Programmation système et reseau

cours 5 : les sockets

Communications inter-machines

Qu'est ce que les Sockets ?

- **Les Sockets** constituent une des **A.P.I** (**A**pplication **p**rogram **I**nterface) permettant aux application (processus) d'accéder au réseau pour communiquer.
  - Ensemble de primitives assurant cette communication
  - Cette A.P.I. s'approche de celle permettant la gestion des fichiers UNIX et réutilisent certains des appels systèmes que nous avons vu.

- **Une Socket** est un point de communication (extrémité) par lequel un processus peut émettre / recevoir des données (messages) à destination / en provenance d'un autre processus.

- L'identificateur (**descripteur**) **de Socket** est similaire à L'identificateur (descripteur) de fichiers.

Communications inter-machines

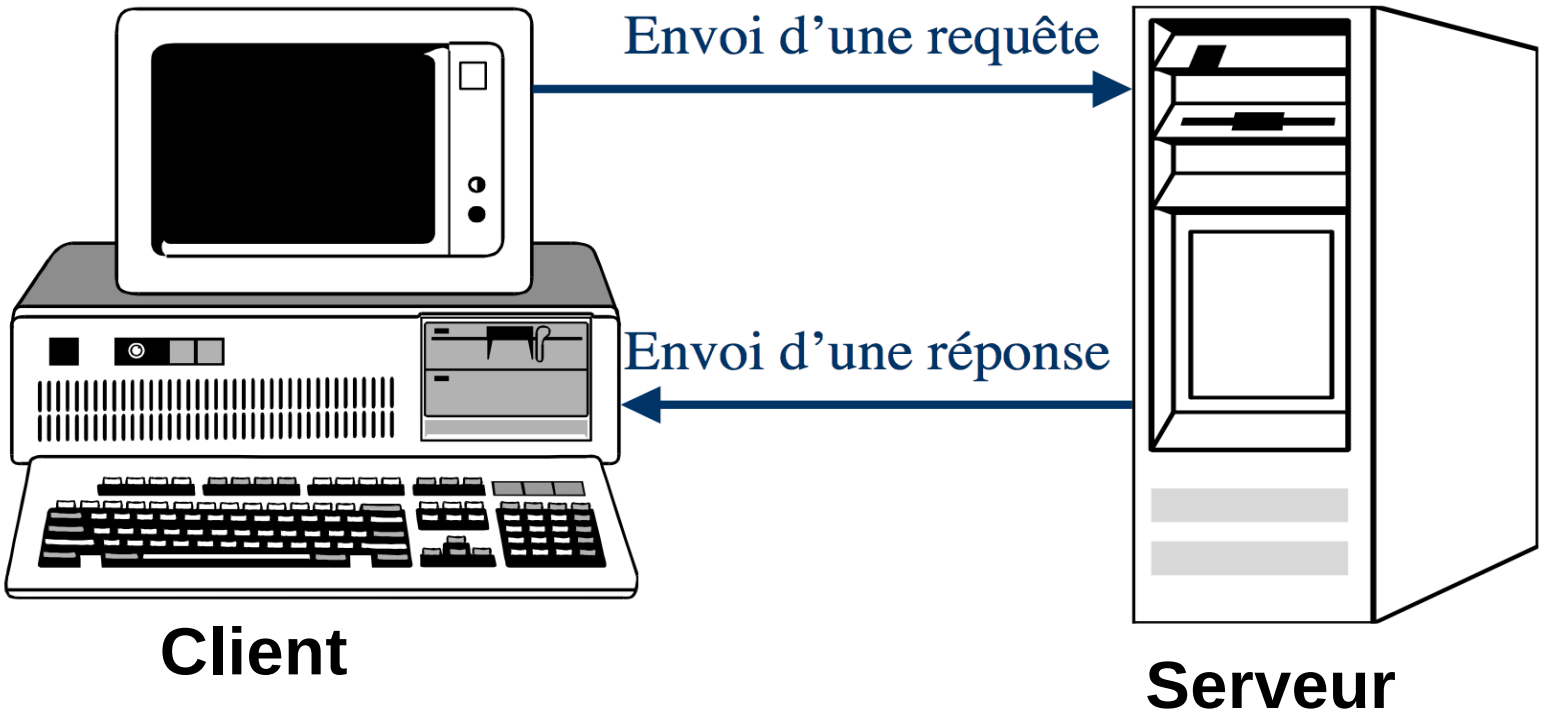
Quelle étendue et pourquoi faire les Sockets ?

- **Les Sockets** peuvent gérer plusieurs familles de protocoles de communication :
  - Internet: **AF_INET**
  - Fichiers locaux: **AF_UNIX**
  - OSI, SN, DEC, CCITT (X25), Appletalk …

- **Modèle Client /serveur:** Le paradigme de communication entre processus le plus commun :
  - **Un serveur :**Processus rendant un service spécifique et installé (puis mis en attente) sur une station bien identifiée (sur un réseau d'ordinateurs).
  - **Un client :** Processus appelant le serveur afin d'obtenir le service. Ce processus est lancé à la demande à partir de n'importe quelle station.

Communications inter-machines

Exemple de communications Client / Serveur



Communications inter-machines sur Internet

Schéma d'adressage TCP / IP et nom de domaine

- Chaque ordinateur (machine) possède une adresse Internet (IP) unique composée de 32 bits
  - L'adresse IP contient assez d'information pour identifier de façon unique un réseau donné et un ordinateur spécifique sur le réseau
  - Exemples d'adresses IP : 132.227.104.15, 193.49.30.38
- Un nom de domaine est organisé de façon hiérarchique :
  - Un code d'identification de l'organisation (com pour commercial, edu pour éducation, gov pour gouvernement, etc...).
  - Un code d'identification du pays (fr pour France, ca pour Canada, us pour les États Unis, etc...).
  - Un code d'identification d'un sous domaine (univ-cergy).
  - Le nom de la machine hôte (www).
  - Exemple **: www.formadep.fr**

