IUT DE PARIS

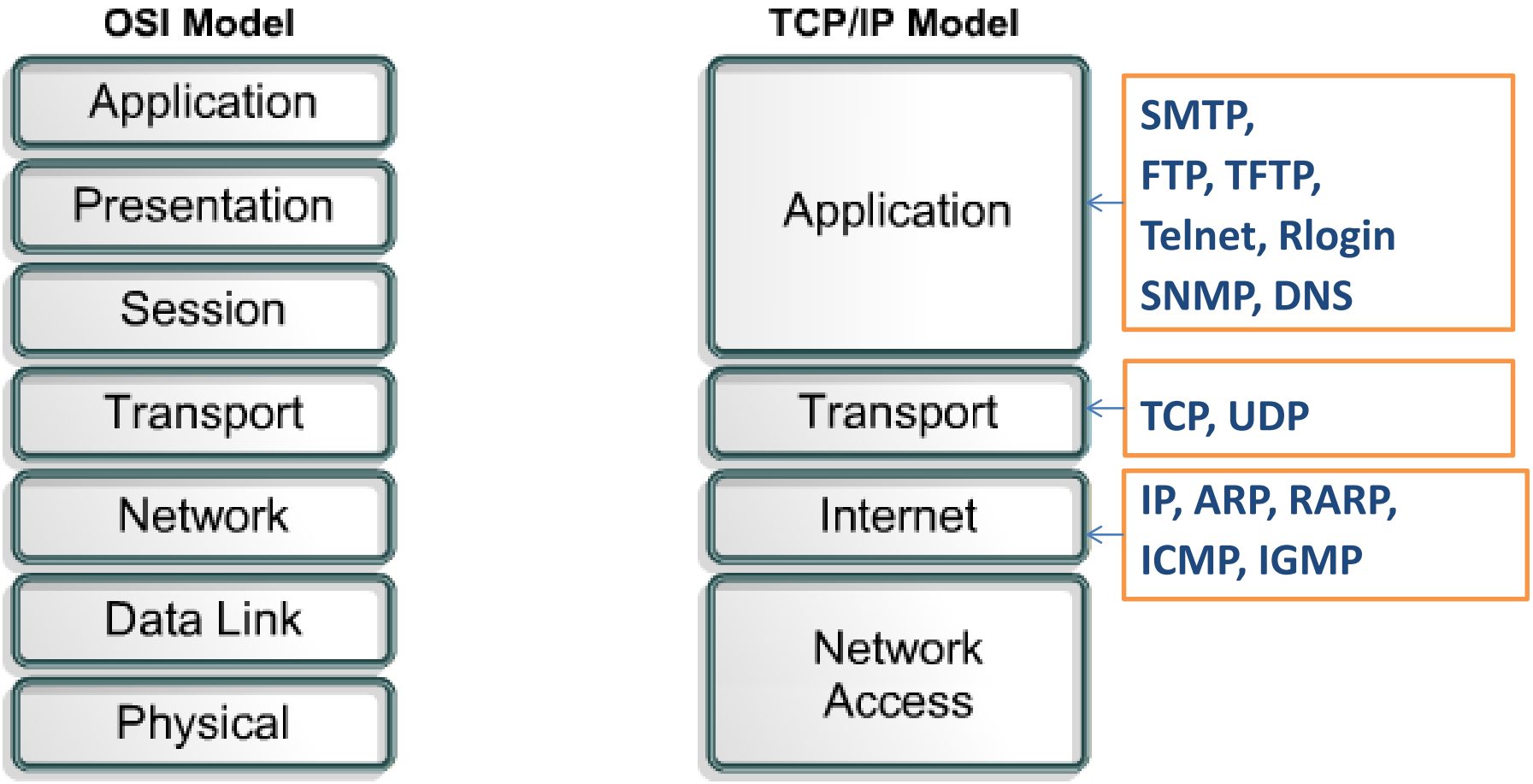
UNIVERSITÉ PARIS DESCARTES

Programmation réseau
+
Utilisation des sockets

Professeur

Jocelyne Elias

Couches de communication



ifconfig

- ***ifconfig*** : affiche l'**adresse IP** et l'**adresse MAC** de la machine

- Exemple: /sbin/ifconfig

```
lo        Link encap:Boucle locale
          inet adr:127.0.0.1  Masque:255.0.0.0
          adr inet6: ::1/128 Scope:HÃ´te
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:4 errors:0 dropped:0 overruns:0 frame:0
          TX packets:4 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 lg file transmission:0
          RX bytes:240 (240.0 b)  TX bytes:240 (240.0 b)

p4p1      Link encap:Ethernet  HWaddr FF:4D:A2:22:87:7C
          inet adr:193.48.200.120  Bcast:193.48.200.255  Masque:255.255.255.0
          adr inet6: fe80::f24d:a2ff:fe22:877c/64 Scope:Lien
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:6346973 errors:0 dropped:0 overruns:0 frame:0
          TX packets:133071 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 lg file transmission:1000
          RX bytes:1078533019 (1.0 GiB)  TX bytes:24453217 (23.3 MiB)
          Interruption:16
```

hostname

- ***hostname*** : affiche ou définit le nom d'hôte du système

Exemples :

- *hostname : affiche le nom de la machine (version longue)*
- *hostname -s : affiche le nom de la machine (version short)*
- *hostname -d* : affiche le nom de domaine DNS
- *hostname -i* : affiche l'adresse IP

