

# Architecture Interne des Systèmes d'Exploitation (AISE)

Master CHPS

Jean-Baptiste BESNARD  
[jb.besnard@paratools.fr](mailto:jb.besnard@paratools.fr)

# Organisation du Cours

## Cour MATIN, TD après-midi

- ➡ 10/01: Généralités sur les OS et Utilisation de base
- ➡ 15/01 : La Chaine de Compilation et l'exécution d'un programme
- ➡ 16/01 : Les I/Os POSIX et Introduction aux Sockets**
- ➡ 30/01 : Méthodes de communication Inter-Processus
- ➡ 01/02 : Mémoire avancée (mmap, madvise, pages, TLB, ...)
- ➡ 06/02 : Programmation et reverse et Q/A projets (journée TD)
- ➡ 13/02 : TD Déboggage (gdb, valgrind) && TD mesure du temps et profilage (perf, kcachegrind) et Q/A projets (journée TD)
- ➡ Un examen final (25 Mars matin)

# Les I/Os POSIX

- Les descripteurs de fichier « bas-niveau »
- Les IO de haut niveau FILE\*
- Création de PIPE
- Projection de fichiers en mémoire
- La couche TCP
- Notion de Socket

# I/Os bas-niveau

# Les Fichiers Bas-Niveau

## L'état d'un descripteur de fichier:

- Droits (lecture ou écriture)
- Bidirectionnel (en fonction des droits)
- Peut correspondre à un flux ou bien un fichier sur le disque
- Possède un offset courant (cas d'un fichier)
- On peut y lire et écrire (Read/Write)

## Des descripteurs spéciaux:

- Stdin (0) (ou STDIN\_FILENO de unistd.h) -> Entrée standard
- Stdout (1) (ou STDOUT\_FILENO de unistd.h) -> Sortie standard
- Stderr (2) (ou STERR\_FILENO de unistd.h) -> Sortie d'erreur standard

## On peut créer un descripteur avec:

- Open (sur fichier)
- Pipe (sur flux)
- socket (pour une connection réseau)

# Ouvrir un Fichier

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int open(const char *path, int oflag, ... /* mode_t mode */);
```

**Soit deux empreintes pour ouvrir:**

```
int open(const char *path, int oflag );
int open(const char *path, int oflag, mode_t mode );
```

**Arguments:**

- **path:** chemin vers le fichier
- **oflag:** ou binaire entre les options (O\_RDWR, O\_CREAT, O\_APPEND, ... )
- **(si O\_CREAT) mode:** ou binaire définissant les droits du fichier

**Retour < 0 si erreur**

# Fermer un Fichier

```
#include <unistd.h>

int close(int fildes);
```

## Arguments:

- **fildes:** descripteur de fichier ouvert

**Retour < 0 si erreur**

# Créer un fichier vide

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>

int main(int argc, char **argv)
{
    int fd = open("./toto", O_RDWR | O_CREAT, S_IRUSR | S_IWUSR );

    if( fd < 0 )
    {
        perror("open");
    }

    close(fd);

    return 0;
}
```

# Créer un fichier vide

```
jbbesnard@deneb | <0> | lun. janv. 14 12:17:03
~/AISE_2
$ls -la toto
-rw----- 1 jbbesnard jbbesnard 0 janv. 14 12:16 toto
```

# Créer un fichier vide (non existant)

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>

int main(int argc, char **argv)
{
    int fd = open("./toto", O_RDWR | O_CREAT | O_EXCL,
                  S_IRUSR | S_IWUSR );

    if( fd < 0 )
    {
        perror("open");
    }

    close(fd);

    return 0;
}
```

# Créer un fichier vide (non existant)

```
jbbesnard@deneb | <0> | lun. janv. 14 12:19:29
~/AISE_2
$ ./a.out
open: File exists
```

# Lire dans un Fichier

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

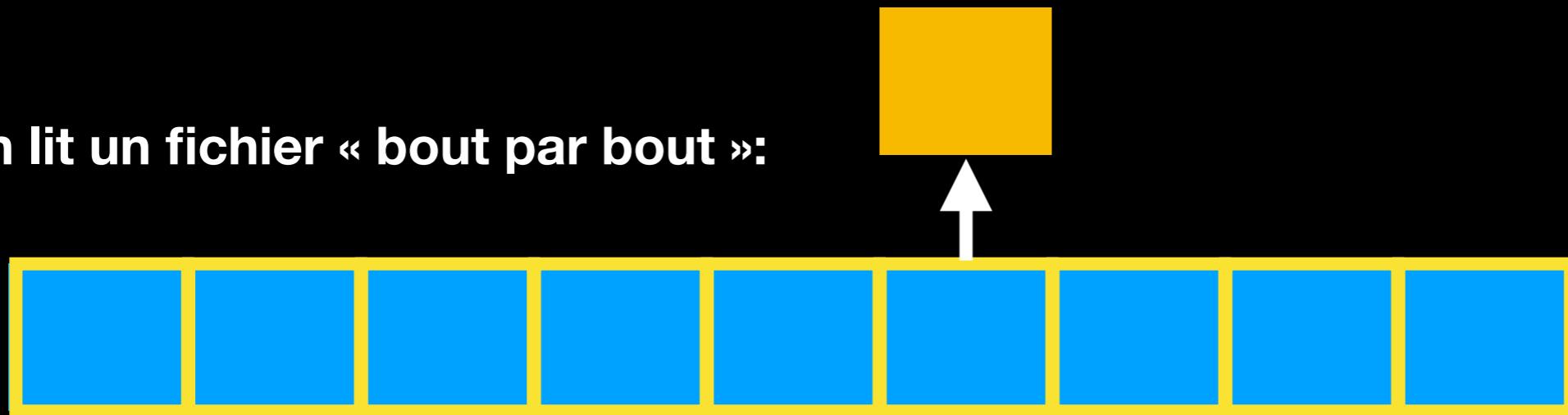
```
ssize_t read(int fd, void *buf, size_t count);
```

Descripteur

Buffer

Taille max

On lit un fichier « bout par bout »:



# Lire dans un Fichier (possible EAGAIN sur socket)

```
#include <stdio.h>
#include <time.h>
#include <unistd.h>
#include <stdlib.h>
#include <fcntl.h>

int main(int argc, char ** argv){

    if( argc != 2 )
    {
        fprintf(stderr, "Usage %s PATH\n", argv[0]);
        return 1;
    }

    int fd = open(argv[1], O_RDONLY);

    if( fd < 0 )
    {
        perror("open");
        return 1;
    }

    ssize_t cnt;
    char buff[500];

    while( (cnt = read(fd, buff, 500)) != 0)
    {
        if( cnt < 0 )
        {
            perror("read");
            return 1;
        }
    }

    close(fd);

    return 0;
}
```

# Écrire dans un Fichier

```
ssize_t safe_write(int fd, void *buff, size_t size)
{
    size_t written = 0;
    while( (size - written) != 0 )
    {
        errno = 0;
        ssize_t ret = write(fd, buff + written, size);

        if( ret < 0 )
        {
            if(errno == EAGAIN)
            {
                continue;
            }

            perror("write");
            return ret;
        }

        written += ret;
    }
}
```

Symétrie avec Read !

(possible EAGAIN sur socket par exemple géré ici)

# Écrire dans un Fichier

```
ssize_t safe_write(int fd, void *buff, size_t size)
{
    size_t written = 0;
    while( (size - written) != 0 )
    {
        errno = 0;
        ssize_t ret = write(fd, buff + written, size);

        if( ret < 0 )
        {
            if((errno == EAGAIN) || (errno == EINTR))
            {
                continue;
            }

            perror("write");
            return ret;
        }

        written += ret;
    }
}
```

On peut aussi vouloir gérer un signal entrant EINTR

# Liste des Cas

## Les cas à gérer pour read bas niveau:

- **EOF** : la fin du fichier (retour 0)
- **>0** : N bytes ont été lus (**peut être moins que SIZE!!**)
- **<0** : Une erreur s'est produite
  - ➔ errno == EINTR l'appel a été interrompu par un signal
  - ➔ errno == EAGAIN si on a marqué le fd O\_NONBLOCK

## Les cas à gérer pour write bas niveau:

- **>0** : N bytes ont été écrits (**peut être moins que SIZE!!**)
- **<0** : Une erreur s'est produite
  - ➔ errno == EINTR l'appel a été interrompu par un signal
  - ➔ errno == EAGAIN si on a marqué le fd O\_NONBLOCK

# Changer d'Offset

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

