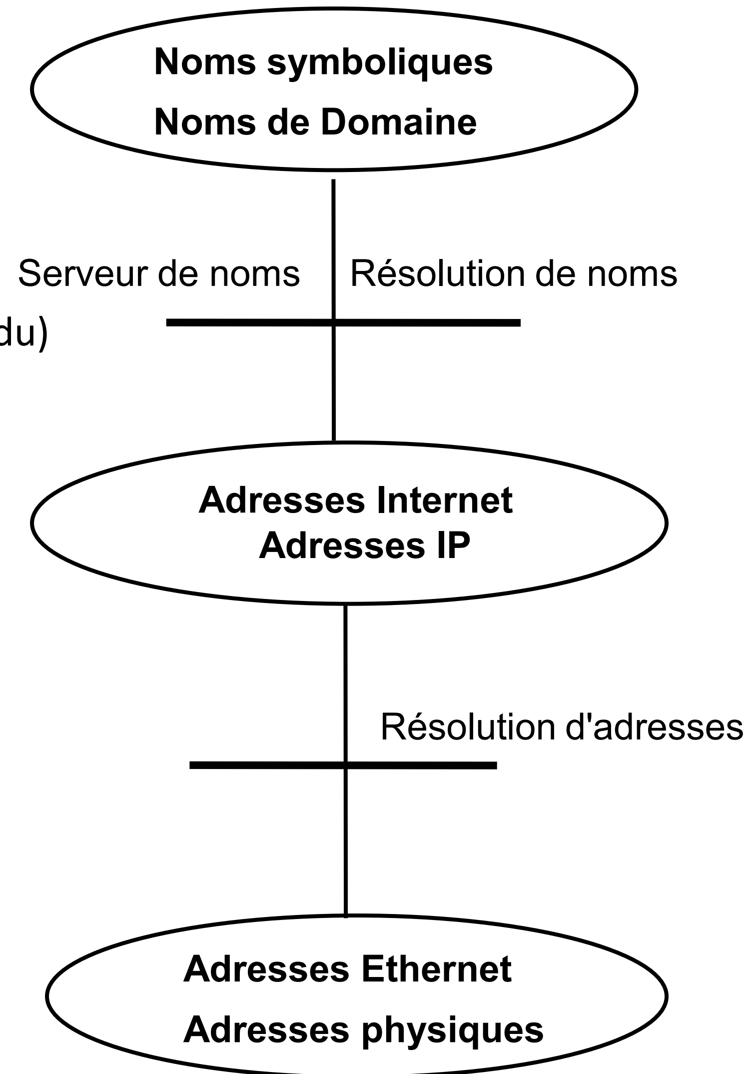
- Quand utiliser TCP ?
  - Si perte d'informations gênante pour application
  - Contrôle des paquets
  - Ex : transfert de fichiers, de texte, …
- Quand utiliser UDP ?
  - Si perte d'informations non gênante et besoin de communications rapides
  - Aucun contrôle des paquets
    - Sinon utiliser TCP ou à la charge de l'application
  - Ex : jeux en ligne, contenu en streaming …

Architectures réseaux

- Architecture Client/Serveur
  - Architecture la plus classique
  - Programmes clients communiquent via un programme serveur (au centre de l'architecture)



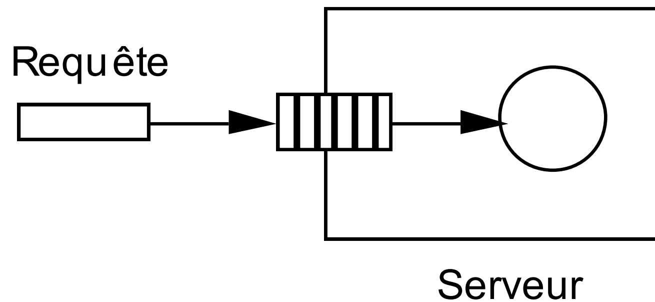
Architectures réseaux



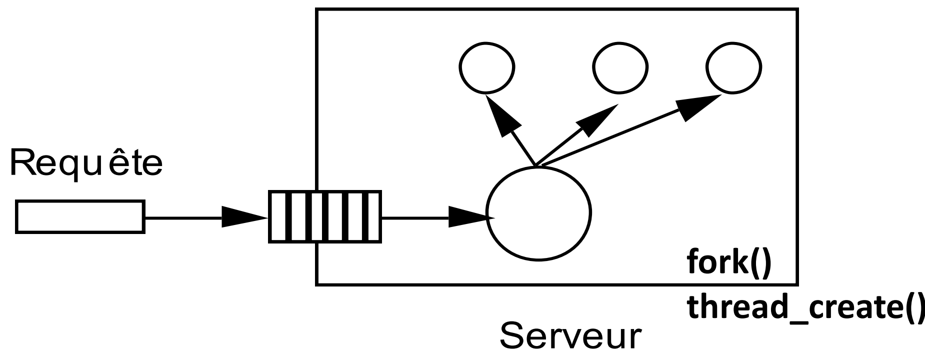
Structure hiérarchique fondée sur le domaine (DNS)
objet.sous-domaine.domaine fermi.cnam.fr
objet : nom d'une machine.
domaine : géographique (fr, jp) ou institutionnel (com, mil, edu)

Adresses IP : 32 bits
Paire (adresse réseau, adresse machine dans le réseau)
Forme pointée : ***10000000 00001010 00000010 00011110***
128.10.2.30

Architectures réseaux



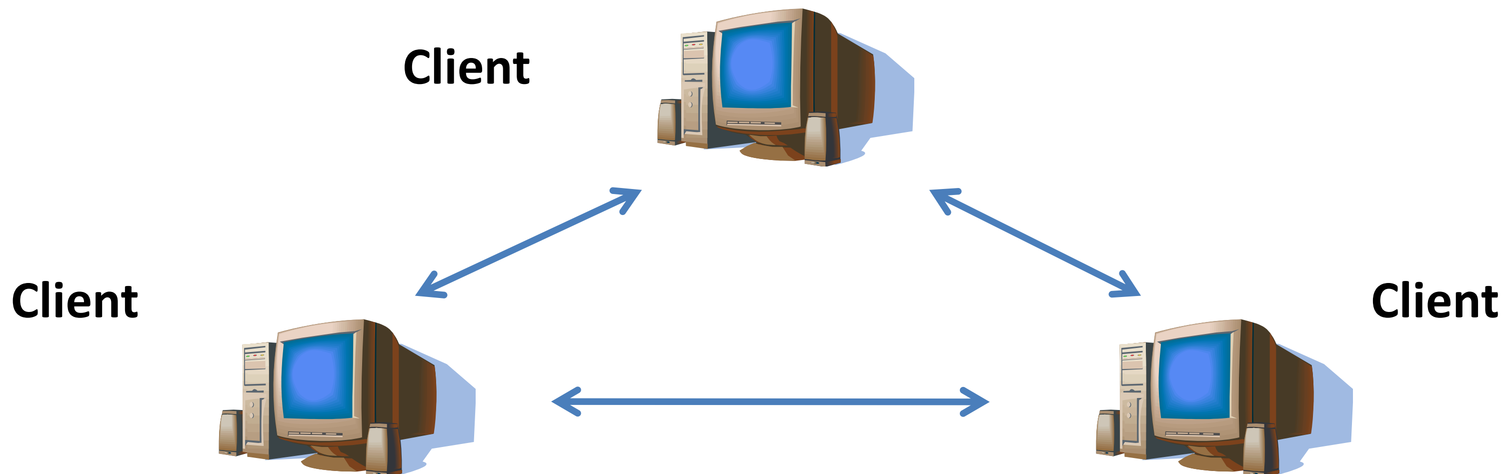
Serveur Itératif
Un seul processus effectue la réception, le traitement et l'émission



Serveur Parallèle
Le processus père effectue la réception. Il crée un fils pour réaliser le traitement et l'émission de la réponse.

