

IO et Réseau

M1 - CHPS

Architecture Interne des Systèmes d'exploitations (AISE)

Jean-Baptiste Besnard
<jbbesnard@paratools.com>



Programme du Semestre

- 1 - Généralités sur les OS
- 2 - **Entrées-Sorties et Réseau**

Type d'Examen	Coefficient
QCMs	10 %
PROJET	40 %
EXAMEN	50 %

Cours et Corrections



https://github.com/besnardjb/AISE_25

Point sur le Projet

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 STDERR_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>
```

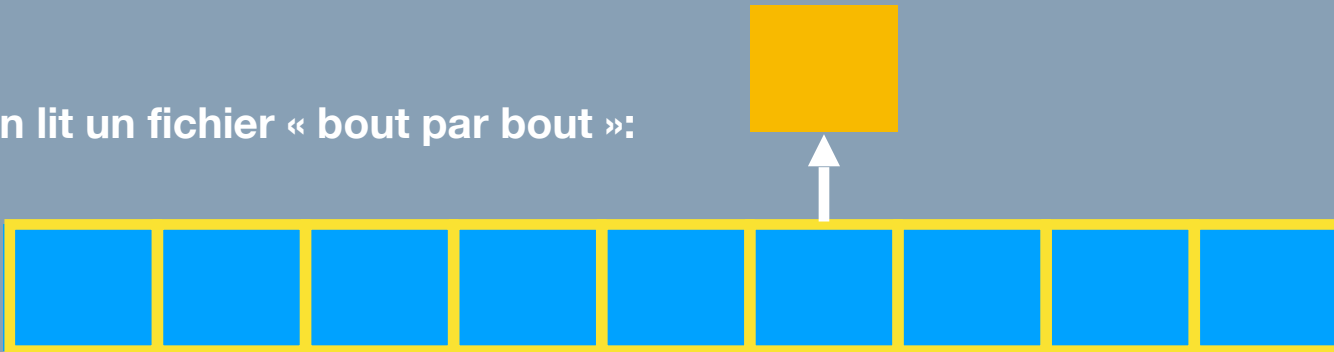
```
ssize_t read(int fd, void *buf, size_t count)
```

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-written);

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

            perror("write");
            return ret;
        }

        written += ret;
    }
}
```

Symétrie avec Read !

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 == 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 on 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 on 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

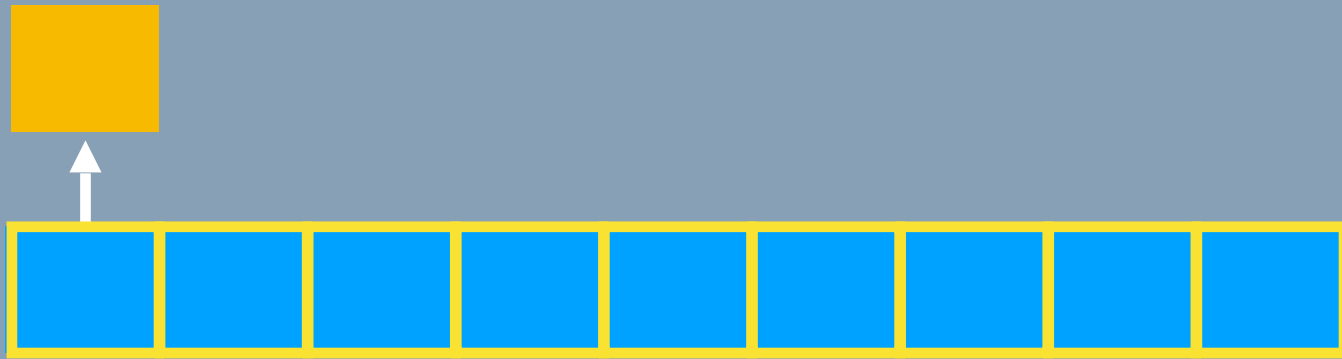
```
#include <sys/types.h>
```

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

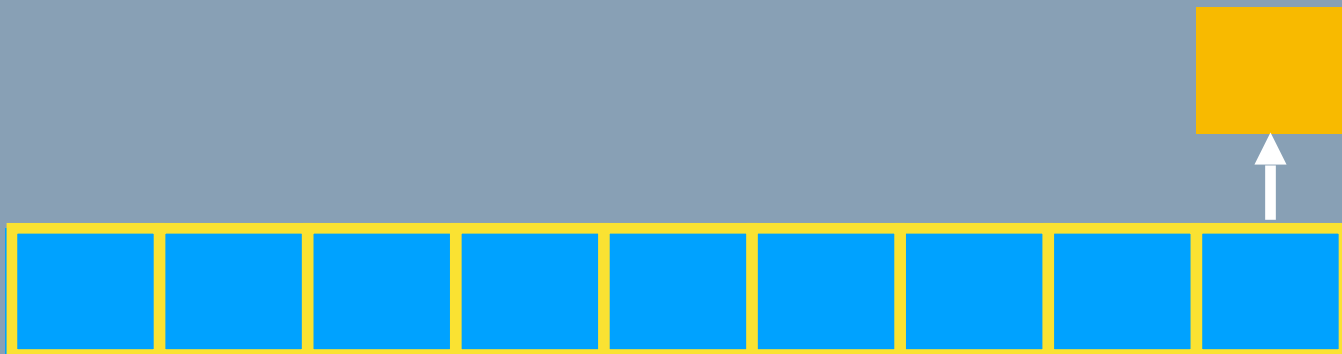
```
off_t lseek(int fd, off_t offset, int whence);
```

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@denéb | <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@deneb | <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@deneb | <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

$$D = \text{PERM} \& (\sim \text{UMASK})$$

Exemple:

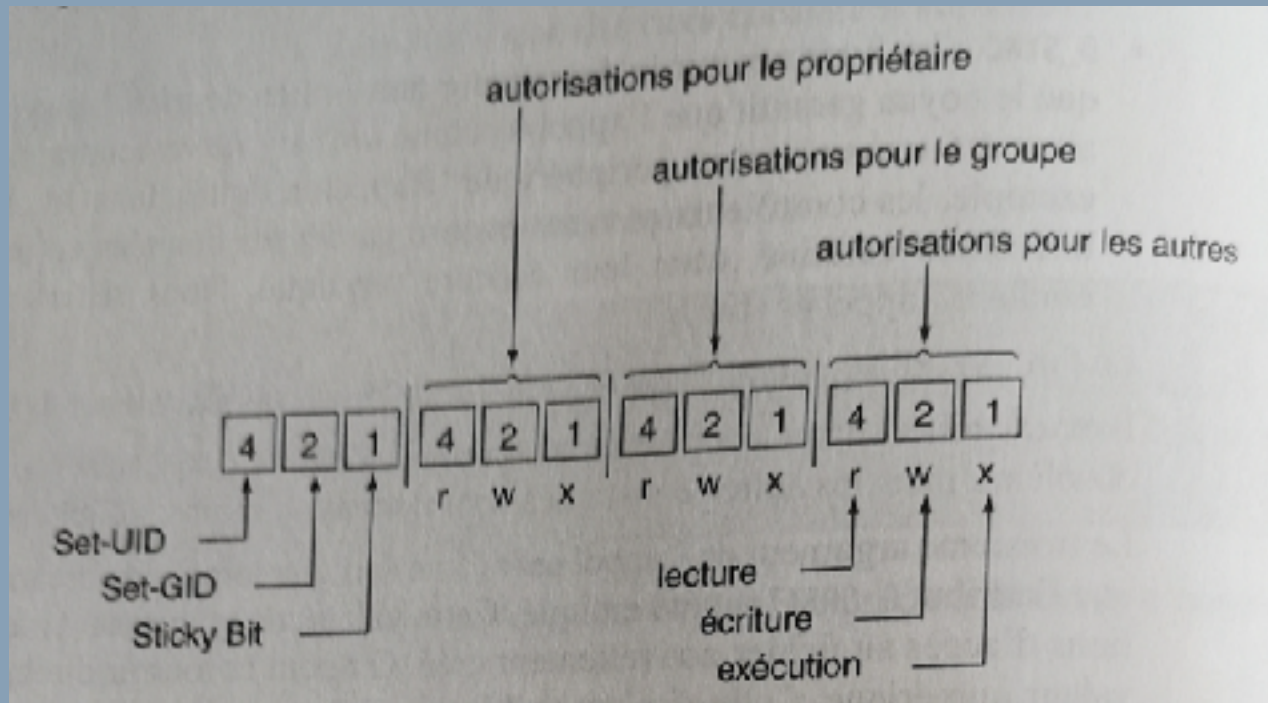
$$D = 0777 \& (\sim 0022)$$

$$D = 111\ 111\ 111 \& (\sim 000\ 010\ 010)$$

$$D = 111\ 111\ 111 \& 111\ 101\ 101$$

$$D = 111\ 101\ 101$$

Droits en Détail



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

set-GID: execution possible avec le groupe du binaire

Sticky Bit: seul le propriétaire du répertoire et du fichier peuvent le supprimer