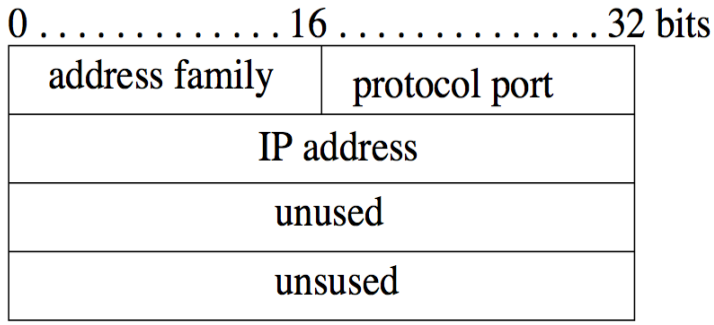
Communications inter-machines sur Internet

Qu'en est il des Sockets sur Internet (AF_INET)

- La structure **struct sockaddr_in** :

- Désignation spécifique permettant d'être identifié de l'extérieur

```
struct in_addr {
        u_long      s_addr;
};
struct sockaddr_in {
        short           sin_family;     /* AF_INET */
        u_short         sin_port;       /* numéro de port */
        struct in_addr  sin_addr;       /* adresse internet */
        char            sin_zero[8];    /* champ de 8 zeros] */
};
```



Structure décrite dans le fichier **<netinet/in.h>.**

TCP Vs. UDP

- **Protocole UDP (*User Datagram Protocol*):**
  - L'entête est minimale.
  - La livraison des données n'est pas fiable.
  - Pas besoin d'établir de connexion.
- **Protocole TCP (*Transport Control Protocol*) :**
  - Le total de contrôle (checksum) est calculé et inclu dans chaque paquet.
  - La livraison des données est fiable.
  - Établir la connexion avant la transmission de données.

Types de Sockets

- **SOCK_DGRAM:** transport de données en mode non connecté. Dans le domaine **AF_INET**, le protocole **UDP** est utilisé. Les frontières de messages sont préservées.

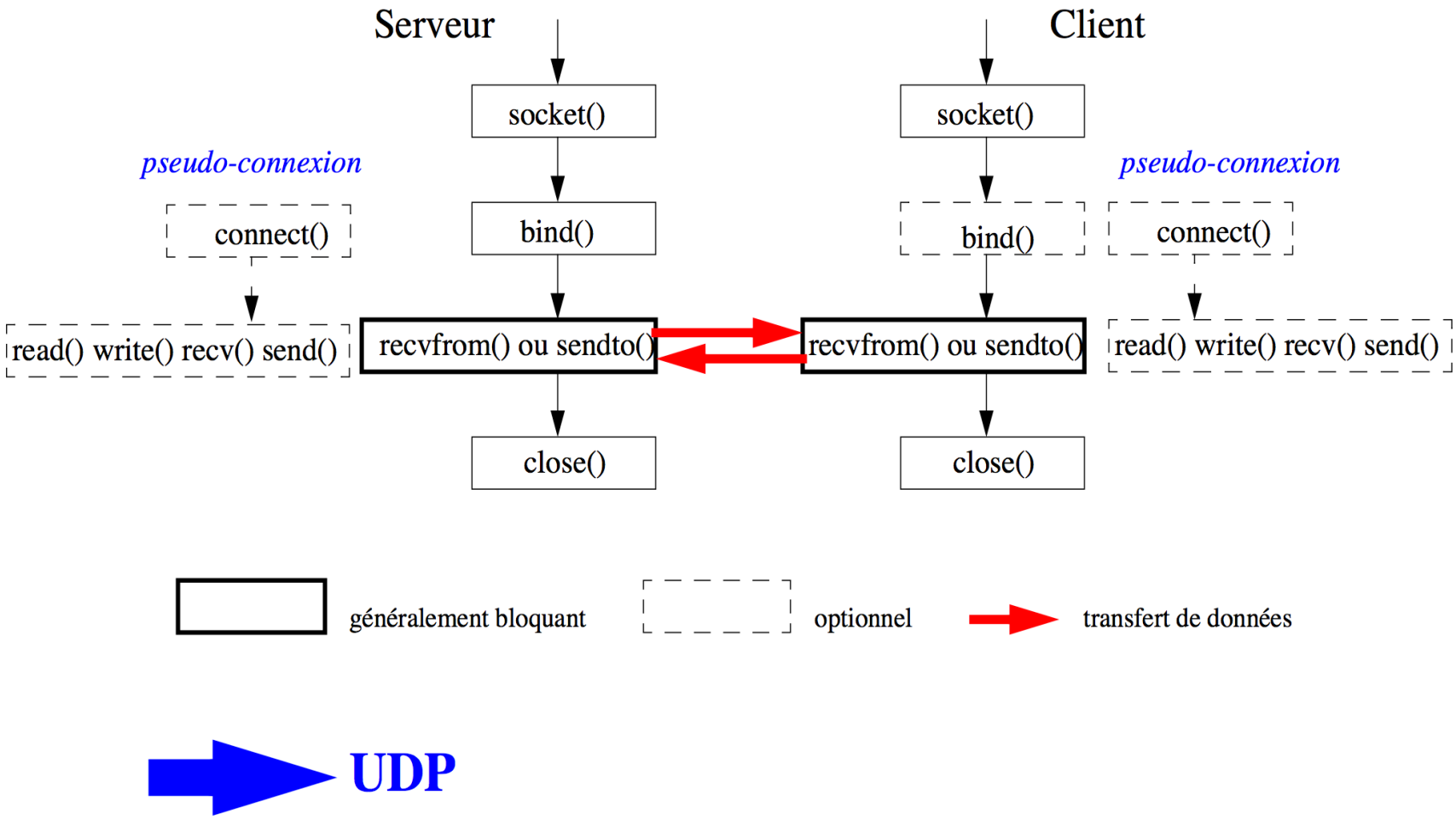
- **SOCK_STREAM:** transport de données en mode connecté. Dans le domaine **AF_INET,** utilisation de **TCP**. Frontières de messages préservées et possibilité d'envoi de messages urgents.

- **SOCK_RAW:** permet l'accès au service de niveau 3 (émission/réception de paquets IP). Utilise des paquets bruts(raw packets) permettant d'envoyer des paquets IP directement sans encapsulation dans les protocoles TCP ou UDP.