```
off_t lseek(int fd, off_t offset, int whence);
```

Whence	Description
SEEK_SET	Règle l'offset à « offset »
SEEK_CUR	Retourne l'offset courant plus le paramètre offset
SEEK_END	Retourne l'offset de fin plus le paramètre offset

# Se déplacer à la Fin d'un Fichier



```
lseek(fd, 0, SEEK_END);
```



Un équivalent ?

# Un équivalent ?

Ouvrir le FD avec le paramètre  
`O_APPEND` dans le Open !



R.T.**!**.M.

**open(2), close(2), stat(2), read(2), write(2),  
pread(2), pwrite(2), fsync(2)**  
**Avancé : fcntl(2)**

# UMASK

# Notion de UMASK

**Il est possible régler un masque de droit par défaut:**

- Pour automatiquement masquer certains droits sur les nouveaux fichiers
- Ces droits s'appliquent à tout fichier nouvellement créé et s'écrivent en octal

```
jbbesnard@deneb | <0> | lun. janv. 14 12:28:38  
~/AISE_2  
$umask  
0022
```

**Quels droits sont masqués ?**

Valeur	R	W	X
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

# Notion de UMASK

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>

int main(int argc, char **argv)
{
    int fd = open("./toto_um", O_RDWR | O_CREAT, 0777);

    if( fd < 0 )
    {
        perror("open");
    }

    close(fd);

    return 0;
}
```

Umask 0022 quels sont les droits de toto\_um ?

# Notion de UMASK

```
jbbesnard@adeneb | <0> | lun. janv. 14 12:35:22
~/AISE_2
$ls -lah toto_um
-rwxr-xr-x 1 jbbesnard jbbesnard 0 janv. 14 12:34 toto_um
```

# Notion de UMASK

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>

int main(int argc, char **argv)
{
    umask(0777);
    int fd = open("./toto_um", O_RDWR | O_CREAT, 0777);

    if( fd < 0 )
    {
        perror("open");
    }

    close(fd);

    return 0;
}
```

Quels sont les droits de toto\_um ?

# Notion de UMASK

```
jbbesnard@adeneb | <0> | lun. janv. 14 12:38:00
~/AISE_2
$ls -lah toto_um
----- 1 jbbesnard jbbesnard 0 janv. 14 12:37 toto_um
```

# Notion de UMASK

Comment régler les droits totalement dans OPEN ?

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>

int main(int argc, char **argv)
{
    int oldm = umask(0000);
    int fd = open("./toto_um", O_RDWR | O_CREAT, 0777 );
    umask(oldm);

    if( fd < 0 )
    {
        perror("open");
    }

    close(fd);

    return 0;
}
```

# Calcul du UMASK

D = PERM & (~ UMASK)

Exemple:

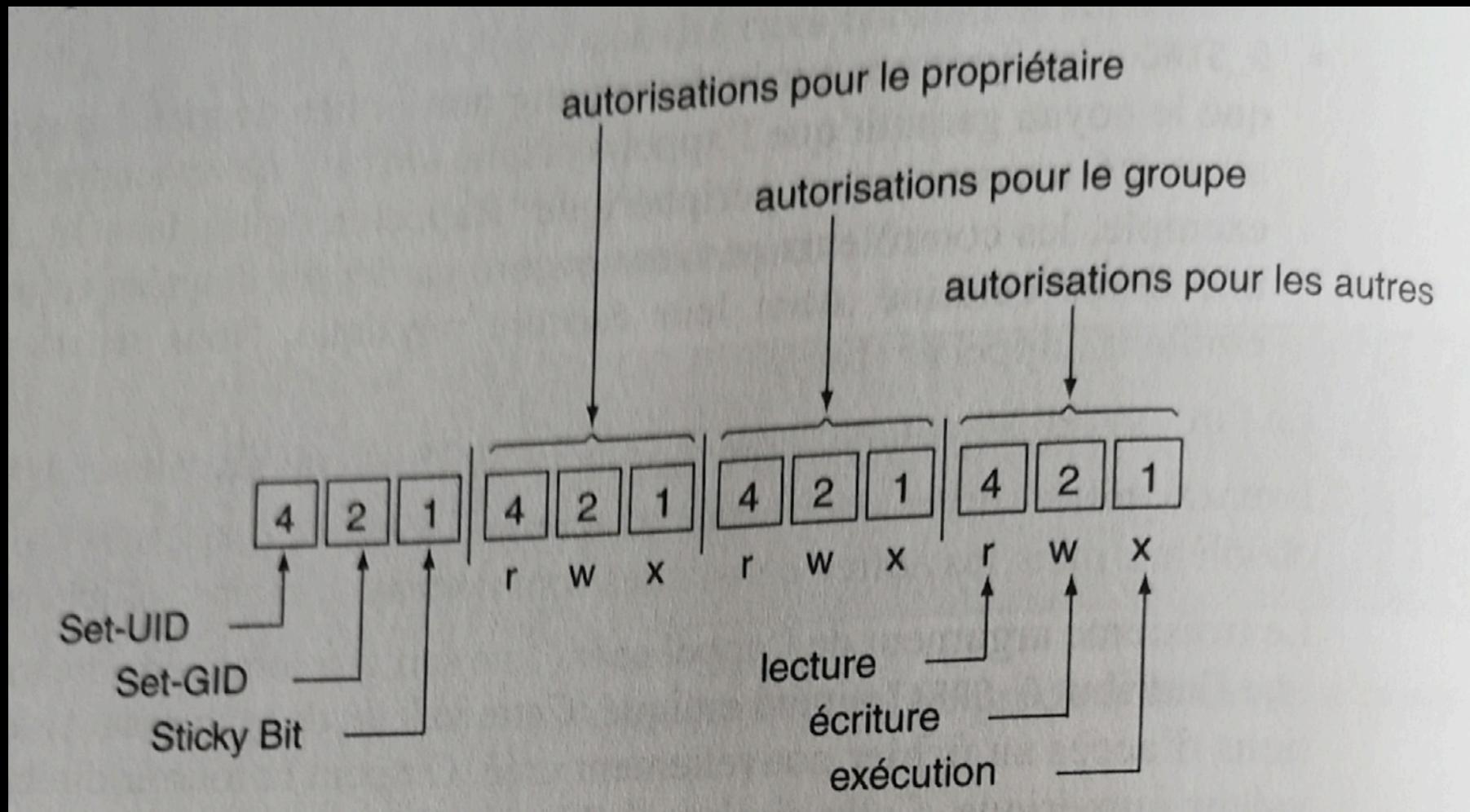
D = 0777 & (~ 0022)

D = 111 111 111 & (~ 000 010 010)

D = 111 111 111 & 111 101 101

D = 111 101 101

# Droits en Détail



**set-UID: execution possible avec l'utilisateur du binaire**

**set-GID: execution possible avec le groupe du binaire**

**Sticky bit (sur répertoire): seul le propriétaire du répertoire et du fichier peuvent le supprimer utilisé pour /tmp**

# I/Os Haut Niveau

# Ouvrir un Fichier (FILE \*)

**#include <stdio.h>**

**FILE \*fopen(const char \*pathname, const char \*mode);**

**FILE \*fdopen(int fd, const char \*mode);**

**Le FILE \* est une sur-couche au FD précédemment vu.**

# Fermer un Fichier (FILE \*)

```
#include <stdio.h>  
  
int fclose(FILE *stream);
```

# Ouvrir un Fichier (FILE \*)

```
#include <stdio.h>

int main(int argc, char ** argv) {

    FILE * fd = fopen(argv[1], "r");

    if(!fd){
        perror("fopen");
        return 1;
    }

    fclose(fd);

    return 0;
}
```