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, fgetc, fputc, 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

Redirection de sortie dans un fichier

```
#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 )
{
    pid_t child = fork();

    if( child == 0)
    {
        int out = open("./out.dat", O_CREAT | O_WRONLY ,
                        0600);
        /* Replace stdout with the file */
        dup2(out, STDOUT_FILENO);
        close(out);
        char * argv[] = {"ls", "-la", NULL};
        execvp( argv[0], argv);
    }
    else
    {
        /* Parent closes out */
        wait(NULL);
    }

    return 0;
}
```

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
```

```
11:10:11 PM ~ % echo "Salut Tout Le Monde " | tac -s " " 14 16 17 19
```

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

In

./a.out

Out

pp[2]

In

echo

Out

pp[2]

In

tac

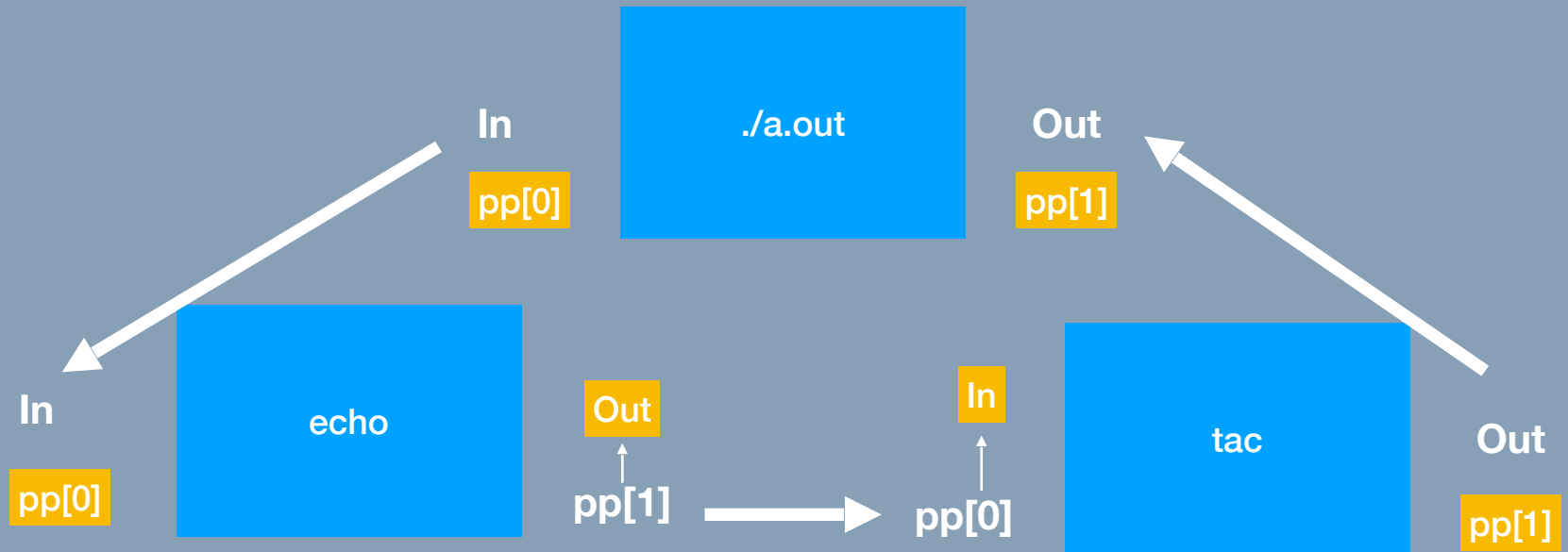
Out

pp[2]

Juste après le fork,
les descripteurs sont dans
tous les fichiers.

Chainer deux Commandes

Ensuite on insère le PIPE entre les deux commandes.



Réseau

Programmation Réseau

En HPC le réseau est un aspect crucial car les machines sont:

- Distribuées (milliers de noeuds)
- La mémoire par noeud est limitée

Il faut donc pouvoir interconnecter des milliers de processus UNIX pour former un calcul cohérent. MPI permet généralement cela Mais il repose sur des réseaux et techniques plus bas niveau dont nous avons déjà vu certaines:

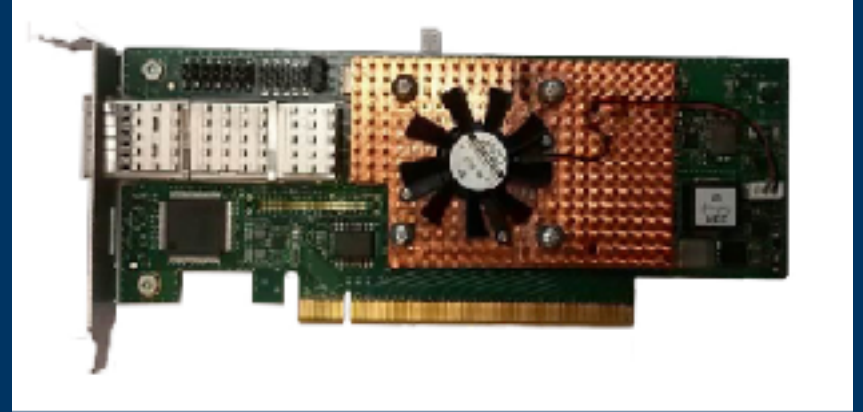
