APPLICATIONS CLIENT-SERVEUR

- Processus:
  - ✓ Client → emet des requêtes
  - ✓ Serveur → répond aux requêtes

- Communications:
  - ✓ Orienté connexion
  - ✓ Non orienté connexion

- avantages client-serveur:

  - ✓ Transparence de la localisation des processus

  - ✓ Transparence des types de machines et du système opératoire

  - ✓ Flexibilité d’ajout de serveurs et clients

  - ✓ Meilleur fiabilité et efficacité

ASSOCIATIONS et SOCKETS

- Données necessaires pour identifier un canal client-serveur
  → **`BIND`** (asociation)

{protocole, adresse locale, processus local, adresse distante, processus distant}

- Moyen d'association dans la machine locale:

  {protocole, adresse locale, processus local}

- Moyen d'asociation dans la machione distante:

  {protocole, adresse distante, processus distant}

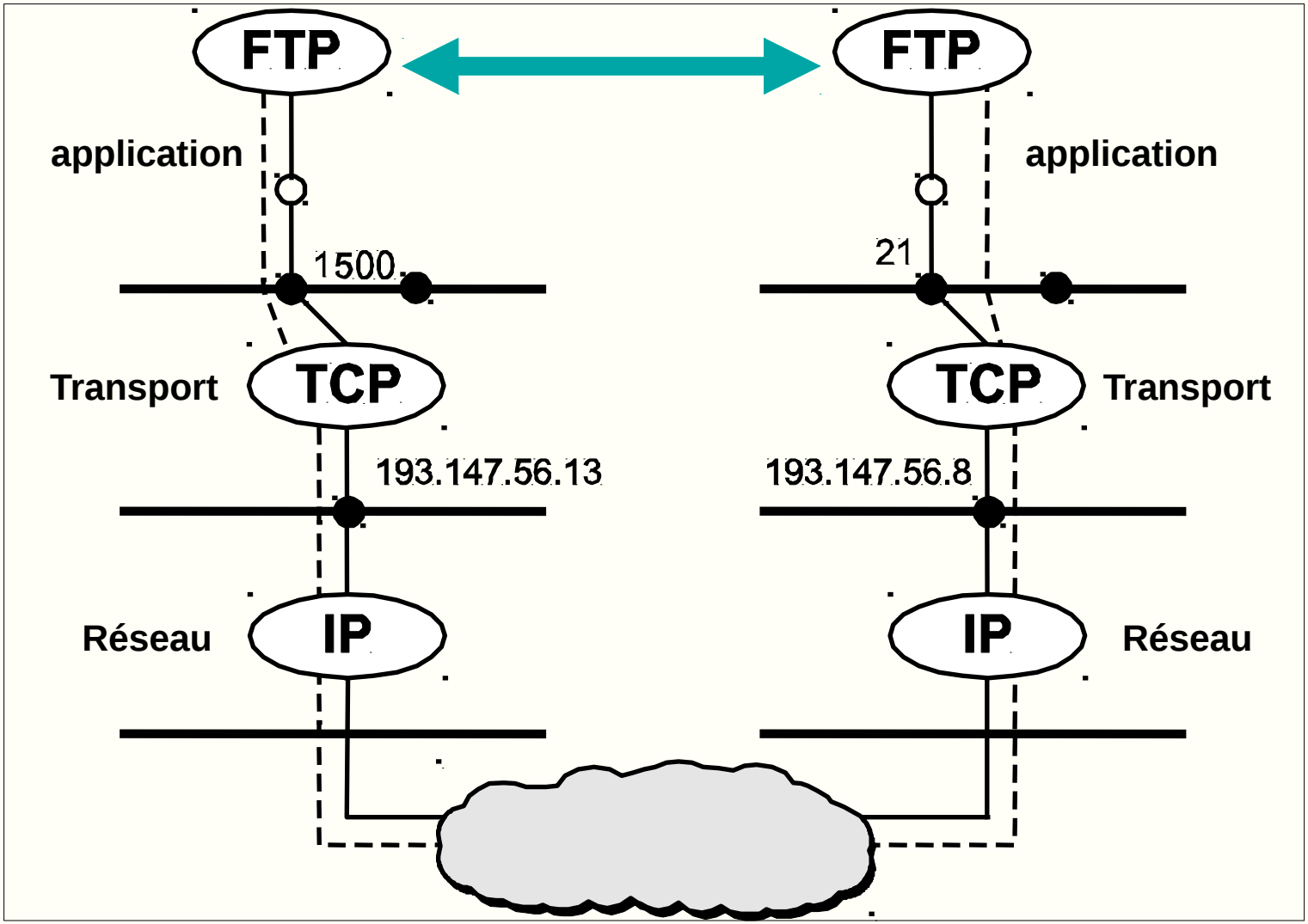
Exemple: #ftp 193.147.56.8

→ IP distant: 193.147.56.8
→ port distant: 21
→ IP local: celle du client (exemple: 193.147.56.13)
→ port local: 1500

