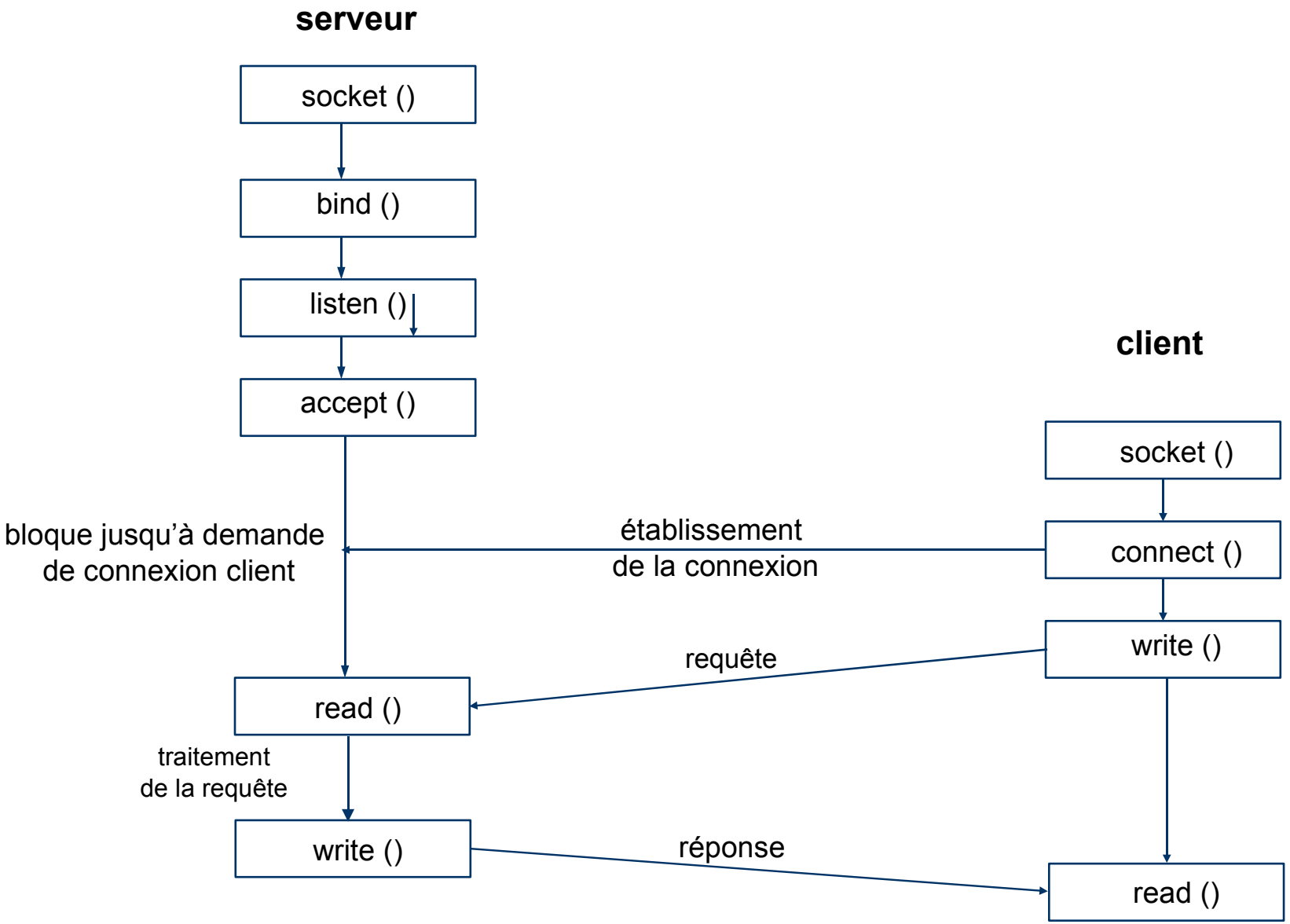
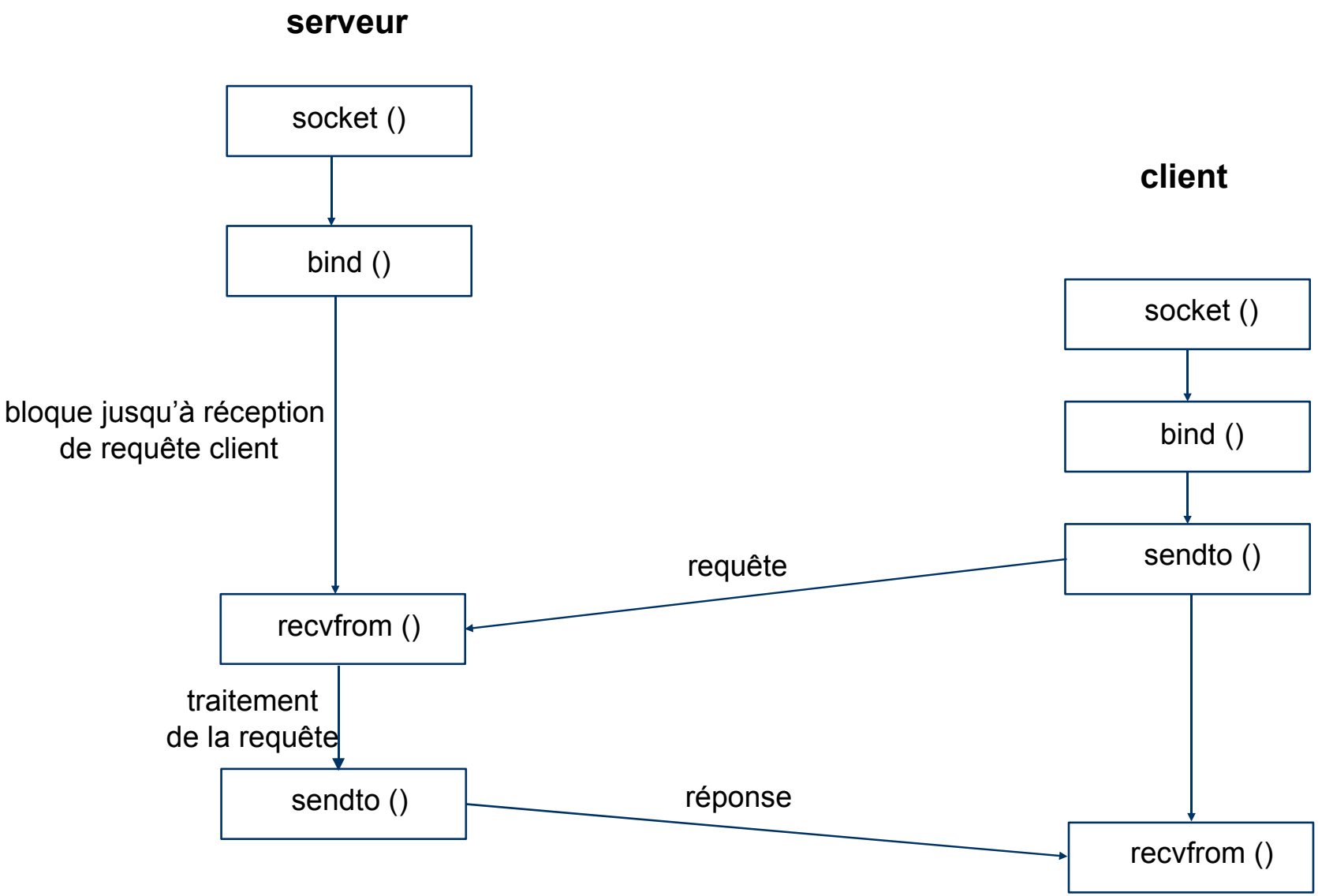
Introduction à la Programmation Réseaux en Langage C