- Segment SHM
- Réseaux TCP (rare)
- Réseaux rapides (voir aperçu slide suivant)

Aujourd'hui nous allons implémenter des appels que vous faites plusieurs milliers de fois par jour à chaque fois que vous allez sur internet, consultez vos mail, parlez à votre assistant vocal ...

Réseaux HPC



Infiniband (libverbs)



Bull Exascale Interconnect (Portals 4)

- Tofu interconnect
- Aries Interconnect
- Cray Slighshot
- TH Express

Base du Réseau

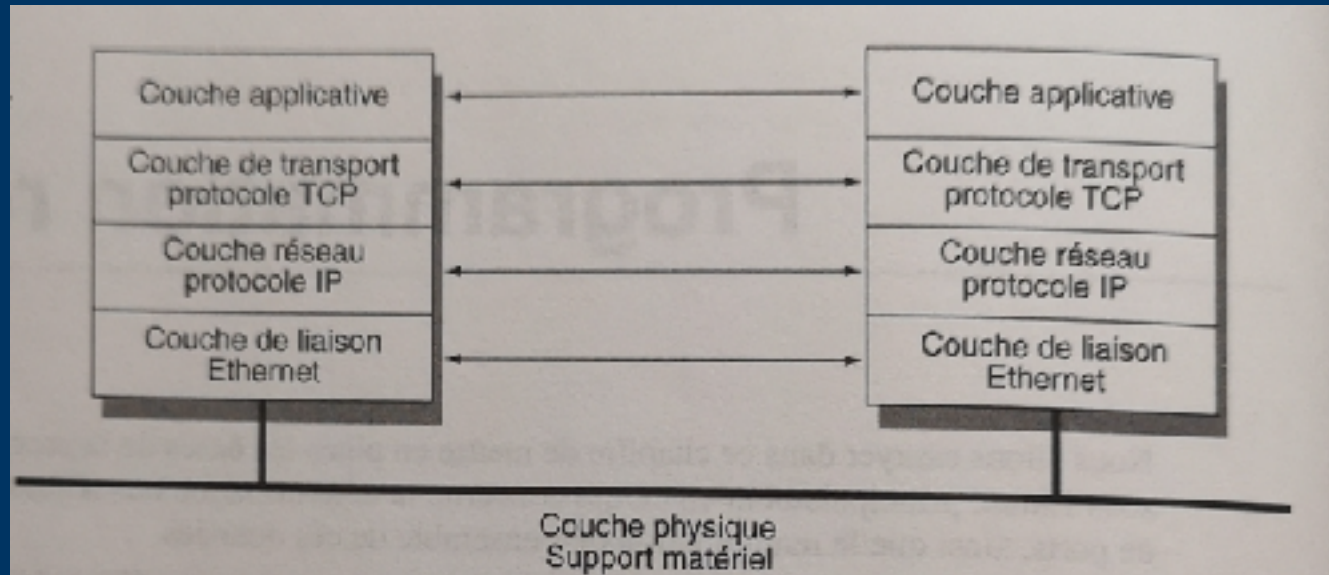
Très rapidement le but d'un réseau est de connecter des machines entre elles on peut citer comme support de réseau:

- Cable Ethernet (RJ45)
- Wifi (802.11a/b/g/n/ac)
- Bluetooth (802.15.1)
- Zigbee (802.15.4) (domotique)

D'un point de vue système ces différents réseaux reposent sur des interfaces relativement similaires nous allons voir aujourd'hui principalement du TCP mais sachez que cela se transpose assez facilement à d'autres types de réseaux.

Modèle en Couche (Rappel)

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 (Class C)
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
= 256 x 256 - 2

Exemple d'adresses non routable sur Internet

RFC1918 name	IP address range	Count	largest subnet mask)	host id size	mask bits	classful description[Note 1]
24-bit block	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
20-bit block	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
16-bit block	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

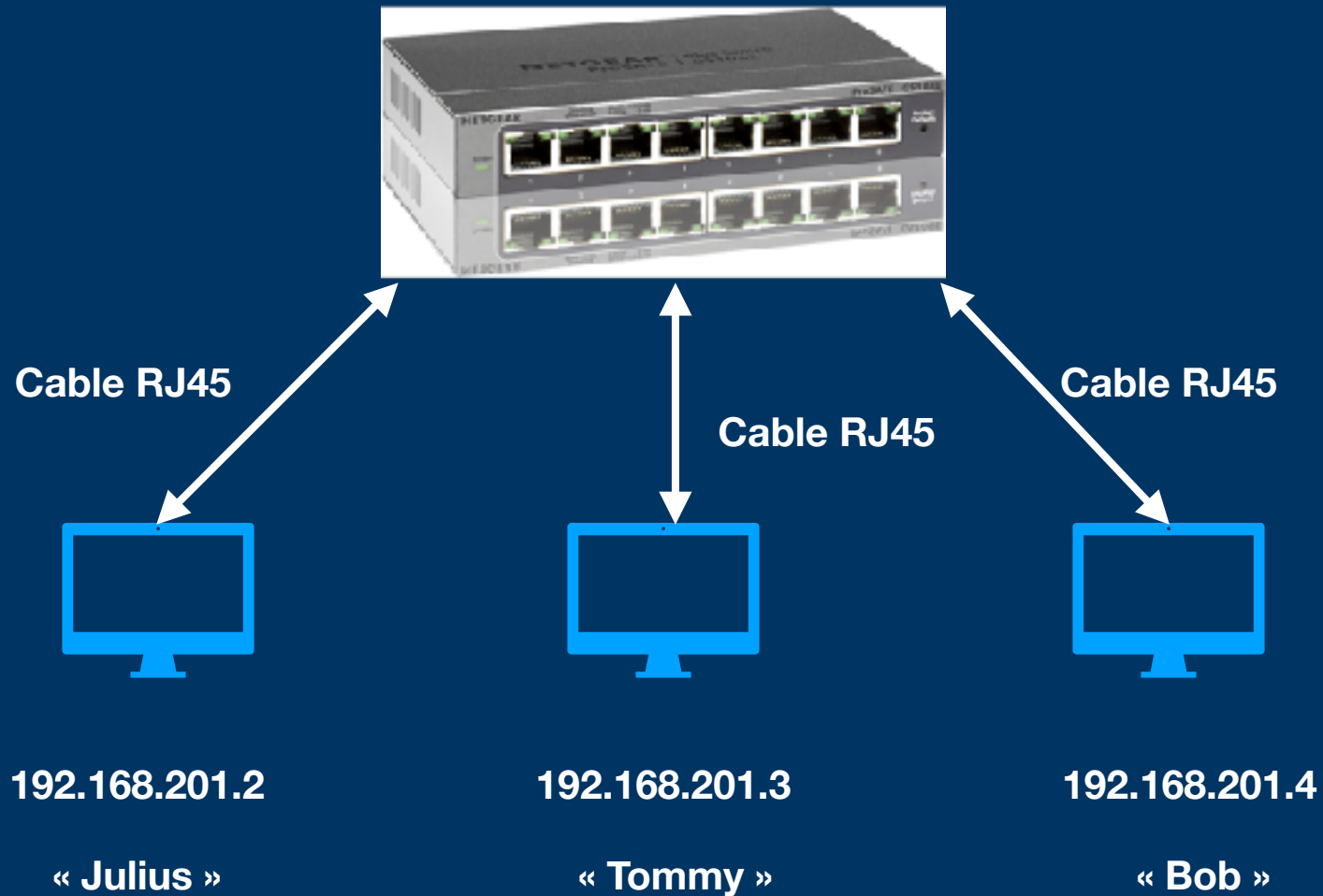
```
$ip route
```