Architectures réseaux

- Architecture Peer-To-Peer (P2P)
  - Architecture décentralisée
  - Pas de serveur centralisateur
  - Chaque programme client intègre partie serveur pour communication inter-clients (plus complexe)



Stockage informations

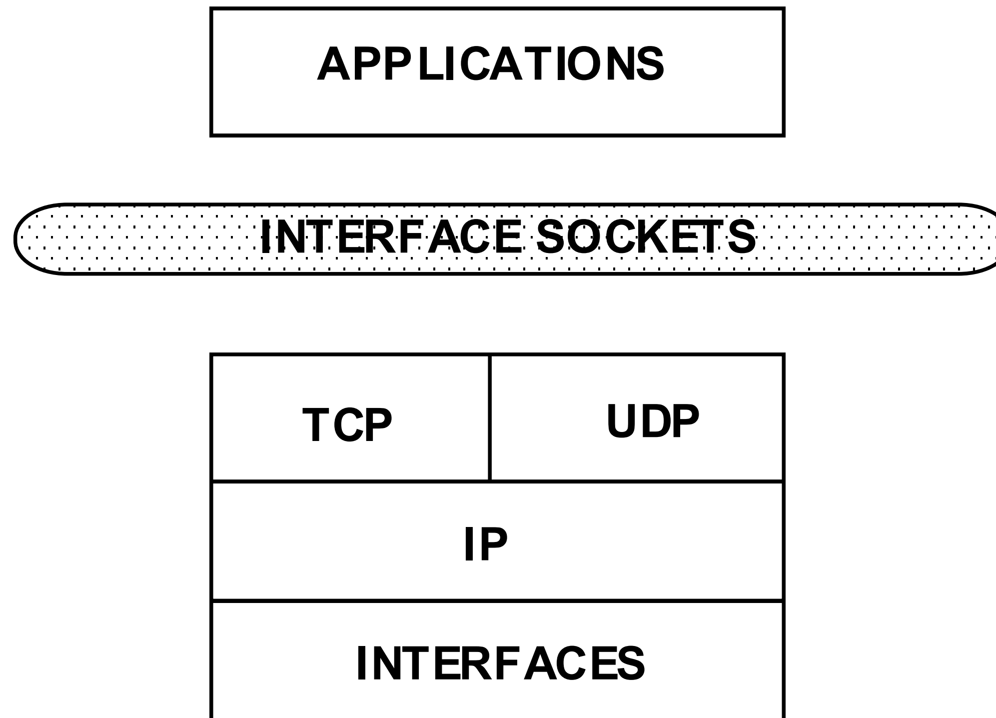
- Constructeurs peuvent utiliser des façons différentes pour stocker l'information
  - Ordre différent pour stocker octets
  - Big-Endian : en partant octet poids fort
    - Stockage de « a42b » : a4 puis 2b
  - Little-Endian : en partant octet poids faible
    - Stockage de « a42b » : 2b puis a4
  - Ex : Intel -> Little-Endian, Motorola -> Big-Endian
- Besoin de choisir un ordre d'échange des données pour communications réseaux (*Network Byte Order*)
  - Utilisation de primitives faisant la conversion

Stockage informations

- Primitives de conversion
  - htons() : **h**ost **to** **n**etwork **s**hort
  - htonl() : **h**ost **to** **n**etwork **l**ong
  - ntohs() : **n**etwork **to** **h**ost **s**hort
  - ntohl() : **n**etwork **to** **h**ost **l**ong

Communication par Sockets

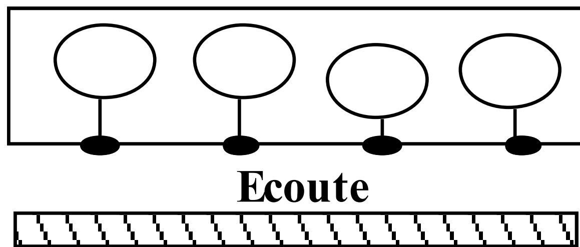
L'interface socket : situation

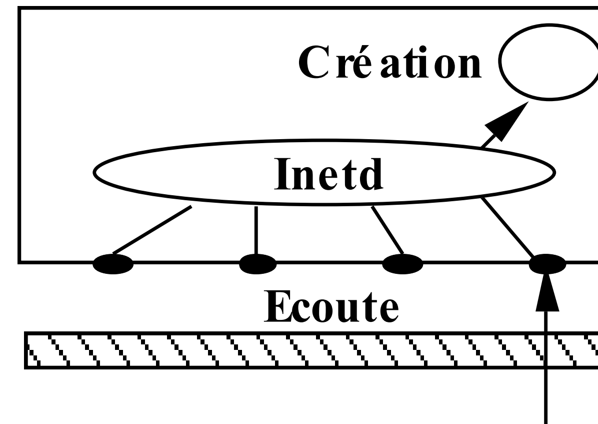


- Interface de programmation (ensemble de primitives)
- Point de communication (adresse d'application)
- Compatible SGF

L'interconnexion de réseau : quelques principes

La couche Application





Un démon par service
=> Encombrement de la table des processus

Le démon inetd
en écoute sur les différents ports
crée le serveur sollicité

Qu’est ce qu’une socket

- Connexion réseau représentée par descripteur (comme pour fichier sur disque)
  - « *Tout est fichier dans un système Unix* »
  - Descripteur de fichier est un entier associé à un fichier ouvert dans le système
  - Possibilité d’utiliser même primitive que pour fichiers classiques
    - Pour avoir plus de contrôle, utilisation primitives dédiées pour communications en réseau

Types de socket

- Plusieurs types de socket
- 3 trois types principaux
  - Stream sockets : communication connectée
    - Utilise le protocole TCP, dénommé SOCK_STREAM (http://tools.ietf.org/html/rfc793)
  - Datagram sockets : communication non-connectée
    - Utilise le protocole UDP, dénommé SOCK_DGRAM (http://tools.ietf.org/html/rfc791)
  - Raw sockets :
    - Permet d'accéder directement à la couche IP (droit super-utilisateur), dénommé SOCK_RAW
- Les deux premiers sont les plus utilisés

Structure de données pour adresses

- Définition de la structure de données stockant adresse(s) de l'hôte
  - Chaînage entre structures (via pointeur ***ai_next***)
  - Permet d'utiliser adresses IPv4 ou IPv6 (indiqué par ***ai_family***)

```
struct addrinfo {
        int ai_flags;                   // AI_PASSIVE, AI_CANONNAME, etc.
        int ai_family;                  // AF_INET, AF_INET6, AF_UNSPEC
        int ai_socktype;                // SOCK_STREAM, SOCK_DGRAM
        int ai_protocol;                // use 0 for "any"
        size_t ai_addrlen;              // size of ai_addr in bytes
        struct sockaddr *ai_addr;       // struct sockaddr_in or _in6
        char *ai_canonname;             // full canonical hostname

        struct addrinfo *ai_next;       // linked list, next node
};
```

Structure de données pour adresses

- Structure *addrinfo* est rempli en faisant appel à
  - La fonction **getaddrinfo()**, permet de ne plus remplir manuellement structure et être IPv4 ou IPv6 compatible
  - Utilisation de la première structure correcte de la liste (voir exemple)
  - Pointeur ***ai_addr*** référence structure ***sockaddr*** contenant l'adresse IP à utiliser
    - ***sa_family*** : AF_INET pour adresse IPv4, AF_INET6 pour adresse IPv6
    - ***sa_data*** : contient adresse IP et numéro de port

```
struct sockaddr {
        unsigned short sa_family;  // address family, AF_xxx
        char sa_data[14];          // 14 bytes of protocol address
};
```

