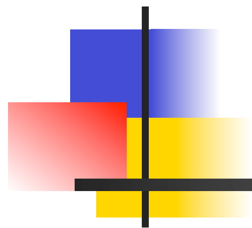
TELECOM PARIS
école nationale supérieure des télécommunications



BSD Sockets

L. Pautet

- L'IETF a défini une série de protocoles : IPv4 et IPv6 pour l'adressage des machines; TCP et UDP échanger des messages au dessus de IP :
  - Ils sont une norme de facto pour la partie « utilisateur » du réseau, celle qui est directement accessible,
  - Les autres protocoles relèvent du cœur du réseau, ou des couches applicatives (e.g. telnet, DHCP, …)
  - L'IETF ne définit qu'un format de messages et les automates pour établir des connexions, échanger des messages, gérer les erreurs, la fragmentation, …

Modes d'adressage IPv4

- Les adresses sont sur 4 octets, en notation dite pointée :
  - 137.194.2.34, avec netid = 137.194, hostid = 2.34
- Deux éléments composent l'adresse : netid et hostid
  - Netid : identifiant du réseau
  - Hostid : identifiant de l'hôte sur le réseau
- Utilisé pour le routage des paquets
- IPv6 lève les limites d'adressage : les adresses passent de 4 à 6 octets, et ajoute des mécanismes de configuration

Modes d’adressage IPv4

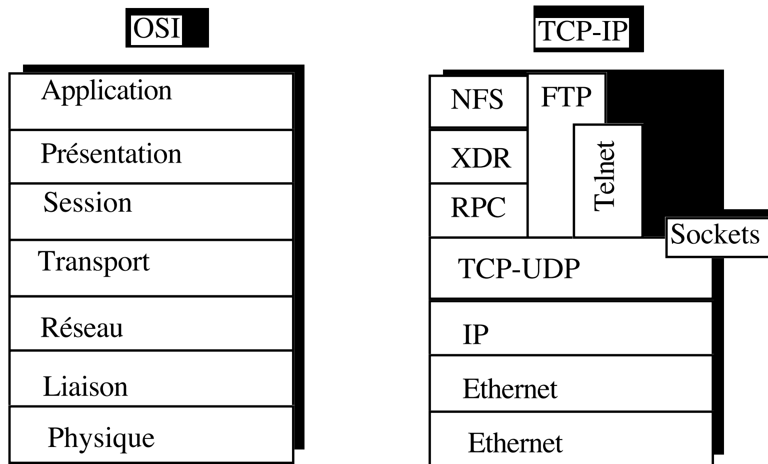
- Les netid sont répartis en classes :
  - Classe A, netid codé sur 1 octet :
    - les adresses réseau vont de 1.0.0.0 à 126.0.0.0 (127 : réservé à localhost),
  - Classe B, sur 2 octets (les 2 premiers bits sont à 10) :
    - les adresses réseau vont ainsi de 128.0.0.0 à 191.255.0.0
  - Classe C, sur 3 octets (les 3 premiers bits sont à 110) :
    - les adresses réseau vont de ainsi 192.0.0.0 à 223.255.255.0
  - Classe D (multicast), netid sur un octet, de 224 à 239

Rappels: modes d’adressage IPv4

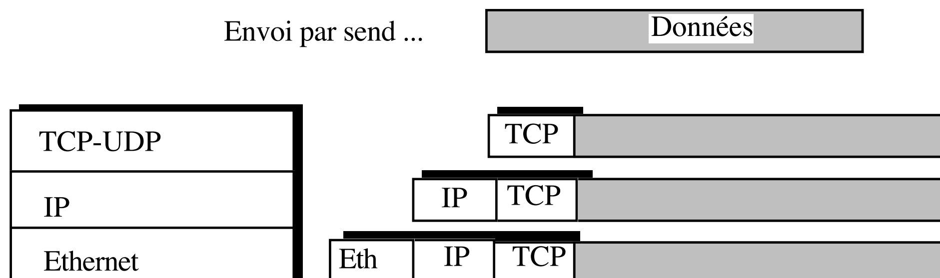
- Adresses spéciales
  - 127.0.0.1 : « localhost », bouclage local
  - 0.0.0.0 : adresse de destination invalide
  - 255.255.255.255 : adresse de diffusion