```
default via 192.168.201.1 dev enp3s0 proto static metric 100
```

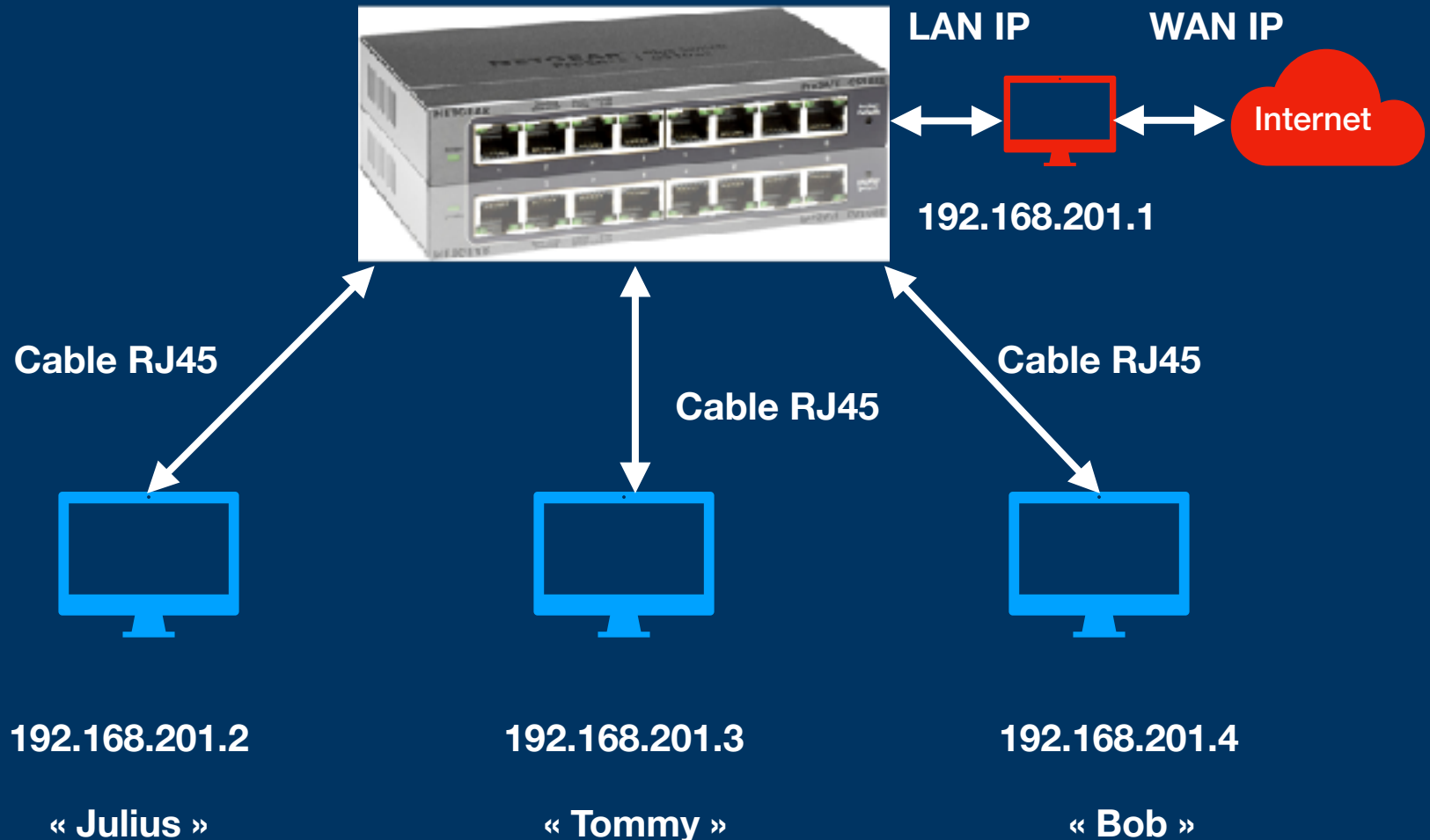
```
192.168.201.0/24 dev enp3s0 proto kernel scope link src 192.168.201.42 metric 100
```

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

Couche Physique



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.

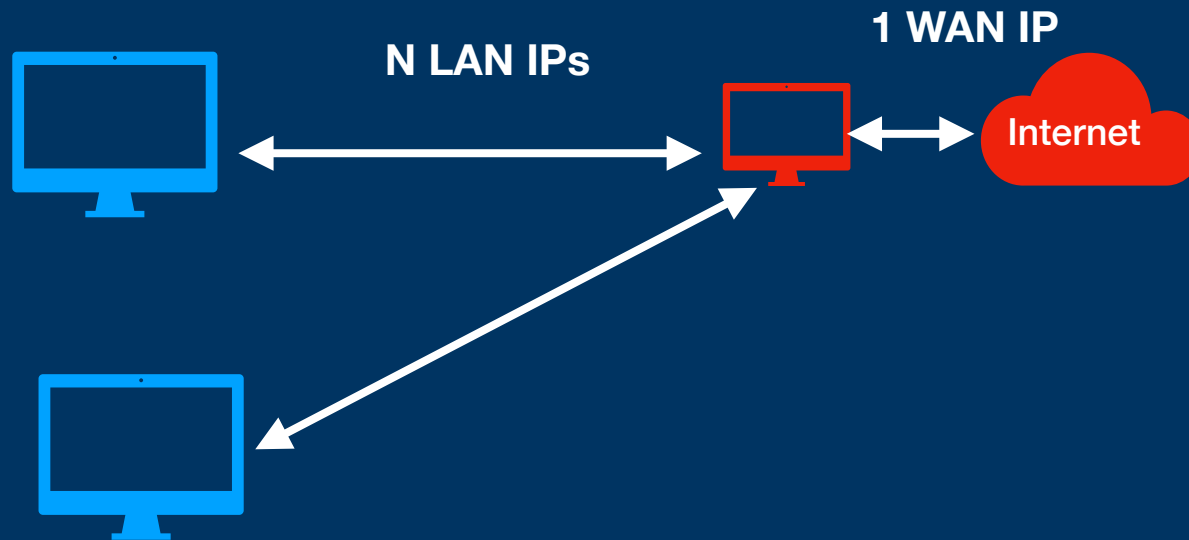


La Passerelle en Entreprise

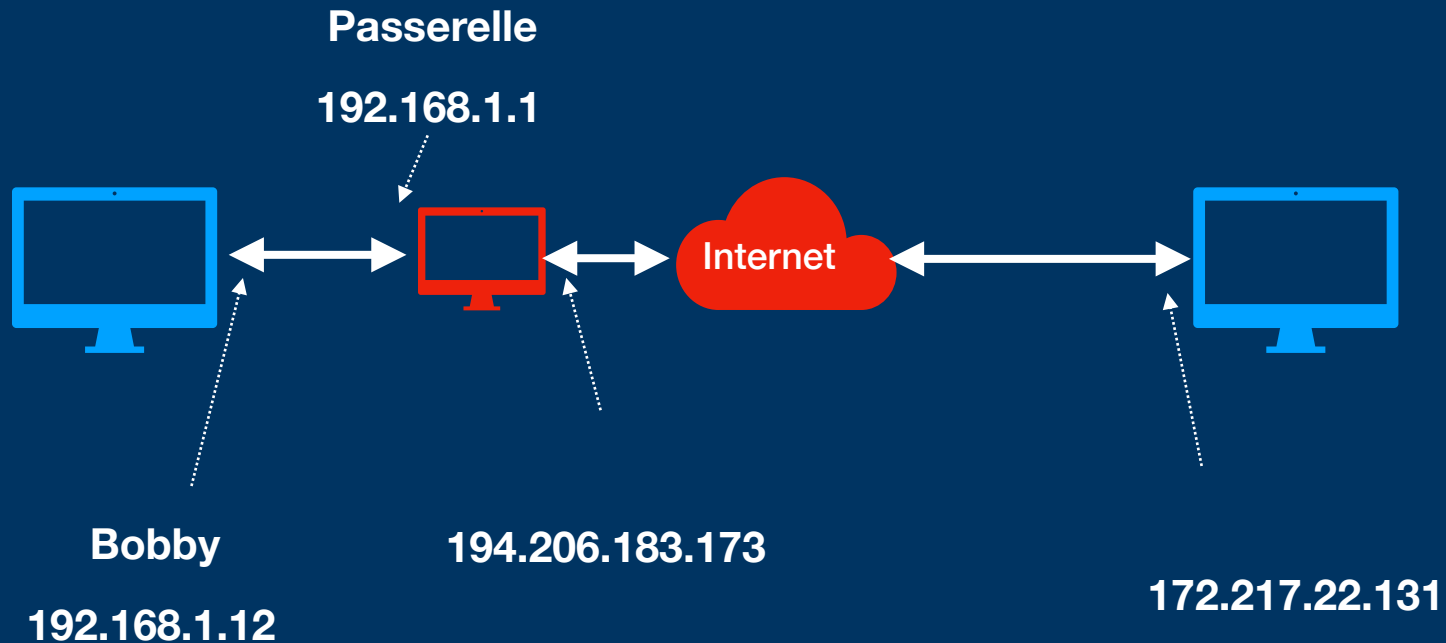


Exemples CISCO de routeurs configurables.

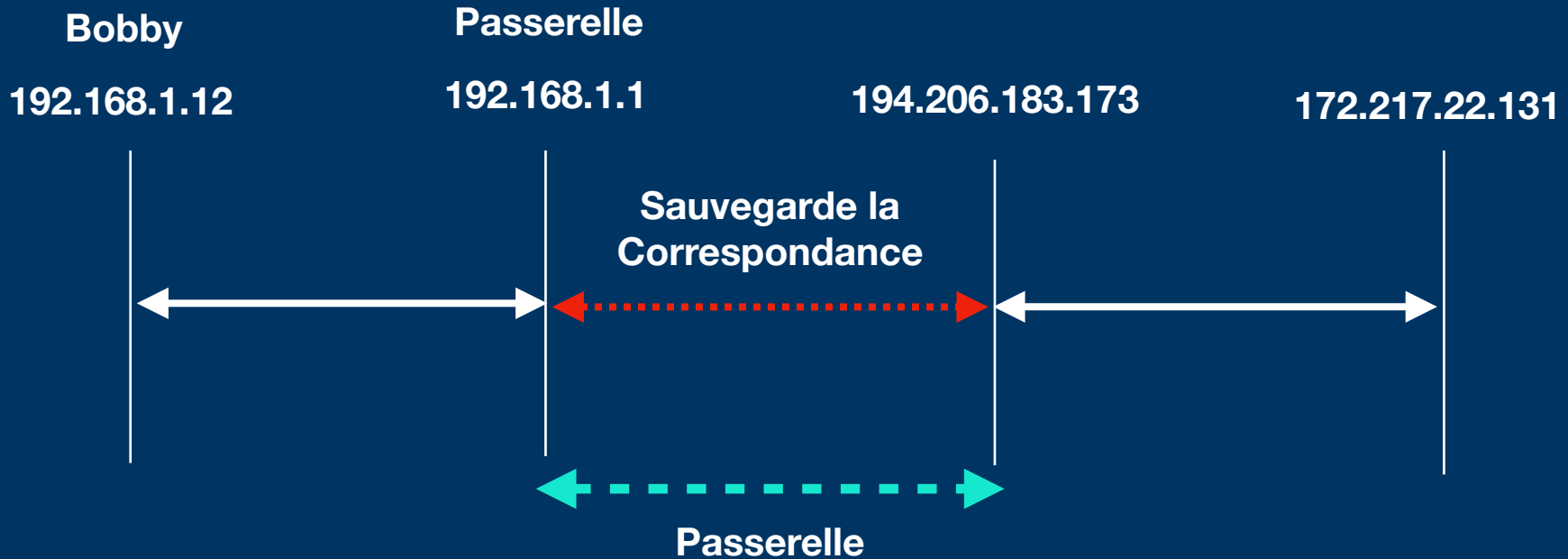
Network Address Translation



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



Services Communs

Protocol	Port	Name	Description
FTP	tcp/20, tcp/21	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

Résolution de Noms

Serveur DNS

Traduire un nom de domaine (Domain Name) en une adresse IP.

Par exemple:

```
$ dig google.com
```

```
; <<>> DiG 9.10.3-P4-Debian <<>> google.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 35279
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
;; EDNS: version: 0, flags::; udp: 512
;; QUESTION SECTION:
;google.com.                IN      A

;; ANSWER SECTION:
google.com.                 270     IN      A      216.58.204.110

;; Query time: 9 msec
;; SERVER: 192.168.201.127#53(192.168.201.127)
;; WHEN: Mon Feb 24 12:05:09 CET 2020
;; MSG SIZE rcvd: 55
```

Serveur DNS

On peut faire cette traduction en IPV6 (record AAAA)

Par exemple:

```
$ dig AAAA google.com
```

```
; <<>> DiG 9.10.3-P4-Debian <<>> AAAA google.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 44141
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
;; EDNS: version: 0, flags::; udp: 512
;; QUESTION SECTION:
;google.com.                IN      AAAA

;; ANSWER SECTION:
google.com.                 299     IN      AAAA    2a00:1450:4007:80f::200e

;; Query time: 10 msec
;; SERVER: 192.168.201.127#53(192.168.201.127)
;; WHEN: Mon Feb 24 12:05:39 CET 2020
;; MSG SIZE rcvd: 67
```

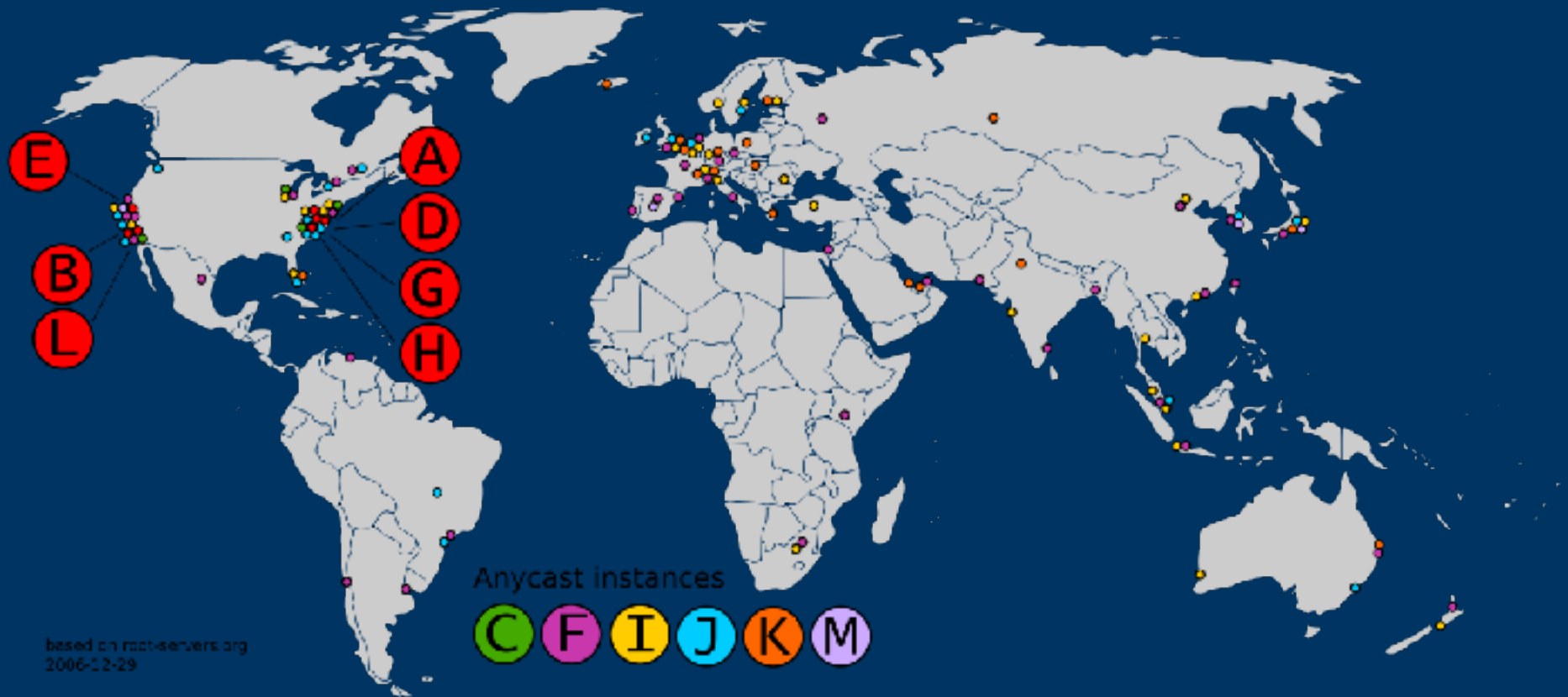
Serveur DNS

13 Serveur racine (source wikipedia)

Lettr e	adresse IPv4	adresse IPv6	Ancien nom	Société	Localisation	Logiciel
A10	198.41.0.4	2001:503:ba3e::2:30	ns.internic.net	VeriSign	trafic distribué par anycast	BIND
B11	199.9.14.201 Notes 1	2001:500:200::b	ns1.isi.edu	Université de Californie du Sud	Marina Del Rey, Californie, États-Unis	BIND
C13	192.33.4.1214	2001:500:2::c	c.psi.net	Cogent Communications	trafic distribué par anycast	BIND
D15	199.7.91.1316 Notes 2	2001:500:2d::d	terp.umd.edu	Université du Maryland	College Park, Maryland, États-Unis	BIND
E17	192.203.230.1017	2001:500:a8::e	ns.nasa.gov	NASA	Mountain View, Californie, États-Unis	BIND
F18	192.5.5.24119	2001:500:2f::f	ns.isc.org	Internet Systems Consortium	trafic distribué par anycast	BIND
G20	192.112.36.4 Notes 3	2001:500:12::d0d	ns.nic.ddn.mil	Defense Information Systems Agency	trafic distribué par anycast	BIND
H21	198.97.190.5322 Note s 4	2001:500:1::53	aos.arl.army.mil	United States Army Research Laboratory (en)	Aberdeen, Maryland, États-Unis	NSD
I23	192.36.148.1724	2001:7fe::53	nic.nordu.net	Autonomica (Netnod (en))	trafic distribué par anycast	BIND
J25	192.58.128.3025 Note s 5	2001:503:c27::2:30		VeriSign	trafic distribué par anycast	BIND
K27	193.0.14.12927	2001:7fd::1		RIPE NCC	trafic distribué par anycast	BIND , NSD27
L29	199.7.83.42 Notes 6	2001:500:3::42		ICANN	trafic distribué par anycast	NSD29
M31	202.12.27.3332	2001:dc3::35		WIDE Project (en)	trafic distribué par anycast	BIND

Serveur DNS

13 Serveur racine (source wikipedia)



gethostbyname

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

The hostent structure is defined in <netdb.h> as follows:

```
struct hostent {  
    char *h_name;          /* official name of host */  
    char **h_aliases;      /* alias list */  
    int  h_addrtype;       /* host address type */  
    int  h_length;         /* length of address */  
    char **h_addr_list;    /* list of addresses */  
}
```