Noms des protocoles

- cat /etc/protocols : les protocoles sont définis dans un fichier système « **/etc/protocols** ».

```
ip          0       IP              # internet protocol, pseudo protocol number
hopopt      0       HOPOPT          # hop-by-hop options for ipv6
icmp        1       ICMP            # internet control message protocol
igmp        2       IGMP            # internet group management protocol
ggp         3       GGP             # gateway-gateway protocol
ipv4        4       IPv4            # IPv4 encapsulation
st          5       ST              # ST datagram mode
tcp         6       TCP             # transmission control protocol
[…]
udp         17      UDP             # user datagram protocol
[…]
ipv6        41      IPv6            # IPv6 encapsulation
[…]
#           255             Reserved                        [IANA]
```

Structure **protoent**

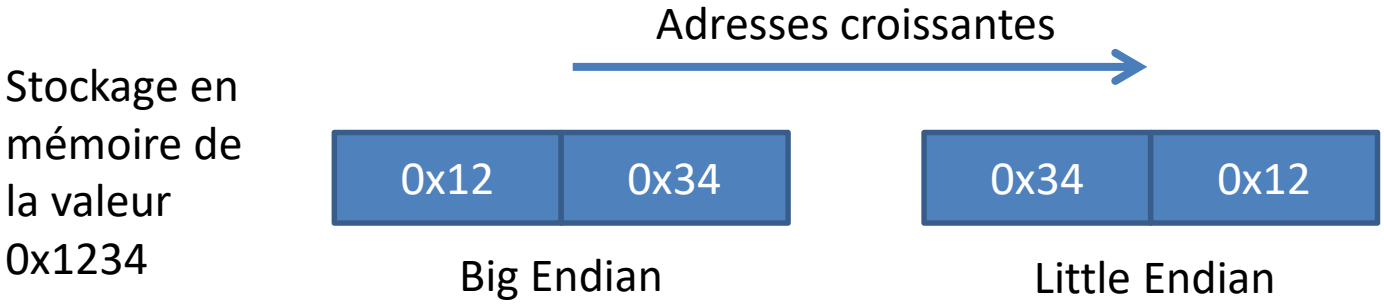
- La structure **protoent** contient les membres suivants :

Nom	Type	Signification
p_name	char *	Nom officiel du protocole (défini par la RFC 1700). RFC : Request for Comments
p_proto	int	Numéro officiel du protocole (dans l'ordre des octets de la machine)
p_aliases	char **	Table de chaînes de caractères correspondants à d'éventuels alias. Cette table est terminée par un pointeur NULL.

- struct protoent * **getprotobyname** (const char * nom);
- struct protoent * **getprotobynumber** (int numero);