- **SOCK_SEQPACKET**: comme le type SOCK_STREAM mais ne permettant pas l'envoi de messages urgents

Enchainement des primitives

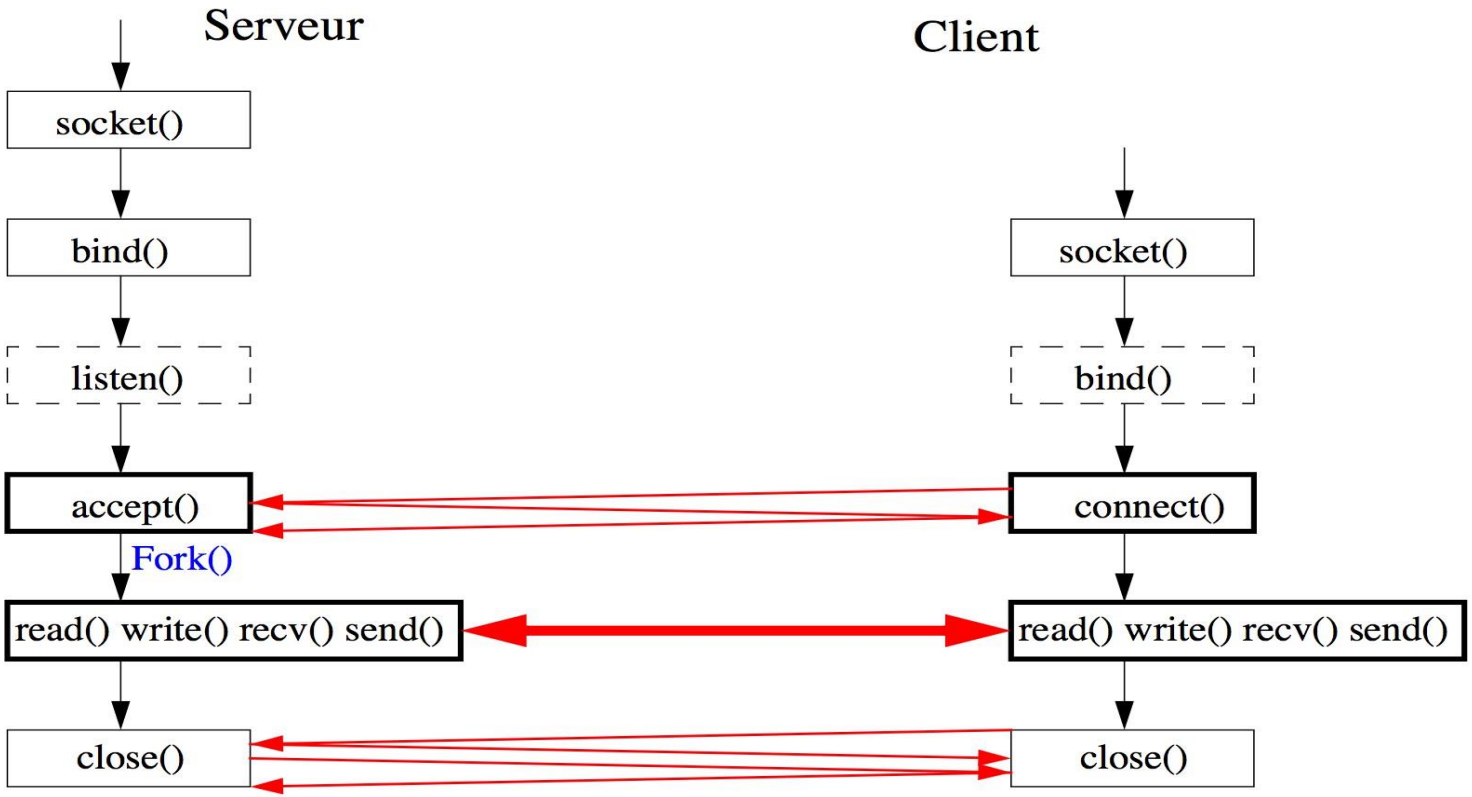
Enchainement simple en mode non-connecté (SOCK_DGRAM)



UDP

Enchainement des primitives

Enchainement simple en mode connecté (SOCK_STREAM)



Chaque "close()" ne ferme qu'un seul sens de communication !

TCP

A.P.I. Sockets : les primitives

Création d'une Socket (socket)

int ***socket***(int *domain*, int *type*, int *protocol*);

- descripteur de la socket, -1 si erreur
- domaine de communication : PF_UNIX ou PF_INET, etc.
  ---
  voir liste dans : **<sys/socket.h>**
- mode de communication : SOCK_STREAM, SOCK_DGRAM, etc.
  ---
  voir liste dans : **<sys/socket.h>**
- protocole à utiliser, (0 => le système choisit le protocole le mieux adapté au mode).
  voir dans : **<netinet/in.h>**

Exemple : s=socket(PF_INET,SOCK_DGRAM,0);

A.P.I. Sockets : les primitives

Renseignement de la Socket avec les informations locales (bind)

int ***bind***(int *s*, struct sockaddr_in * *addsock*, int *lg_addsock*);

- 0 si succès, -1 si erreur
- id. local de la socket créée
- pointeur sur la structure contenant les informations sur la machine locale (domaine, n° de port, et adresse IP) **cf <netinet/in.h>**
- taille de la structure précédente (obtenue avec sizeof)

INADDR_ANY: remplace n'importe quelle adresse

si le numéro de port n'est pas précisé, il est choisi par le noyau

Exemple :
```
if (bind(s, &add_sock,sizeof(add_sock)))
{fprintf(stderr,"Echec dans la réalisation du bind(...)");
 exit(EXIT_BINDING_SOCK);
 }
```

A.P.I. Sockets : les primitives

Renseignement de la Socket avec les informations locales (bind)

- **Socket():** permet de créer une socket (**descripteur de socket**) pour établir un canal de communication.
- **bind():** permet d'associer un nom ou une adresse et un numéro de port au descripteur de socket retourné par la fonction socket( ). Ses arguments sont :
  - le descripteur **s** de la socket considérée.
  - Un pointeur **addr** vers une structure qui contient l'adresse ou le nom assigné et le numéro de port utilisé.
  - La taille **namelen** de la structure vers laquelle pointe **addr**
- Des fonctions de conversion d'adresses doivent être utilisées pour passer d'une représentation binaire à une représentation ASCII et inversement.