- Adresses réservées pour les réseaux locaux (non accessibles depuis l’extérieur) :
  - Classe A : netid 10, hostid de 0.0.1 à 255.255.254
  - Classe B : netid 172.16 à 172.31, hostid de 0.1 à 255.254
  - Classe C : netid 192.168.0 à 192.167.255, hostid de 1 à 254

Modèle OSI vs TCP/IP



Structure des trames:



- Exemple de services sur IP :
  - `ftp-data 20/tcp`
  - `ftp 21/tcp`
  - `telnet 23/tcp`
  - `smtp 25/tcp`
  - `time 37/tcp`

  - `sunrpc 111/udp`
  - `sunrpc 111/tcp`

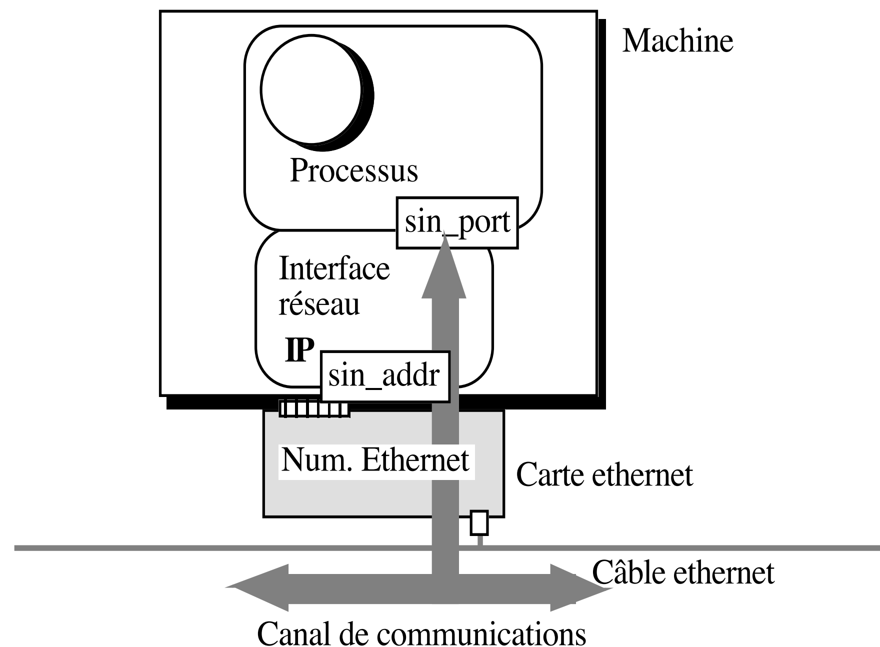
- TCP : protocole orienté connexion au dessus de IP
  - Protocole fiable (avec gestion des erreurs)
    - Paquets de taille maximale (MTU), avec un mécanisme de segmentation pour les données volumineuses
  - Mécanisme de gestion de flux pour éviter de saturer le réseau (algo. de Nagle)
- UDP : protocole plus simple que TCP
  - Si, « tout va bien » on évite la complexité de TCP
    - On perd le séquencement des paquets, la gestion du flux
    - Utile pour les applications légères, le temps réel mou (multimédia)

BSD Sockets
une API pour IP, TCP et UDP

- Aucune API définie par l' IETF
- BSD Sockets : modèle employé en Unix
  - Repris sur d'autres plates-formes, dont Windows
- Calquée sur le modèle de ressources Unix :
  - Tout est fichier
  - Les sockets suivent la sémantique producteur/consommateur et s' utilisent avec open/read/write
  - Liens forts avec l'API standard pour le cycle de vie de la connexion

Domaine des sockets: AF_INET

- Dans AF_INET, le socket est un port vers la couche 4 du protocole Internet (IP) :



- L'appelant (ou client) :
  - Crée une socket ;
  - Construit l’adresse réseau (IP + port) du serveur (connu généralement par le nom)
    - Pas d’annuaire pour les services, seulement pour les noms de sites (DNS)
  - Se connecte au serveur (avec l'adresse construite) (attribution automatique d’un port au client) ;
  - Lit ou écrit sur la socket ;
  - Ferme la socket.