# Lire un Fichier

```
#include <stdio.h>

int main(int argc, char ** argv){
    FILE * fd = fopen(argv[1], "r");
    if(!fd){
        perror("fopen");
        return 1;
    }
    char buff[500];
    size_t cnt;

    while( 1 )
    {
        cnt = fread(buff, sizeof(char), 500, fd);
        if( cnt == 0 )
        {
            if( feof(fd) )
            {
                break;
            }
            else
            {
                perror("fread");
                return 1;
            }
        }
        /* USE your buff here */
    }
    fclose(fd);
    return 0;
}
```

# Lire ligne par ligne

```
#include <stdio.h>

int main(int argc, char ** argv){

    if( argc != 2 )
        return 1;

    FILE * fd = fopen(argv[1], "r");

    if(!fd){
        perror("fopen");
        return 1;
    }

    char buff[500];
    char * ret;

    while(1)
    {
        ret = fgets(buff, 500, fd);

        if(!ret)
        {
            if( feof(fd) )
            {
                /* EOF all OK */
                break;
            }
            else
            {
                /* Error */
                perror("fgets");
                return 1;
            }
        }

        /* USE your buff here */
        fprintf(stdout, "%s", ret );
    }

    fclose(fd);
    return 0;
}
```

# Ecrire dans un Fichier

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

int main(int argc, char ** argv){
    if(argc != 2 )
        return 1;

    FILE * fd = fopen(argv[1], "w");

    if(!fd){
        perror("fopen");
        return 1;
    }

    char data[ ] = "Hello I/Os\n";
    size_t cnt;

    cnt = fwrite(data, sizeof(char),
                 strlen(data), fd);

    if( cnt == 0 )
    {
        perror("fwrite");
        return 1;
    }

    fclose(fd);

    return 0;
}
```



**RT!M**

Before you ask those kinds of questions.

**fopen, fclose, fread, fwrite, fgets, ftell, fseek,  
feof, fileno, fdopen**

# Redirection de Flux

# Redirection de Flux

```
#include <unistd.h>

int dup2(int oldfd, int newfd);
```

**Dup2 remplace « newfd » par « oldfd » et se charge de fermer « newfd ».**

# Exemple

```
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/wait.h>

int main(int argc, char ** argv )
{
    int out = open("./out.dat", O_CREAT | O_WRONLY , 0600);

    pid_t child = fork();

    if( child == 0)
    {
        /* Replace stdout with the file */
        dup2(out, STDOUT_FILENO);
        char * argv[ ] = {"ls", "-la", NULL};
        execvp( argv[0], argv);
    }
    else
    {
        /* Parent closes out */
        close(out);
        wait(NULL);
    }

    return 0;
}
```

## Redirection de sortie dans un fichier

# Création de Pipe

```
#include <unistd.h>  
  
int pipe(int pipefd[2]);
```

Crée un « tuyau » == PIPE en anglais.

**pipefd[2] = { READ\_END, WRITE\_END };**



Un pipe est UNIDIRECTIONNEL

# Chainer deux Commandes

```
echo "Salut Tout Le Monde " | tac -s " "
```

```
$echo "Salut Tout Le Monde " | tac -s " "  
Monde Le Tout Salut
```

# Chainer deux Commandes

```
echo "Salut Tout Le Monde " | tac -s " "
```

```
./a.out
Monde Le Tout Salut
```

```
#include <fcntl.h>
#include <sys/types.h>
#include <sys/wait.h>

int main(int argc, char ** argv )
{
    int pp[2];
    pipe(pp);

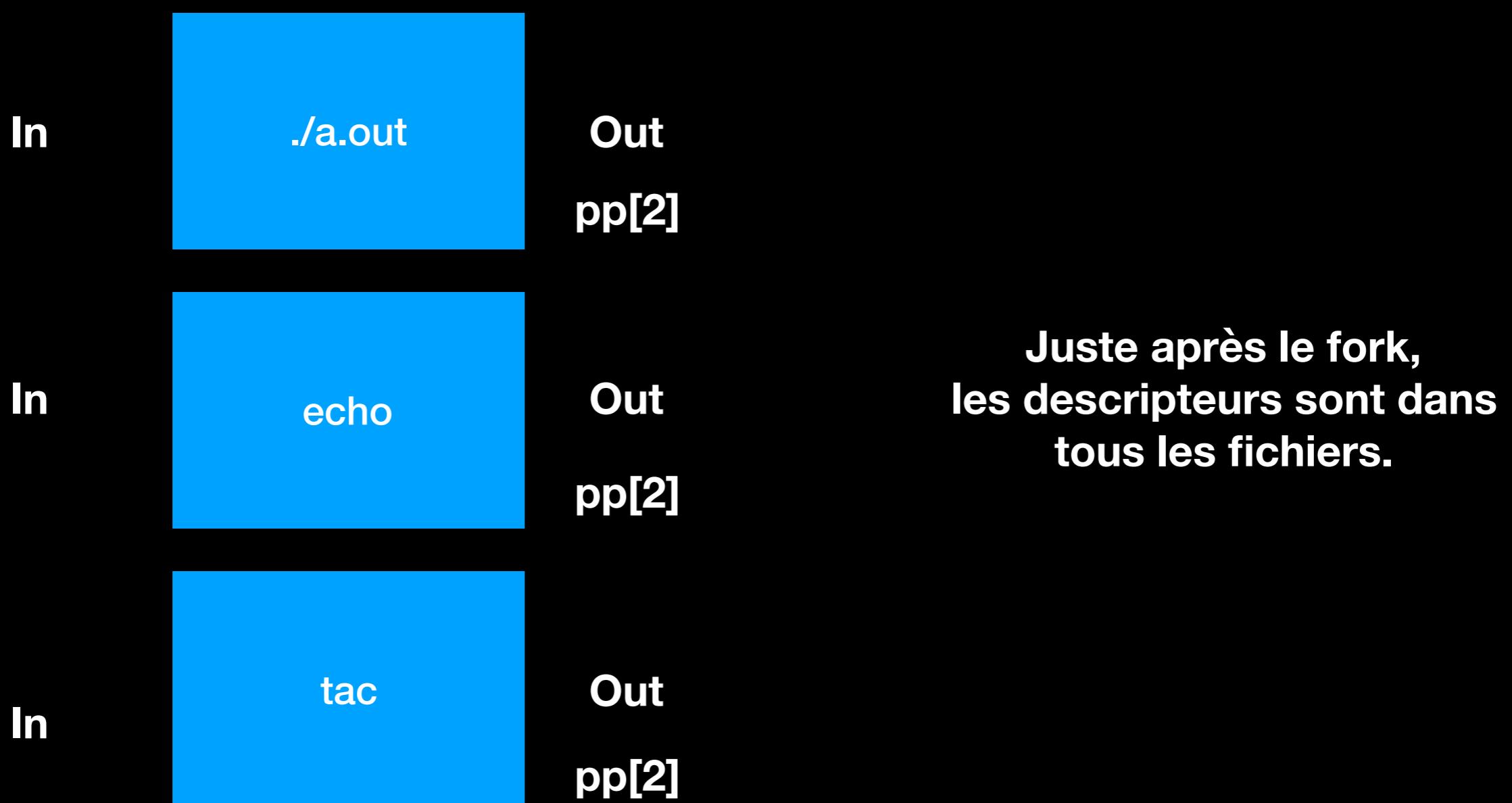
    pid_t child1 = fork();

    if( child1 == 0 )
    {
        /* Replace stdout with the write end of the pipe */
        dup2(pp[1], STDOUT_FILENO);
        /* Close read end of the pipe */
        close(pp[0]);
        /* Run command */
        char * argv[] = {« printf», "Salut Tout Le Monde « , NULL};
        execvp( argv[0], argv);
    }
    else
    {
        pid_t child2 = fork();

        if(child2 == 0)
        {
            /* Replace stdin with the read end of the pipe */
            dup2(pp[0], STDIN_FILENO);
            /* Close write end of the pipe */
            close(pp[1]);
            /* Run command */
            char * argv[] = {"tac", "-s", " ", NULL};
            execvp( argv[0], argv);
        }
        else
        {
            /* Close both end of the pipe */
            close(pp[0]);
            close(pp[1]);
            /* wait for two child */
            wait(NULL);
            wait(NULL);
        }
    }