A.P.I. Sockets : les primitives

Renseignement de la Socket avec les informations locales (bind)

- Dans le domaine Internet, le domaine **sockaddr** est une structure: **sockaddr_in**

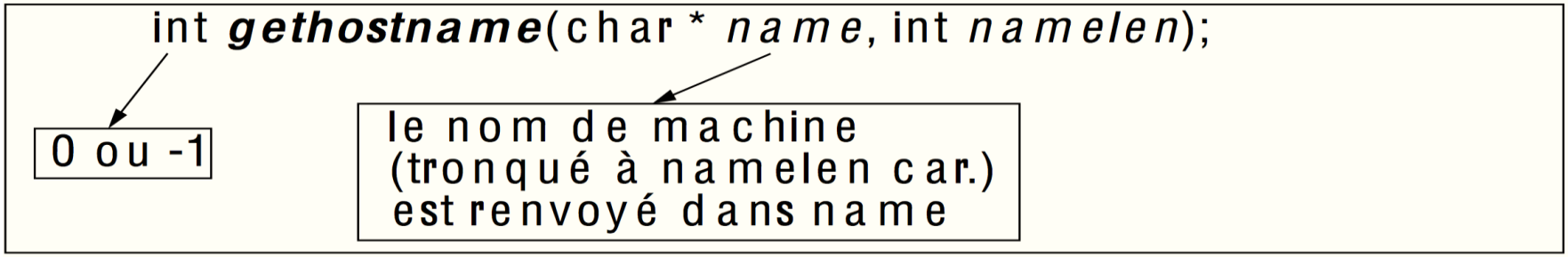
  - Choisir une des adresse, quelconque, de la machine (**INADDR_ANY**) ou spécifier l'adresse d'une interface IP particulière via (**gethostname()** et **gethostbyname()**).

  - Choisir le numéro de port (**getservbyname()** dans le cas d'un service existant) ou bien fixer un numéro non réservé (>= 1024) ou encore laisser le système choisir un numéro de port (en initialisant le champs **sin_port** à **0**).

A.P.I. Sockets : les primitives

Conversions d'adresses (gethostname, gethostbyname)

- Obtenir le **nom** d'une machine locale




```
#include <unistd.h>
#include <stdio.h>

int main() {
   char hostname[256];

   if (gethostname(hostname, sizeof(hostname)) == 0) {
      printf("Nom d'hôte : %s\n", hostname);
   } else {
      perror("gethostname");
   }

   return 0;
}
```

A.P.I. Sockets : les primitives

Conversions d'adresses (gethostname, gethostbyname)

Obtenir **l'adresse** d'une machine

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

voir définition de la structure dans **<netdb.h>**

- La fonction **gethostbyname()** permet de résoudre un nom d'hôte en adresse IP.
- **Paramètre d'entrée:** est le nom de la machine
- **Valeur de retour**
  - pointeur vers une structure **hostent** en cas de succès
  - **NULL** en cas d'erreur

```
struct hostent {
    char *h_name; // nom officiel de l'hôte
    char **h_aliases; // liste d'alias
     int h_addrtype; // type d'adresse (AF_INET)
     int h_length; // longueur de l'adresse
     char **h_addr_list; // liste d'adresses IP
};
```

A.P.I. Sockets : les primitives

Conversions d'adresses (gethostname, gethostbyname)

```
#include <netdb.h>
#include <stdio.h>
#include <arpa/inet.h>

int main() {
   struct hostent *host = gethostbyname("www.google.com");

   if (host == NULL) {
      perror("gethostbyname");
      return 1;
   }

   printf("Nom : %s\n", host->h_name);

   char *ip = inet_ntoa(*(struct in_addr *)host->h_addr_list[0]);
        //inet_ntoa() sert à convertir une adresse IP IPv4 binaire en chaîne de caractères lisible.

printf("IP : %s\n", ip);
   return 0;
}
```

A.P.I. Sockets : les primitives

gethostbyaddr(): est une fonction qui permet de résoudre une adresse IP en nom d'hôte (DNS inverse).

```
struct hostent *gethostbyaddr(const void *addr,  socklen_t len,  int type)
```

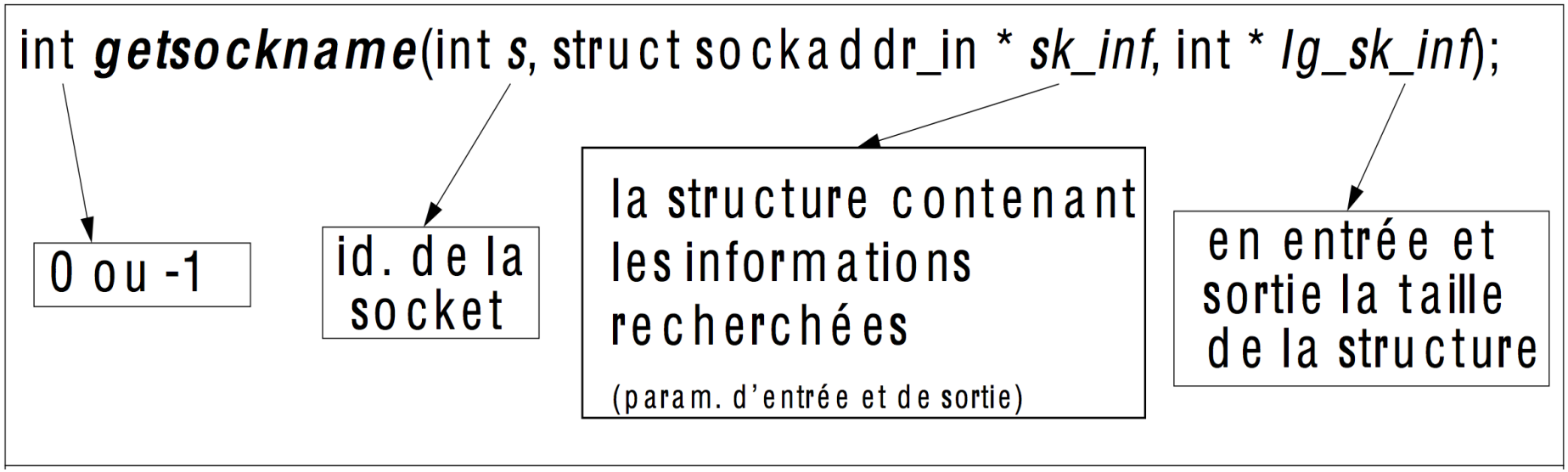
- addr: Pointeur vers l'adresse IP (binaire)
- len: la taille de l'adresse
- type: type d'adresse AF_INET pour IPv4 et AF_INET6 pour IPv6

```
#include <netdb.h>
#include <arpa/inet.h>
#include <stdio.h>

int main() {
   struct in_addr addr;        addr.s_addr = inet_addr(« 100.10.20.5");
   struct hostent *host =   gethostbyaddr(&addr, sizeof(addr), AF_INET);

   if (host != NULL) {
      printf("Nom d'hôte : %s\n", host->h_name);
   } else {
      perror("gethostbyaddr");
   }

   return 0;
}
```