12. Appels Systèmes Elémentaires

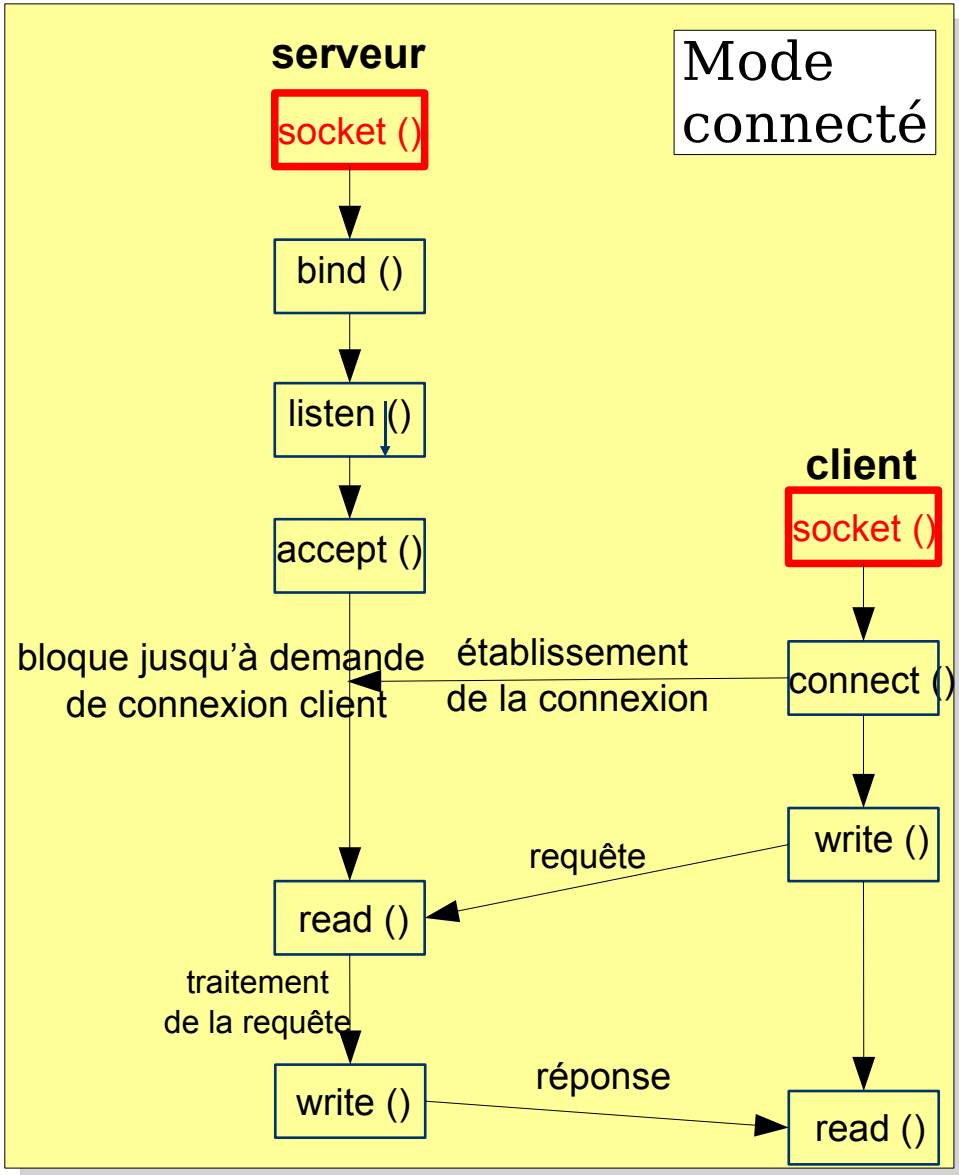
Synoptique en mode connecté

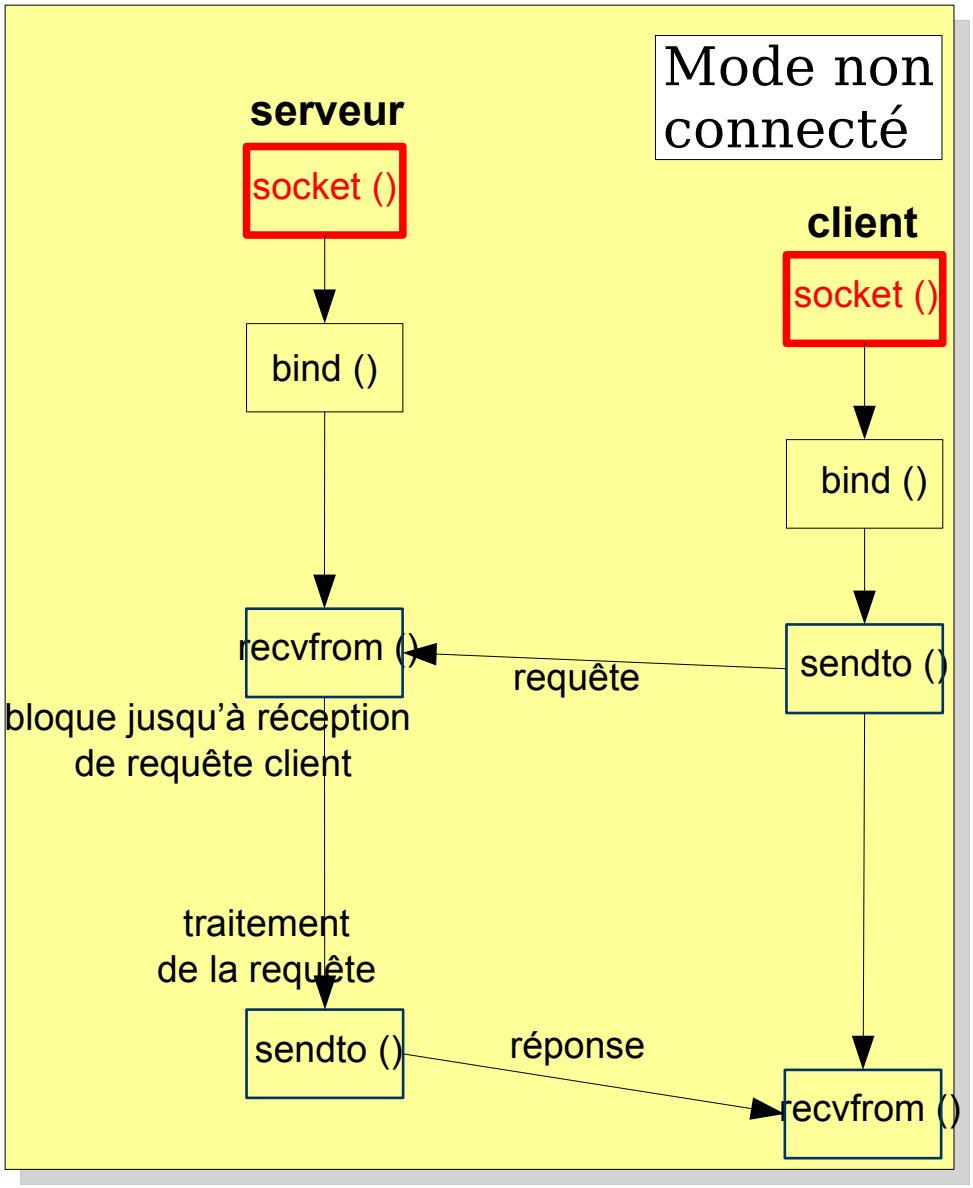


Synoptique en mode non connecté



Création de Socket





Création de Socket

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

int socket(int domain,int type,int protocole)
```

domain : Famille de Protocole (Domaine)