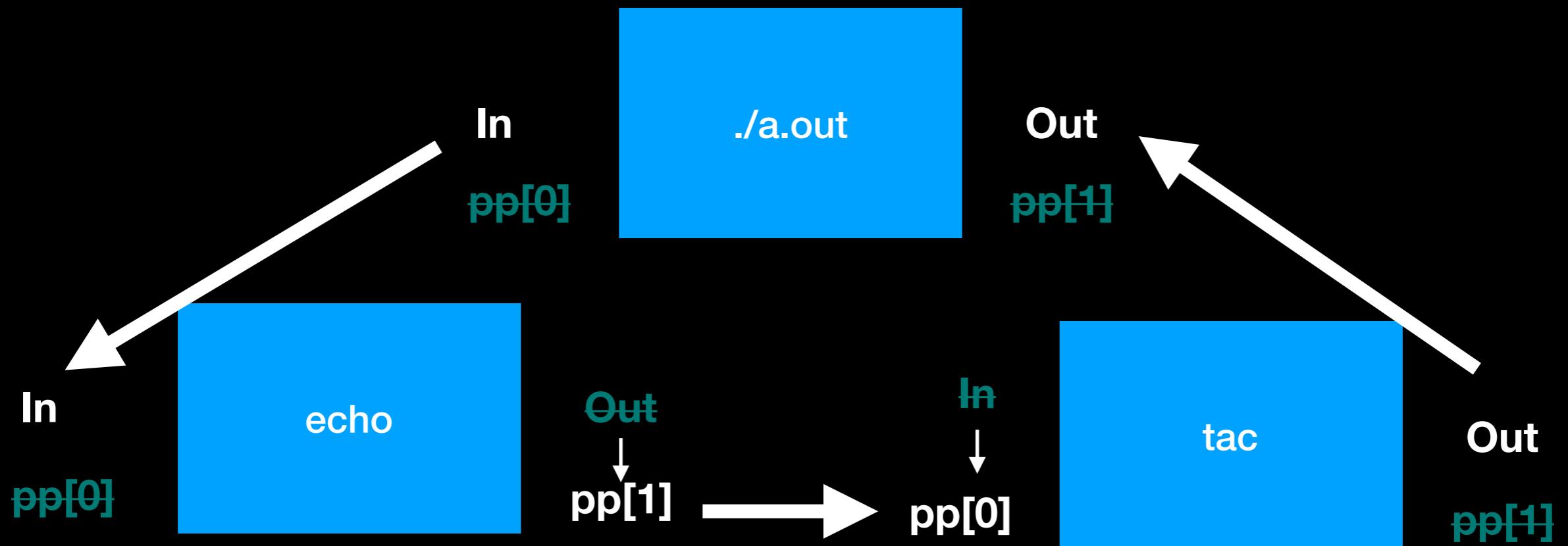
    return 0;
}
```

# Chainer deux Commandes



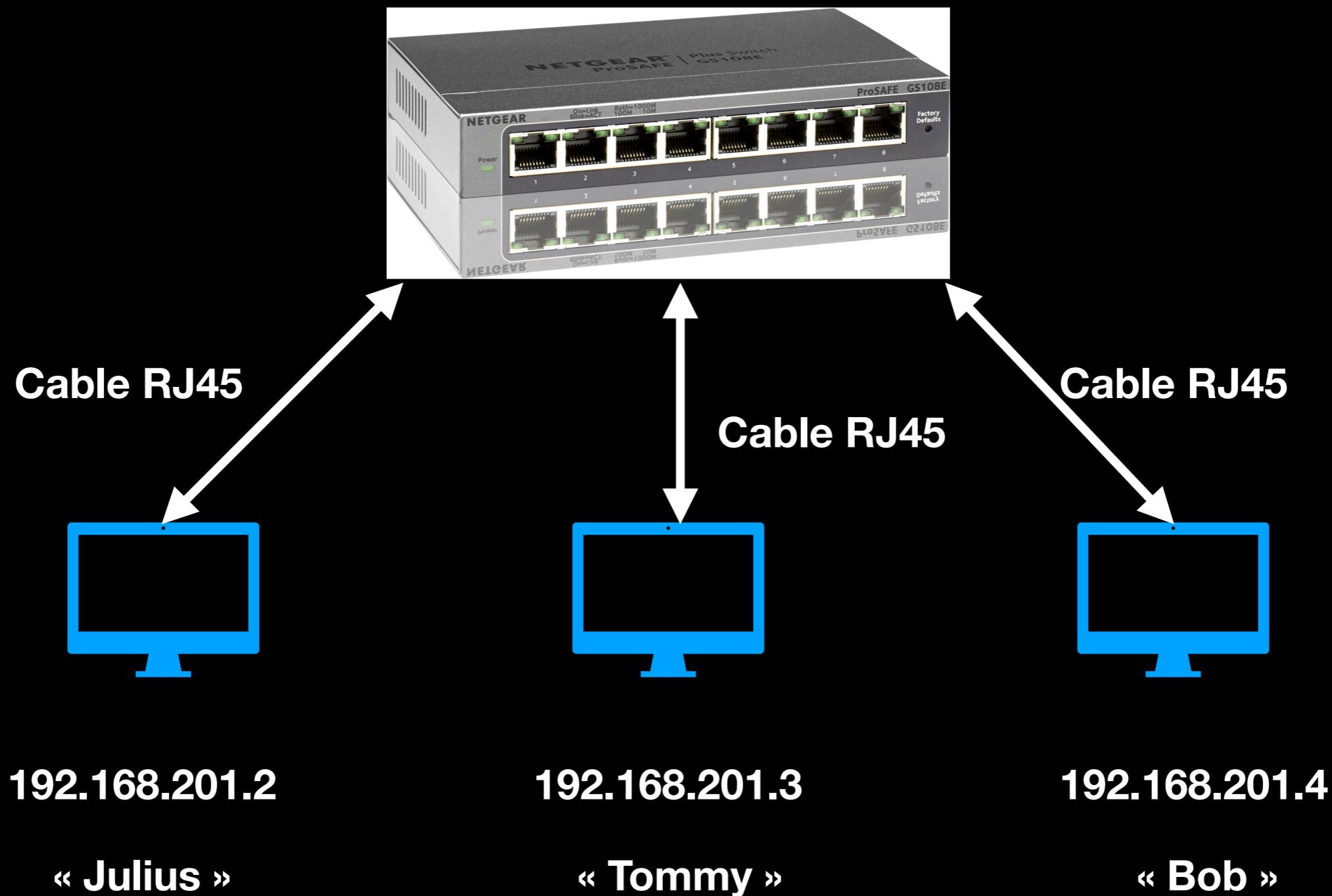
# Chainer deux Commandes

Ensuite on insère le PIPE entre les deux commandes.



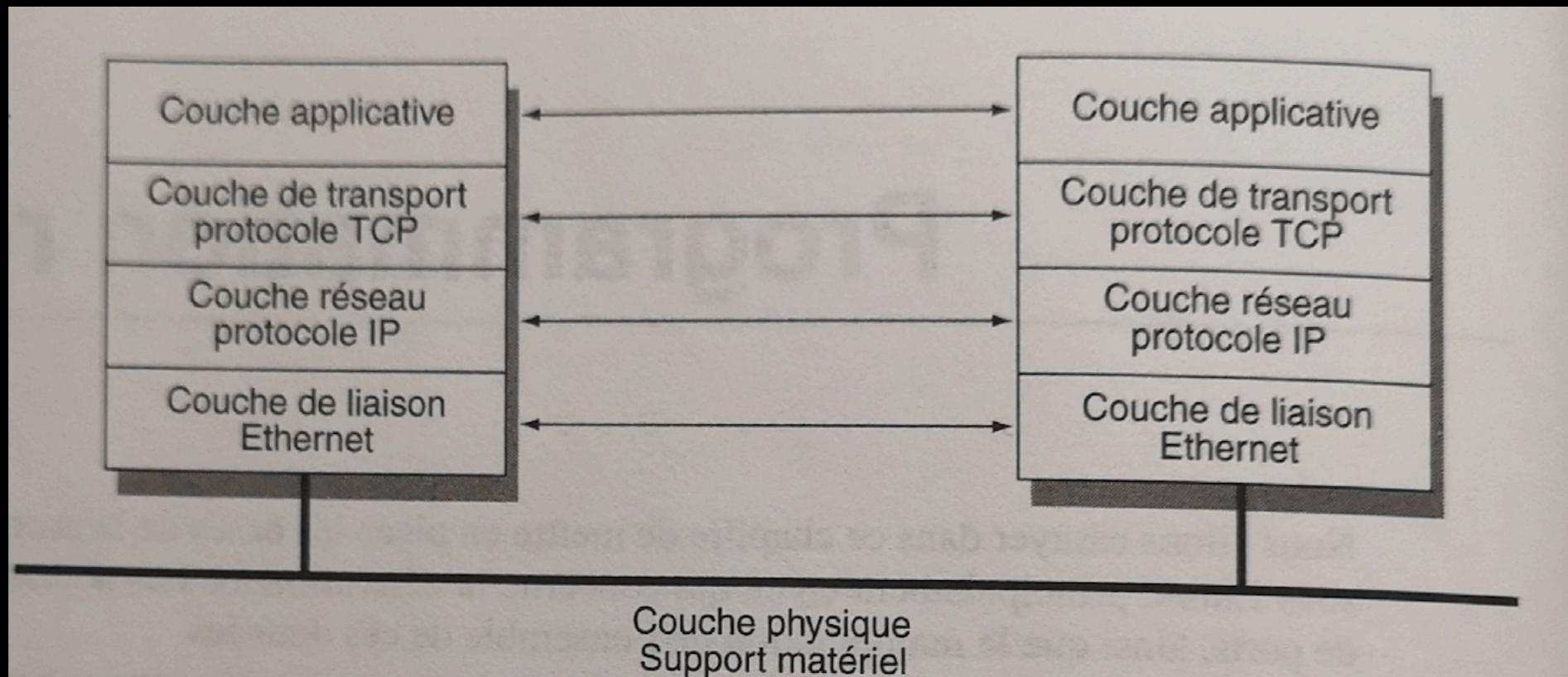
# Notion sur les Réseaux

# Couche Physique



# Modèle en Couche

Version simplifiée (à 5 couche) du modèle OSI (Open System Interconnection)



- **Couche ethernet -> adresses MAC**
- **Couche IP -> addresses IP**
- **Couche TCP -> Modèle d'encapsulation des données**
- **Couche applicative -> Ce qui est transmis**

TCP = Transmission Control Protocol

**TCP/IP**

# Couche IP

- Chaque machine possède une adresse IP
  - sur 4 Octets (32 bits) pour IPV4
  - Sur 16 octets (128 bits) IPV6
- Elle communique sur un réseau identifié par un masque:
  - 255.255.255.0 (IPV4) -> 254 machines (la valeur 255 est réservée)