A.P.I. Sockets : les primitives

Conversions d'adresses (getsockname)

- Obtenir l'adresse IP d'une socket dont on connaît le descripteur :



A.P.I. Sockets : les primitives

Attention au format d'échange des information sur le réseau (Little Vs. Big Endian)

- La représentation des nombre sur une machine dépend de son architecture interne et de son OS :
  - Sur les machines Windows, Linux et Mac OS X : Little Endian
  - Sur le réseau Internet : Big Endian (standard réseau universel)
- Il faut donc toujours **convertir** du **Little Endian** au **Big Endian** et vice versa avant et après communication.

@IP	Big Endian	Little Endian
132.227.70.77	84.E3.46.4D	4D.46.E3.84

A.P.I. Sockets : les primitives

Conversion Little Vs. Big Endian (htons, htonl, ntohs, ntohl)

1. Conversion format **machine** vers format **short réseau**

- **htons(short nombre)**
  - ✓ **htons** signifie "**h**ost **to** **n**etwork **s**hort".
  - ✓ Elle prend un nombre de type short (**16 bits**) en entrée, qui est souvent un numéro de port.

```c
#include <stdio.h>
#include <arpa/inet.h>

int main() {
    short port = 8080;  // exemple de port
    short network_port = htons(port);  // conversion en big-endian

    printf("Port local : %d\n", port);
    printf("Port après htons : %d\n", network_port);

    return 0;
}
```

A.P.I. Sockets : les primitives

Conversion Little Vs. Big Endian (htons, htonl, ntohs, ntohl)

1. Conversion format **machine** vers format **long réseau**

- **long htonl(long nombre)**
  - ✓ **htonl** signifie "**h**ost **to** **n**etwork **l**ong“
  - ✓ Elle prend un nombre de type long (32 bits) en entrée, souvent utilisé pour des adresses IP ou des nombres de 32 bits, et convertit ce nombre en format **big-endian** (ordre des octets réseau).

```c
#include <stdio.h>
#include <arpa/inet.h>

int main() {
    long ip_address = 3232235776; // Exemple d'adresse IP 192.168.1.0 en format décimal (4
    long network_order_ip = htonl(ip_address); // Conversion en big-endian (network byte o

    printf("Adresse IP locale : %ld\n", ip_address);
    printf("Adresse IP en network byte order : %ld\n", network_order_ip);

    return 0;
}
```

Conversion Little Vs. Big Endian (htons, htonl, ntohs, ntohl)

2. Conversion format **réseau** vers **format machine**

- **short ntohs(short nombreRes)**
  - ✓ **ntohs** signifié **n**etwork **to h**ost **s**hort
  - ✓ Elle prend un short (16 bits) en entrée, qui a été précédemment converti en network byte order (par exemple avec htons), et le convertit dans l'ordre des octets de la machine locale.

```c
#include <stdio.h>
#include <arpa/inet.h>

int main() {
    short network_port = 0x1234;  // Exemple de port en réseau, format big-endian
    short local_port = ntohs(network_port);  // Conversion en format local (host byte orde

    printf("Port en network byte order : 0x%X\n", network_port);
    printf("Port après conversion en host byte order : 0x%X\n", local_port);

    return 0;
}
```

Conversion Little Vs. Big Endian (htons, htonl, ntohs, ntohl)

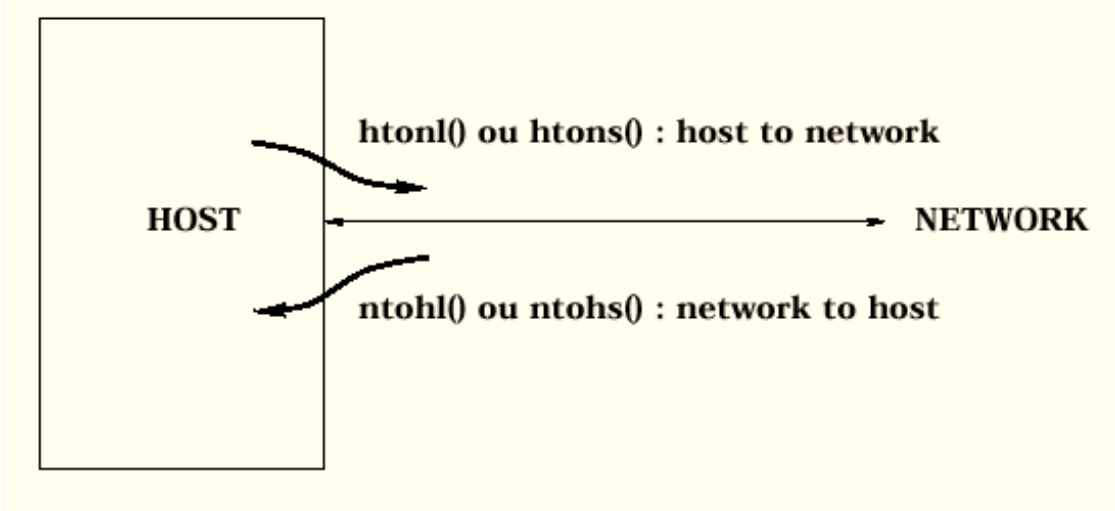
2. Conversion format **réseau** vers **format machine**