Mode connecté TCP pour le serveur

- L'appelé (ou serveur) :
  - Crée une socket ;
  - Associe une adresse socket (adresse Internet et numéro de port) au service ;
  - Se met en attente des connexions entrantes ;
  - Pour chaque connexion entrante :
    - Accepte la connexion (une nouvelle socket est créée);
    - Lit ou écrit sur la nouvelle socket ;
    - Ferme la nouvelle socket.

Canevas avec API C d’un client TCP

1. Construire l’adresse du serveur
2. Annuaire pour hôte mais pas pour port
3. Demander l’établissement de la communication

```
Socket = socket(PF_INET, SOCK_STREAM, 0);
struct sockaddr ServerAddr;
ServerAddr.sin_addr = gethostbyname(« www.enst.fr »);
ServerAddr.sin_port = ServerPort;
connect(Socket, &ServerAddr, sizeof(struct sockaddr));
```

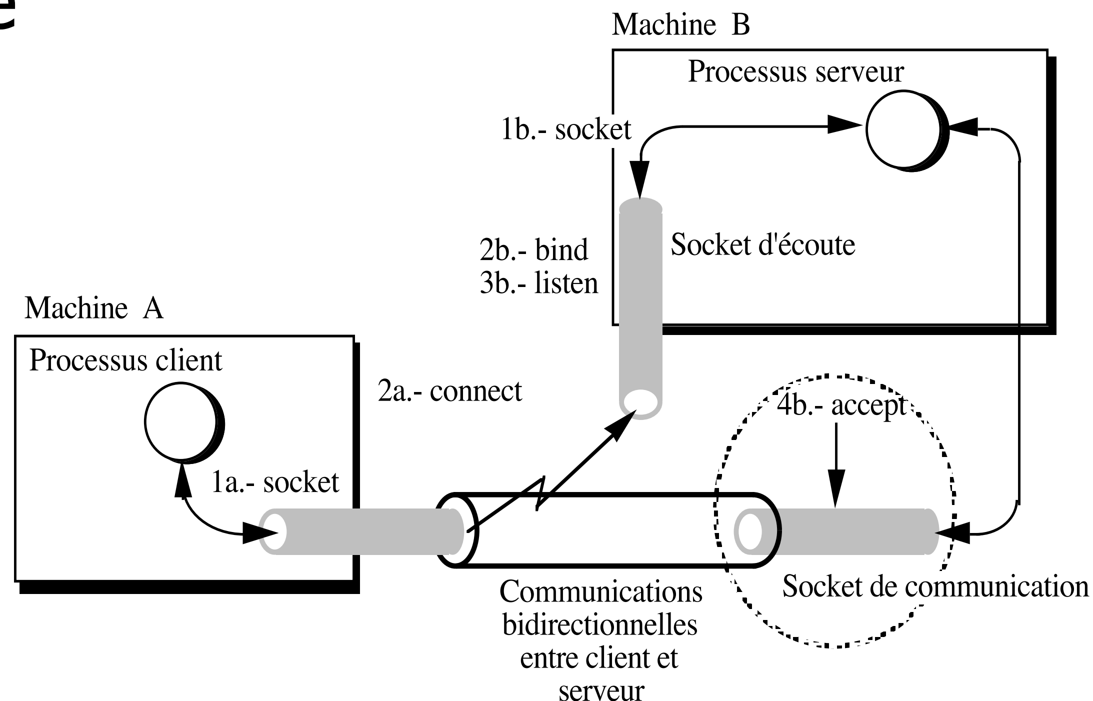
Canevas avec API C d'un serveur multi-tâches TCP

1. Création d'une socket d'écoute
2. Attente d'une demande de connexion
3. Gestion concurrente des communications (délégation à un fils)

```
struct sockaddr ServerAddr, ClientAddr;
int ClientAddrLen;
ServerSocket = socket(PF_INET, SOCK_STREAM, 0);
bind(ServerSocket, &ServerAddr, sizeof(struct sockaddr));
while (1){// Boucle d'attente sur le port d'ecoute
   Socket = accept(ServerSocket, &ClientAddr, &ClientAddrLen);
   if (fork() == 0){
      close(ServerSocket);
      Handle(Socket);
   } else
      close(Socket);
}
```