```
Address: 192.168.0.1          11000000.10101000.00000000 .00000001
Netmask: 255.255.255.0 = 24   11111111.11111111.11111111 .00000000
Wildcard: 0.0.0.255           00000000.00000000.00000000 .11111111
=>
Network: 192.168.0.0/24       11000000.10101000.00000000 .00000000 (Class C)
Broadcast: 192.168.0.255      11000000.10101000.00000000 .11111111
HostMin: 192.168.0.1          11000000.10101000.00000000 .00000001
HostMax: 192.168.0.254        11000000.10101000.00000000 .11111110
Hosts/Net: 254                (Private Internet)
```

Ce réseau est 192.168.0.0/24 (selon les bits de masquage)

# Couche IP

Address:	192.168.0.1	11000000.10101000 .00000000.00000001
Netmask:	255.255.0.0 = 16	11111111.11111111 .00000000.00000000
Wildcard:	0.0.255.255	00000000.00000000 .11111111.11111111
=>		
Network:	192.168.0.0/16	11000000.10101000 .00000000.00000000 ( <b>Class C</b> )
Broadcast:	192.168.255.255	11000000.10101000 .11111111.11111111
HostMin:	192.168.0.1	11000000.10101000 .00000000.00000001
HostMax:	192.168.255.254	11000000.10101000 .11111111.11111110
Hosts/Net:	65534	(Private Internet)

**65534 machines pour un réseau de masque /16**

## Adresses non routable sur Internet

RFC1918 name	IP address range	Count	largest subnet mask)	host id size	mask bits	classful description[Note 1]
<b>24-bit block</b>	10.0.0.0 – 10.255.255.255	16777216	10.0.0.0/8 (255.0.0.0)	24 bits	8 bits	single class A network
<b>20-bit block</b>	172.16.0.0 – 172.31.255.255	1048576	172.16.0.0/12 (255.240.0.0)	20 bits	12 bits	16 contiguous class B networks
<b>16-bit block</b>	192.168.0.0 – 192.168.255.255	65536	192.168.0.0/16 (255.255.0.0)	16 bits	16 bits	256 contiguous class C networks

# Afficher l'adresse IP

\$ ip address

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: enp3s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 98:90:96:d2:65:0e brd ff:ff:ff:ff:ff:ff
    inet 192.168.201.42/24 brd 192.168.201.255 scope global dynamic enp3s0
        valid_lft 26772sec preferred_lft 26772sec
    inet6 fe80::9a90:96ff:fed2:650e/64 scope link
        valid_lft forever preferred_lft forever
3: wlp4s0: <BROADCAST,MULTICAST> mtu 1500 qdisc mq state DOWN group default qlen 1000
    link/ether 8a:8d:54:80:7b:23 brd ff:ff:ff:ff:ff:ff
```

# Afficher l'adresse IP

```
2: enp3s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 98:90:96:d2:65:0e brd ff:ff:ff:ff:ff:ff
    inet 192.168.201.42/24 brd 192.168.201.255 scope global dynamic enp3s0
        valid_lft 26772sec preferred_lft 26772sec
    inet6 fe80::9a90:96ff:fed2:650e/64 scope link
        valid_lft forever preferred_lft forever
```

- Interface -> enp3s0
- Connectée -> oui (UP)
- Adresse IPV4 -> 192.168.201.42/24
- Adresse IPV6 -> fe80::9a90:96ff:fed2:650e/64

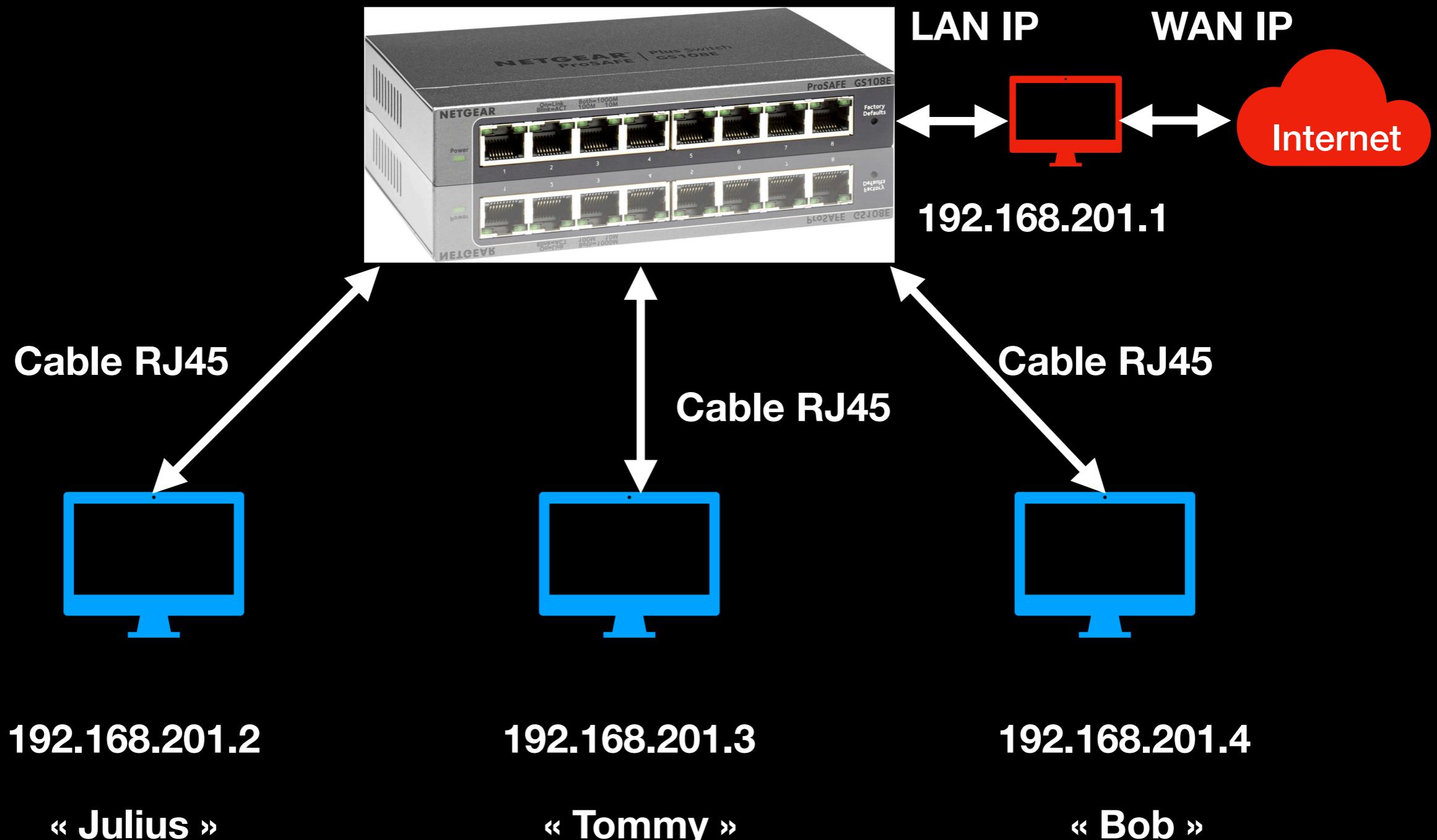
# Afficher les Routes

```
$ ip route
```

```
default via 192.168.201.1 dev enp3s0 proto static metric 100  
192.168.201.0/24 dev enp3s0 proto kernel scope link src 192.168.201.42 metric 100
```