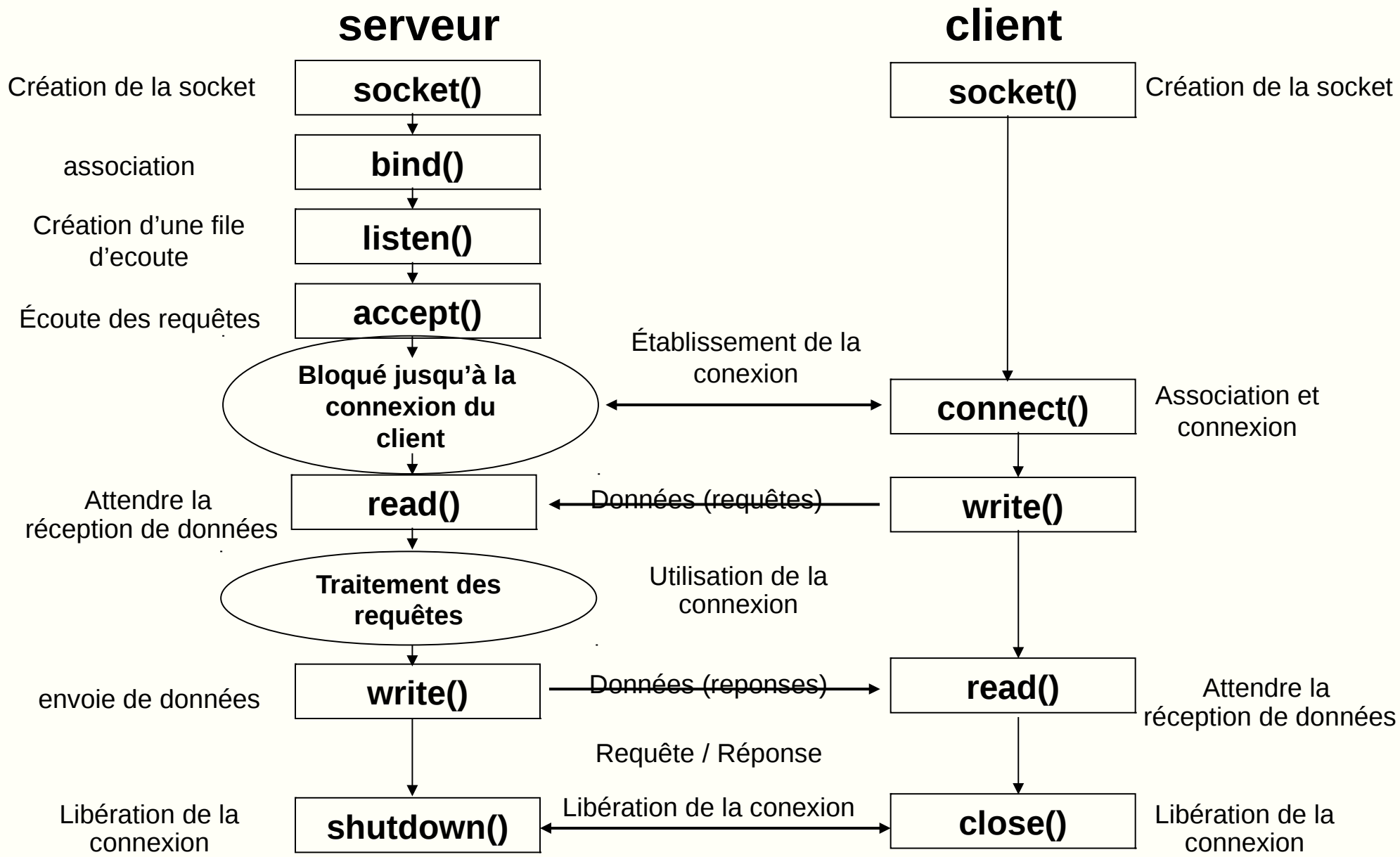
{TCP, 193.147.56.8, 21, 193.147.56.13,21} ➔
{TCP, 193.147.56.8, 21} → socket dans le serveur
{TCP, 193.147.56.13,1500} → socket dans le client