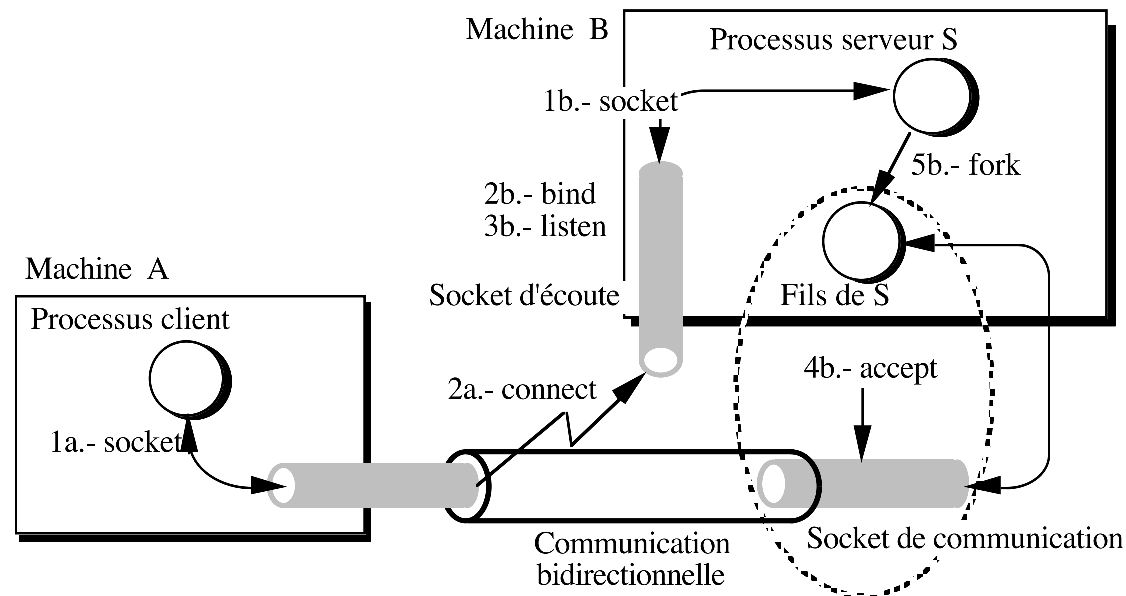
Mode connecté (TCP)

- Schéma fonctionnel de la communication de base



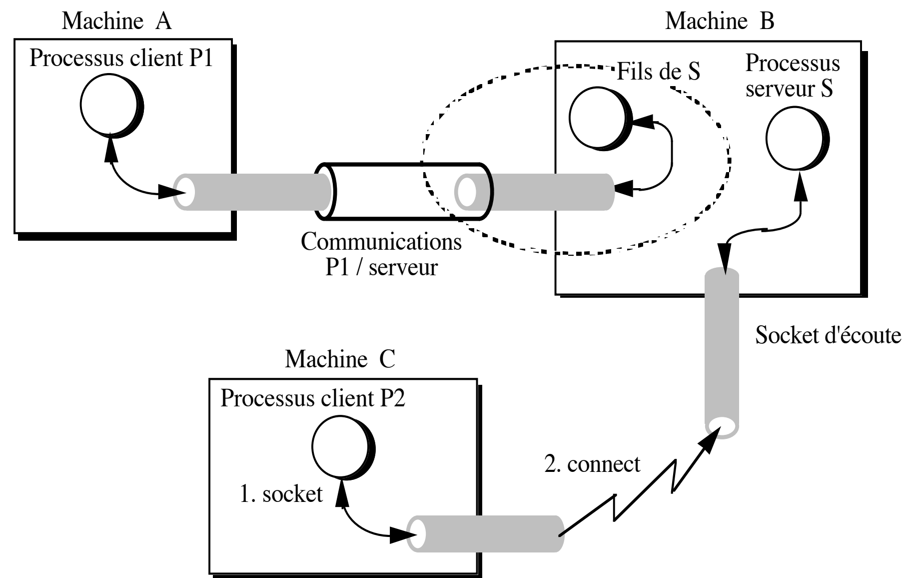
Mode connecté : gestion concurrente des clients (1/2)

- Communication avec un premier client



Mode connecté : gestion concurrente des clients (2/2)