```
#define h_addr h_addr_list[0] /* for backward compatibility */
```

The members of the hostent structure are:

h_name The official name of the host.

h_aliases

An array of alternative names for the host, terminated by a null pointer.

h_addrtype

The type of address; always AF_INET or AF_INET6 at present.

h_length

The length of the address in bytes.

h_addr_list

An array of pointers to network addresses for the host (in network byte order), terminated by a null pointer.

h_addr The first address in h_addr_list for backward compatibility.

gethostbyname

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

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

    struct hostent *ret = gethostbyname(argv[1]);

    if(!ret)
    {
        perror("gethostbyname");
        return 1;
    }

    printf("Host resolves to %s\n", ret->h_name);

    unsigned int i=0;
    while ( ret->h_addr_list[i] != NULL) {
        printf( "%s\n", inet_ntoa( *( struct in_addr*) ( ret->h_addr_list[i])));
        i++;
    }

    return 0;
}
```

gethostbyname

```
$ ./a.out cnn.com
```

```
Host resolves to cnn.com
```

```
151.101.65.67
```

```
151.101.1.67
```

```
151.101.193.67
```

```
151.101.129.67
```

```
$ ./a.out elysee.fr
```

```
Host resolves to elysee.fr
```

```
45.60.151.214
```

```
45.60.155.214
```

```
$ ./a.out facebook.com
```

```
Host resolves to facebook.com
```

```
157.240.21.35
```

```
$ ./a.out www.uvsq.fr
```

```
Host resolves to preprod.uvsq.fr
```

```
193.51.33.8
```

UNIQUEMENT IPV4

getaddrinfo

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

```
int getaddrinfo(const char *node,    // Domaine par exemple google.com
                const char *service, // Port ou service (80 ou www par exemple)
                const struct addrinfo *hints, // Hints souvent un memset 0 de la struct
                struct addrinfo **res); // Valeur de retour
```

```
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
};
```

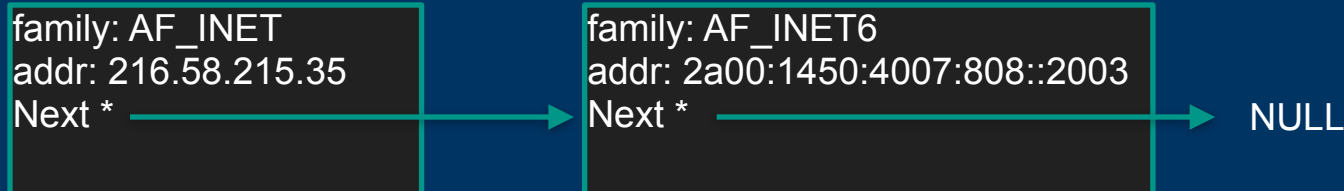
Convertir adresse DNS google.fr en:

IPV4 : 172.217.22.131

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

getaddrinfo

```
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 : 216.58.215.35

IPV6 : 2a00:1450:4007:808::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
```

getaddrinfo

```
$ ./getaddr cnn.com
```

```
IPV4 : 151.101.193.67
```

```
IPV4 : 151.101.129.67
```

```
IPV4 : 151.101.1.67
```

```
IPV4 : 151.101.65.67
```

```
IPV6 : 2a04:4e42:600::323
```

```
IPV6 : 2a04:4e42:400::323
```

```
IPV6 : 2a04:4e42::323
```

```
IPV6 : 2a04:4e42:200::323
```

```
$ ./getaddr google.com
```

```
IPV4 : 216.58.209.238
```

```
IPV6 : 2a00:1450:4007:812::200e
```

```
$ ./getaddr free.fr
```

```
IPV4 : 212.27.48.10
```

```
IPV6 : 2a01:e0c:1::1
```

```
$ ./getaddr facebook.com
```

```
IPV4 : 157.240.21.35
```

```
IPV6 : 2a03:2880:f130:83:face:b00c:0:25de
```

Créer un Socket

Créer un Socket TCP

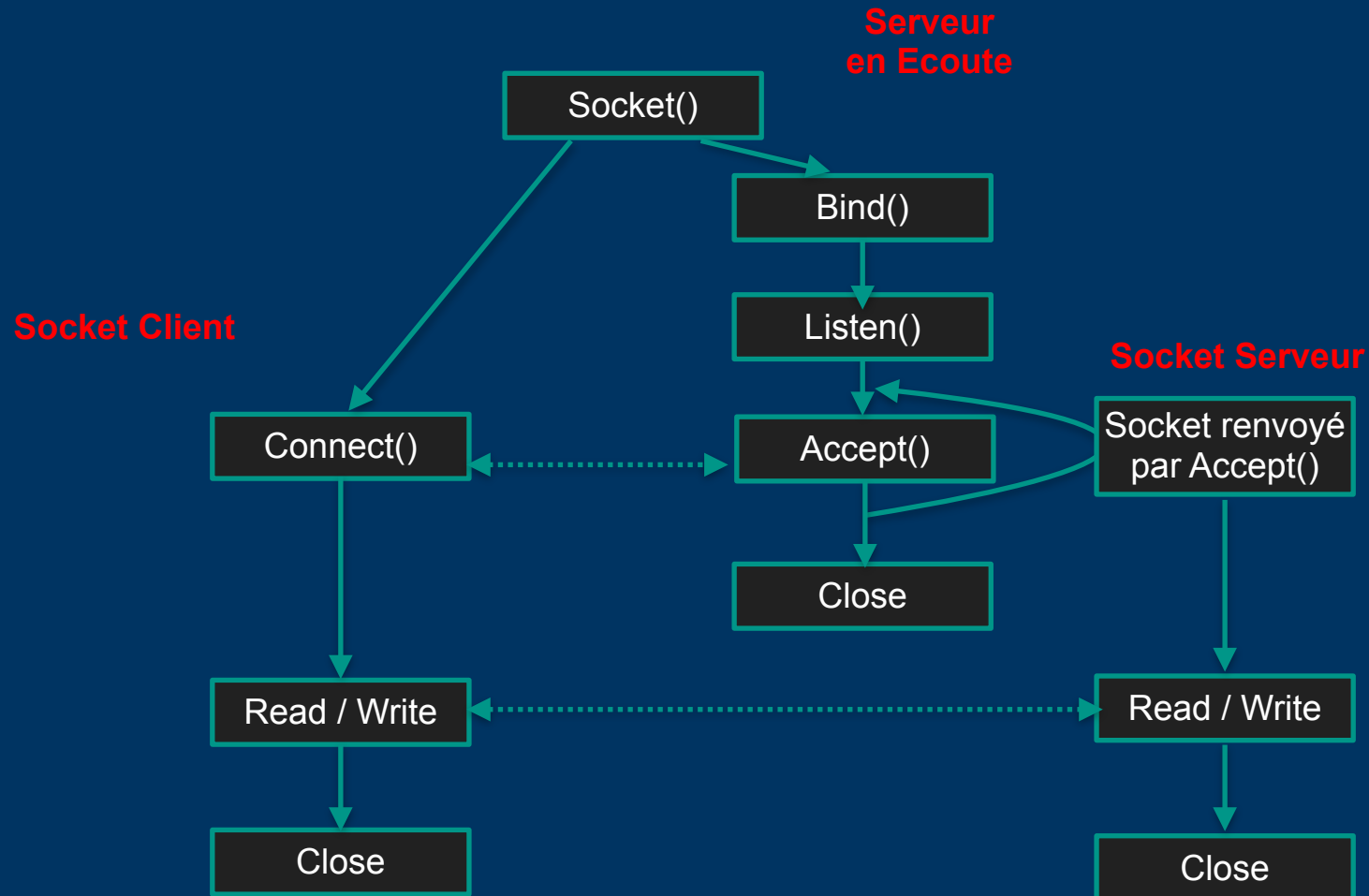
Un socket est un descripteur de fichier correspondant à un flux réseau

```
#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_RAW -> accès bas niveau à l'interface (uniquement root)
- Protocol -> Protocole à utiliser
 - ➔ En général on laisse 0

Etats d'un Socket



Connecter un Socket

Connect

NAME

connect – initiate a connection on a socket

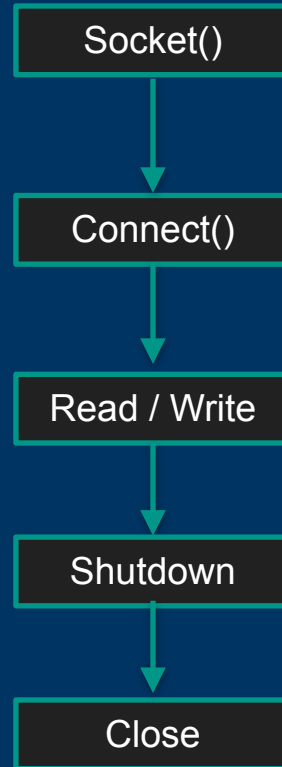
SYNOPSIS