ASSOCIATIONS et SOCKETS

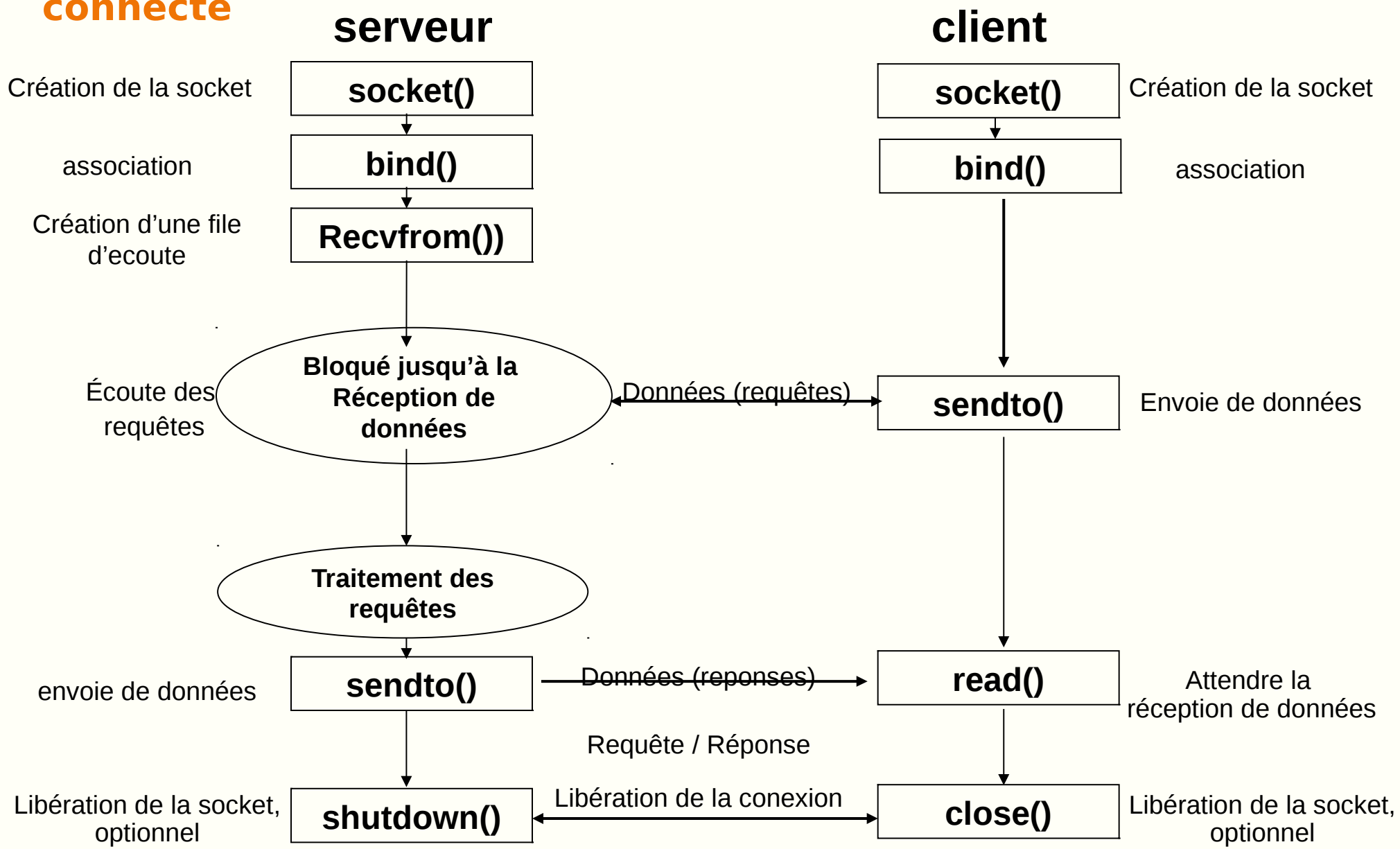
Exemple: #ftp 193.147.56.8



SOCKETS: schéma de fonctionnement en mode connecté



SOCKETS: schéma de fonctionnement en mode non connecté



DOMAINES DES SOCKETS

- Indique sous quel réseau la communication va se réaliser ➔ adressage à employer

- Structure d'adresse d'un socket (<sys/socket.h>)

```
struct sockaddr
{
   u_short sa_family;    /* famille d'adresse, 2 bytes */
   char    sa_data[14];          /* 14 bytes d'adresse max */
};
```

Domaine UNIX (AF_UNIX)

- Dans la même machine

```
struct sockaddr_un
{
      short sun_family;
      char sun_path[108]
};
```

Domaine Internet (AF_INET)

- Dans des machines distantes

```
struct in_addr{
  u_long s_addr; };

struct sockaddr_in {
   short sin_family;
   u_short sin_port;
   struct in_addr s_addr;
   char sin_zero[8]};
```