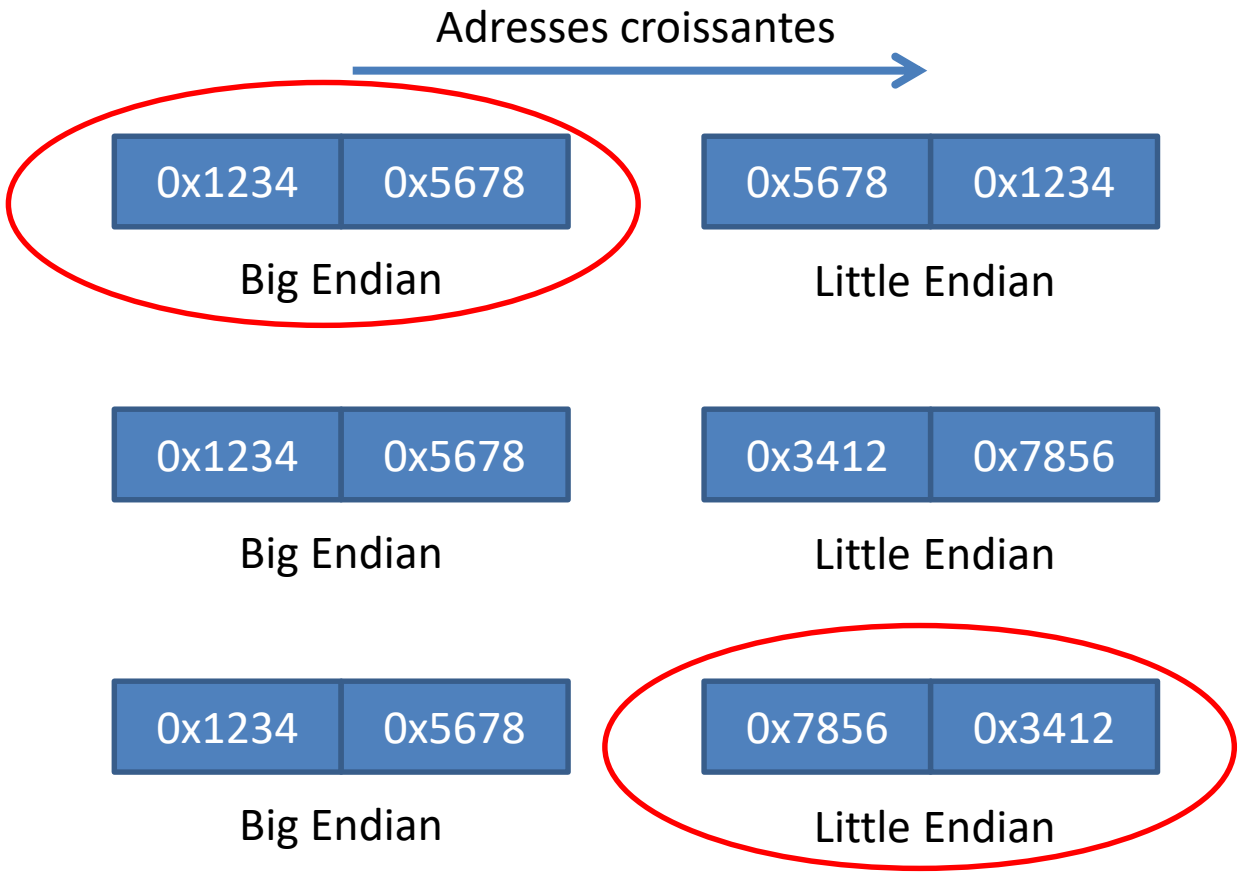
Ordre des octets

- Les communications et les échanges de données entre ordinateurs hétérogènes sont confrontés au problème d'*ordre des octets* dans les valeurs entières.
- Pour stocker en mémoire une valeur tenant sur 2 octets, certains processeurs placent en première position l'octet de poids faible, puis celui de poids fort. Cette organisation est nommée ***Little Endian***.
- A l'opposé, il existe des machines rangeant d'abord l'octet de poids fort, suivi de celui de poids faible. C'est le ***Big Endian***.



Ordre des octets

- Un autre exemple avec des données sur 32 bits; la valeur 0x12345678 peut être stockée sous 4 formes :



Ordre des octets

- Les protocoles basés sur IP emploient la solution suivante :
  - Figer l'ordre des octets dans tous les en-têtes des paquets de données circulant sur le réseau.
  - La forme retenue est ***Big Endian***.

htonl, htons, ntohl, ntohs

- unsigned long int ***htonl*** (unsigned long int valeur);
- unsigned short int ***htons*** (unsigned short int valeur);
- unsigned long int ***ntohl*** (unsigned long int valeur);
- unsigned short int ***ntohs*** (unsigned short int valeur);

- Les fonctions ***htonl()*** et ***htons()*** convertissent respectivement depuis l'ordre des octets de l'hôte (h) vers (*to*) celui du réseau (*network n*).
- Les fonctions ***ntohl()*** et ***ntohs()*** convertissent respectivement les entiers depuis l'ordre des octets du réseau vers celui de l'hôte (h).
- LONG ⇒ adresses IP sur 32 bits en version 4
- SHORT ⇒ numéros de ports

Services et numéros de ports

- Pour connaître l'association entre un numéro de port et un service particulier, on peut consulter le fichier /etc/services : **cat /etc/services**

```
tcpmux          1/tcp                   # TCP port service multiplexer
tcpmux          1/udp                   # TCP port service multiplexer
rje             5/tcp                   # Remote Job Entry
rje                     5/udp                   # Remote Job Entry
echo            7/tcp
echo            7/udp
[…]
ftp             21/tcp
ftp             21/udp          fsp fspd
ssh             22/tcp                  # The Secure Shell (SSH) Protocol
ssh             22/udp                  # The Secure Shell (SSH) Protocol
telnet          23/tcp
telnet          23/udp
[…]
http            80/tcp          www www-http    # WorldWideWeb HTTP
http            80/udp          www www-http    # HyperText Transfer Protocol
```

Structure **servent**

- La structure **servent** contient les membres suivants :

Nom	Type	Signification
s_name	char *	Nom officiel du service (défini par la RFC 1700).
s_port	short int	Numéro du service dans l'ordre des octets du réseau.
s_proto	char *	Nom du protocole associé.
s_aliases	char **	Liste d'éventuels alias, terminée par un pointeur NULL.

- struct servent * **getservbyname** (const char* nom, const char * protocole);
- struct servent * **getservbyport** (short int numero, const char * protocole);

Manipulation des adresses IP

Structure **in_addr**

- Le protocole IP version 4 offre le routage des paquets à destination d'hôtes dont le nom est indiqué par une adresse sur 32 bits.
- La définition de l'adresse est donc assurée par la structure **in_addr** (définie dans <netinet/in.h>).
- La structure **in_addr** contient un seul membre :

Nom	Type	Signification
s_addr	unsigned int long	Adresse IP dans l'ordre des octets du réseau.

- La structure in6_addr est utilisée pour les adresses IPv6. Cette structure est plus compliquée (Nous la considérerons comme un type opaque).

inet_ntoa, inet_aton, inet_addr

- Fonctions déclarées dans <arpa/inet.h>

- char * ***inet_ntoa*** (struct in_addr adresse);
- int ***inet_aton*** (const char * chaine, struct in_addr * adresse);
- unsigned long int ***inet_addr*** (const char * chaine);

IPv4

- La fonction ***inet_ntoa()*** renvoie une chaine de caractères statique représentant en notation pointée l'adresse transmise en argument.
- ***inet_aton()*** remplie la structure in_addr (second argument) en ayant converti la chaine pointée transmise (premier argument).
- ***inet_addr()*** prend en argument la chaine en notation pointée et renvoie directement l'adresse sous forme d'un entier long non signé.