- **long ntohl(long nombreRes)**
  - ✓ **ntohl** signifié **n**etwork **to h**ost **l**ong
  - ✓ Elle prend un nombre de type long (32 bits) en entrée, qui a été précédemment converti en network byte order (par exemple, avec htonl), et le convertit dans l'ordre des octets de la machine locale.

```c
#include <stdio.h>
#include <arpa/inet.h>

int main() {
    long network_ip = 0x7F000001;  // Exemple d'adresse IP en format réseau (127.0.0.1)
    long local_ip = ntohl(network_ip);  // Conversion en format local (host byte order)

    printf("Adresse IP en network byte order : 0x%lX\n", network_ip);
    printf("Adresse IP après conversion en host byte order : 0x%lX\n", local_ip);

    return 0;
}
```

A.P.I. Sockets : les primitives

Conversion Little Vs. Big Endian (htons, htonl, ntohs, ntohl)



A.P.I. Sockets : les primitives

Exemple d'ouverture et d'attachement d'une socket UDP

```
int ouvreSocket(int port)
{
  int skD;
  size_t tailleAd;
  struct sockaddr_in adLocale;
  adLocale.sin_family = AF_INET;                    /* Type de la socket (TCP/IP) */
  adLocale.sin_port = htons(port);                  /* Affectation du port local */
  adLocale.sin_addr.s_addr = htonl(INADDR_ANY); /* Identificateur de l'hote */
  skD = socket(AF_INET, SOCK_DGRAM, 0);             /* Créé socket UDP (déconnectée) */
  if (skD == -1)
  {
    perror("Erreur lors de la création de la socket\n");
    return -1;
  }
  tailleAd = sizeof(adLocale);
  retVal = bind(skD, (struct sockaddr*) &adLocale, tailleAd); /* Attache la socket
  */ if (retVal == -1)
  {
    perror("Erreur lors du bind\n");
    close(skD);
    return -1;
  }
  return skD;
}
```

A.P.I. Sockets : les primitives

Longueur maximum de la file d'attente des communications en cours d'établissement (listen)

int ***listen***(int *s*, int *backlog*);

- 0 si succès, -1 si erreur
- id. de la socket locale
- taille de la file d'attente des demandes de communication (par défaut= 5)

Exemple :

```
if (listen(sock,MAXCONREQ) < 0)
    {perror("Echec du listen");
     exit(EXIT_ERR_LISTEN);
    }
```

A.P.I. Sockets : les primitives

Etablissement de la (pseudo-)connexion en renseignant la Socket pour le destinataire (connect)

int ***connect***(int *s*, struct sockaddr_in * *dest*, int *lg_dest*);

- 0 si succès
-1 si erreur
- id. de la socket locale
- pointeur sur la structure contenant les informations sur la **socket distante** (récupérable avec *gethostbyname(...)*)
- taille de la structure précédente

Exemple :
```
if (connect(sock, &addsock, sizeof(addsock)) < 0)
    {perror("Echec de lien de socket");
     exit(EXIT_ERR_CONNECT_SOCK);
    }
```

A.P.I. Sockets : les primitives

Définition de l'ensemble des connexions acceptables (accept)

```
int accept(int s, struct sockaddr_in * from, int * fromlen);
```

- `s` : id. de la **nouvelle** socket ou -1 si erreur
- `from` : la structure contenant les informations sur les émetteurs acceptables
- `fromlen` : la taille de la structure est mise ici

Exemple :

```
new_sock = accept(s, &add_initiator, &size_add_initiator);
if (new_sock < 0)
        {perror("Echec de l'accept");
         exit(EXIT_ERR_ACCEPT);
        }
```

A.P.I. Sockets : les primitives

Réception de données – strictement identique à l'appel système habituelles read

```
int read(int s, char * buf, int nbyte);
```

- **s** : nombre d'octets lus
  0 si la connexion est fermée
  ou -1 si erreur
- ***buf*** : les données reçues sont mises dans la zone pointée par *buf*
- ***nbyte*** : nombre maximum d'octets à lire

Exemple :

```
if ((nblu=read(new_sock, buf, MAXBUFFER)) <= 0)
        {perror("Echec en réception de données");
         exit(EXIT_ERR_RECV_STREAM);
        }
```

A.P.I. Sockets : les primitives

Réception de données – autres primitives similaires à read

```
int recvfrom(int s, char * buf, int nbyte, int flag,
                    struct sockaddr_in * from, int * fromlen);
```

=> utile surtout en mode non-connecté pour identifier l'émetteur

---

```
int recvmsg(int s, struct msghdr * msg, int flag);
```

=> pour réduire le nombre de paramètres à fournir directement

---

```
int recv(int s, char * buf, int nbyte, int flag);
```

=> utilisable plutôt en mode connecté

A.P.I. Sockets : les primitives

Emission de données – Identiques ou similaires à l'appel système write

```
int write(int s, char * buf, int nbyte);
```

```
int sendto(int s, char * buf, int nbyte, int flag,
                        struct sockaddr_in * dest, int    destlen);
```

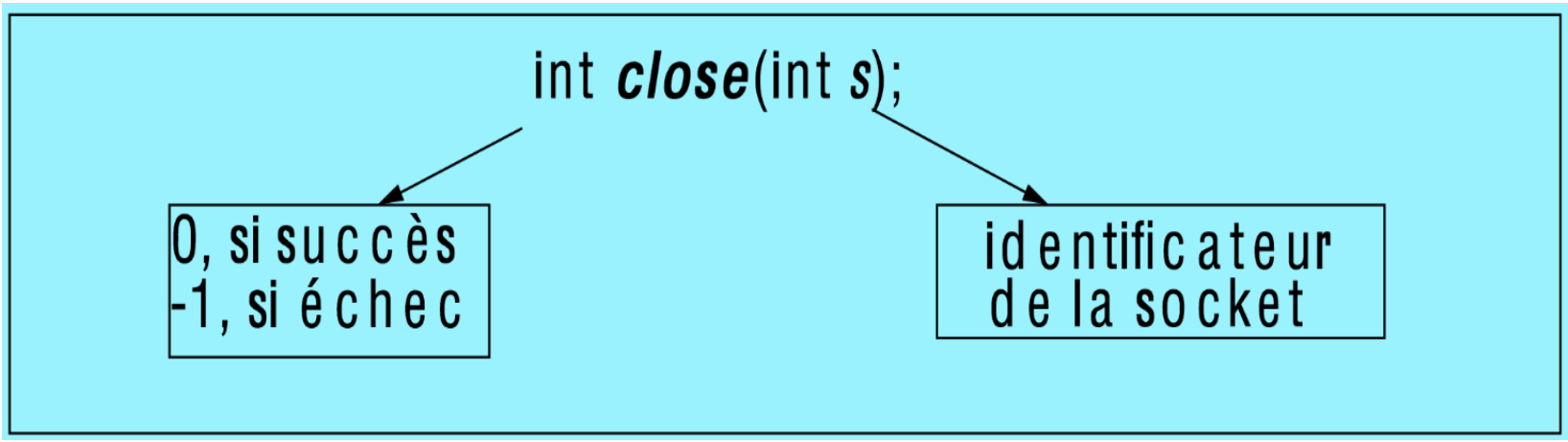
```
int sendmsg(int s, struct msghdr * msg, int flag);
```

```
int send(int s, char * buf, int nbyte, int flag);
```