- Communication avec un second client

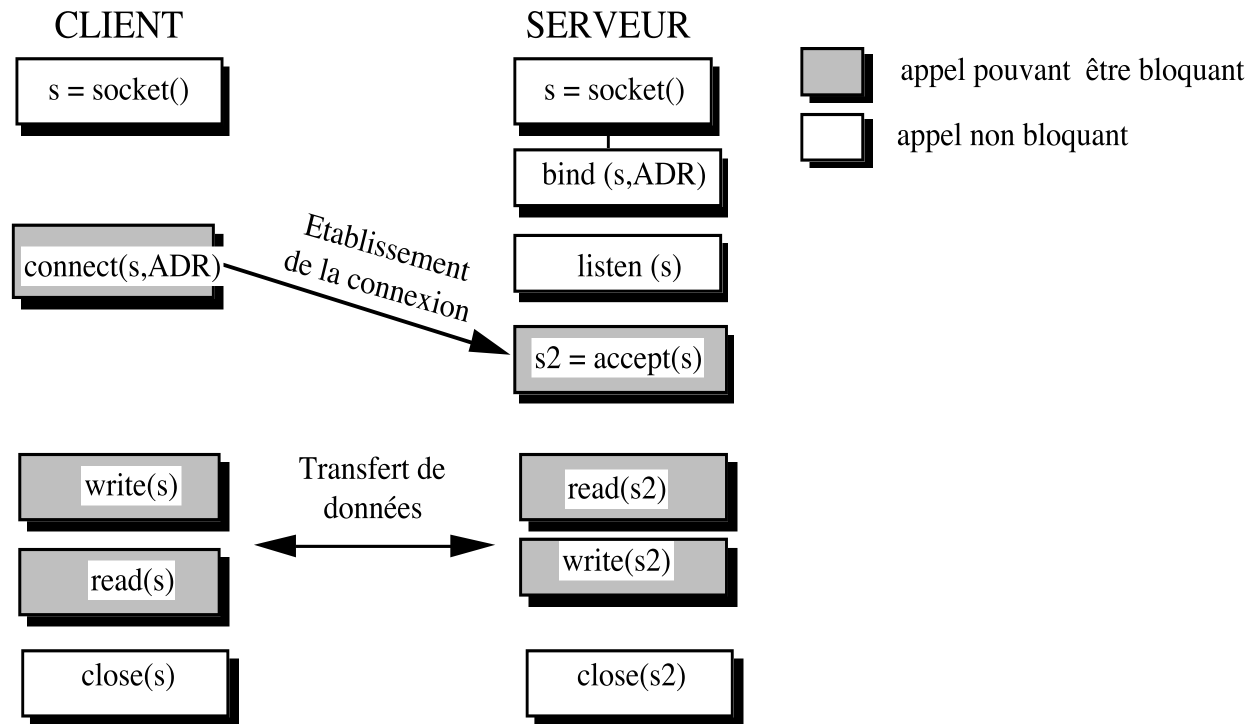


Création de sockets

- Socket = socket(Domain, Type, Protocol)
  - Socket: entrée dans la table des fichiers ouverts par le processus

- Pour lier une socket à un port (serveur):
  - bind(Socket, &ServerAddr, ServerAddrLen);
    - Socket numéro renvoyé par la primitive socket
    - ServerAddr structure définissant le serveur
    - ServerAddrLen taille de cette structure (sizeof)

Schéma de communication en mode connecté (TCP)



Communication en mode connecté (TCP)

- Côté client
  - Appel à la fonction :
  - connect(Socket, &ServerAddr, ServerAddrLen)

  - Cette fonction lance une demande d’établissement de communication vers un serveur dont on a donné l’adresse (IP, port) dans la structure ServerAddr

- Côté serveur de socket d'écoute ServerSocket
  - listen(ServerSocket, Nb_Clients)
    - taille de la file d'attente sur la socket d'écoute du serveur
  - ClientSocket =
    accept(ServerSocket, &ClientAddr, &ClientAddrLen)
  - Attente d'une demande de connexion (par connect) :
    - ClientSocket : socket renvoyé par accept dès qu'il reçoit une demande de connexion par un client
    - ServerSocket : socket d'écoute du serveur
    - ClientAddr : adresse décrivant le client appelant