inet_ntoa, inet_aton, inet_addr

- ***inet_aton()*** n'est disponible que sur peu de systèmes. Si possible on peut utiliser ***inet_addr()***.
- ***inet_addr()*** renvoie une valeur particulière (INADDR_NONE) en cas de problème.
  - La constante INADDR_NONE peut représenter une adresse valide : 255.255.255.255 !
  - INADDR_NONE a la même valeur que INADDR_BROADCAST !

inet_ntop, inet_pton

- Fonctions prêtes à être utilisées avec l'adressage IP version 6.
- Offrent les mêmes services que les fonctions ***inet_ntoa()*** et ***inet_aton()***
- p ⟹ présentation

Noms d'hôtes et noms de réseaux

Noms d'hôtes

- Les adresses IP des machines ne sont pas faciles à mémoriser, surtout avec IPv6 (128 bits).
- On associe donc des noms à ces machines.
- On a besoin d'un serveur de noms ou serveur DNS (*Domaine Name Server*) !
- Rechercher un hôte à partir de son nom (***gethostbyname()***) ou de son adresse (***gethostbyaddr()***) .
  - struct hostent * ***gethostbyname*** (const char * nom);
  - struct hostent * ***gethostbyaddr*** (const char * adresse, int longueur, int format); format : AF_INET/AF_INET6
- Les informations concernant un hôte sont regroupées dans la structure **hostent** (diapositif suivant).

Structure **hostent**

- La structure **hostent** contient les membres suivants :

Nom	Type	Signification
h_name	char *	Nom officiel d'hôte.
h_aliases	char **	Liste d'alias, terminée par un pointeur NULL.
h_addrtype	int	Type d'adresse, AF_INET ou AF_INET6.
h_length	int	Longueur (en octets) des adresses du type indiqué ci-dessus.
h_addr_list	char **	Liste d'adresses correspondant à cet hôte. Les adresses sont données dans l'ordre des octets du réseau.
h_addr	char *	Équivalent de h_addr_list[0].

Noms de réseaux

- Il existe une base de données de sous-réseaux.
- Moins connue de la base de données d'hôtes.
- En général employée par l'administration du système et constituée par le fichier /etc/networks.
- Structure **netent**.
- Fonctions d'accès sont ***getnetbyname()*** et ***getnetbyaddr()***.
  - struct netent * ***getnetbyname*** (const char * nom);
  - struct netent * ***getnetbyaddr*** (unsigned long int adresse, int type);
- Fonctions pour balayer la base des sous-réseaux : ***setnetent()***, ***getnetent()*** et ***endnetent()***.
  - …

Structure **netent**

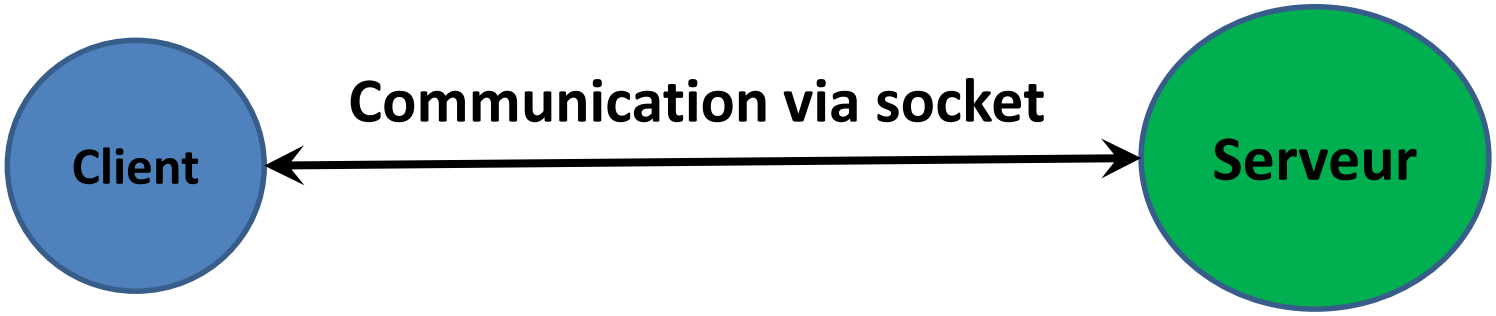
- La structure **netent** contient les membres suivants :

Nom	Type	Signification
n_name	char *	Nom du sous-réseau.
n_aliases	char **	Liste d'alias, terminée par un pointeur NULL.
n_addrtype	int	Type d'adresses sur le sous-réseau (uniquement AF_INET pour le moment).
n_net	unsigned long int	Adresse du sous-réseau, dans l'ordre des octets de l'hôte.

Utilisation des sockets

Concept de socket

- Les sockets permettent de faire dialoguer des processus s'exécutant sur différentes machines.
- Les sockets sont représentées dans un programme par des entiers comme les descripteurs de fichiers.
- Appels système : ***read()***, ***write()***, ***select()***, ***close()*** …



Adresse du socket :
@ IP + num. de port

@ IP + num. de port
--- Serveur Web ➔
Num. de port = 80

Création d'une socket

- Appel système **socket()** défini dans <sys/socket.h>
- #include <sys/types.h> et #include <sys/socket.h>
- *int **socket** (int domaine, int type, int protocole);*
- **domaine** : domaine de communication
  - AF_INET : protocole basé sur IP ver. 4
  - AF_INET6 : protocole IPv6
  - AF_UNIX : communication entre processus sur la même machine
- **type** : type du socket
  - SOCK_STREAM : mode connecté + contrôle de flux
  - SOCK_DGRAM : mode non connecté
  - SOCK_RAW : dialoguer d'une manière brute avec le protocole
- **protocole** : protocole désiré. Champ p_proto de la structure *protoent*