- Pb : structure pas pratique à remplir manuellement -> ***sockaddr_in***

Structure de données pour adresses

- Structure ***sockaddr_in*** utilisée pour IPv4
  - Voir ***sockaddr_in6*** pour IPv6

```
struct sockaddr_in {
        short int   sin_family;              // Address family, AF_INET
        unsigned short int sin_port;    // Port number, Network Byte Order
        struct in_addr        sin_addr;                 // Internet address
        unsigned char         sin_zero[8];              // Same size as struct sockaddr
};
```

  - Possibilité de passer de l'un à l'autre en faisant « cast »
  - ***sin_zero*** : complétée avec zéros via fonction **memset()**
  - ***sin_port*** : à remplir en utilisant fonction **htons()**

```
struct in_addr {
        uint32_t  s_addr;                // that's a 32-bit int (4 bytes)
};
```

Structure de données pour adresses

- Structure ***sockaddr_in6*** utilisée pour IPv6
  - Structure similaire pour adresse IPv6

```
struct sockaddr_in6 {
        u_int16_t        sin6_family;     // address family, AF_INET6
        u_int16_t        sin6_port;       // port number, Network Byte Order
        u_int32_t        sin6_flowinfo;   // IPv6 flow information
        struct in6_addr  sin6_addr;       // IPv6 address
        u_int32_t        sin6_scope_id;   // Scope ID
};

struct in6_addr {
        unsigned char s6_addr[16]; // IPv6 address
};
```

Structure de données pour adresses

- Conversion adresse IP au format indiqué par ***sockaddr_in*** ou ***sockaddr_in6***

```
struct sockaddr_in sa;        // IPv4
struct sockaddr_in6 sa6;      // IPv6

inet_pton(AF_INET, "192.0.2.1", &(sa.sin_addr));                    // IPv4
inet_pton(AF_INET6, "2001:db8:63b3:1::3490", &(sa6.sin6_addr));     // IPv6
```

  - ***pton*** : pour **P**resentation **to** **N**etwork
  - Code de retour doit être supérieur à 0

Structure de données pour adresses

- Récupérer adresse IP sous forme de caractère indiqué par ***sockaddr_in*** ou ***sockaddr_in6***
  - Avec ***INET_ADDRSTRLEN*** ou ***INET6_ADDRSTRLEN*** taille max adresse IPv4 ou IPv6

```
// IPv4:
char ip4[INET_ADDRSTRLEN];          // space to hold the IPv4 string
struct sockaddr_in sa;              // pretend this is loaded with something
inet_ntop(AF_INET, &(sa.sin_addr), ip4, INET_ADDRSTRLEN);
printf("The IPv4 address is: %s\n", ip4);

// IPv6:
char ip6[INET6_ADDRSTRLEN];         // space to hold the IPv6 string
struct sockaddr_in6 sa6;            // pretend this is loaded with something
inet_ntop(AF_INET6, &(sa6.sin6_addr), ip6, INET6_ADDRSTRLEN);
printf("The address is: %s\n", ip6);
```

Changements pour compatibilité IPv6

- Utiliser fonction ***getaddrinfo()*** au lieu de ***gethostbyname()***
- Utiliser fonction ***getnameinfo()*** au lieu de ***gethostbyaddr()***

- Ajouter le chiffre « 6 » dans les macros ou types :
  - Utiliser ***AF_INET6*** au lieu de ***AF_INET***
  - Utiliser ***in6addr_any*** au lieu de ***INADDR_ANY***
  - Utiliser structure ***sockaddr_in6*** au lieu de ***sockaddr_in***
  - Utiliser ***inet_pton()*** au lieu de ***inet_aton()*** ou ***inet_addr()***
  - Utiliser ***inet_ntop()*** au lieu de ***inet_ntoa()***

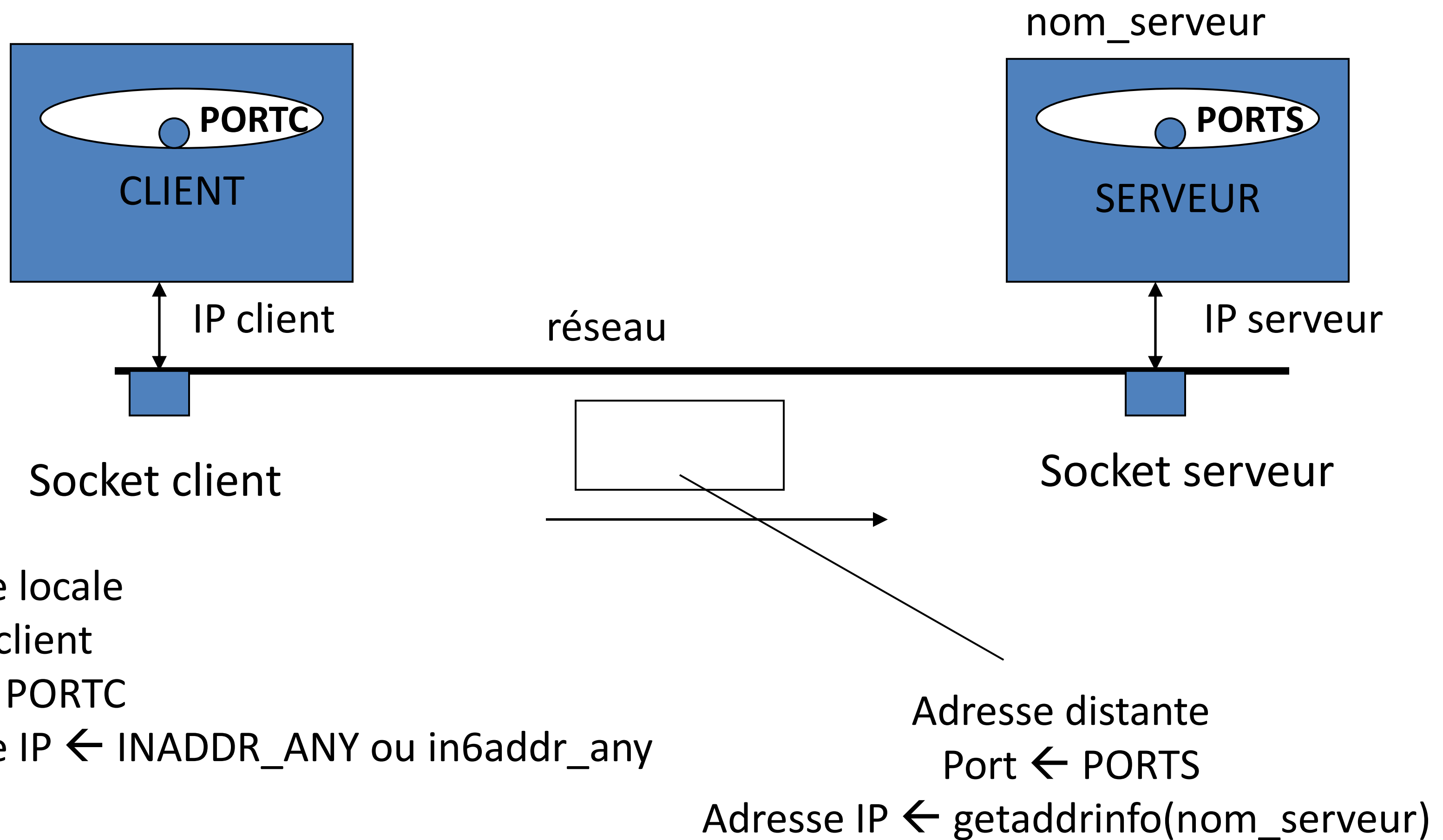
Initialisation structures pour socket

- Initialisation de la structure ***addrinfo*** :