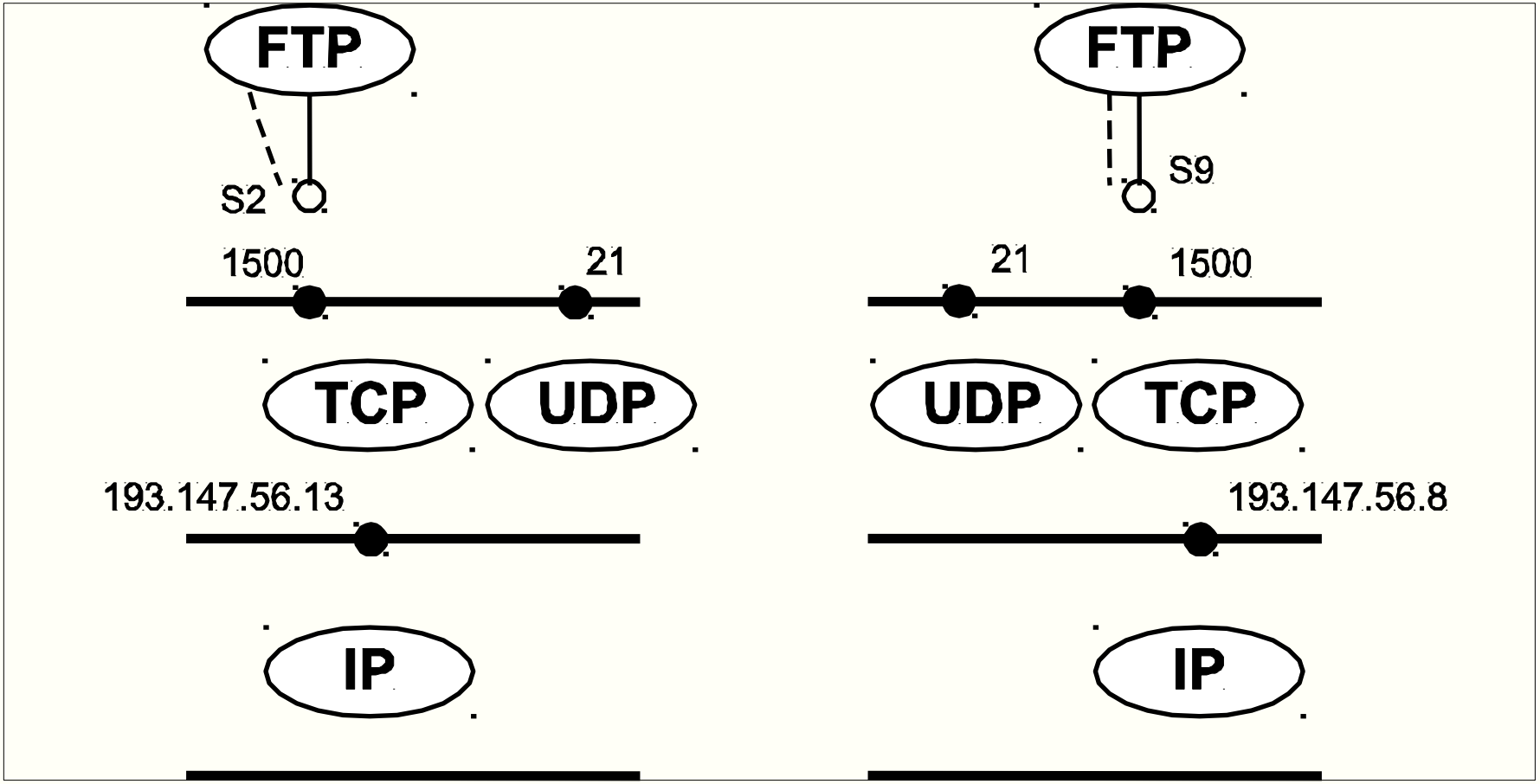
CREATION D'UN SOCKET

```
#include <sys/types.h>
#include <sys/socket.h>
int socket (domaine, type, protocole)
   int domaine;          /* AF_INET, AF_UNIX, ...*/
   int type;             /* SOCK_DGRAM, SOCK_STREAM, …*/
   int protocole;        /* 0: protocole par defaut */
```

- Type:
  - **SOCK_DGRAM**. Communication avec des protocoles non orienté connecxion
  - **SOCK_STREAM**. Communication sous des protocoles orienté connexion
  - **SOCK_RAW**. Accès au protocoles de plus bas niveau

Type	Protocole	processus local	processus distant
Serveur orienté connexion	*socket()*	*bind()*	*listen(), accept()*
Client orienté connexion	*socket()*	*connect()*	
Serveur non orienté connexion	*socket()*	*bind()*	*recvfrom()*
Client non orienté connexion	*socket()*	*bind()*	*sendto()*