```
#include <sys/types.h>           /* See NOTES */
#include <sys/socket.h>

int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
```

Astuce : utiliser les valeurs de retour de gethostbyname !

Socket Client



Connect via Gethostbyname

```
/* Argument check */
if ( argc != 3 )
{
    return 1;
}

/* Name resolution */
struct hostent *server_info = gethostbyname( argv[1] );

if ( !server_info )
{
    perror( "gethostbyname" );
    return 1;
}

/* Create Socket */
int sock = socket( AF_INET, SOCK_STREAM, 0 );

if ( sock < 0 )
{
    perror( "socket" );
    return 1;
}

/* Configure client socket */
struct sockaddr_in server_conf;
/* Insert address family */
server_conf.sin_family = server_info->h_addrtype;
/* Insert PORT */
server_conf.sin_port = htons( atoi( argv[2] ) );
/* Copy destination addr from server_info */
memcpy( &server_conf.sin_addr, server_info->h_addr_list[0], server_info->h_length );

/* Connect the socket to the server */
if ( connect( sock, ( struct sockaddr * )&server_conf, sizeof( struct sockaddr_in ) ) < 0 )
{
    perror( "connect" );
    return 1;
}
```

Connect via getaddrinfo

```
#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;
}
```

Créer un Serveur (Socket en écoute)

Bind & Listen

Attacher le socket à un port donné:

```
#include <sys/types.h>           /* See NOTES */
#include <sys/socket.h>

int bind(int sockfd, const struct sockaddr *addr,
         socklen_t addrlen);
```

Se mettre en attente de connexions avec Listen:

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

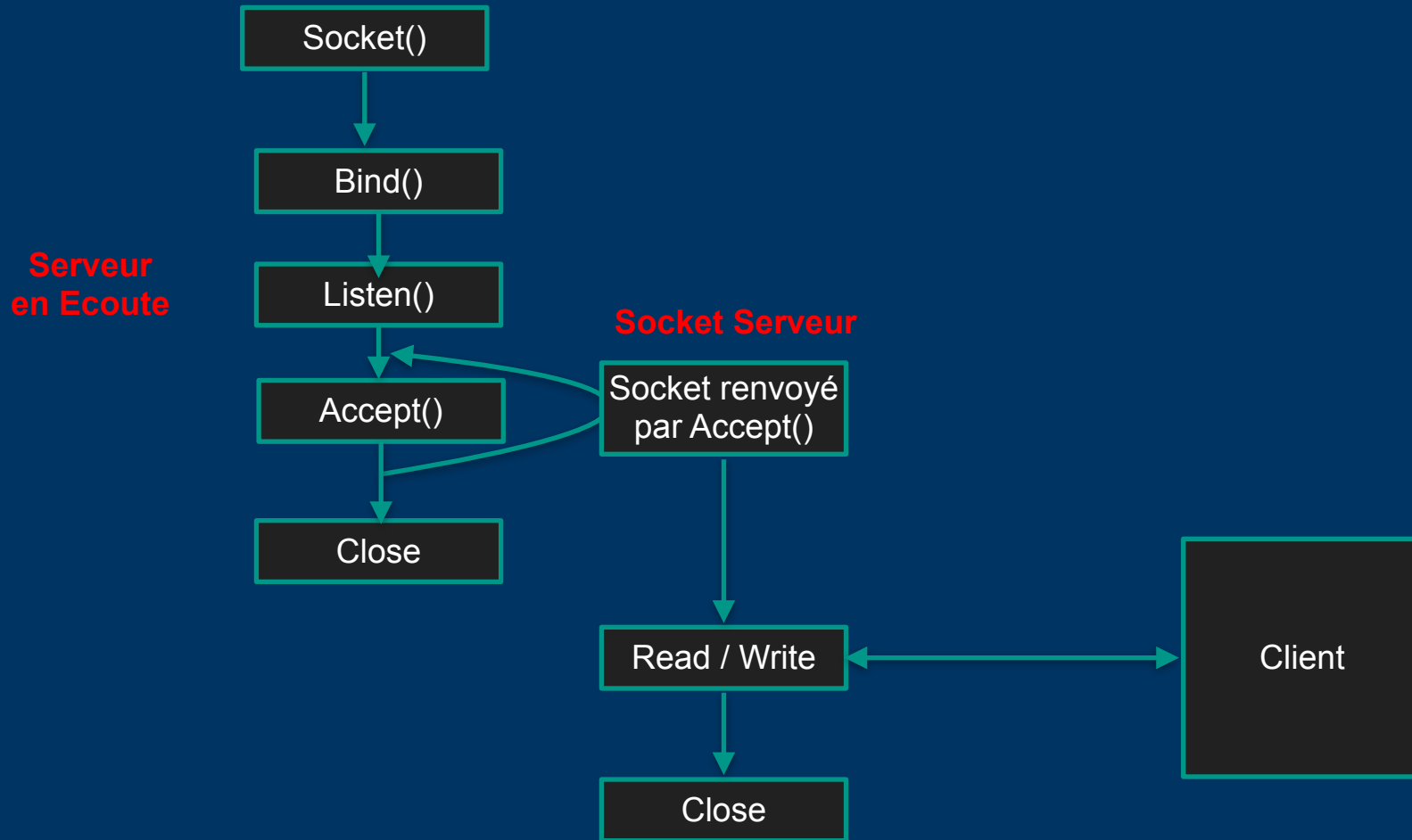
int listen(int sockfd, int backlog);
```

Accepter une connexion entrante:

```
#include <sys/types.h>           /* See NOTES */
#include <sys/socket.h>

int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
```


Etats d'un Socket



Avec Getaddrinfo

```
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;
```

```
int ret = getaddrinfo(NULL, PORT, &hints, &res);
```

PORT : est un string avec soit, un numéro ou bien un descripteur de service par exemple « www ».

Comme pour connect, cet appel prépare pour différentes configuration les paramètre à bind.

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;
        }
        binded = 1;
    }
}
```

```
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);