Création d'une socket

- *int **socket** (int domaine, int type, int protocole);*

- Si **protocole** est NULL, les combinaisons suivantes sont automatiquement réalisées :
  - AF_INET SOCK_STREAM IPPROTO_TCP
  - AF_INET SOCK_DGRAM IPPROTO_UDP
  - ---------- SOCK_RAW IPROTO_RAW (1)
  - ---------- SOCK_RAW IPROTO_ICMP (2)

  (1) Communication directe avec la couche IP

  (2) Communication utilisant le protocole ICMP (Internet Control Message Protocol), qui est utilisé dans l'utilitaire /bin/ping !

- Pour les communications fondées sur le protocole IP, l'identité d'une socket contient l'***adresse IP*** de la machine et le ***numéro de port*** employé.

Création d'une socket

- Pour stocker l'adresse complète d'une socket, on emploie la structure **sockaddr**, définie dans <sys/socket.h>
- **sockaddr** contient les membres suivants :
  - sa_family (unsigned short int) : famille de communication
  - sa_data (char [ ]) : Données propres au protocole.
- Type donné (homogène) pour toutes les communications réseau.
- AF_INET ⟹ structure sockaddr_in définie dans <netinet/in.h>

Affectation d'adresse

- Appel système **bind()** permet d'affecter une identité à notre socket
- *int **bind** (int sock, struct sockaddr * adr, socklen_t longueur);*
- La socket représentée par le descripteur passé en premier argument (**sock**) est associée à l'adresse passée en 2ème position (***adr**)
- **longueur** est la longueur de l'adresse.
- Le type *socklen_t* n'est pas disponible sur tous les Unix. Dans ce cas le troisième argument de *bind()* est un *int*.
- int ***getsockname*** (int sock, struct sockaddr * adr, socklen_t * longueur);
  - Renvoie l'adresse d'une socket
  - Utile si elle a été identifiée automatiquement par le noyau et on désire connaître ses caractéristiques.

Mode connecté et non connecté

- **Mode non connecté** (SOCK_DGRAM, AF_INET) : le processus identifie sa socket avec **bind(…)** et cette socket est accessible de l'extérieur du serveur. Le processus côté serveur attend de recevoir des requêtes de la part d'autres programmes. Du côté client, l'identification est établie automatiquement par le noyau.
  - Est nécessaire d'indiquer le destinataire du message à chaque envoi (comme le ***service postal***) !

- **Mode connecté** : il reste du travail à faire, tant du côté serveur que du côté client.
  - 2 interlocuteurs face à face. Pas d'ambiguité lors de l'émission/réception d'un message (comme un **appel téléphonique**).

Mode connecté et non connecté

- **Mode non connecté** (SOCK_DGRAM, AF_INET) :
  - ***sendto***(…) : envoyer un paquet à destination d'un processus passé en argument
  - ***recvfrom***(…) : recevoir un paquet en récupérant l'adresse de l'émetteur
- **Mode connecté** :
  - ***send***(…)
  - ***recv***(…)
  - ***read***(…)
  - ***write***(…)

  } comme avec un tube classique
  - …

Attente des connexions

- Appel système **listen()** défini dans <sys/socket.h>
- #include <sys/types.h> et #include <sys/socket.h>
- *int **listen** (int sock, int nb_en_attente);*
  - nb_en_attente : permet de dimensionner une file d'attente des requêtes de connexions.
    - Si la file est pleine, les nouvelles connexions sont rejetées !
    - En général, on utilise la **constante 5** (sockets BSD), mais Linux accepte jusqu'à 128 connexions en attente.
  - ***listen()*** n'est pas bloquant
  - Mettre effectivement le processus attente : appel système ***accept()***

Attente des connexions

- Appel système **accept()** défini dans <sys/socket.h>
- #include <sys/types.h> et #include <sys/socket.h>
- *int **accept** (int sock, struct sockaddr * adr, socklen_t longueur);*
  1. Prendre une demande de connexion en attente (file d'attente de *listen(…)*)
  2. Ouvrir une nouvelle socket du côté serveur
  3. Etablir la connexion sur cette nouvelle socket
- adr et longueur fournissent l'identité du client
- Le serveur a besoin de dialoguer avec plusieurs clients simultanément ?
  1. accept(…)
  2. Un fork() pour chaque requête client ⟹ le processus fils traite la communication

Demande d'une connexion

- Appel système **connect()** défini dans <sys/socket.h>
- #include <sys/types.h> et #include <sys/socket.h>
- *int **connect** (int sock_client, struct sockaddr * servadr, socklen_t longueur);*
  - Cette fonction contacte le serveur dont l'adresse est passée en argument (**servadr**) et en établissant la connexion sur la socket (côté client) **sock_client.**