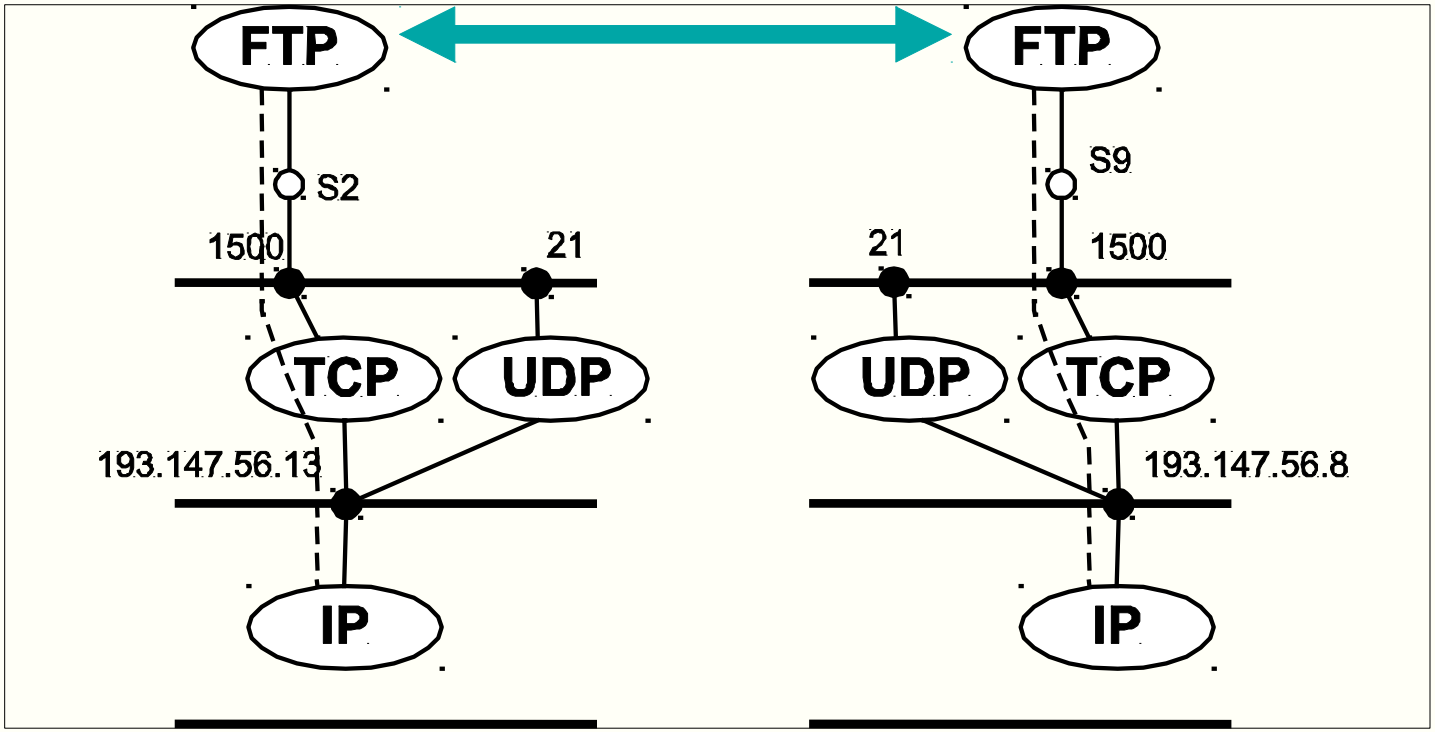
CREATION D'UN SOCKET



ASSOCIATION D'UN SOCKET

```
#include <sys/types.h>
#include <sys/socket.h>
int bind (s_descripteur, p_adresse, longadr)
   int descripteur;              /* descripteur de la socket*/
   struct sockaddr *p_adresse    /* pointeur à l'adresse */
   int longadr;                  /* longueur de l'adresse */
```

- Bind permet de lier deux élements: l'adresse IP et le port local.



ENVOI ET RECEPTION DE DONNEES (N.O.C.)

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

int sendto (desc, msg, lg, option, p_dest, lg_dest)
   int desc;       /* descripteur du socket d'émission */
   char *msg ;     /* adresse du message à envoyer */
   int lg;         /* longueur du message */
   int option;     /* 0 */
    struct sockaddr *p_dest;/*pointeur à l'adresse du socket
                            destination */
   int lg_dest; /*longueur de l'adresse du socket destination*/

int recvfrom (desc, msg, lg, option, p_exp, p_lgexp)
   int desc;       /* descripteur du socket de reception */
   char *msg ;     /* adresse du buffer pour le message */
   int lg;         /* longueur du buffer */
   int option;     /* 0 normalement, MSG_PEEK*/
   struct sockaddr *p_exp;/* recupérer l'adresse d'éxpédition */
   int *p_lgexp;
```

APPELS ORIENTES CONNEXION

Attente de connexion par le serveur

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

int listen (desc, nb)
   int desc;      /* descripteur du socket d'écoute */
   int nb;        /* nombre max de requêtes à accepter */
```

Acceptation de connexion dans le serveur

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

int accept (desc, p_adr, p_lgadr)
   int desc ;              /* descripteur du socket */
   struct sockaddr *p_adr ;/* adresse du socket connecté */
   int *p_lgadr;  /* pointeur vers la longueur de l'adresse */
```

APPELS ORIENTES CONNEXION

Etablissement de connexion dans le client

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

int connect (desc, p_adr, lgadr)
   int desc;                /* descripteur du socket locale */
   struct sockaddr *p_adr; /* adresse du socket distant */
   int lgadr;               /* longueur de l'adresse distante*/
```