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

int getaddrinfo(const char *node,              // e.g. "www.example.com" or IP
                const char *service,           // e.g. "http" or port number
                const struct addrinfo *hints, // additional infos on socket type
                struct addrinfo **res);
```

- Retour : 0 si succès, erreur sinon récupérable via *gai_strerror()*
- Résultat de la fonction est retournée dans champ ***res***

Attachement d'une adresse d'application



Adresse locale
Socket client
Port ← PORTC
Adresse IP ← INADDR_ANY ou in6addr_any

Communication en mode connecté

Serveur | **Client**

Socket ()

Bind ()

Listen ()

Socket ()

Sock = socket (AF_INET, SOCK_STREAM, IPPROTO_TCP)

Serveur en attente de connexion

Accept() ← Connect()

Etablissement de connexion avec échange
des adresses participantes

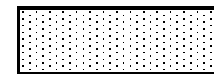
Connexion établie

Recv() ← Send()

Send() → Recv()

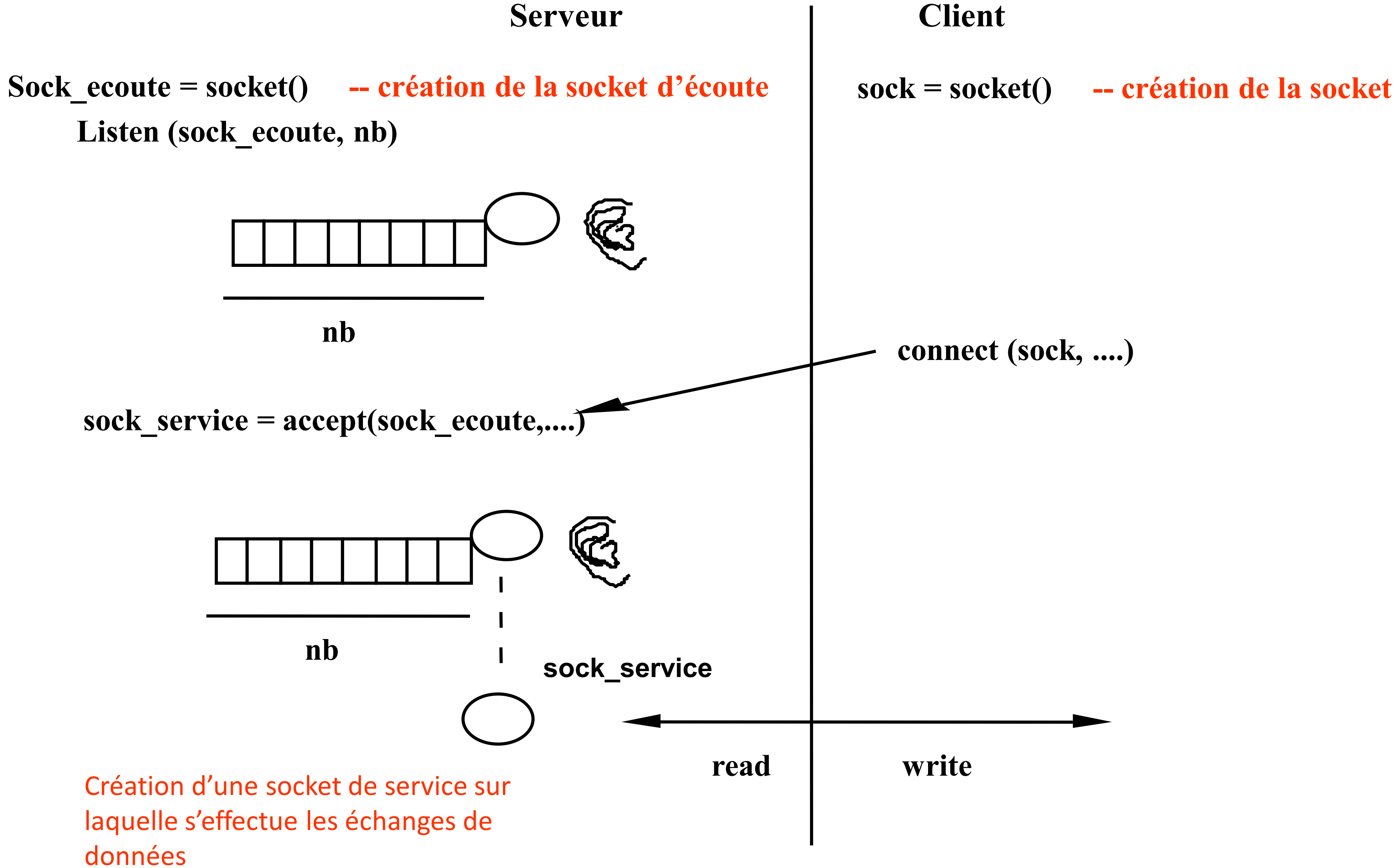
Close() | Close()

→ Les messages échangés ne contiennent pas l'adresse du destinataire



Appels bloquants

Communication en mode connecté



Création d'une socket

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

sockfd = socket(int af, int type, int protocole)
```

Famille d'adresse de la socket :
AF_INET : protocole internet
AF_UNIX : protocole interne à UNIX

Type de service :
SOCK_STREAM
SOCK_DGRAM
SOCK_RAW

IPPROTO_TCP
IPPROTO_UDP
0 par défaut

- **Retourne un descripteur de socket ayant les mêmes propriétés qu'un descripteur de fichier (héritage) accessible par le créateur et les fils de celui-ci**
- **Retourne -1 si erreur**

Exemple création d’une socket

```
#define PORT “3490”

int status,sockfd;
struct addrinfo hints;
struct addrinfo *res;

memset(&hints, 0, sizeof(hints)); // make sure the struct is empty
hints.ai_family = AF_UNSPEC;       // don't care IPv4 or IPv6, otherwise use AF_INET or AF_INET6
hints.ai_socktype = SOCK_STREAM; // TCP stream sockets
hints.ai_flags = AI_PASSIVE;        // fill in my IP, or use IP address (first parameter of getaddrinfo())

status = getaddrinfo(NULL, PORT, &hints, &res);

if (status!=0)
{           fprintf(stderr, "getaddrinfo error: %s\n", gai_strerror(status));
            exit(1);}

sockfd = socket(res->ai_family, res->ai_socktype, res->ai_protocol);
```

Informations sur extrémités socket

```
#include <sys/socket.h>
int getpeername(int sockfd, struct sockaddr *addr, int *addrlen);
```

Socket d'écoute côté serveur → sockfd

Structure avec adresse et port rétournés → *addr

Taille structure adresse (en octets) → *addrlen

- Retourne l'adresse et port nœud connecté à la socket
- Retourne 0 si succès, -1 sinon

```
#include <unistd.h>
int gethostname(char *hostname, size_t size);
```

Adresse buffer où stocker nom de l'hôte → *hostname

Taille de la chaîne de caractère → size

- Retourne le nom de la machine hôte exécutant le programme
- Retourne 0 si succès, -1 sinon

Attachement d’une adresse d’application

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

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

Longueur de la structure mon_adr

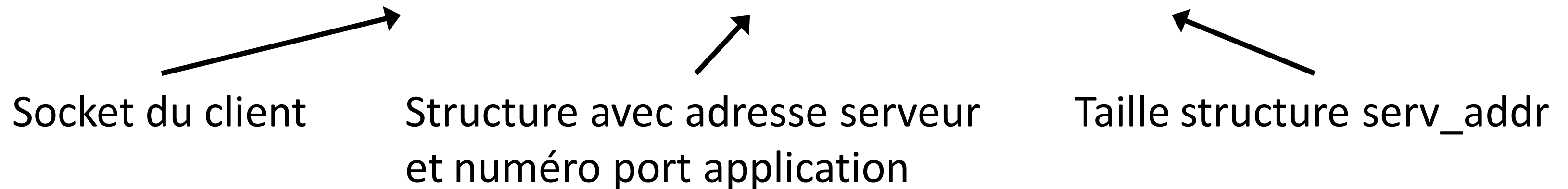
Pointeur en mémoire sur l’adresse
dans le domaine TCP/IP