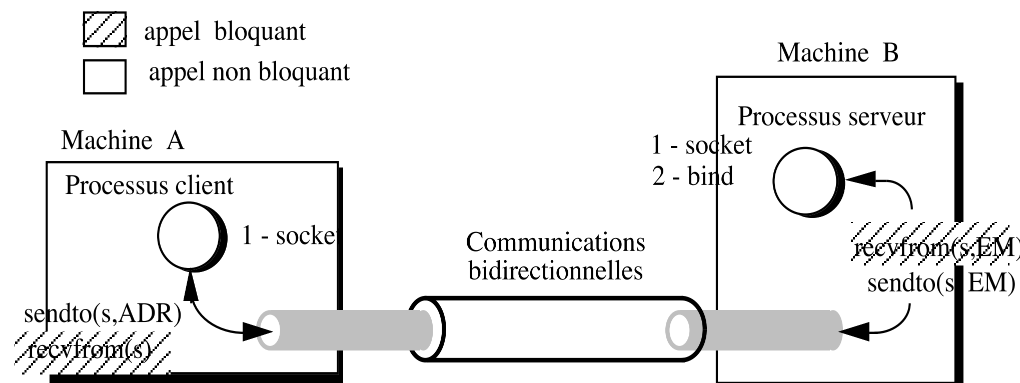
Emissions/Réceptions

1. Fonctions standards destinées aux fichiers :
   - read/write(Socket, Message, MessageLen)

2. Fonctions spécifiques (gestion plus fine) :
   - send/recv(Sock, Message, MessageLen, Option)
     - Exemple d'options : MSG_OOB,
     - Une façon élégante de récupérer un caractère envoyé "OOB", est de traiter le signal SIG_URG envoyé par le noyau lors de la réception de ce caractère

Communications en mode datagramme (UDP)

- Client (appelant) :
  - Crée une socket ;
  - Lit ou écrit sur la socket ;
- Serveur (appelé) :
  - Crée une socket ;
  - Lie la socket à un port (bind)
  - Lit ou écrit sur la socket.



Communications en mode datagramme (UDP)

- Emission
  - sendto(Socket, Message, MessageLen, 0, &ReceiverAddr, ReceiverAddrLen)
  - ReceiverAddr indique l’adresse du destinataire

- Reception
  - recvfrom(Socket, Message, MessageLen, Flags, &SenderAddr, &SenderAddrLen)
  - SenderAddr indique l’adresse de l’émetteur

Canevas avec API C de client UDP

```
char Message[] =    »Hello World»;

Socket = socket(AF_INET, SOCK_DGRAM, 0);

ReceiverAddr.sin_addr = gethostbyname(ReceiverName);
ReceiverAddr.sin_port = htons(ReceiverPort);

sendto (Socket, Message, strlen(Message), 0,
        &ReceiverAddr, ReceiverAddrLen)
```

```
#define ReceiverAddrPort 7777
struct sockaddr ReceiverAddr, SenderAddr;
int SenderAddrLen;
char Message[256]
Socket = socket(AF_INET, SOCK_DGRAM,0);
ReceiverAddr.sin_port = htons(ReceiverAddrPort);
bind(Socket, &ReceiverAddr, ReceiverAddrLen);
recvfrom(Socket, Message, sizeof(Message), …,
              &SenderAddr, &SenderAddrLen);
```

Multiplexage: select()

- Pour attendre sur plusieurs ports simultanément, on utilise la fonction select() qui permet de manipuler des ensembles de descripteurs de fichier (masques)
- SetLen= select(SetLenMax, &R_Set, &W_Set, &E_Set, Timeout)
  - MaskLen nombre de fichiers ayant répondu
  - MaskLenMax dernier fichier sur lequel on attend
  - R_Set fichiers sur lesquels on attend une écriture
  - W_Set fichiers sur lesquels on attend une lecture
  - E_Set fichiers sur lesquels on attend une erreur
  - Timeout temps d’attente avant réponse