- Dans le processus d'établissement de la connexion client/serveur se fait l'échange des paramètres pour régir cette connexion.
- **Les deux sockets doivent avoir le même type et famille**
- Dans le processus d'établissement de la connexion, les 4 coordonnées restantes de l'association sont connu. Donc, ce n'est pas necessaire d'utiliser l'appel `bind()`

APPELS ORIENTES CONNEXION

Envoi et réception de données

```
int write (desc, msg, lg)
   int desc ;     /* descripteur du socket local */
   char *msg ;    /* adresse du message a envoyer */
   int lg;        /* longueur du message */

int send (desc, msg, lg, option)
   int desc ;     /* descripteur du socket local */
   char *msg ;    /* adresse du message à envoyer */
   int lg;        /* longueur du message */
   int option;    /* 0, MSG_OOB, MSG_PEEK */

int read (desc, msg, lg)
   int desc ;     /* descripteur du socket local */
   char *msg ;    /* descripteur pour garder le message */
   int lg;        /* longueur de la zône reservée au msg */

int recv (desc, msg, lg, option)
   int desc ;     /* descripteurr du socket local */
   char *msg ;    /* adresse pour garder le message */
   int lg;        /* longueur de la zône réservée a msg */
   int opcion;    /* 0, MSG_OOB, MSG_PEEK */
```

APPELS ORIENTES CONNEXION

Fermer une connexion

```
int close (descripteur)
  int descripteur;
```

Essaye d'envoyer les données qui sont déjà dans la file

```
int shutdown (desc, sens)
   int desc;              /* descripteur du socket */
   int sens ;             /* control de la déconnexion */
```

Donne un meilleur controle sur la connexion dans les deux sens.

L'argument `sens` nous permet de controler le processus de déconnexion, selon les valeurs:

- 0: si on ne veut pas recevoir
- 1: si on ne veut pas émettre
- 2: si on ne veut ni émettre ni recevoir

Avec `shutdown()` c'est possible de fermer la connexion dans l'un des sens en laissant l'autre sens ouvert.