A.P.I. Sockets : les primitives

Primitive de libération de la socket (close)

- Libère le descripteur de socket
- Libère l'espace de stockage associé à la socket
- Libère la connexion si elle existe.



A.P.I. Sockets : les primitives

Exemple de code courant en TCP (échange de messages: client /serveur)

Partie serveur

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <arpa/inet.h>
#include <unistd.h>

#define PORT 8080  // Le port d'écoute
#define BUFFER_SIZE 1024  // Taille du buffer de réception

int main() {
int server_fd, new_socket;
struct sockaddr_in address;
char buffer[BUFFER_SIZE] = {0};

if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
   perror("Erreur lors de la création du socket");
    exit(EXIT_FAILURE);
                          }
address.sin_family = AF_INET;
address.sin_addr.s_addr = INADDR_ANY;  // Accepte les connexions sur n'importe quelle adresse
address.sin_port = htons(PORT);
```

A.P.I. Sockets : les primitives

Exemple de code courant en TCP(échange de messages: client /serveur)

Partie serveur

```
// Lier le socket à l'adresse et au port
 if (bind(server_fd, (struct sockaddr*)&address, sizeof(address)) < 0) {
                perror("Erreur de binding");
                                exit(EXIT_FAILURE);    }
// Écouter les connexions entrantes
 if (listen(server_fd, 3) < 0) {       perror("Erreur lors de l'écoute");       exit(EXIT_FAILURE);    }
                printf("En attente de connexion...\n");
new_socket= accept(server_sd , clien_address, sizeof(client_address));
// Lire le message du client
read(new_socket, buffer, BUFFER_SIZE);    printf("Message du client : %s\n", buffer);
// Répondre au client
char *response = "Message reçu!";
send(new_socket, response, strlen(response), 0);    printf("Réponse envoyée au client\n");
close(new_socket);
close(server_fd);
return 0;
}
```

A.P.I. Sockets : les primitives

Exemple de code courant en TCP (échange de messages: client /serveur)

Partie Client

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <arpa/inet.h>
#include <unistd.h>

#define SERVER_IP "127.0.0.1"  // Adresse IP du serveur
#define PORT 8080  // Le port d'écoute
#define BUFFER_SIZE 1024  // Taille du buffer de réception

int main() {
int int sock = 0;
struct sockaddr_in server_address;
char buffer[BUFFER_SIZE] = {0};

// Création du socket
if ((sock = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
   perror("Erreur lors de la création du socket");
    exit(EXIT_FAILURE);
                          }
address.sin_family = AF_INET;
address.sin_port = htons(PORT);
```

A.P.I. Sockets : les primitives

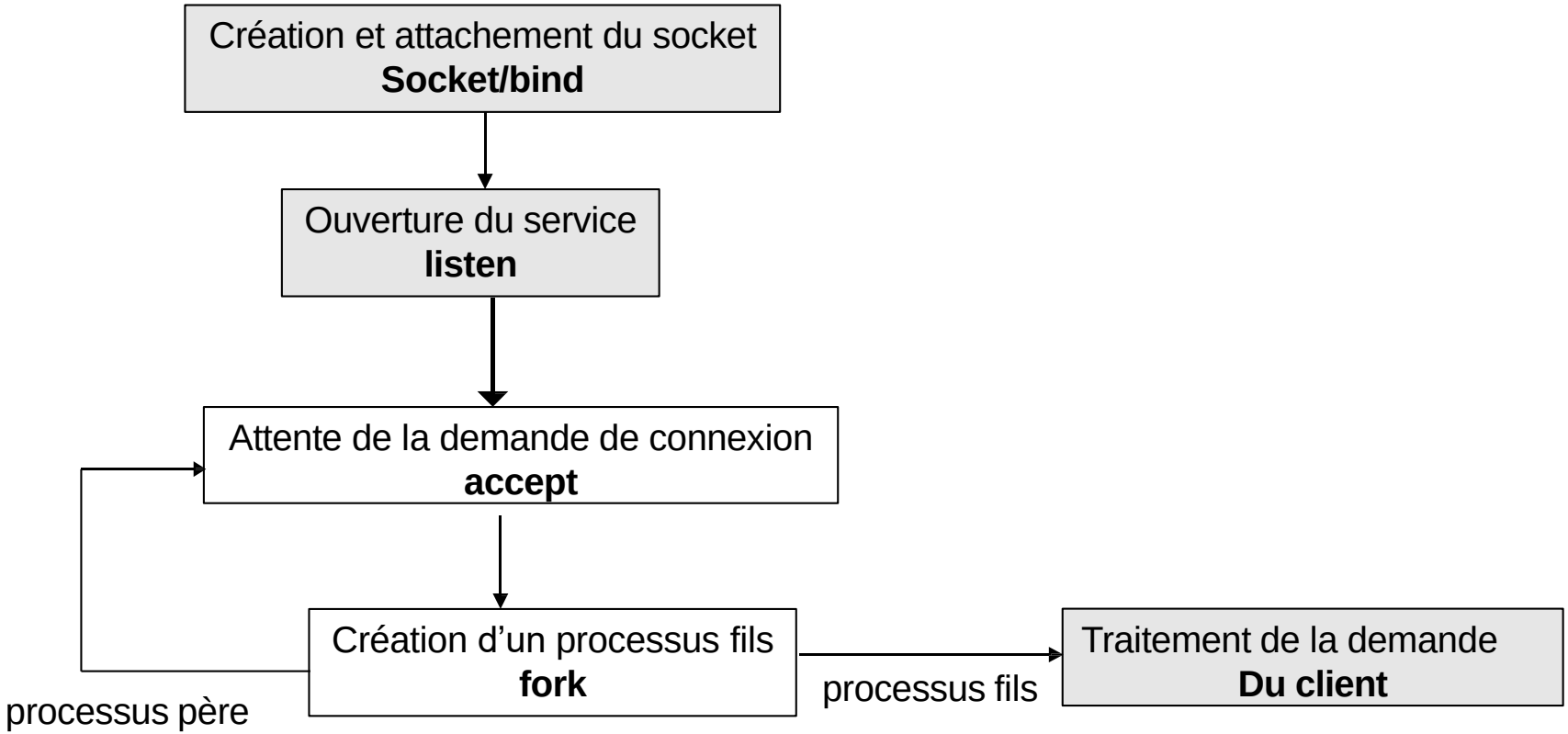
Exemple de code courant en TCP(échange de messages: client /serveur)

```
// Se connecter au serveur
if (connect(sock, (struct sockaddr*)&server_address, sizeof(server_address)) < 0) {
        perror("Erreur lors de la connexion");
         exit(EXIT_FAILURE);    }


// Envoyer un message au serveur
char *message = "Bonjour, serveur !";
send(sock, message, strlen(message), 0);
printf("Message envoyé au serveur\n");


// Lire la réponse du serveur
read(sock, buffer, BUFFER_SIZE);
 printf("Réponse du serveur : %s\n", buffer);


// Fermer le socket
close(sock);
return 0;
}
```