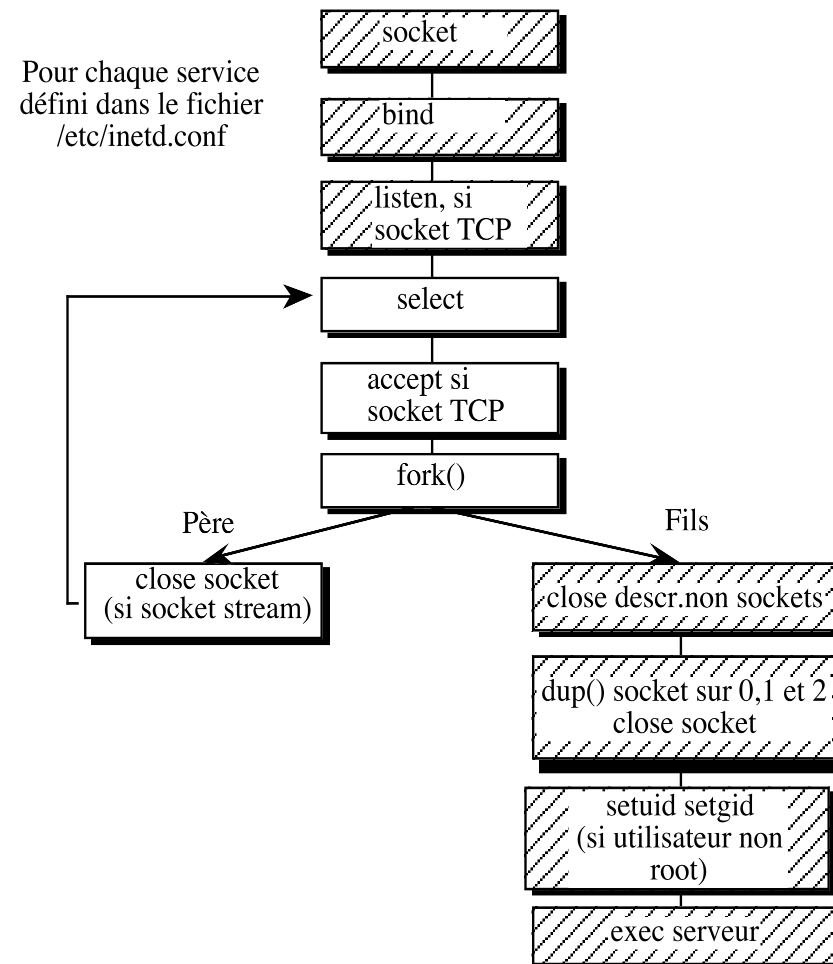
Multiplexage: select()

- select() : fonction Unix qui permet de multiplexer des fichiers en mode producteur/consommateur
- Les ensembles en paramètre (ex en écriture) sont mis à jour pour indiquer les fichiers sur lesquels ont survenu des événements (en écriture)
- Pour positionner les ensembles, utiliser :
  - FD_ZERO(&fdset) met l'ensemble à vide (zero)
  - FD_SET(fd, &fdset) met le fichier fd à 1 dans l'ensemble
  - FD_CLR(fd, &fdset) met le fichier fd à 0 dans l'ensemble
  - FD_ISSET(fd, &fdset) renvoie le fichier fd dans l'ensemble

Multiplexage : le serveur `inetd`

- `inetd` trouve la liste les serveurs qu'il doit lancer dans le fichier de configuration :

  `/etc/inetd.conf`



API C : divers utilitaires

- Informations sur les machines
  - gethostbyaddr(struct sockaddr *HostAddr, int HostAdddrLen, int Type);
  - gethostbyname(char *HostName),
  - gethostent()
- Informations sur les services
  - getservbyport(int Port, char *Proto)
  - getservbyname(char *Nom, char *Proto)
- Informations sur les ports
  - getsockname(Socket, &sa, &len)
  - getpeername(Socket, &sa, &len)
- Fin de communication et fermeture de socket
  - shutdown(Socket, Direction)
  - close(Socket)

API C : normalisation des données

- Les processeurs ne représentent pas les données de la même manière (endianess)
- Pas de protocole de normalisation des données
- Il existe des fonctions convertissant des informations sur 16 ou 32 du format réseau au format local et vice-versa :
  - htonl(net_long host_long)
  - htons(net_short host_short)
  - ntohl(host_long net_long)
  - ntohs(host_short net_short)

API C : traitements spéciaux

- setsockopt(…)
  - Exemple d'options : taille des buffers, réutilisation d'un port
- readv(), writev()
  - Pour et de recevoir des données éparpillées sans avoir à les rassembler dans un buffer unique
  - Mode « scatter/gather »

Bilan API C sockets (1)

- Faiblesses :
  - Service d'annuaire qui associe nom de machine et adresse IP (gethostbyname), pas d'annuaire au niveau service (ie pour les numéros de ports, sauf ceux des services bien connus dans /etc/services)
  - Représentation des données
    - Pas de normalisation sur les données applicatives
    - Conservions des informations de niveau réseau (htons, htonl, etc), utilisables pour les données applicatives.