- 192.168.201.0/24 est sur enp3s0
- « default » tout le reste est sur la machine 192.168.201.1

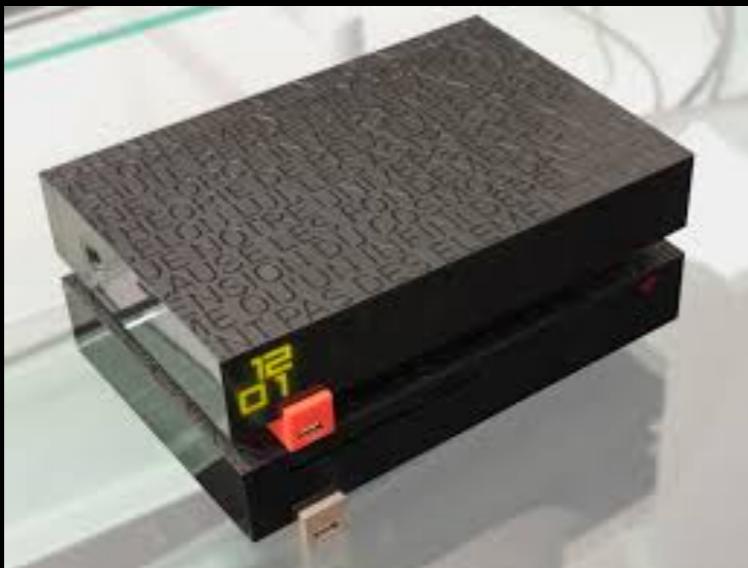
# Couche Physique



WAN = Wide Area Network  
LAN = Local Area Network

# La Passerelle

- La machine qui est connectée à la fois à internet et au réseau local est appelée la « passerelle ».
- Chez les particulier c'est souvent une « Box » qui opère comme routeur entre ces deux réseau. Plus généralement c'est une machine avec au moins deux interface réseau.