- L’opération d’attachement permet d’étendre le groupe des processus pouvant accéder à la socket.
- Fonction à utiliser uniquement pour le programme serveur (client utilise fonction ***connect()***)
- Retourne : 0 si succès, -1 si erreur

Communication en mode connecté

Connexion du client sur une application serveur

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

int connect(int sockfd, struct sockaddr *serv_addr, int addrlen);
```



- Client essaie de se connecter à une application serveur
- Retourne 0 si succès, -1 sinon

Communication en mode connecté

Ecoute et acceptation connexion de clients

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

```
int listen(int sockfd, int backlog);
```

Socket d'écoute côté serveur → sockfd

Nb max de connexion en attente de traitement (20 max système, en pratique entre 5 et 10) → backlog

- Serveur écoute et est en attente d'une demande de connexion d'un client
- Retourne 0 si succès, -1 sinon

```
newfd = accept(int sockfd, struct sockaddr *client_addr, socklen_t *addrlen);
```

Socket d'écoute côté serveur → sockfd

Structure avec adresse et port du client → client_addr

Taille structure client_addr → addrlen

- Accepte une connexion client et retourne nouveau descripteur de socket de service (newfd) pour traiter requête du client
- Retourne -1 si echec

Communication en mode connecté

Envoie et réception de données sur une socket

```
int send(int sockfd, const void *msg, int len, int flags);
```

- **sockfd** : Socket où envoyer les données
- **msg** : Pointeur sur les données à envoyer
- **len** : Taille des données à envoyer (en octets)
- **flags** : Par défaut à 0, données normales

- Envoie des données sur sockfd indiquées par pointeur msg
- Retourne nombre d’octets envoyés si succès (***peut être < len***), -1 sinon

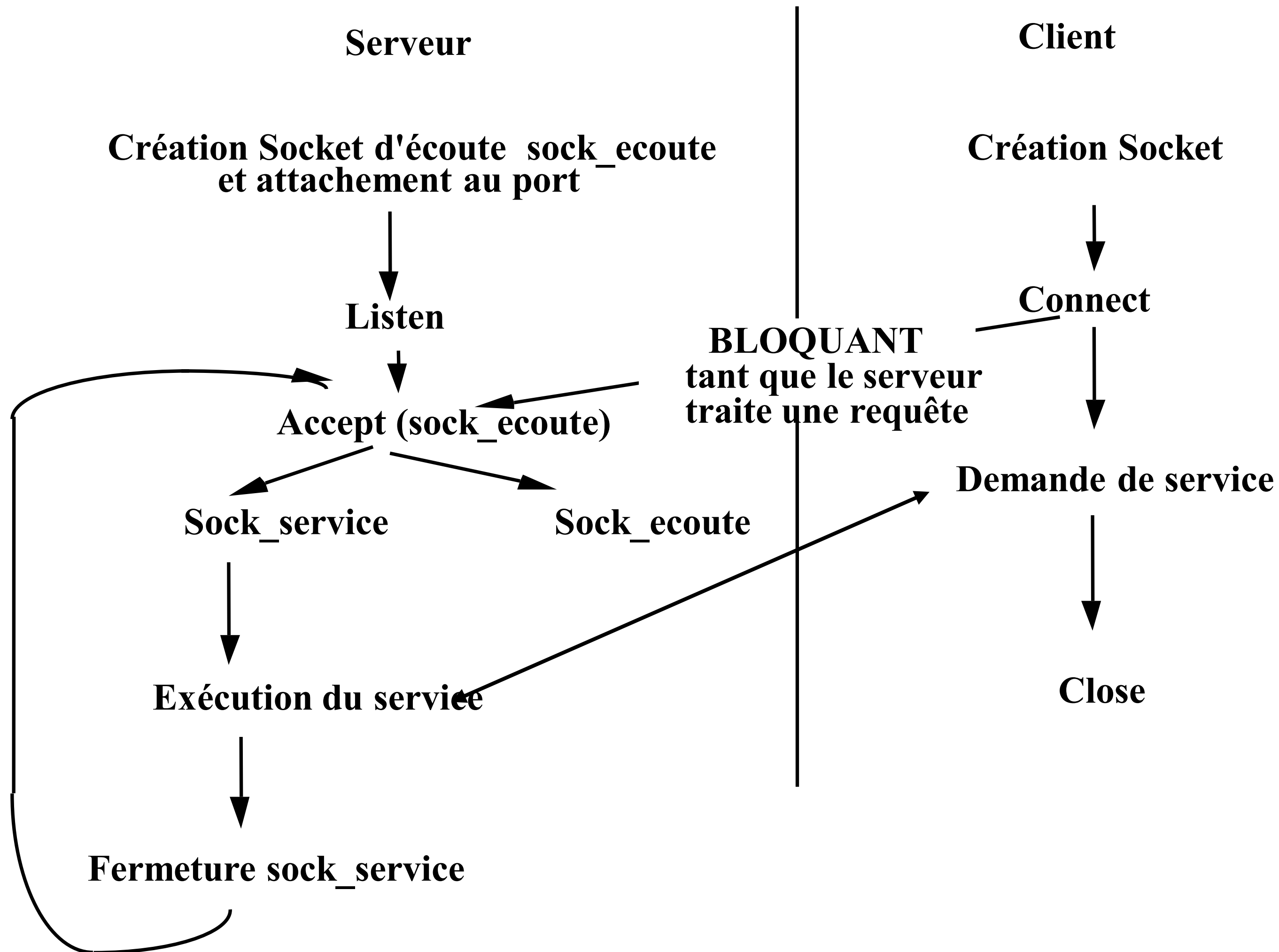
---