```
<sys/socket.h>
#define AF_UNSPEC       0               /* unspecified */
#define AF_UNIX         1               /* local to host (pipes, portals) */
#define AF_INET         2               /* internetwork: UDP, TCP, etc. */
...
#define PF_UNSPEC       AF_UNSPEC
#define PF_UNIX         AF_UNIX
#define PF_INET         AF_INET
...
```

Création de Socket

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

int socket(int domain,int type,int protocole)
```

type : Type de Protocole

```
<sys/socket.h>
#define SOCK_STREAM     NC_TPI_COTS     /* stream socket */
#define SOCK_DGRAM      NC_TPI_CLTS     /* datagram socket */
#define SOCK_RAW        NC_TPI_RAW      /* raw-protocol interface */
#define SOCK_RDM        5               /* reliably-delivered message */
#define SOCK_SEQPACKET  6               /* sequenced packet stream */
```

Création de Socket

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

int socket(int domain,int type,int protocole)
```

- ***protocole*** : Explicite le protocole

➔Généralement ignoré (défaut = 0)

```
<netinet/in.h>
#define IPPROTO_IP              0               /* dummy for IP */
#define IPPROTO_ICMP            1               /* control message protocol */
...
#define IPPROTO_TCP             6               /* tcp */
...
#define IPPROTO_UDP             17              /* user datagram protocol */
...
#define IPPROTO_RAW             255             /* raw IP packet */
```

➔Combinaisons valides pour le domaine PF_INET

Type	Protocole	rotocole I
SOCK_STREAM	IPPROTO_TCP	TCP/IP
SOCK_DGRAM	IPPROTO_UDP	UDP/IP
SOCK_RAW	IPPROTO_ICMP	ICMP
SOCK_RAW	IPPROTO_RAW	RAW IP

Cas Particulier du Domaine UNIX

- Un mécanisme d'IPC
- Transmission fiable (interne noyau)
- Deux modes de transmission
  - Connecté (SOCK_STREAM) : contrôle de flux

  L'écriture bloque lorsque le socket est plein
  - Non Connecté (SOCK_DGRAM) : sans contrôle de flux

  Une écriture peut échouer

- Propriété intéressante :
  - Echange de descripteurs entre TOUT processus

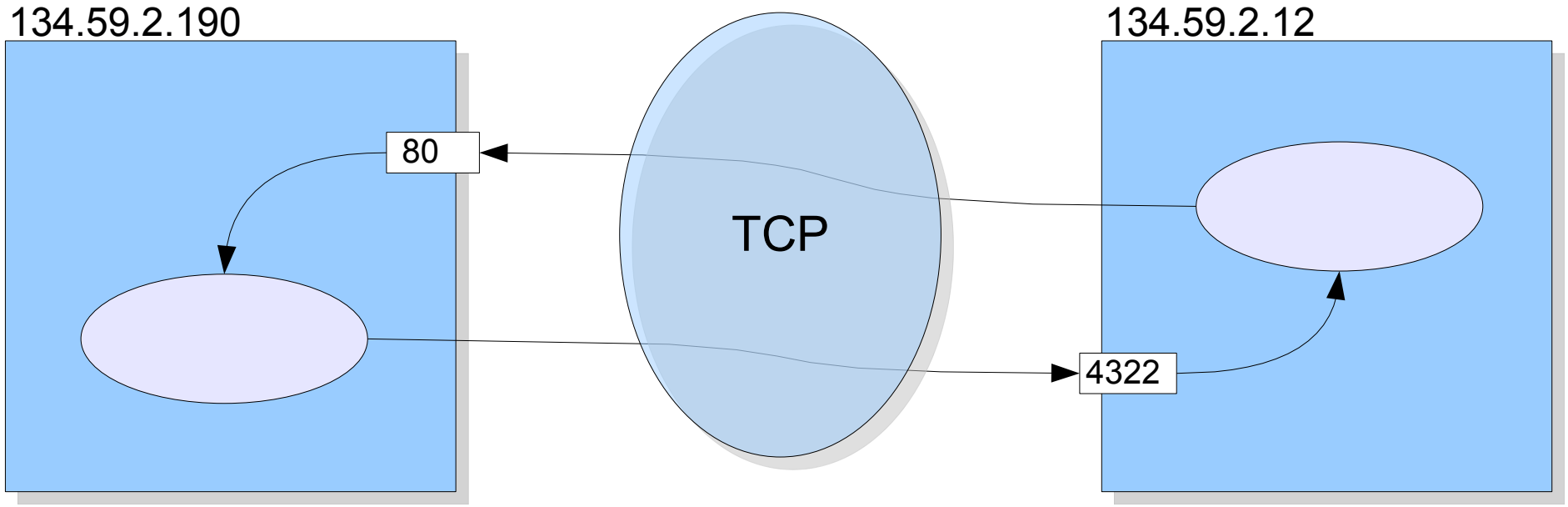
Définition d'une Association entre Processus

- Complètement définie par le 5-uplet :

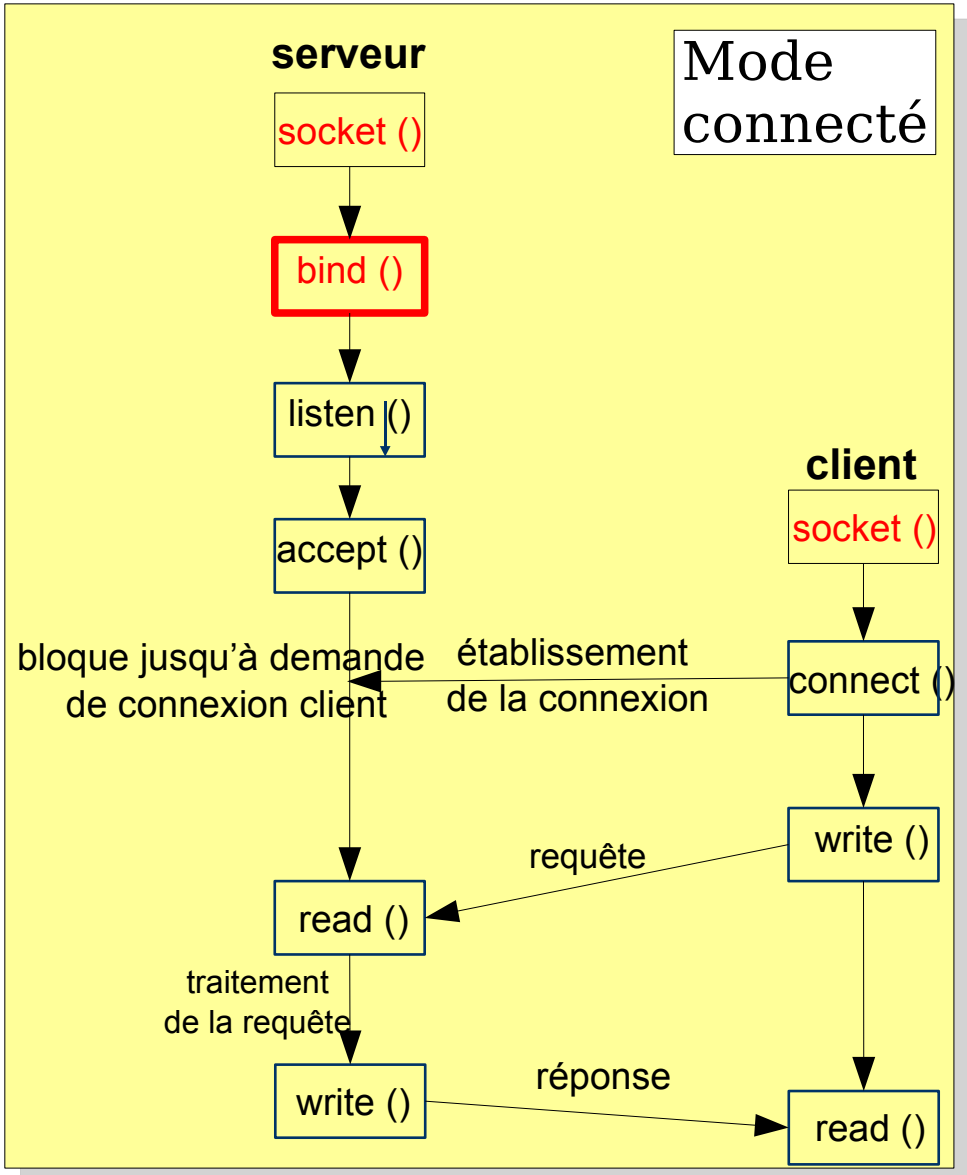
(protocole, @hôte local, @process local,
@hôte distant, @process distant)

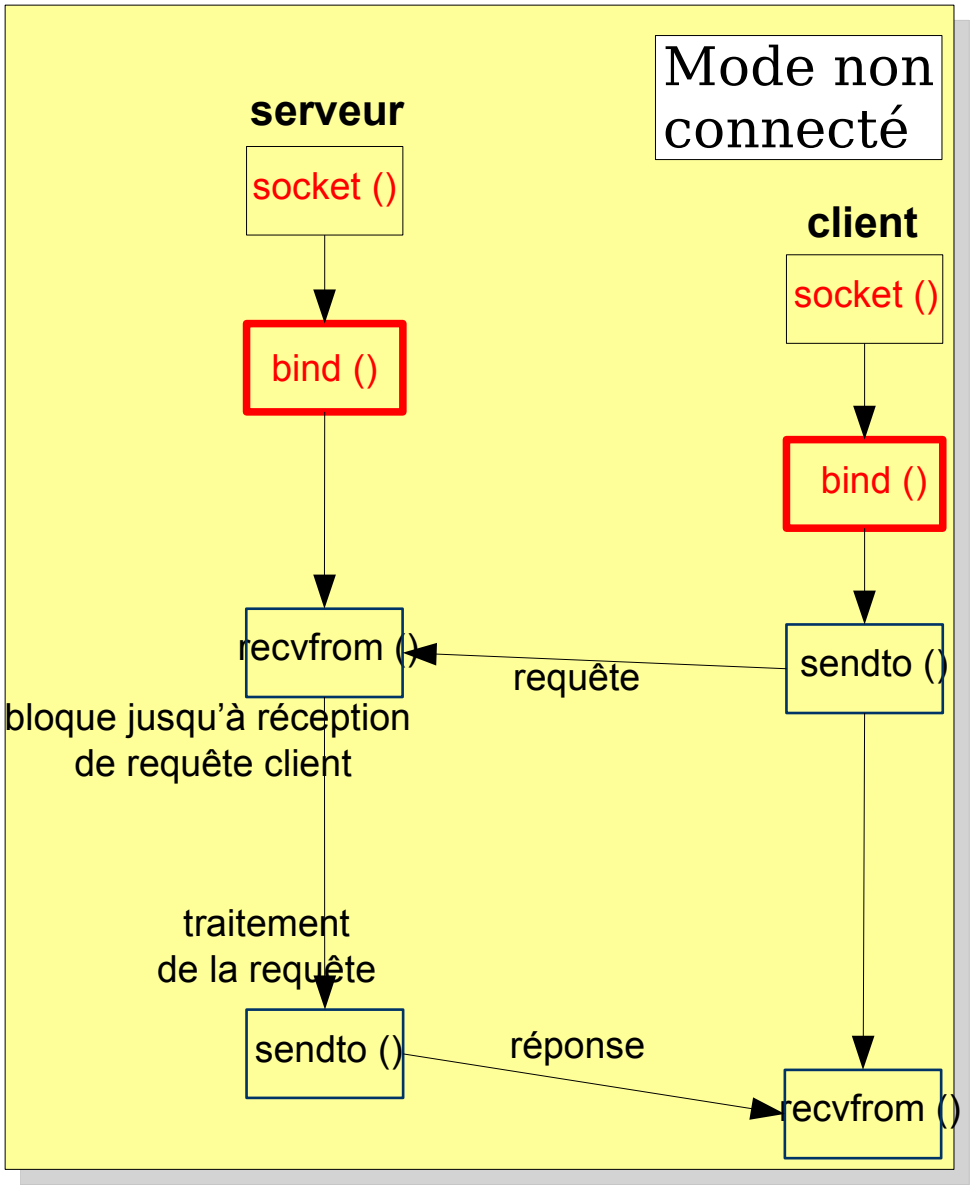
- Exemple :

(TCP, 134.59.2.190, 80, 134.59.2.12, 4322)



Attachement d'un nom à un Socket





Attachement d'un « nom » à un Socket

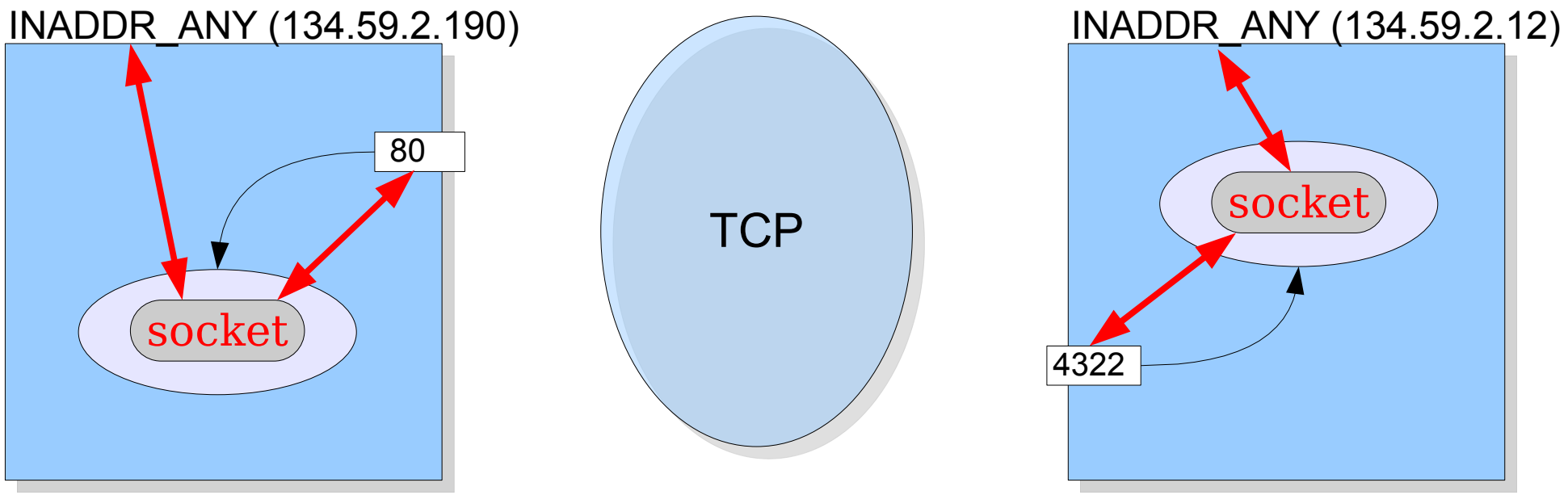
```
#include <sys/types.h>
#include <sys/socket.h>