Réception et envoi de données

- Appels système ***recvfrom()*** et ***sendto()***
- *int **recvfrom** (int sock, char * buffer, int taille_buffer, int attributs, struct sockaddr * source, socklen_t * taille);*
- *int **sendto** (int sock, char * buffer, int taille_buffer, int attributs, struct sockaddr * destination, socklen_t taille);*
  - La seule différence entre les 2 prototypes est le dernier argument : pointeur (*recvfrom*) et valeur (*sendto*)
  - sock : socket employée
  - buffer : transmettre et recevoir les données
  - source/destination : pointeur sur l'adresse de l'émetteur/destinataire du message
  - taille : longueur de l'adresse
  - attributs : diapositif n° 19

Réception et envoi de données

- Appels système ***recv()*** et ***send()***
- *int **recv** (int sock, char * buffer, int taille_buffer, int attributs);*
- *int **send** (int sock, char * buffer, int taille_buffer, int attributs);*
  - L'appel système *send* ne peut pas être utilisé qu'après avoir invoqué *connect()* sur la socket (surtout utilisé en mode connecté)
  - sock : socket employée
  - buffer : transmettre et recevoir les données
  - attributs : diapositif n° 19

Réception et envoi de données

- attributs :

Nom	Signification
MSG_DONTROUTE	Cette option permet d'ignorer le routage. Valable uniquement pour les sockets avec la famille d'adresses AF_INET et ignorée pour les autres familles d'adresses. Utilisée avec *sendto()* ou *send()*.
MSG_OOB	Le message doit être considéré comme des données TCP hors bande. Traiter le message avec une sorte de priorité supérieure à d'autres du côté récepteur.
MSG_PEEK	Lire/obtenir une copie des données désirées sur une socket TCP, sans les extraire de la file d'attente. Utilisée avec *recvfrom()* ou *recv()*.

Fermeture d'un socket

- Appel système **close()**
- *int **close** (int sock);*
  - Cette primitive est automatiquement invoquée lorsque le processus se termine.
  - La socket est refermée et devient inutilisable !

Exemples d'utilisation de *socket(), bind(), listen(), accept(), connect(), …*

Exemple 1

Exemple 1

int cree_socket_stream (const char * nom_hote, const char * nom_service, const char * nom_proto)

```
{
 int sock;
 struct sockaddr_in  adresse;
 struct hostent *  hostent;
 struct servent *  servent;
 struct protoent *  protoent;

 if ( ( hostent = gethostbyname(nom_hote) ) == NULL) {
  perror (‘‘gethostbyname’’);
  return(-1);
 }
 if ( ( protoent = getprotobyname(nom_proto) ) == NULL) {
  perror (‘‘getprotobyname’’);
  return(-1);
 }
 if ( ( servent = getservbyname(nom_service, protoent->p_name) ) == NULL) {
  perror (‘‘getservbyname’’);
  return(-1);
 }
```

Exemple 1

```
if( (sock = socket( AF_INET, SOCK_STREAM, 0) ) < 0) {
perror(‘‘ socket’’);
return(-1);
}
memset ( & adresse, 0, sizeof( struct sockaddr_in) );
adresse.sin_family = AF_INET;
adresse.sin_port = servent->s_port;
(adresse.sin_addr).s_addr  = ( ( struct in_addr *) (hostent -> h_addr)) -> s_addr;

if ( bind( sock, (struct sockaddr *) & adresse, sizeof( struct sockaddr_in) ) < 0 )  {
  close(sock);
  perror( ‘‘bind’’);
  return(-1);
 }
 return(sock);
}
```

Exemple 1

```
int affiche_adresse_socket (int sock)
{
 struct sockaddr_in adresse;
 socklen_t longueur;

 longueur = sizeof( struct sockaddr_in);

 if ( getsockname(sock, & adresse, & longueur ) < 0 ) {
  perror (‘‘getsockname’’);
  return(-1);
 }
 fprintf( stdout, ‘‘IP = %s, Port = %u \n ’’, inet_ntoa(adresse.sin_addr),
ntohs(adresse.sin_port) );
 return(0);
}
```

Exemple 1

```
int serveur_tcp (void) {
 int sock_contact;
 int sock_connectee;
 struct sockaddr_in adresse;
 socklen_t longueur;

 sock_contact = cree_socket_stream (NULL, NULL, '' tcp '');
 if (sock_contact < 0 )
  return(-1);

 listen(sock_contact, 5);
 fprintf( stdout, ''Mon adresse >> '');
 affiche_adresse_socket (sock_contact);
 while ( ! quitter_le_serveur() ) {
  longueur = sizeof( struct sockaddr_in);
  sock_connectee = accept( sock_contact, (struct sockaddr *) & adresse, & longueur);
  if ( sock_connectee < 0 ){
   perror(''accept'') ;
   return(-1);
  }
```

⟸ Cette routine renvoie toujours 0 !!

Exemple 1

```
 switch( fork() ) {
  case 0 : //fils
    close(sock_contact);
    traite_connexion (sock_connectee);
    exit(EXIT_SUCCESS);
  case -1 :
    perror(‘‘ fork’’);
    return(-1);
  default : //pere
      close(sock_connectee);
  }
 }
 return(0);
}
```

Exemple 1

```
void traite_connexion (int sock)
{
 struct sockaddr_in adresse;
 socklen_t longueur;
 char buffer[256];

 longueur = sizeof( struct sockaddr_in);
 if ( getpeername(sock,  (struct sockaddr *) & adresse, & longueur ) < 0 ) {
  perror (‘‘getpeername’’);
  return;
  }
  sprintf(buffer, ‘‘IP = %s, Port = %u \n ’’, inet_ntoa(adresse.sin_addr), ntohs(adresse.sin_port)
);
  fprintf(stdout, ‘‘ Connexion  :  locale ’’);
  affiche_adresse_socket(sock);
  fprintf(stdout, ‘‘     distante  %s ’’, buffer);

  write (sock, ‘‘Votre adresse : ’’, 16);
  write( sock, buffer, strlen(buffer) );
  close(sock);
}
```