```
int recv(int sockfd, void *buf, int len, int flags);
```

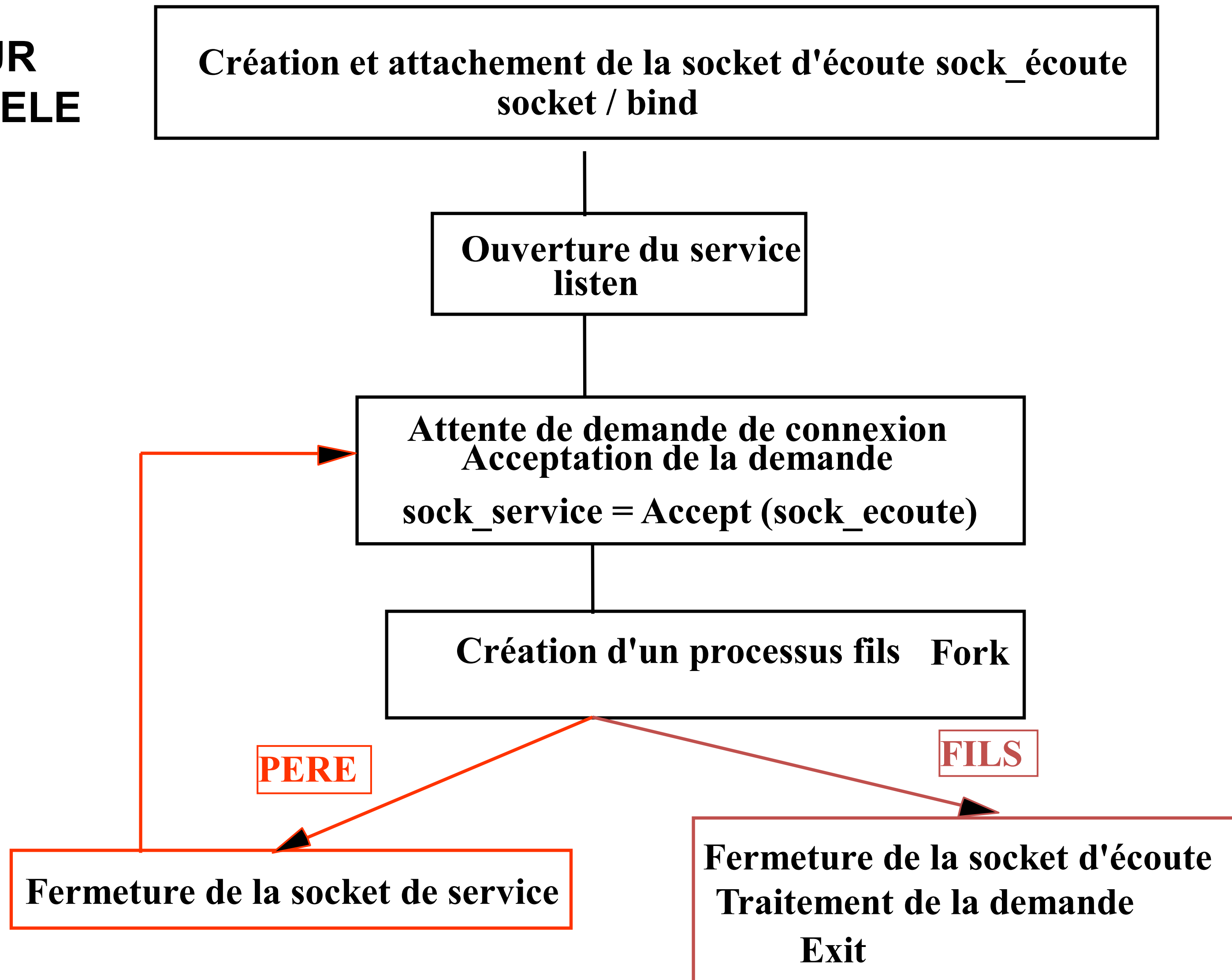
- **sockfd** : Socket de réception de données
- **buf** : Pointeur buffer stocker données
- **len** : Taille maximale du buffer (en octet)
- **flags** : Par défaut à 0 données normales

- Réceptionne des données d’une socket et les stocke dans un buffer (***appel bloquant***)
- Retourne nb octets lu si succès, 0 indique socket a été fermée, -1 sinon

Communication en mode connecté



Communication en mode connecté



Communication en mode connecté

```
/*********** CLIENT TCP  **************/

#include      <stdio.h>
#include      <sys/types.h>
#include      <sys/socket.h>
#include      <netinet/in.h>
#include      <netdb.h>
#include      <netdb.h>

#define SERVEUR “127.0.0.1”
#define PORTS “2058”

main()
{
int sockfd, rv;
struct addrinfo hints, *servinfo;
char buf[100];

memset(&hints, 0, sizeof(hints));
hints.ai_family = AF_UNSPEC;
hints.ai_socktype = SOCK_STREAM;
rv = getaddrinfo(SERVEUR, PORTS, &hints, &servinfo);

if (rv != 0)
{ fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(rv));
return 1;}
```

```
// Création  socket  et  attachement
if ((sockfd = socket(servinfo->ai_family, servinfo->
ai_socktype,  servinfo->ai_protocol)) == -1) {
        perror(« client: socket");
      }
if ((connect(sockfd, servinfo->ai_addr, servinfo->
ai_addrlen) == -1) {
        close(sockfd);
        perror("client: connect");
      }

freeaddrinfo(servinfo);    // Libère structure

if ((numbytes = recv(sockfd, buf, 100-1, 0)) == -1)
{   perror("recv");
    exit(1);
}
printf(‘’Message reçu : %s\n’’,buf);

close(sockfd);
return 0;
}
```

Communication en mode connecté

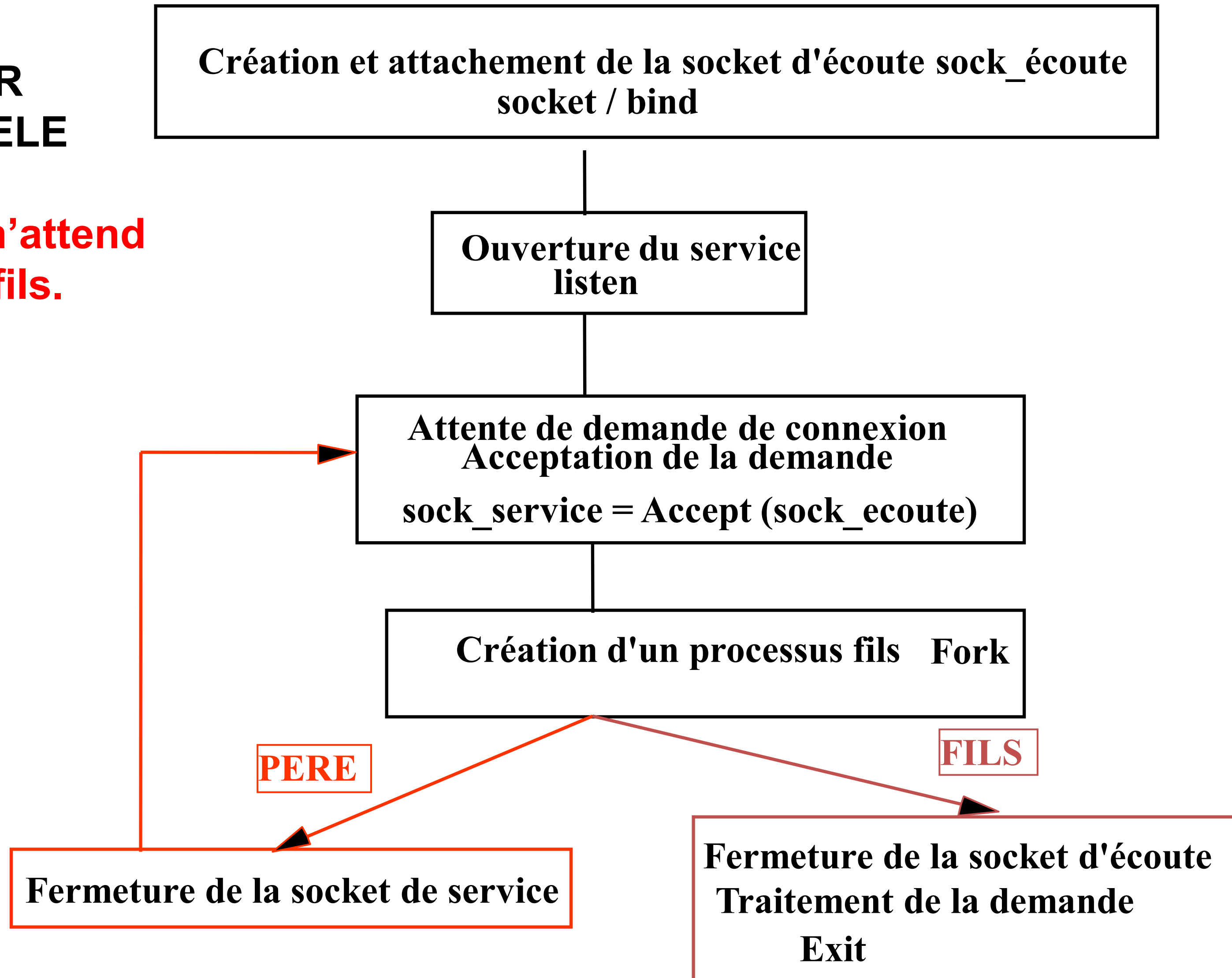
```
/******* SERVEUR TCP  **************/