int bind(int sockfd,const struct sockaddr *myaddr,int addrlen)
```

- Assigne l'adresse **locale** d'un socket : hôte + processus (= port pour IP)

➔ Un serveur enregistre son port de service

➔ Un client peut enregistrer son adresse (inutile avec TCP)

➔ **INADDR_ANY** = @IP locale (indifférente)



Attachement d'un « nom » à un Socket

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

int bind(int sockfd,const struct sockaddr *myaddr,int addrlen)
```

- ***sockfd*** : Descripteur de fichier

➔retourné par socket()

- ***myaddr*** : « nom » = adresse locale

➔IP = struct sockaddr_in

généralement :

- `myaddr.sin_addr.s_addr = INADDR_ANY`
- `myaddr.sin_port = htons(hport)`

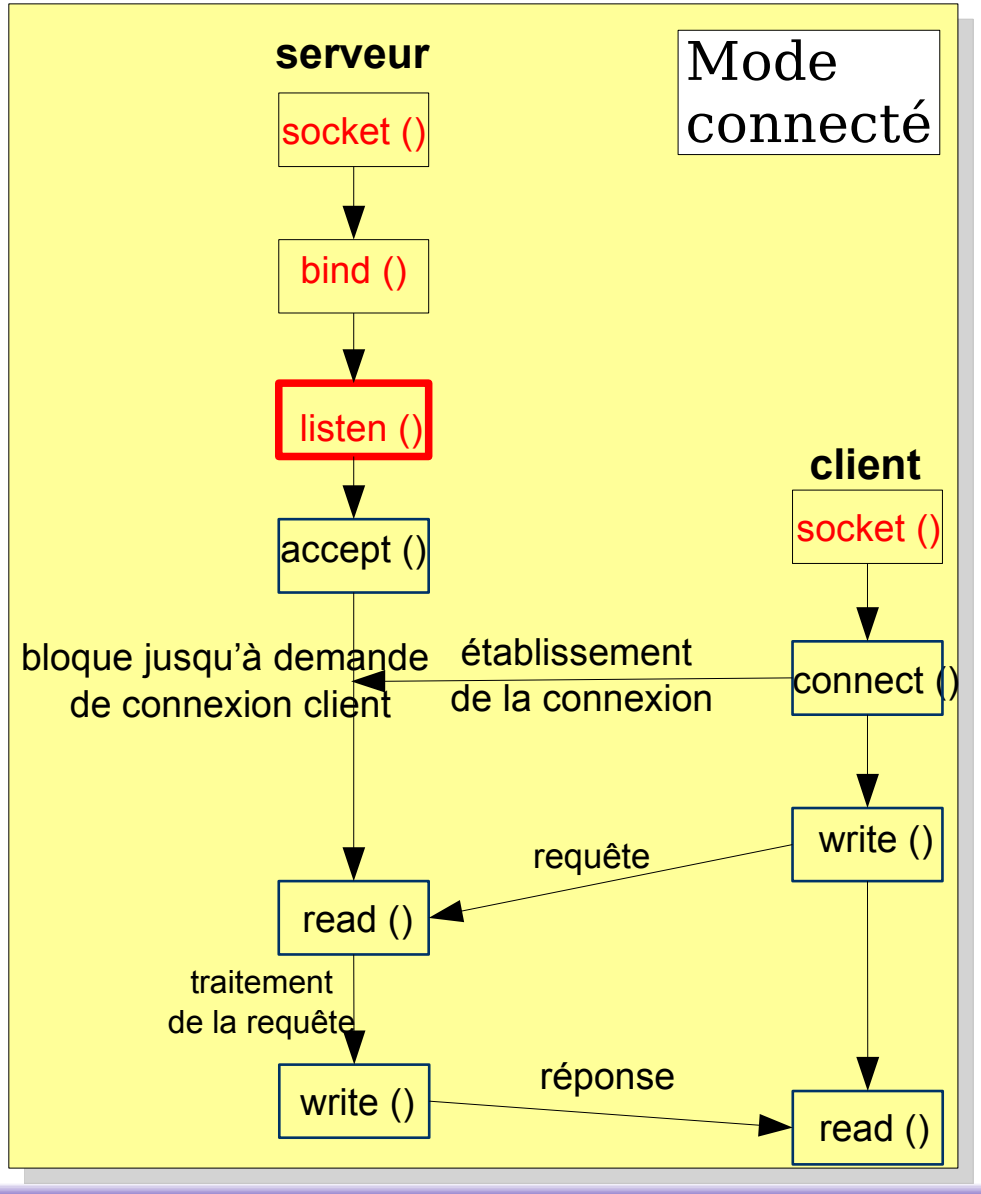
- ***addrlen*** : Taille de l'adresse

➔généralement : sizeof(myaddr)

Choix du numéro de Port (IP) ?

- « *Well known port* »
  - Service connu universellement
    - Telnet, ftp, http, ssh, DNS, ...
  - Généralement n° port < 1024
    - Unix : attachement port < 1024 Ssi utilisateur = root
      - /etc/services
- Port « privé » (> 1024)
  - Port = 16 bits = max 65535
  - Quel port choisir ? comment éviter les conflits ?
    - Protocole ad-hoc :
      - port codé en dur + solution de replis (scan port)
    - RPC portmap
      - Ex: commande rpcinfo -p host …

Transformation d'un Socket en Socket de Service



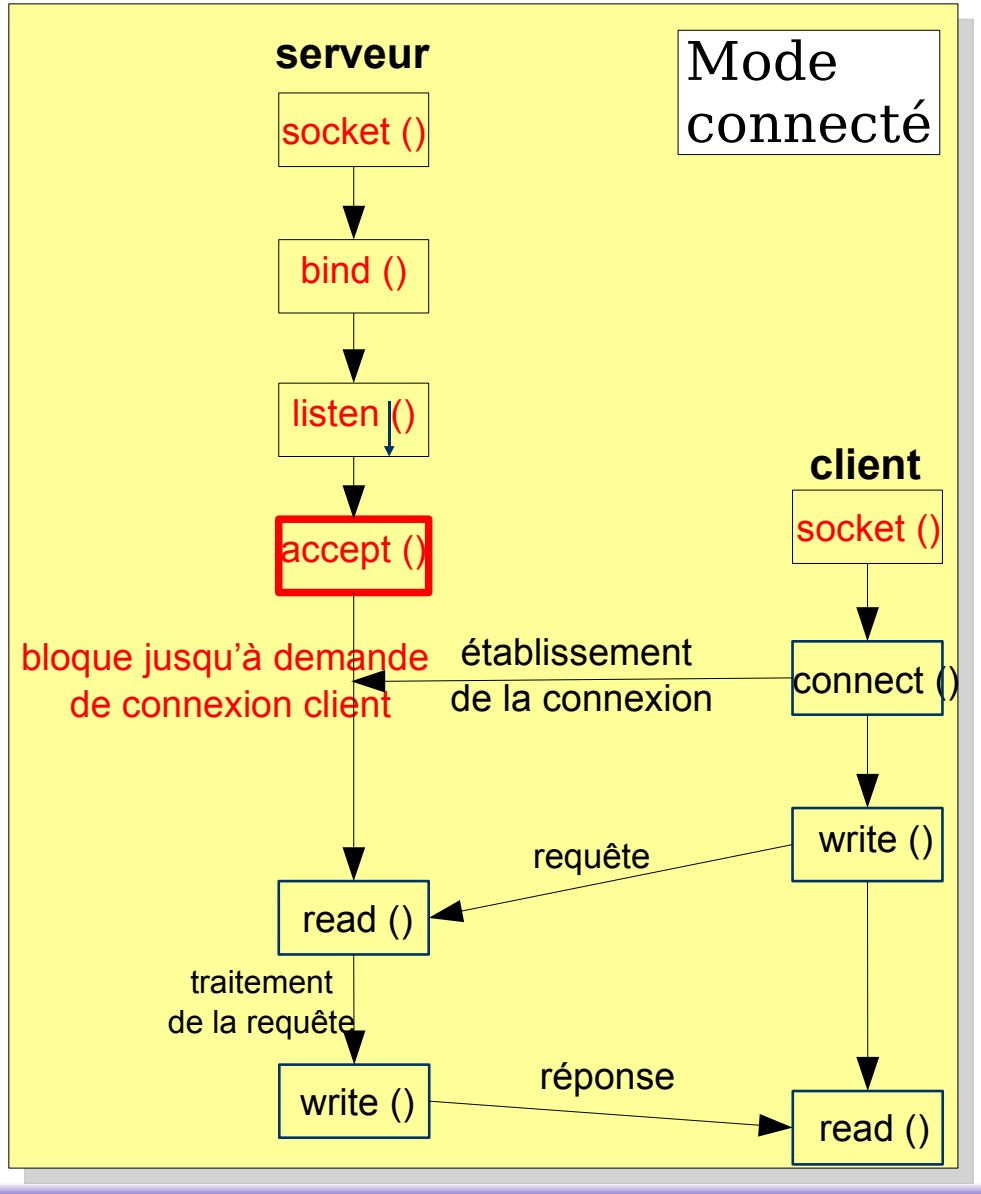
Transformation d'un Socket en Socket de Service