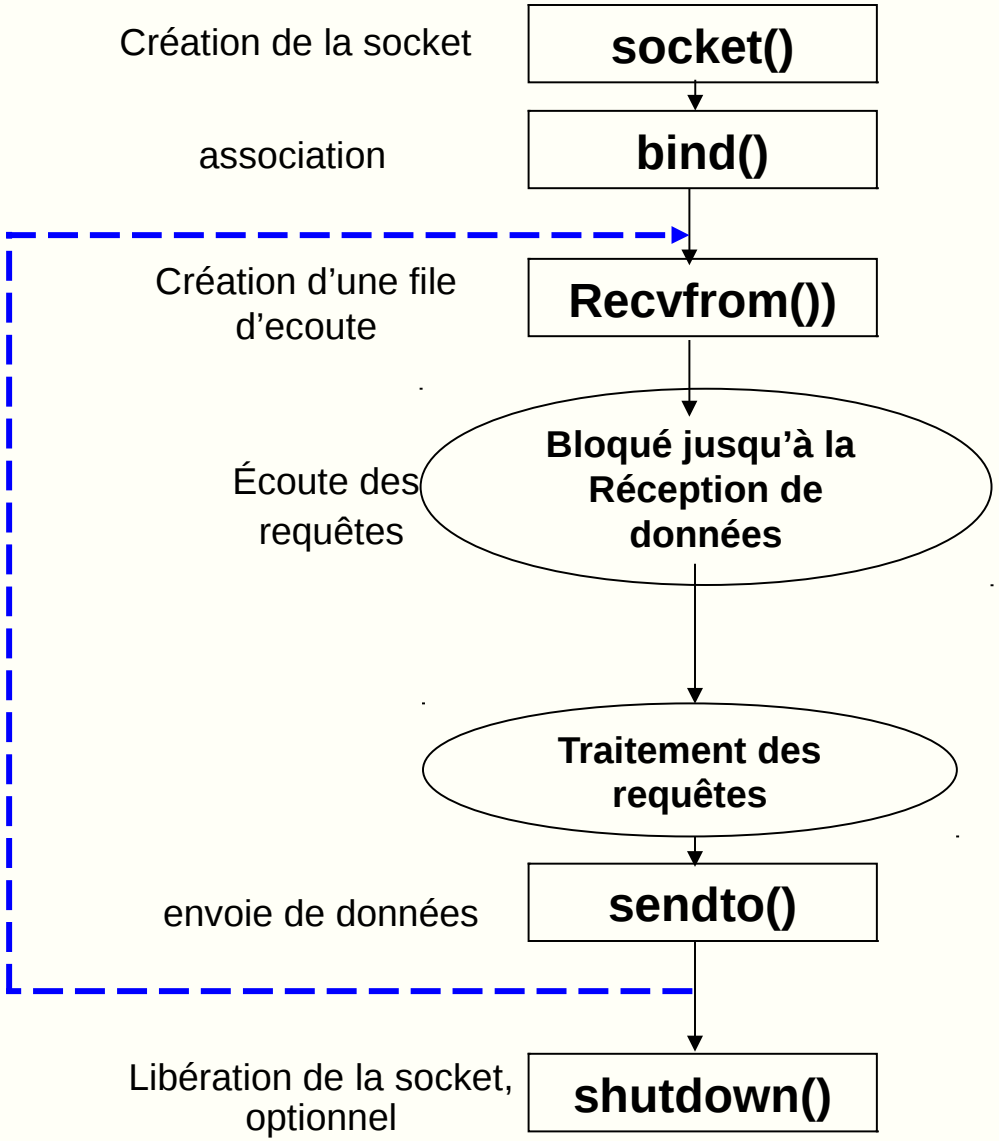
TYPES DE SERVEURS

Critère	Type	Description
Type de connexion	**Orienté connexion**	Établissement d'une **connexion** puis **transmission** puis **déconnexion** entre le serveur et le client.
	Non orienté connexion	Transmission sans connexion au début et sans déconnexion à la fin.
Concurrence	**Iteratif**	Il y a un seul serveur qui entend un seul client à chaque instant.
	Concurrent	Il y a plusieurs serveurs simultanés qui entendent plusieurs clients.

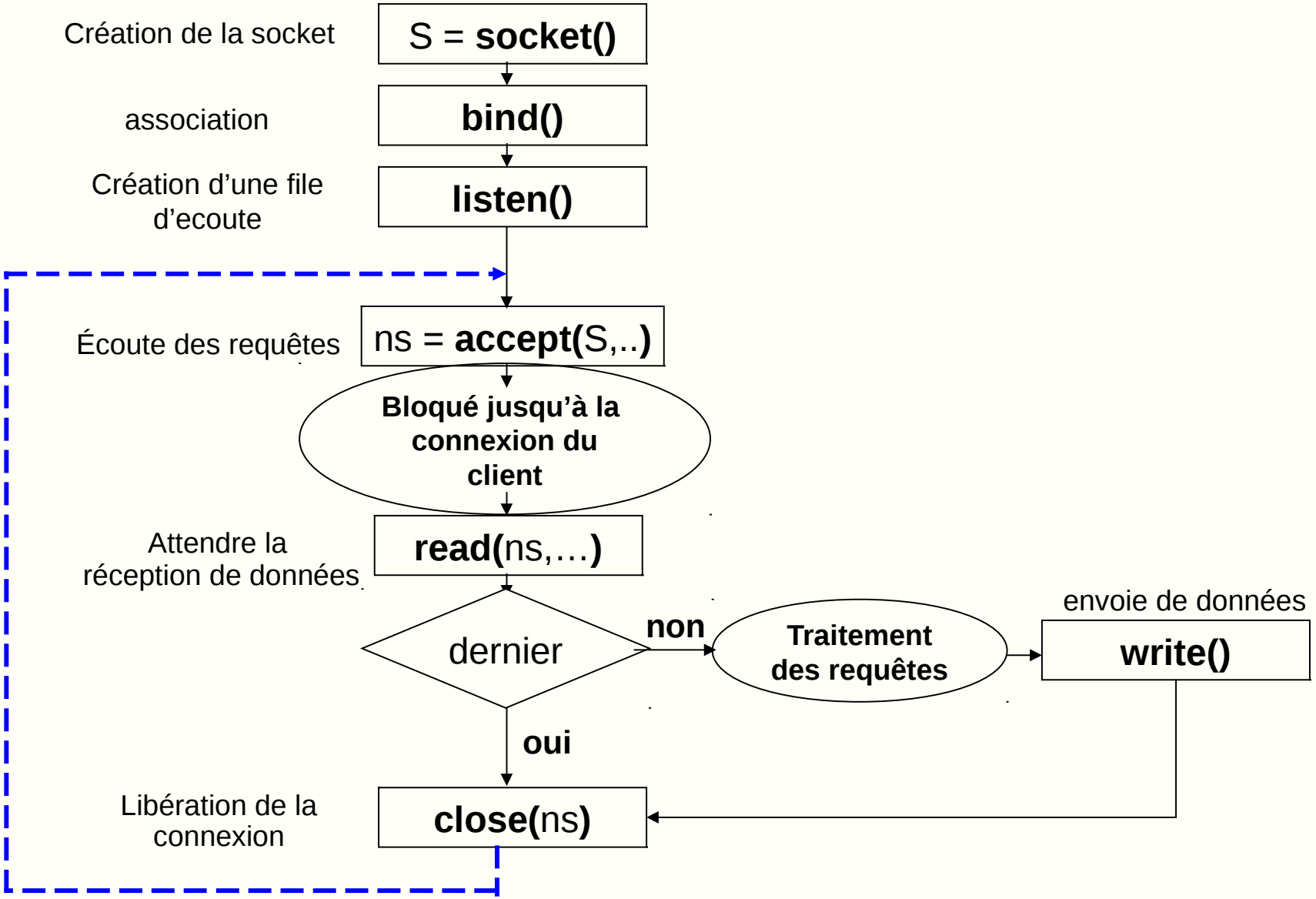
- Un serveur concurrent orienté connexion s'emploie dans des applications types dans lesquelles un client établit une connexion avec le serveur qui dure beaucoup de temps (`telnet`, `FTP`, etc). Dans ce cas, il y a un procesus serveur qui entend à chaque client.