#include      <stdio.h>
#include      <sys/types.h>
#include      <sys/socket.h>
#include      <netinet/in.h>
#include      <netdb.h>
#include      <netdb.h>

#define PORTS "2058"

main()
{
int sockfd, new_fd, rv, sin_size;
struct addrinfo hints, *servinfo, *p;
struct sockaddr their_adr;

memset(&hints, 0, sizeof(hints));
hints.ai_family = AF_UNSPEC;
hints.ai_socktype = SOCK_STREAM;
hints.ai_flags = AI_PASSIVE;            // use my IP
rv = getaddrinfo(NULL, PORTS, &hints, &servinfo);

if (rv != 0)
{ fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(rv));
return 1;}
```

```
// Création  socket  et  attachement
for(p = servinfo; p != NULL; p = p->ai_next)
{
   if ((sockfd = socket(p->ai_family, p->ai_socktype,
   p->ai_protocol)) == -1) {
      perror("server: socket");
      continue;}
   if (bind(sockfd, p->ai_addr, p->ai_addrlen) == -1)
   {   close(sockfd);
       perror("server: bind");
       continue;}
   break;}
if (p == NULL) {
   fprintf(stderr, "server: failed to bind\n");
   return 2;}
freeaddrinfo(servinfo);   // Libère structure

listen(sockfd, 5);

while(TRUE)
{  sin_size = sizeof(their_addr);
   new_fd = accept(sockfd, &their_addr, &sin_size);
   if(!fork())
   {  close(sockfd);
      send(new_fd, "Hello!", 6, 0);
      close(new_fd);  exit(0);  } } }
```

Communication en mode connecté

SERVEUR PARALLELE

Le père n’attend pas son fils.

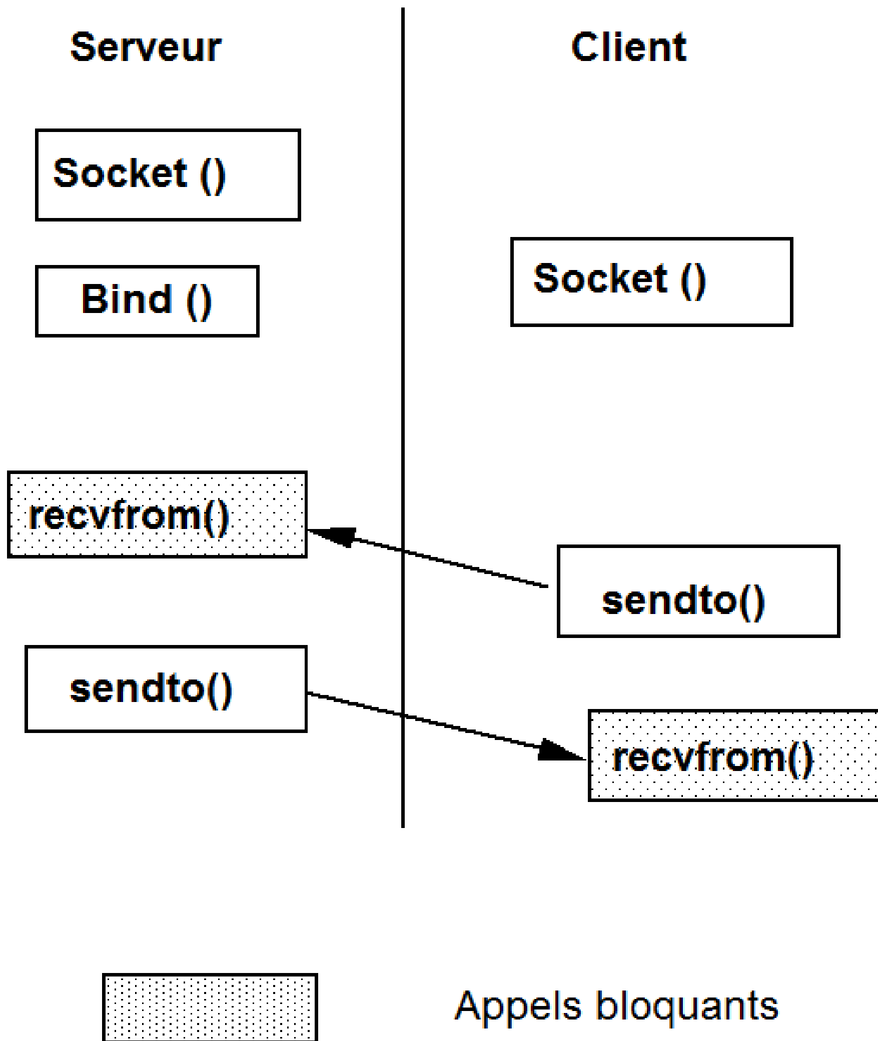


Communication en mode connecté

```
/******* SERVEUR TCP **************/
#include    <stdio.h>
#include    <sys/types.h>
#include    <sys/socket.h>
#include    <netinet/in.h>
#include    <netdb.h>
#include    <netdb.h>
#include    <signal.h>

#define PORTS "2058"

main()
{
int sockfd, new_fd, rv, sin_size;
struct addrinfo hints, *servinfo, *p;
struct sockaddr their_adr;

memset(&hints, 0, sizeof(hints));
hints.ai_family = AF_UNSPEC;
hints.ai_socktype = SOCK_STREAM;
hints.ai_flags = AI_PASSIVE;          // use my IP
rv = getaddrinfo(NULL, PORTS, &hints, &servinfo);

if (rv != 0)
{ fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(rv));
return 1;}
```

```
// Création  socket  et  attachement
for(p = servinfo; p != NULL; p = p->ai_next)
{
   if ((sockfd = socket(p->ai_family, p->ai_socktype,
   p->ai_protocol)) == -1) {
      perror("server: socket");
      continue;}
   if (bind(sockfd, p->ai_addr, p->ai_addrlen) == -1)
   {   close(sockfd);
       perror("server: bind");
       continue;}
   break;}
if (p == NULL) {
   fprintf(stderr, "server: failed to bind\n");
   return 2;}
freeaddrinfo(servinfo);   // Libère structure

listen(sockfd, 5);
signal(SIGCHLD, SIG_IGN);
while(TRUE)
{  sin_size = sizeof(their_addr);
   new_fd = accept(sockfd, &their_addr, &sin_size);
   if(!fork())
   {  close(sockfd);
      send(new_fd, "Hello!", 6, 0);
      close(new_fd);  exit(0);  } } }
```

Communication en mode datagramme



Sock = socket (AF_INET, SOCK_DGRAM, IPPROTO_UDP)

Pas d'établissement de connexion

→ Chaque message échangé contient l'adresse du destinataire

Communication en mode non-connecté

Envoie et réception de données sur une socket

int sendto(int sockfd, void *msg, int len, int flags, struct sockaddr *to, socklen_t tolen);

- sockfd : Socket où envoyer les données
- msg : Pointeur données à envoyer
- len : Taille données (en octets)
- flags : Par défaut à 0, données normales
- to, tolen : Structure adresse destinataire et Taille structure

* Envoie des données sur sockfd indiquées par pointeur msg
* Retourne nombre d'octets envoyés si succès (***peut être < len***), -1 sinon

---

int recvfrom(int sockfd, void *buf, int len, int flags, struct sockaddr *from, int *fromlen);

- sockfd : Socket de réception de données
- buf : Pointeur buffer stocker données
- len : Taille maximale (en octet)
- flags : Par défaut à 0 données normales
- from, fromlen : Structure adresse émetteur et Taille structure

* Réceptionne des données d'une socket et les stocke dans un buffer (***appel bloquant***)
* Retourne nb octets lu si succès, 0 indique socket a été fermée, -1 sinon

Communication en mode non-connecté