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

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

- Utilisé par un serveur pour indiquer qu'il est prêt à recevoir des connexions

Taille file d'attente
combien de demandes de connexions simultanées peuvent être mises en attente

Création d'un Socket de Dialogue



Création d'un Socket de Dialogue

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

int accept(int sockfd, struct sockaddr *peer, int *addrlen)
```

- Utilisé par un serveur pour indiquer qu'il est prêt à recevoir des connexions
- Bloque jusqu'à l'arrivée d'une connexion

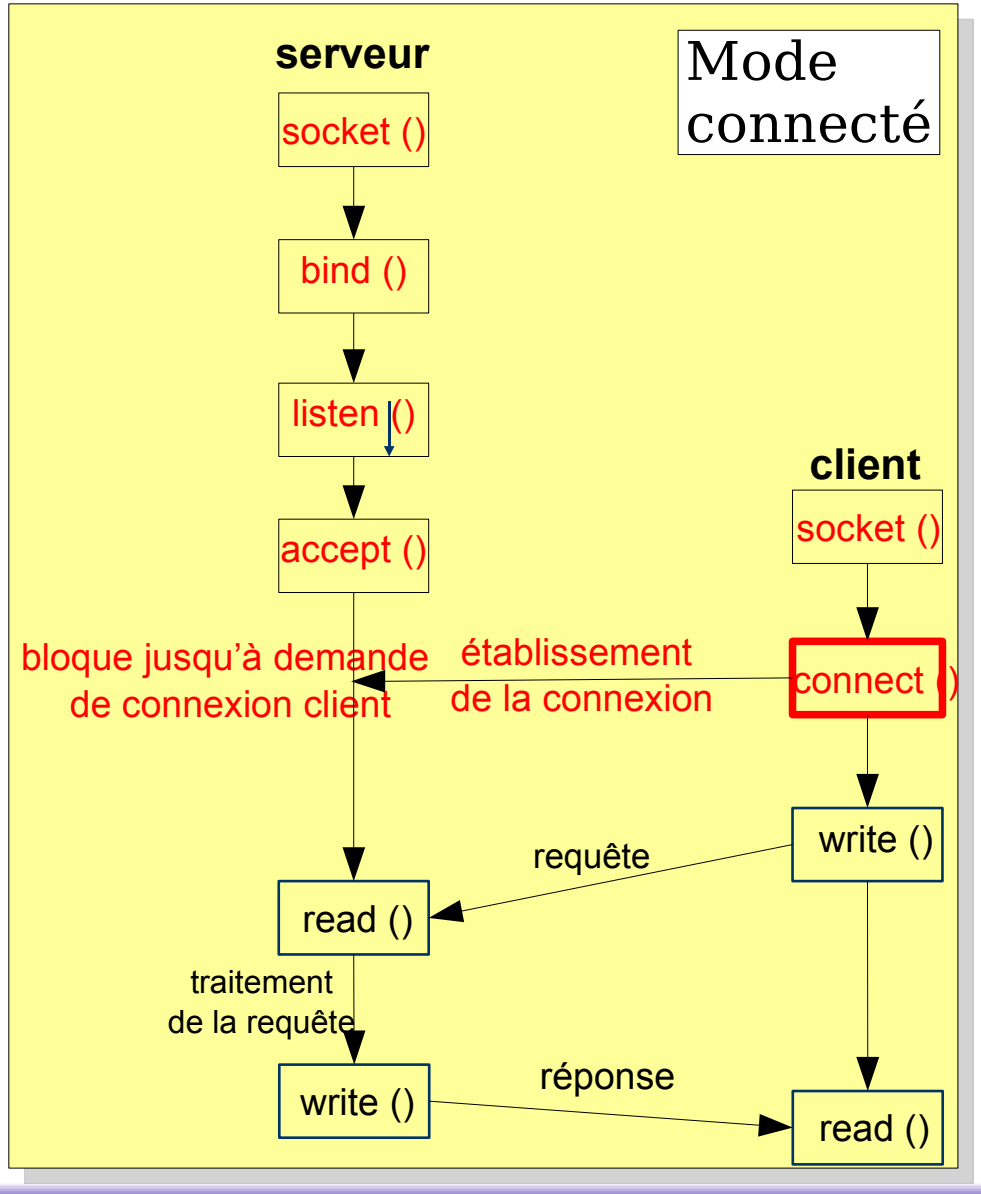
Adresse client

Descripteur initial
Il reste valide *: le descripteur initial ne sert qu'à recevoir des demandes de connexion des clients*

Taille adresse
Attention : passage par adresse !
variable input/output :
Input : taille de la zone mémoire fournie
Output : taille de l'@ reçue dans peer

Un **nouveau descripteur**
c'est par ce nouveau descripteur que le serveur pourra communiquer avec le client

Connexion à un Serveur (client)



Connexion à un Serveur (client)

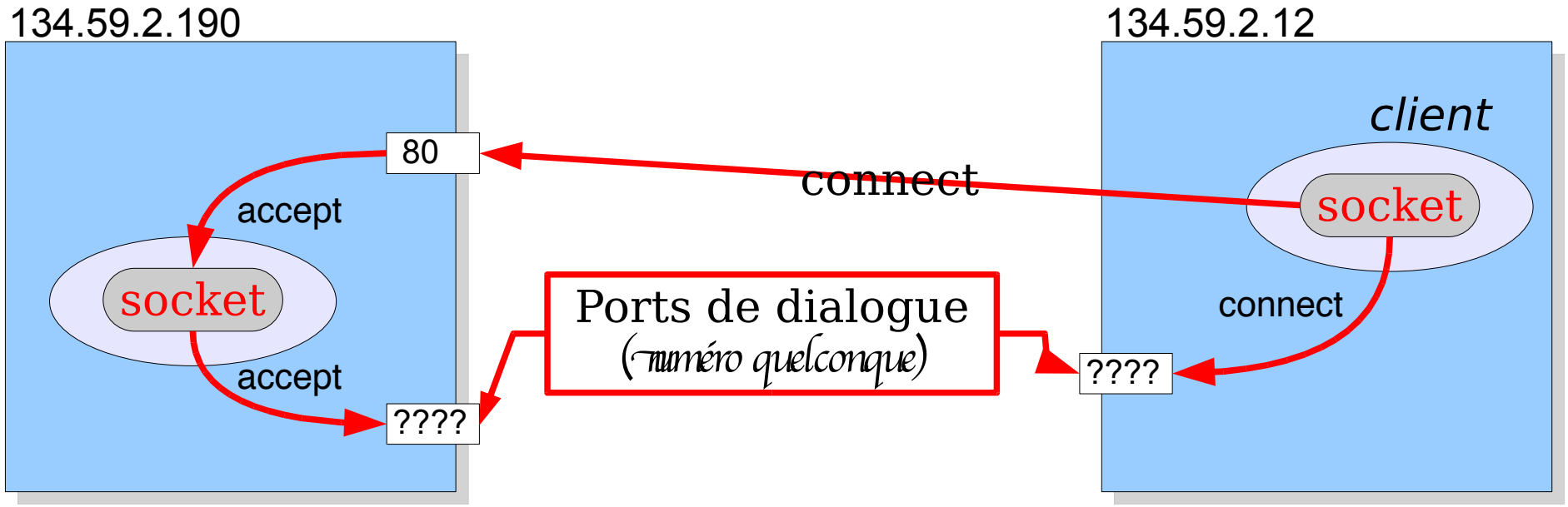
```
#include <sys/types.h>
#include <sys/socket.h>