Bilan API C sockets (2)

- API très puissante:
  - Multicast, mode RAW (setsockopt), etc

- … mais syntaxe lourde :
  - Exemple :

```
sockaddr_ecou.sin_family        = AF_INET;
sockaddr_ecou.sin_addr.s_addr = INADDR_ANY;
sockaddr_ecou.sin_port          = htons (PORT_SERV);
bind(sock_ecou, (struct sockaddr *)&sockaddr_ecou,
     sizeof(sockaddr_ecou));
```

Canevas avec API Java d'un serveur TCP

- L' API Java reprend les concepts de l'API C standard
- Squelette Java pour le serveur
  (objets de type ServerSocket et Socket) :

```
// Create socket and bind using constructor
ServerSocket  serverSocket=  new ServerSocket(Port);

// Block until accept a client
Socket socket = serverSocket.accept();
```

Canevas avec API Java de serveur multi-tâches TCP (1/3)

- Serveur qui multiplexe les communications en utilisant des threads :
  - ServerSocket serverSocket = new ServerSocket(Port);
  - Socket socket = serverSocket.accept();
  - new Thread_Dialogue(socket).start();

Canevas avec API Java de serveur multi-tâches TCP (2/3)

- Thread qui communique avec un client :

```
class Thread_Dialogue extends Thread {
    Socket s;
    public Thread_Dialogue(Socket socket){s = socket;}
    public void run(){
        BufferedReader in = new BufferedReader(
            new InputStreamReader(s.getInputStream()))
        PrintWriter out = new PrintWriter(new BufferedWriter(
            new OutputStreamWriter(s.getOutputStream(), true);
        …
    }
}
```

Canevas avec API Java de serveur multi-tâches TCP (3/3)

- Thread qui communique avec un client :

```
class Thread_Dialogue extends Thread {
    public void run(){
        PrintWriter out = ...
        BufferedReader in = ...
        while (true) {
            String str = in.readLine(); // lecture du message
            System.out.println(str);    // écriture de l'écho
            out.println(str);           // trace sur le serveur
        }
    }
}
```

Canevas avec API Java d’un client TCP

- Client envoie au serveur des chaînes de caractères puis attend une réponse qu'il affiche à l’écran

```
Socket s = new Socket("machine", port);
BufferedReader in = new BufferedReader(
    new InputStreamReader(s.getInputStream()));
PrintWriter out = new PrintWriter(new BufferedWriter(
   new OutputStreamWriter(s.getOutputStream())), true);
String str =    »Hello World!»;
for (int i = 0; i < 10; i++) {
    out.println(str);           // envoi d'un message
    System.out.println (str); // trace sur le client
    str = in.readLine();       // lecture de l'écho
}
```


A. Diaconescu, L. Pautet & B. Dupouy

```
DatagramSocket  socket;
DatagramPacket  message = null;
byte[]          buffer;
InetAddress     address;
int             port = 0;
// Initialisations
socket          = new DatagramSocket(PORT_SERV_UDP);
buffer          = new byte[256];
message         = new DatagramPacket(buffer, buffer.length);
// Attendre un message
socket.receive(message);
// Envoyer la reponse vers le client
address         = message.getAddress();
port            = message.getPort();
Message = new DatagramPacket(buffer, buffer.length, address, port);
socket.send(message);
```

```
DatagramSocket  socket;
DatagramPacket  message;
byte[]          buffer;
InetAddress     address;

// Initialisations
buffer          = new byte[256];
socket          = new DatagramSocket();
address         = InetAddress.getByName(…);
// Emission
packet          = new DatagramPacket(buffer, buffer.length, address, port);
socket.send(packet);
// Reception
Packet           = new DatagramPacket(buffer, buffer.length);
socket.receive(packet);
String message = new String(packet.getData());
```