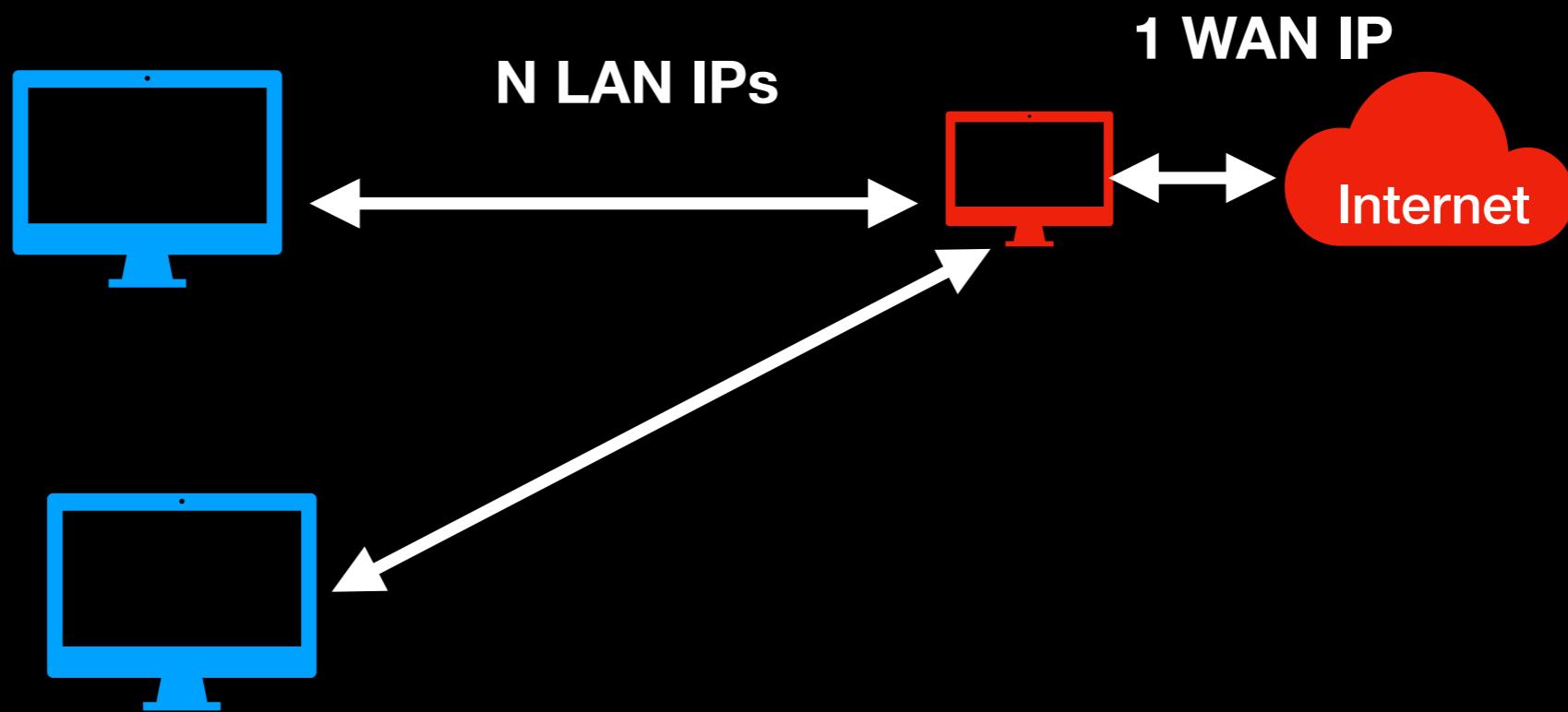


# La Passerelle en Entreprise

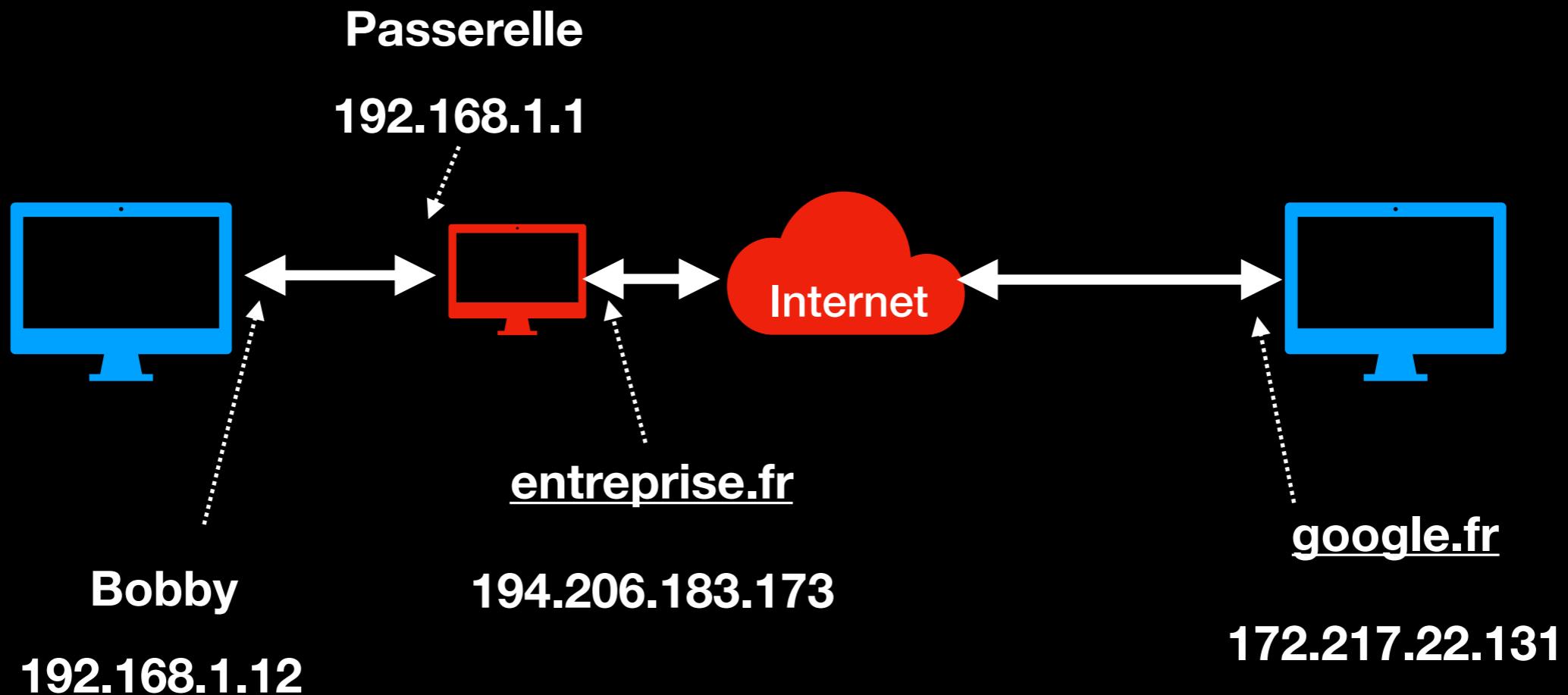


**Exemples CISCO de routeurs configurables.**

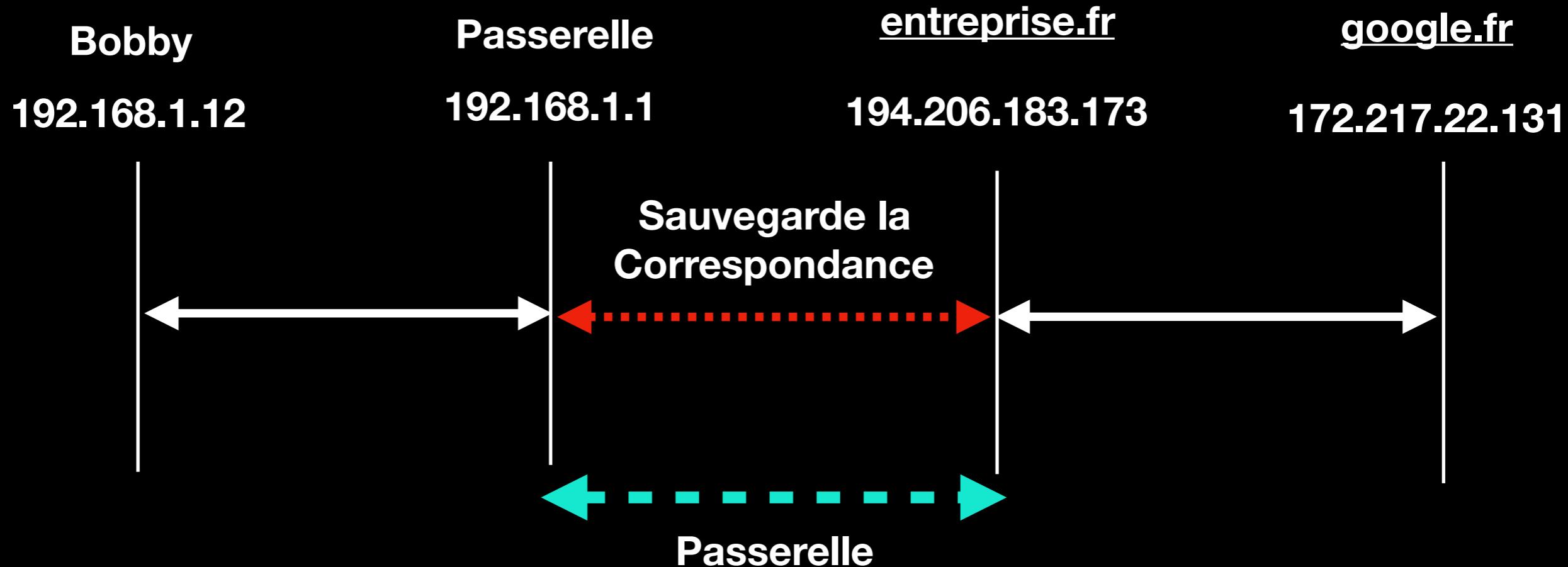
# Network Address Translation



# Network Address Translation



# Network Address Translation



Google ne voit que votre passerelle qui se fait passer pour vous et vous renvoie les données

# Transmission Control Protocol

- TCP permet d'établir des connections « fiables » entre machines en permettant:
  - ➔ Une validation de l'intégrité des données
  - ➔ Un mode connecté (A parle à B)
  - ➔ Un contrôle de flux (l'envoi bloque si la source ne lit pas assez vite par exemple)
- TCP connecte deux **IP** et deux **PORTS**
  - ➔ Un port est une « porte » vers votre machine
  - ➔ Il peut être « ouvert » ou « sortant »
  - ➔ Toute connection TCP est bidirectionnelle (deux FD)
  - ➔ Un port peut être en écoute (en attente de nouvelle connexions)
  - ➔ Plusieurs connexions peuvent utiliser le même port en écoute



192.168.201.2 : 3689

192.168.201.9 : 80

# Services Communs

Protocol	Port	Name	Description
FTP	tcp/20, tcp21	File Transfer Protocol	Sends and receives files between systems
SSH	tcp/22	Secure Shell	Encrypted console access
Telnet	tcp/23	Telecommunication Network	Insecure console access
SMTP	tcp/25	Simple Mail Transfer Protocol	Transfer email between mail servers
DNS	udp/53, tcp/53	Domain Name System	Convert domain names to IP addresses
HTTP	tcp/80	Hypertext Transfer Protocol	Web server communication
POP3	tcp/110	Post Office Protocol version 3	Receive email into a email client
IMAP4	tcp/143	Internet Message Access Protocol v4	A newer email client protocol
HTTPS	tcp/443	Hypertext Transfer Protocol Secure	Web server communication with encryption
RDP	tcp/3389	Remote Desktop Protocol	Graphical display of remote devices
NetBIOS	udp/137	NetBIOS name service	Register, remove, and find Windows services by name
NetBIOS	udp/138	NetBIOS datagram service	Windows connectionless data transfer
NetBIOS	tcp/139	NetBIOS session service	Windows connection-oriented data transfer
SLP	tcp/427, udp/427	Service Location Protocol	Find Mac OS services by name
SMB	tcp/445	Server Message Block	Windows file transfers and printer sharing
AFP	tcp/548	Apple Filing Protocol	Mac OS file transfers