Partie Client

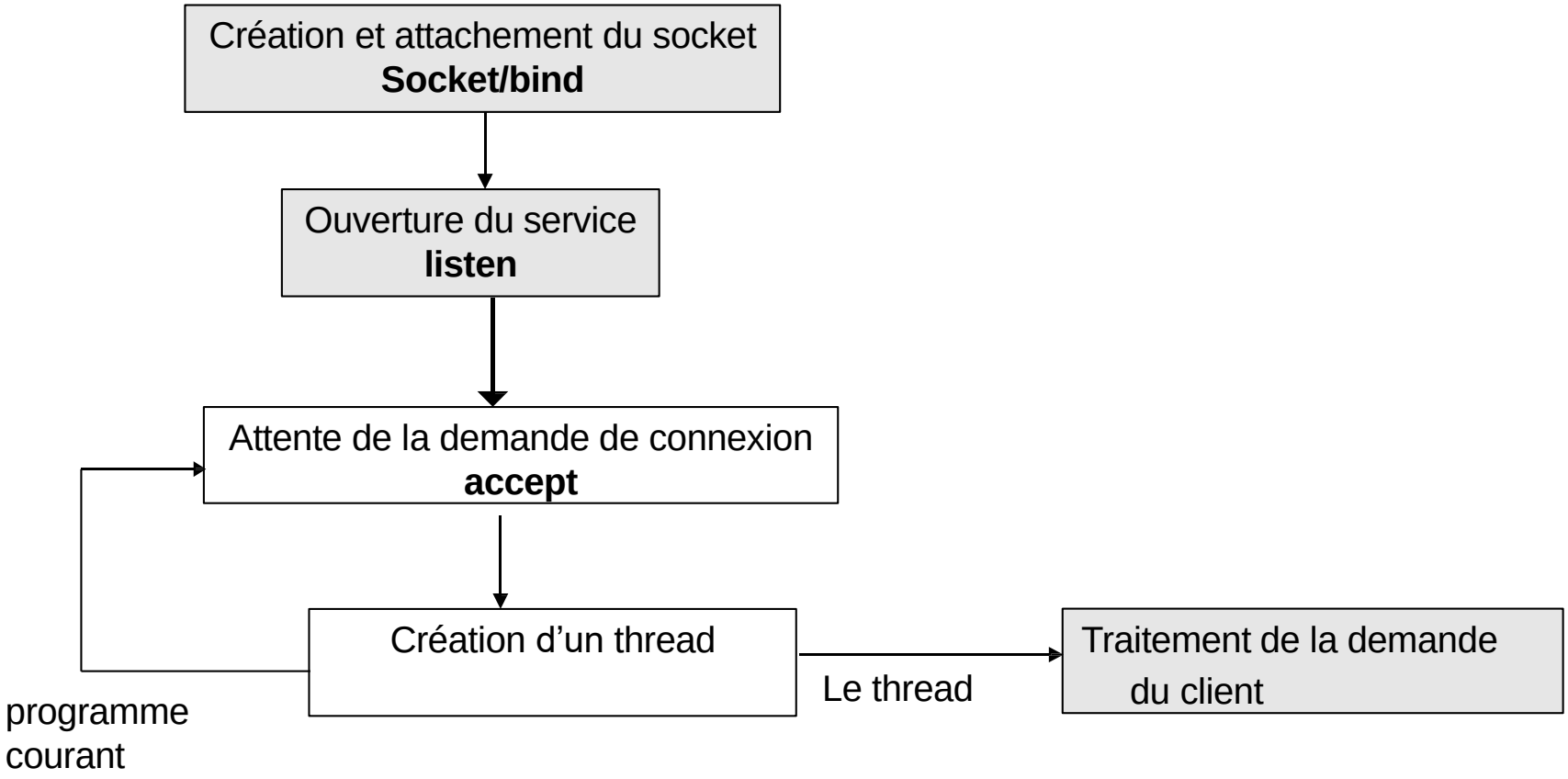
A.P.I. Sockets : les primitives

Différentes étapes de communications – cas de connexions concurrentes



A.P.I. Sockets : les primitives

Différentes étapes de communications – cas de connexions concurrentes



A.P.I. Sockets : les primitives

Exemple de connexions concourants (TCP)

```
skDesc = ouvreSocket(portLocal);
tailleAd = sizeof(adClient);
while (1)
{
  skServDesc = accept(skDesc, &adClient, &tailleAd);
  /* Connexion établie, AdClient renseigné */

  numPID = fork();
  if (numPID == 0)     /* Fils, s'occuper  du client actuel */
  {


    /* TRAITEMENT DU CLIENT */

    close (skServDesc);                    /* Ferme socket de service */

    return 0;
  }
  else
  {
    close(skServDesc);    /* Le père n'a pas besoin de socket de service
  }
}
```