int connect(int sockfd,const struct sockaddr *myaddr,int addrlen)
```

- **Utilisé par un client** pour se connecter à un serveur

➔Principalement en mode connecté (!)

➔Bloque l'appelant jusqu'à établissement de la connexion

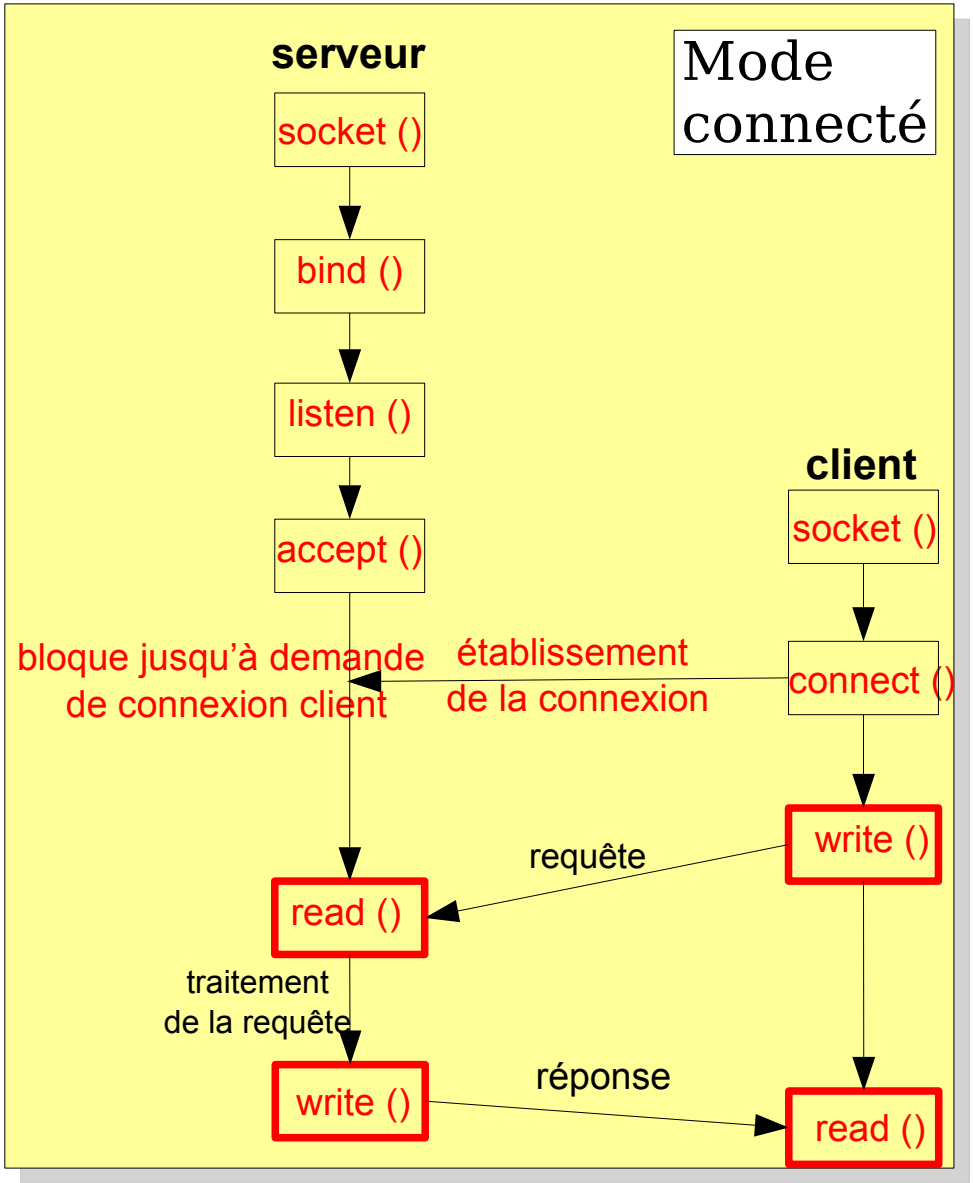
➔Initialise paramètres locaux du client quand bind() est omis (le plus souvent)



Exemple de Connexion (serveur « concurrent »)

```
int sockfd, newsockfd;
struct sockaddr_in sin_local,sin_remote;

/* On suppose que sin_local a ete initialise au prealable */

if ((sockfd = socket(...)) < 0)
    err_sys("socket error");
if ((bind(sockfd, &sin_local, sizeof sin_local)) < 0)
    err_sys("bind error");
if ((listen(sockfd,5) < 0))
    err_sys("listen error");

while (1) {
    newsockfd = accept(sockfd,&sin_remote, sizeof sin_remote);  /* bloquant */
    if (newsockfd < 0) err_sys("accept err");

    if (fork() == 0) {
        close(sockfd); /* fils */
        traite_connexion(newsockfd,&sin_remote);
        exit(0);
    }

    close(newsockfd);  /* parent */
}
```

Echanges de Données



Echanges de Données

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

int write(int sockfd,const char *buff,int nbytes)
int read(int sockfd,const char *buff,int nbytes)

int send(int sockfd,const char *buff,int nbytes,int flags)
int recv(int sockfd,const char *buff,int nbytes,int flags)
```

Fonctionnent de la même façon que pour une lecture/écriture

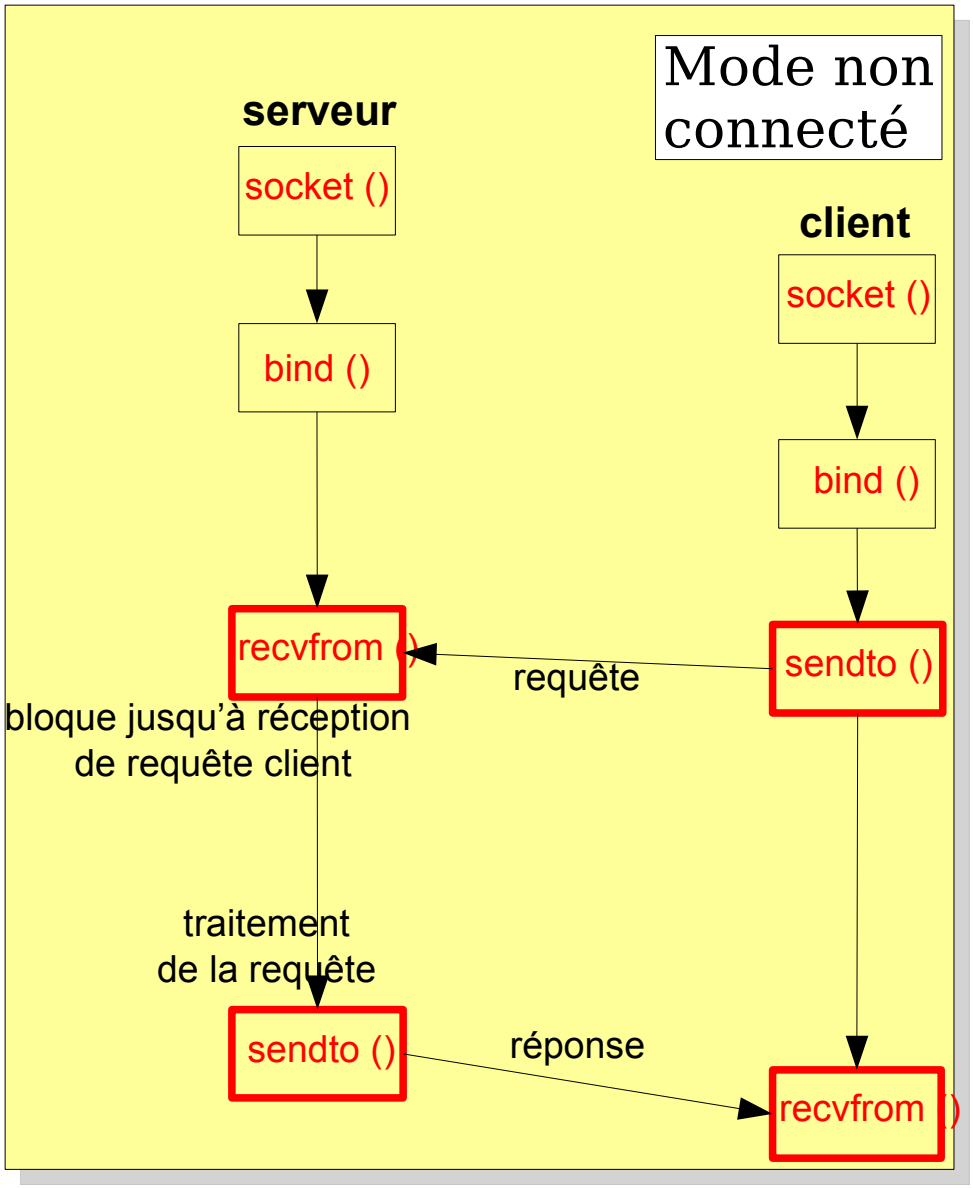
flags : Drapeaux spécifiques aux sockets

```
<sys/socket.h>
#define AF_UNSPEC       0               /* unspecified */
#define MSG_OOB         0x1             /* process out-of-band data */
#define MSG_PEEK        0x2             /* peek at incoming message */
#define MSG_DONTROUTE   0x4             /* send without using routing
tables */
```