Exemple 1

exemple-serveur-tcp.c

```
# include <stdio.h>
# include <stdlib.h>
# include <string.h>
# include <unistd.h>
# include <arpa/inet.h>
# include <netdb.h>
# include <netinet/in.h>
# include <sys/types.h>
# include <sys/socket.h>

int cree_socket_stream(const char * nom_hote, const char * nom_service, const char * nom_proto);
int affiche_adresse_socket (int sock);
int serveur_tcp(void);
int quitter_le_serveur(void);
void traite_connexion (int sock);
[…] //Présentés dans les diapositifs précédents !!
int main (int nbarg, char * tbarg[])
{
 return( serveur_tcp() );
 }
```

Exemple 1 - résultats

Côté serveur :

cree_socket_stream
listen
Mon adresse >> affiche_adresse_socket
IP = 127.0.0.1, Port = 23
quitter_le_serveur
accept
Connexion : locale IP = 127.0.0.1, Port = 23
distante IP = 127.0.0.1, Port = 53356
quitter_le_serveur
accept
Connexion : locale IP = 127.0.0.1, Port = 23
distante IP = 127.0.0.1, Port = 53358
quitter_le_serveur
accept
Connexion : locale IP = 127.0.0.1, Port = 23
distante IP = 127.0.0.1, Port = 53359
quitter_le_serveur
accept
Connexion : locale IP = 127.0.0.1, Port = 23
distante IP = 127.0.0.1, Port = 53360

Côté client :

telnet localhost 23
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Votre adresse : IP = 127.0.0.1, Port = 53356
Connection closed by foreign host.

telnet localhost 23
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Votre adresse : IP = 127.0.0.1, Port = 53358
Connection closed by foreign host.

…

Exemple 2

Exemple 2

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<unistd.h>
#include<arpa/inet.h>
#include<netdb.h>
#include<netinet/in.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<errno.h>
#define    LG_BUFFER    1024

int lire_arguments (int argc, char * argv[], struct sockaddr_in * adresse, char * protocole) {
 char * liste_options = "a:p:h";
 char * hote = "localhost"; // par défaut
 char * port = "2000";
 struct hostent * hostent;
 struct servent * servent;
 int numero;
 int option;

while( ( option = getopt(argc, argv, liste_options) ) != -1 )
 {
        switch(option) {
        case 'a' :
                hote = optarg;
                break;
        case 'p' :
                port = optarg;
                break;
        case 'h' :
                printf("Syntaxe : %s [-a adresse] [-p port] \n", argv[0]);
                return -1;
        default :
                break;
   }
 }
 memset ( adresse, 0, sizeof( struct sockaddr_in) );
 if (inet_aton(hote, & (adresse->sin_addr) ) == 0 ) {
        if ( ( hostent = gethostbyname(hote) ) == NULL) {
                fprintf(stderr, "hote %s inconnu \n", hote);
                return(-1);
        }
        (adresse->sin_addr).s_addr = ( ( struct in_addr *)(hostent->h_addr))->s_addr;
 }
```

Exemple 2

```
 if(sscanf(port, "%d", & numero) == 1) {
                adresse -> sin_port = htons(numero);
                return(0);
 }
 if ( ( servent = getservbyname(port, protocole) ) == NULL)
{
  fprintf(stderr, "Service %s inconnu \n", port);
  return(-1);
 }
 adresse -> sin_port = servent->s_port;
 return 0;
 }


int main (int argc, char *argv[])
{
  struct sockaddr_in adresse;
  char buffer[LG_BUFFER];
  int sock;
  int nb_lus;
       if ( lire_arguments (argc, argv, & adresse, "tcp")   < 0 )
              exit(EXIT_FAILURE);
       adresse.sin_family = AF_INET;
```

```
     if ( (sock = socket(AF_INET, SOCK_STREAM, 0) ) < 0 ) {
            perror("socket");
            exit(EXIT_FAILURE);
     }
     if ( connect(sock, (struct sockaddr *) & adresse,
sizeof( struct sockaddr_in) ) < 0 ) {
            perror("connect");
            exit(EXIT_FAILURE);

     }
     setvbuf(stdout, NULL, _IONBF, 0);
     while (1) {
            if ( (nb_lus = read(sock, buffer, LG_BUFFER) ) == 0 )
                  break;
            if (nb_lus < 0 ) {
                  perror("read");
                  break;
            }
            write(STDOUT_FILENO, buffer, nb_lus);
     }
     return( EXIT_SUCCESS);
}
```

Exemple 2 - résultats

Côté serveur :

Mon adresse >> affiche_adresse_socket

IP = 127.0.0.1, Port = 23

quitter_le_serveur

accept

Connexion : locale IP = 127.0.0.1, Port = 23

distante IP = 127.0.0.1, Port = 53426

….

quitter_le_serveur

accept

Connexion : locale IP = 127.0.0.1, Port = 23

distante IP = 127.0.0.1, Port = 53427

…

Côté client :

./test_client -p 23

debut du main

Votre adresse : IP = 127.0.0.1, Port = 53426

…

./test_client -p 23

debut du main

Votre adresse : IP = 127.0.0.1, Port = 53427

…

Exemple 3
sockets UDP

Exemple 3 - sockets UDP

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

#include<arpa/inet.h>
#include<netdb.h>
#include<netinet/in.h>

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

#define    LG_BUFFER    1024

int  lire_arguments (int  argc, char * argv[], struct sockaddr_in * adresse, char * protocole);

int main (int  argc, char * argv[]) {
  struct sockaddr_in adresse;
  int sock;
  char buffer[LG_BUFFER];
  int nb_lus;
```