```
/*********** CLIENT UDP  **************/
#include      <stdio.h>
#include      <sys/types.h>
#include      <sys/socket.h>
#include      <netinet/in.h>
#include      <netdb.h>
#include      <netdb.h>

#define SERVEUR "127.0.0.1"
#define PORTS "2058"

main()
{
int sockfd, rv;
struct addrinfo hints, *servinfo;
char buf[100];

memset(&hints, 0, sizeof(hints));
hints.ai_family = AF_UNSPEC;
hints.ai_socktype = SOCK_DGRAM;
rv = getaddrinfo(SERVEUR, PORTS, &hints, &servinfo);

if (rv != 0)
{ fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(rv));
return 1;}
```

```
// Création  socket  et  attachement
if ((sockfd = socket(servinfo->ai_family, servinfo->
ai_socktype, servinfo->ai_protocol)) == -1) {
        perror(« client: socket");
      }

freeaddrinfo(servinfo);    // Libère structure

if ((numbytes = sendto(sockfd, ''HELLO!'', 6, 0, servinfo->
ai_addr, servinfo->ai_addrlen)) == -1)
{   perror("sendto");
    exit(1);
}

close(sockfd);
return 0;
}
```

Communication en mode non-connecté

```
/******* SERVEUR UDP  **************/
#include      <stdio.h>
#include      <sys/types.h>
#include      <sys/socket.h>
#include      <netinet/in.h>
#include      <netdb.h>
#include      <netdb.h>

#define PORTS "2058"

main()
{
int sockfd, new_fd, rv, addr_len;
struct addrinfo hints, *servinfo, *p;
struct sockaddr their_adr;
char buf[100];

memset(&hints, 0, sizeof(hints));
hints.ai_family = AF_UNSPEC;
hints.ai_socktype = SOCK_DGRAM;
hints.ai_flags = AI_PASSIVE;               // use my IP
rv = getaddrinfo(NULL, PORTS, &hints, &servinfo);

if (rv != 0)
{ fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(rv));
return 1;}
```

```
// Création  socket  et  attachement
for(p = servinfo; p != NULL; p = p->ai_next)
{
   if ((sockfd = socket(p->ai_family, p->ai_socktype,
   p->ai_protocol)) == -1) {
      perror("server: socket");
      continue;}
   if (bind(sockfd, p->ai_addr, p->ai_addrlen) == -1)
   {   close(sockfd);
       perror("server: bind");
       continue;}
   break;}
if (p == NULL) {
   fprintf(stderr, "server: failed to bind\n");
   return 2;}
freeaddrinfo(servinfo);    // Libère structure

while(TRUE)
{
   if ((numbytes = recvfrom(sockfd, buf, 100-1, 0,
                               &their_addr, &addr_len)) == -1)
   {  perror("recv");
      exit(1);}
   printf("Chaine reçue %s\n", buf);
}
close(sockfd);  } }
```

Sockets avancés

Fonctions bloquantes

- Certaines fonctions de l’API Socket sont bloquantes
  - Attente de l’arrivée d’un évènement
  - Exemple de fonctions
    - *connect, accept, recv/recvfrom*
- Besoin parfois de socket non bloquant
  - Attente d’évènements sur plusieurs sockets
- Possibilité de rendre socket non bloquant
  - Fonction *fcntl()*
  - Pb : utilisation attente active pour recevoir données
    - Utilisation inutile du CPU

Utilisation de plusieurs Sockets

- Fonction donnant l'état de plusieurs sockets

```
int select(int numfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval *timeout);
```

Numéro plus grand
Socket descriptor +1

Ensembles de sockets en attente de :
- Lecture
- Ecriture
- Exceptions

Timer attente,
Désactivé si NULL

- Fonction attendant évènement sur l'un des sockets des ensembles (ou attente période indiquée par ***timeout***)
- Retourne nb sockets avec évènements, 0 si timeout, -1 si erreur

```
struct timeval {
        int tv_sec; // seconds
        int tv_usec; // microseconds
};
```

Utilisation des ensembles de Sockets

- Plusieurs macro utiles
  - Ajout d’un descripteur de socket
    - **FD_SET(int fd, fd_set *set);**
  - Suppression d’un descripteur de socket
    - **FD_CLR(int fd, fd_set *set);**
  - Test si descripteur socket est dans l’ensemble
    - **FD_ISSET(int fd, fd_set *set);**
  - Supprime tous les éléments de l’ensemble
    - **FD_ZERO(fd_set *set);**

Exemple utilisation Select()

```
/******* SERVEUR TCP  **************/
#include      <stdio.h>
#include      <sys/types.h>
#include      <sys/socket.h>
#include      <netinet/in.h>
#include      <netdb.h>

#define PORTS "2058"

main()
{
int sockfd, new_fd, rv,;
struct addrinfo hints, *servinfo, *p;
struct sockaddr their_adr;
socklen_t addrlen;
char buf[256];  // buffer données client

fd_set master;  // master file descriptor list
fd_set read_fds;  // temp file descriptor list for select()
int fdmax;  // maximum file descriptor number

FD_ZERO(&master);  // clear the master and temp sets
FD_ZERO(&read_fds);

// Création  socket  et  attachement
               ……………
listen(sockfd, 5);

FD_SET(sockfd, &master);   // Ajout sockfd à ensemble
fdmax = sockfd;   // Garde valeur max socket

while(TRUE)
{  read_fds = master;     // ensemble socket attente lecture
   if (select(fdmax+1, &read_fds, NULL, NULL, NULL) == -1)
   {  perror("select"); exit(4);}

   for(i = 0; i <= fdmax; i++)
   {
      if (FD_ISSET(i, &read_fds))
      {   if (i == sockfd)
          {      addrlen = sizeof(their_adr);
                 new_fd = accept(sockfd,
                                 &their_adr, &addrlen);
                  if (new_fd == -1)
                 {  perror("accept");}
                 else
                 {  // Ajout new_fd  à ensemble
                    FD_SET(new_fd, &master);
                    if (new_fd > fdmax)
                    {  fdmax = new_fd; }
                    printf("Nouvelle connexion au serveur.\n");
                 }
          }
```

Exemple utilisation Select()

```
else
{   // gestion données client i
   if ((nbytes = recv(i, buf, sizeof buf, 0)) <= 0)
   {   // erreur ou connexion fermée par client
      if (nbytes == 0)
      { printf(« Connexion %d fermée.\n", i);}
      else
      {  perror("recv"); }
      close(i);
      FD_CLR(i, &master); // Supprime ensemble
    }
    else
    {  // Données reçu du client
       for(j = 0; j <= fdmax; j++)
       {  // Envoie données à tous les autres clients j
          if (FD_ISSET(j, &master))
          {
              // Sauf serveur et client source données
              if (j != sockfd && j != i)
              {
                  send(j, buf, nbytes, 0);
              }
          }
       }
     }
```

```
        }  // Fin bloc ELSE client
      }  // Fin bloc IF  FD_ISSET
    }  // Fin bloucle FOR sur i
  }  // Fin boucle WHILE

  return 0;
}
```

Echange de données

- Envoie de données en mode caractères
  - Utilisation fonction *sprintf / snprintf*
- Définition d'un format de message
  - Type du message
  - Taille total du message
    - Utiliser fonction *htons()*
  - Nom expéditeur (nb caractères max fixé)
    - Complété par caractère '\0' si inférieur à max
  - Données (nb caractères max fixé)

Envoie/Réception données

- Envoie
  - Faire attention lors de l'envoi
    - **Tester nombre caractères réellement envoyés !**
  - Possible de faire appel plusieurs fois à *send()*
    - Si message à envoyer dépasse taille max paquet
- Réception
  - Utiliser taille message max avec fonction *recv()*
    - Attention indication taille max tampon et pas taille message à recevoir