- **En lecture** : permet de lire les données mais sans
- vider le tampon du socket

➔autorise de multiples lectures d'un même message

Echanges de Données



Echanges de Données

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

int sendto(int sockfd,const char *buff,int nbytes,int flags
           struct sockaddr *to,int tolen)
int recvfrom(int sockfd,const char *buff,int nbytes,int flags,
           struct sockaddr *from,int *fromlen)
```

- Fonctionnent avec **tous les types de socket**

➔ **to** : adresse destinataire (ignoré avec socket en mode connecté)

➔ **tolen** : taille du tampon **to**

➔ **from** : adresse expéditeur (NULL information inutilisée)

➔ **fromlen** : taille du tampon **from**

Attention : variable input/output ...

Fermeture de Socket

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

int close(int sockfd)
int shutdown(int sockfd,int how)
```

- Relâche le desscripteur (en fermant la connexion si mode connecté)

➔Fermeture garantie sans perte

➔Permet au noyau d'envoyer les dernières données en attente

➔Peut bloquer (mais existe pour empêcher le blocage)

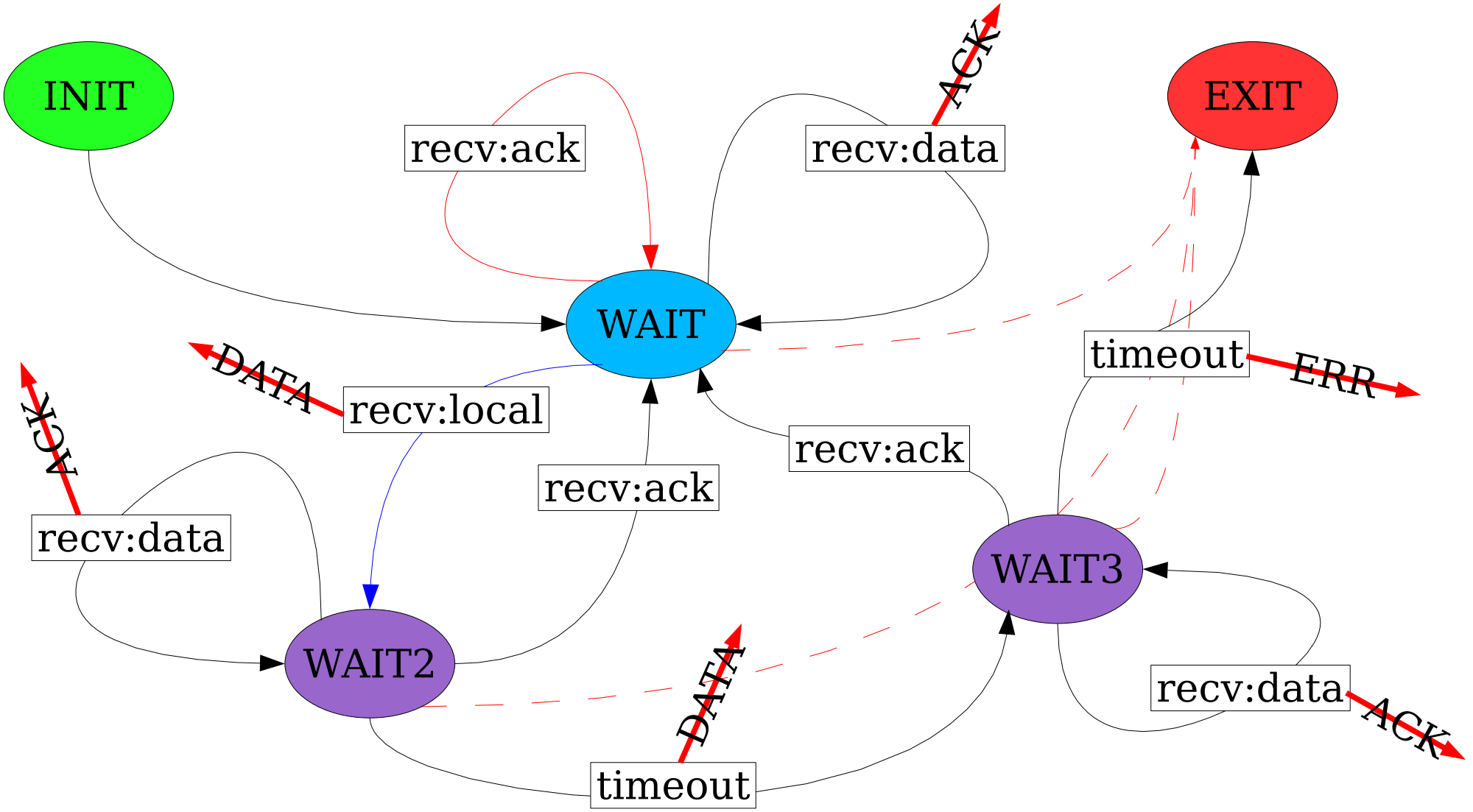
how : fermeture éventuellement asymétrique

- 0 : dans le sens des réceptions
- 1 : dans le sens des émissions
- 2 : dans les deux sens

Conception de protocole

- Protocole :
  - Format des messages
    - en-tête = x octets, données utiles = entre x et y octets, ...
  - Sémantique des messages
    - Message echo, reply, request_truc, reply_truc, ...
  - Règles algorithmiques
    - Lors de la réception d'un message « echo », répondre avec un message « reply »
    - Problème : les algorithmes deviennent vite complexes
      - Risque important d'oubli de cas de figures ...
    - Méthodologie usuelle :
      - Expliciter le ou les automates des états du protocole
      - Vérifier la complétude (toutes les transitions prévues dans tout état)

Exemple de Protocole

- Protocole (naïf) d'échange fiable de données sur UDP :
  - Fonctionnement symétrique du client et du serveur
    - Attendre l'un des événements suivants
      - Réception d'une donnée distante du pair (=> traitement local)
      - Réception d'une donnée locale (=>transmission **bloquante** au pair)
  - En cas de réception d'un message, retourner un message accusé de réception
  - Lorsqu'un envoi de message est demandé localement :
    - Transmission + attente **bornée** d'un accusé de réception
    - Si délai expire sans réception accusé :
      - Retransmission du message avec nouvelle attente bornée
      - Si délai expire à nouveau : erreur puis terminaison

Exemple d'Automate