Exemple 3 - sockets UDP

```
if ( lire_arguments (argc,  argv , & adresse, ‘‘udp’’ ) < 0 )
            exit(EXIT_FAILURE);
  adresse.sin_family = AF_INET;
  if (  (sock = socket (AF_INET, SOCK_DGRAM, 0 ) )  < 0 ) {
            perror (‘‘socket ’’);
            exit(EXIT_FAILURE);
  }
  while (1) {
            if ( (nb_lus = read ( STDIN_FILENO, buffer, LG_BUFFER) ) == 0 )
                        break;
            if ( nb_lus  < 0 ) {
                        perror (‘‘read ’’);
                        exit(EXIT_FAILURE);
            }
            sendto ( sock, buffer, nb_lus, 0 , (struct sockaddr *) & adresse, sizeof
(struct sockaddr_in));
  }
  return 0;
}
```

Exemple 3 - sockets UDP

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

#include<arpa/inet.h>
#include<netdb.h>
#include<netinet/in.h>

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

#define    LG_BUFFER     1024

int  lire_arguments (int  argc, char * argv[], struct sockaddr_in * adresse, char * protocole);

int main (int  argc, char * argv[]) {
  struct sockaddr_in adresse;
  int sock;
  char buffer[LG_BUFFER];
  int nb_lus;
```

Exemple 3 - sockets UDP

```
if ( lire_arguments (argc,  argv , & adresse, ‘‘udp’’ ) < 0 )
        exit(EXIT_FAILURE);
adresse.sin_family = AF_INET;
if (  (sock = socket (AF_INET, SOCK_DGRAM, 0 ) )  < 0 ) {
        perror (‘‘socket ’’);
        exit(EXIT_FAILURE);
}
if ( bind( sock, (struct sockaddr *) & adresse, sizeof( struct sockaddr_in) ) < 0 ) {
        perror (‘‘bind ’’);
        exit(EXIT_FAILURE);
}
setvbuf( stdout, NULL, _IONBF, 0);
while (1) {
        if ( ( nb_lus = recv ( sock, buffer, LG_BUFFER, 0)  ) == 0 )
                break;
        if ( nb_lus  < 0 ) {
                perror (‘‘recv ’’);  break;
        }
        write ( STDOUT_FILENO, buffer, nb_lus );
}
return EXIT_SUCCESS;
}
```

Exemple 4
sockets TCP
Serveur d'anagrammes …

Exemple 4 - sockets TCP – côté **serveur**

```
void traite_connexion (int sock) {
 int longueur;
 char buffer[256];

 while(1) {
        longueur = read (sock, buffer, 256);
        if ( longueur == 0 )
                break;
        if ( longueur  < 0 ) {
                perror (‘‘read ’’);
                exit( EXIT_FAILURE);
        }
        buffer[longueur] = ‘\0’;
        strfry(buffer);
        write ( sock, buffer, longueur);
  }
  close(sock);
}
```

Exemple 4 - sockets TCP – côté **client**

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

#include<arpa/inet.h>
#include<netdb.h>
#include<netinet/in.h>

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

#define    LG_BUFFER    1024

int  lire_arguments (int  argc, char * argv[], struct sockaddr_in * adresse, char * protocole);

int main (int  argc, char * argv[]) {
  struct sockaddr_in adresse;
  int sock;
  char buffer[LG_BUFFER];
  int nb_lus;
```

Exemple 4 - sockets TCP – côté **client**

```
if ( lire_arguments (argc,  argv , & adresse, ‘‘tcp’’ ) < 0 )
        exit(EXIT_FAILURE);
adresse.sin_family = AF_INET;
if (  (sock = socket (AF_INET, SOCK_STREAM, 0 ) )  < 0 ) {
        perror (‘‘socket ’’);
        exit(EXIT_FAILURE);
}
if ( connect( sock, & adresse, sizeof( struct sockaddr_in) ) < 0 ) {
        perror (‘‘connect’’);
        exit(EXIT_FAILURE);
}
```

Exemple 4 - sockets TCP – côté **client**

```
while (1) {
        if ( fgets ( buffer, 256, stdin)  == NULL )
                break;
        if (  buffer[ strlen(buffer) -1 ] == '\n' )
                 buffer[ strlen(buffer) -1 ] == '\0';
        if ( write ( sock, buffer, strlen(buffer)  )  < 0 ) {
                perror (''write '');
                break;
        }
        if ( ( nb_lus = read ( sock, buffer, LG_BUFFER, 0)  ) == 0 )
                break;
        if ( nb_lus  < 0 ) {
                perror (''read '');
                break;
        }
        fprintf (stdout, ''%s\n'', buffer);
}
return EXIT_SUCCESS;
}
```