# Créer un Socket TCP

**Un socket est un descripteur de fichier correspondant à un flux réseau**

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

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

- Domain -> type de socket
  - ➔ AF\_UNIX -> socket UNIX reposant sur un fichier (socket local)
  - ➔ AF\_INET -> socket IPV4
  - ➔ AF\_INET6 -> socket IPV6
- Type -> Mode de communications
  - ➔ SOCK\_STREAM -> flux de donnée TCP
  - ➔ SOCK\_DGRAM -> datagramme (UDP)
  - ➔ SOCK\_RAX -> accès bas niveau à l'interface (uniquement root)
- Protocol -> Protocole à utiliser
  - ➔ En général on laisse 0

# Résolution de Nom DNS

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

int getaddrinfo(const char *node, const char *service,
                const struct addrinfo *hints,
                struct addrinfo **res);

struct addrinfo {
    int                         ai_flags;          // AI_PASSIVE, AI_CANONNAME, etc.
    int                         ai_family;         // AF_INET, AF_INET6, AF_UNSPEC
    int                         ai_socktype;       // SOCK_STREAM, SOCK_DGRAM
    int                         ai_protocol;       // use 0 for "any"
    size_t                      ai_addrlen;        // size of ai_addr in bytes
    struct sockaddr *ai_addr;        // struct sockaddr_in or _in6
    char                        *ai_canonname;     // full canonical hostname

    struct addrinfo *ai_next;        // linked list, next node
};
```

Convertir adresse DNS google.fr en:

IPV4 : 172.217.22.131

IPV6 : 2a00:1450:4007:815::2003

# Résolution de Nom DNS

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

int main( int argc, char ** argv ) {
    struct addrinfo *res = NULL;
    struct addrinfo hints;
    memset(&hints, 0, sizeof(hints));
    hints.ai_family = AF_UNSPEC;
    hints.ai_socktype = SOCK_STREAM;
    int ret = getaddrinfo(argv[1], argv[2],
        &hints,
        &res);
    if( ret < 0 ) {
        perror("getaddrinfo");
        return 1;
    }
    struct addrinfo *tmp;
    for (tmp = res; tmp != NULL; tmp = tmp->ai_next) {
        char ip[INET6_ADDRSTRLEN];
        ip[0] = '\0';
        if(tmp->ai_family == AF_INET) {
            struct sockaddr_in* saddr = (struct sockaddr_in*)tmp->ai_addr;
            inet_ntop(AF_INET, &saddr->sin_addr, ip, sizeof(ip));
            printf("IPV4 : %s\n", ip);
        }
        else if(tmp->ai_family == AF_INET6) {
            struct sockaddr_in6* saddr6 = (struct sockaddr_in6*)tmp->ai_addr;
            inet_ntop(AF_INET6, &saddr6->sin6_addr, ip, sizeof(ip));
            printf("IPV6 : %s\n", ip);
        }
    }
    return 0;
}
```

```
hints.ai_family = AF_UNSPEC;
IPV4 : 172.217.22.131
IPV6 : 2a00:1450:4007:815::2003

hints.ai_family = AF_INET;
IPV4 : 172.217.22.131

hints.ai_family = AF_INET6;
IPV6 : 2a00:1450:4007:815::2003
```

# Client-Serveur TCP (version simplifiée)

# Client

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

int main( int argc, char ** argv )
{
    struct addrinfo *res = NULL;
    struct addrinfo hints;
    memset(&hints, 0, sizeof(hints));
    hints.ai_family = AF_UNSPEC;
    hints.ai_socktype = SOCK_STREAM;

    int ret = getaddrinfo(argv[1], argv[2],
                          &hints,
                          &res);

    if( ret < 0 )
    {
        perror("getaddrinfo");
        return 1;
    }
```

Suite du code ici ...

```
    struct addrinfo *tmp;
    int sock = -1;
    int connected = 0;

    for (tmp = res; tmp != NULL; tmp = tmp->ai_next) {
        sock = socket(tmp->ai_family,
                      tmp->ai_socktype,
                      tmp->ai_protocol);

        if( sock < 0 ) {
            perror("socket");
            continue;
        }

        int ret = connect( sock, tmp->ai_addr,
                           tmp->ai_addrlen);

        if( ret < 0 ) {
            close(sock);
            perror("connect");
            continue;
        }

        connected = 1;
        break;
    }

    if( !connected ) {
        fprintf(stderr, "Failed to connect to %s:%s\n",
                argv[1], argv[2]);
        return 1;
    }

    /* Use the socket */
    close(sock);
    return 0;
}
```

# Serveur

## (Suite)

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

int main( int argc, char ** argv ) {
    struct addrinfo *res = NULL;
    struct addrinfo hints;
    memset(&hints, 0, sizeof(hints));
    hints.ai_family = AF_UNSPEC;
    hints.ai_socktype = SOCK_STREAM;
    hints.ai_flags = AI_PASSIVE;

    if(argc != 2)
        return 1;

    int ret = getaddrinfo(NULL, argv[1], &hints, &res);

    if( ret < 0 ) {
        perror("getaddrinfo");
        return 1;
    }

    struct addrinfo *tmp;
    int listen_sock = -1;
    int binded = 0;

    for (tmp = res; tmp != NULL; tmp = tmp->ai_next) {
        listen_sock = socket(tmp->ai_family,
                            tmp->ai_socktype,
                            tmp->ai_protocol);

        if( listen_sock < 0 ) {
            perror("sock");
            continue;
        }

        ret = bind( listen_sock, tmp->ai_addr, tmp->ai_addrlen);

        if( ret < 0 ) {
            close(listen_sock);
            perror("bind");
            continue;
        }
    }
}
```

```
if(!binded)
{
    fprintf(stderr, "Failed to bind on 0.0.0.0:%s\n", argv[1]);
    return 1;
}

/* Start listening */
ret = listen(listen_sock, 2);

if( ret < 0 )
{
    perror("listen");
    return 1;
}

/* Now accept one connection */
struct sockaddr client_info;
socklen_t addr_len;
fprintf(stderr,"Before accept\n");
int client_socket = accept(listen_sock, &client_info, &addr_len);
fprintf(stderr,"After accept\n");

if( client_socket < 0 )
{
    perror("accept");
    return 1;
}

fprintf(stderr,"Closing client socket\n");
close(client_socket);

close(listen_sock);

return 0;
}
```

# Serveur

Paramétrage serveur Démarrage du serveur

Création Socket Attache adresse au Socket

Accueil des clients

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

int main( int argc, char ** argv ) {
    struct addrinfo *res = NULL;
    struct addrinfo hints;
    memset(&hints, 0, sizeof(hints));
    hints.ai_family = AF_UNSPEC;
    hints.ai_socktype = SOCK_STREAM;
    hints.ai_flags = AI_PASSIVE;

    if(argc != 2)
        return 1;

    int ret = getaddrinfo(NULL, argv[1], &hints, &res);

    if( ret < 0 ) {
        perror("getaddrinfo");
        return 1;
    }

    struct addrinfo *tmp;
    int listen_sock = -1;
    int binded = 0;

    for (tmp = res; tmp != NULL; tmp = tmp->ai_next) {
        listen_sock = socket(tmp->ai_family,
                            tmp->ai_socktype,
                            tmp->ai_protocol);

        if( listen_sock < 0 ) {
            perror("sock");
            continue;
        }

        ret = bind( listen_sock, tmp->ai_addr, tmp->ai_addrlen);

        if( ret < 0 ) {
            close(listen_sock);
            perror("bind");
            continue;
        }
    }
}
```

```
if(!binded)
{
    fprintf(stderr, "Failed to bind on 0.0.0.0:%s\n", argv[1]);
    return 1;
}

/* Start listening */
ret = listen(listen_sock, 2);

if( ret < 0 )
{
    perror("listen");
    return 1;
}

/* Now accept one connection */
struct sockaddr client_info;
socklen_t addr_len;
fprintf(stderr,"Before accept\n");
int client_socket = accept(listen_sock, &client_info, &addr_len);
fprintf(stderr,"After accept\n");

if( client_socket < 0 )
{
    perror("accept");
    return 1;
}

fprintf(stderr,"Closing client socket\n");
close(client_socket);

close(listen_sock);

return 0;
}
```