- Un serveur itératif non orienté connexion, s'utilise quand le service sollicité reste un peu de temps (`echo`, `time`...etc).

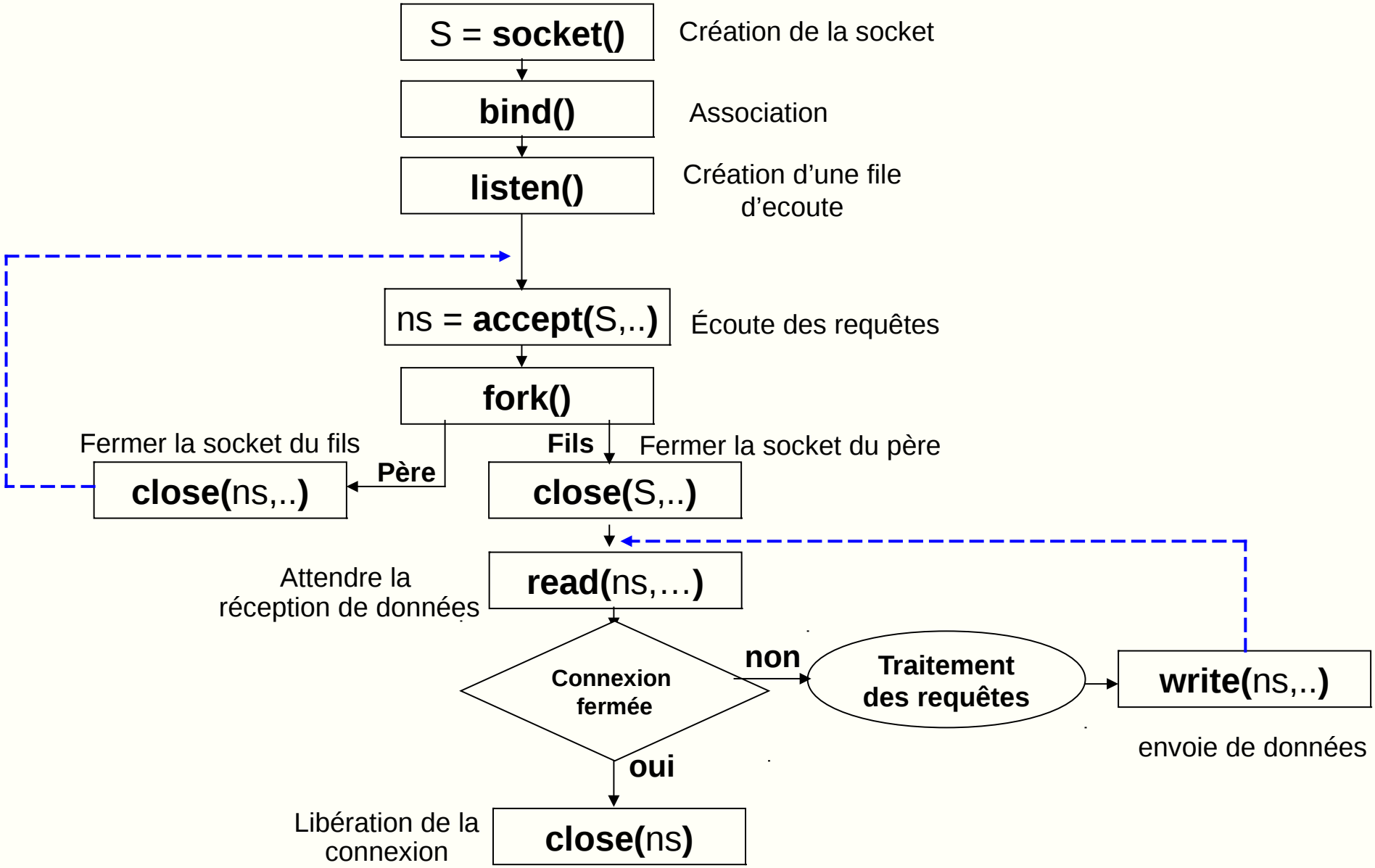
SERVEUR ITERATIF N.O.C.



SERVEUR ITERATIF O.C.



SERVEUR ITERATIF O.C.



SERVEUR CONCURRENTIEL N.O.C.