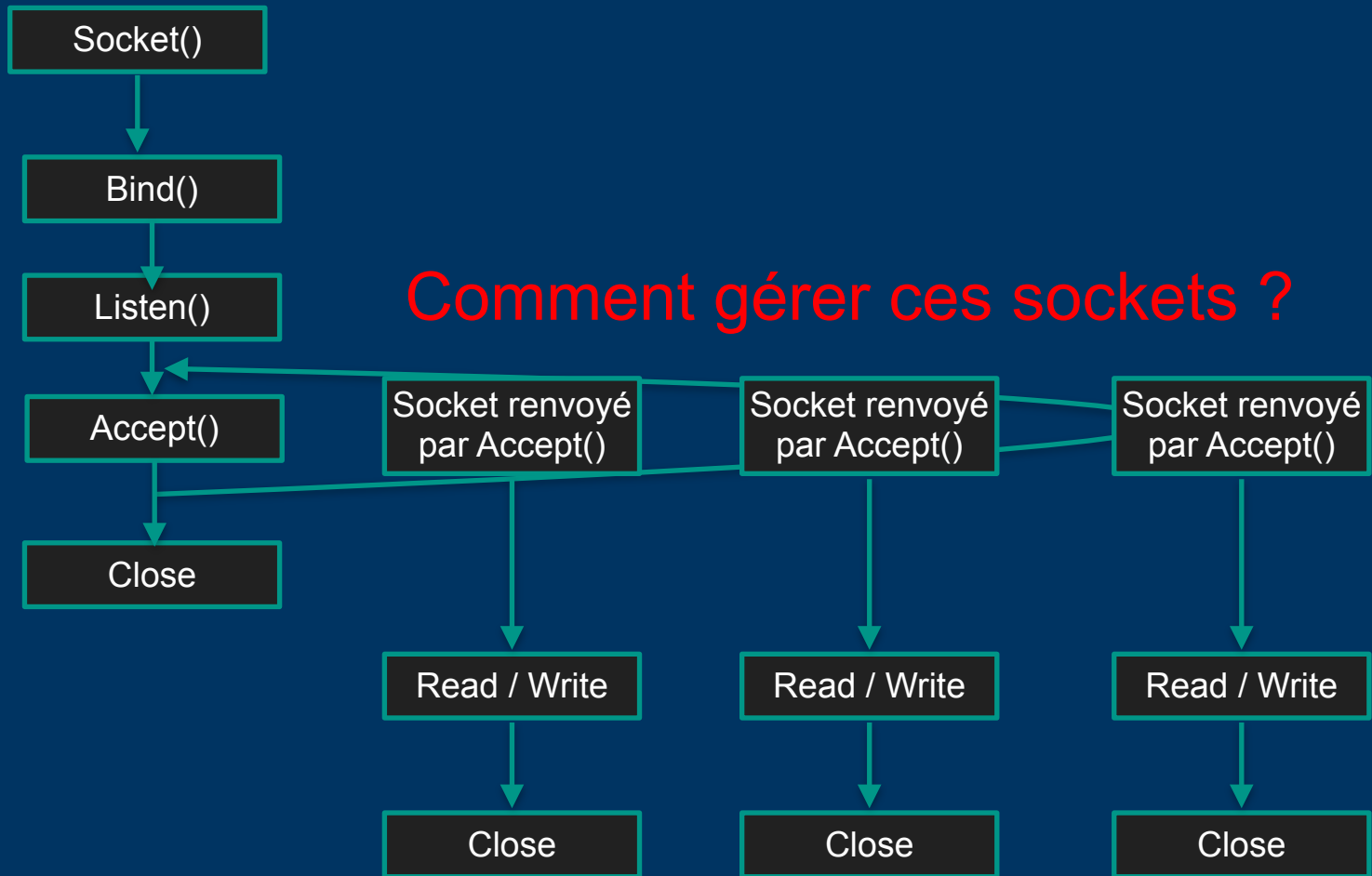
close(listen_sock);

return 0;
}
```

Parallélisme de Serveur

Serveur Multi-Clients

Serveur
en Ecoute



Serveur qui « Fork »

```
while ( 1 )
{
    /* On accepte un client et on récupère un nouveau FD */
    int client_socket = accept( listen_sock, &client_info, &addr_len );
    fprintf( stderr, "After accept\n" );

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

    pid_t c = fork();

    if ( !c )
    {
        int j;

        for ( j = 0 ; j < 128 ; j++ )
        {
            char message[128];
            snprintf( message, 128, "Salut %d\n", j );
            /* On salut a celui qui s'est connecté */
            write( client_socket, message, strlen( message ) );
            sleep( 1 );
        }

        fprintf( stderr, "Closing client socket\n" );
        /* On se déconnecte du client */
        close( client_socket );
    }
    else
    {
        close( client_socket );
    }
}
```

Serveur multi-thread

```
while ( 1 )
{
    /* On accepte un client et on récupère un nouveau FD */
    int client_socket = accept( listen_sock, &client_info, &addr_len );
    fprintf( stderr, "After accept\n" );

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

    struct client_infos *infos = ( struct client_infos * )malloc( sizeof( struct client_infos ) );

    infos->client_socket = client_socket;

    pthread_t th;

    pthread_create( &th, NULL, client_loop, ( void * ) infos );
}
```


Appel Select

```
/* According to POSIX.1-2001, POSIX.1-2008 */  
#include <sys/select.h>
```

```
/* According to earlier standards */  
#include <sys/time.h>  
#include <sys/types.h>  
#include <unistd.h>
```

```
int select(int nfd, /* Nombre de FDs */  
           fd_set *readfds, /* Sortie: FD prêts en lecture */  
           fd_set *writefds, /* Sortie: FD prêts en écriture */  
           fd_set *exceptfds, /* Sortie FD eb erreur */  
           struct timeval *timeout); /* Timeout optionnel */
```

Ensemble de FDs:

```
void FD_CLR(int fd, fd_set *set);  
int  FD_ISSET(int fd, fd_set *set);  
void FD_SET(int fd, fd_set *set);  
void FD_ZERO(fd_set *set);
```

Appel Select

Le but est de déléguer le blocage au noyau (au lieu de le faire en espace utilisateur). De plus on peut alors avoir une seule boucle répondant aux différents évènements aussi bien en lecture qu'en écriture.

Select manipule des ensemble de descripteurs de fichier:

Appel	Description
FD_ZERO	Initialise un set à 0
FD_SET	Ajoute au set
FD_CLR	Retire du set
FD_ISSET	Test de présence

L'assignation de sets est possible.

```
void fdsetprint(fd_set * set)
{
    int i;

    printf("-----\n");
    for( i = 0 ; i < FD_SETSIZE; i++)
    {
        if( FD_ISSET(i, set))
        {
            printf("%d is in set\n", i);
        }
    }
    printf("-----\n");
}

int main(int argc, char ** argv )
{
    fd_set set;

    FD_ZERO(&set);

    FD_SET(19, &set);
    FD_SET(20, &set);
    fdsetprint(&set);

    FD_CLR(19, &set);
    fdsetprint(&set);

    fd_set second = set;
    fdsetprint(&second);

    return 0;
}
```

Serveur multiplexé

```
fd_set active_fd_set, read_fd_set;
/* Initialize the set of active sockets. */
FD_ZERO ( &active_fd_set );
FD_SET ( listen_sock, &active_fd_set );

while ( 1 )
{
    /* Block until input arrives on one or more active sockets. */
    read_fd_set = active_fd_set;

    if ( select ( FD_SETSIZE, &read_fd_set, NULL, NULL, NULL ) < 0 )
    {
        perror ( "select" );
        exit ( EXIT_FAILURE );
    }

    int i;

    /* Service all the sockets with input pending. */
    for ( i = 0; i < FD_SETSIZE; ++i )
        if ( FD_ISSET ( i, &read_fd_set ) )
        {
            if ( i == listen_sock )
            {
                /* Event was on the listen socket */
                struct sockaddr_in client_info;
                /* Connection request on original socket. */
                int new;
                unsigned int addr_size = sizeof ( struct sockaddr_in );
                new = accept ( listen_sock, ( struct sockaddr * ) &client_info, &addr_size );

                if ( new < 0 )
                {
                    perror ( "accept" );
                    exit ( EXIT_FAILURE );
                }

                fprintf ( stderr,
                    "Server: connect from host %s, port %d\n",
                    inet_ntoa ( client_info.sin_addr ),
                    ntohs ( client_info.sin_port ) );
                FD_SET ( new, &active_fd_set );
            }
            else
            {
                /* Data arriving on an already-connected socket. */
                if ( read_from_client ( i ) < 0 )
                {
                    fprintf ( stderr, "Client left\n" );
                    close ( i );
                    FD_CLR ( i, &active_fd_set );
                }
            }
        }
    }
}
```

Socket UNIX

Socket UNIX

- Classiquement un socket est identifié par un couple

HOST:PORT

- Cependant, il existe des sockets associés à un fichier ils sont donc locaux à un système et soumis aux droits standards sur les fichiers.

/tmp/my.sock (par exemple)

```

/* UNIX socket descriptor */
struct sockaddr_un addr;
/* Clear it */
memset( &addr, 0, sizeof( addr ) );
/* Set family to UNIX */
addr.sun_family = AF_UNIX
/* Set socket PATH */;
strcpy( addr.sun_path, argv[1], sizeof( addr.sun_path ) - 1 );
/* Create socket FD */
int listen_socket = socket( AF_UNIX, SOCK_STREAM, 0 );

if ( listen_socket < 0 )
{
    perror( "socket" );
    return 1;
}

/* BIND the socket to UNIX socket */
int ret = bind( listen_socket, ( struct sockaddr * )&addr, sizeof( addr ) );

if ( ret < 0 )
{
    perror( "bind" );
    fprintf( stderr, "Failed to bind on 0.0.0.0:%s\n", argv[1] );
    return 1;
}

/* On commence à écouter */
ret = listen( listen_socket, 7 );

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

```

Socket UNIX

Socket UNIX

```
$ ./server /tmp/test.sock
```

ATTENTION: toutes les version ne netcat ne supportent pas les sockets UNIX il vous faut une version qui comprend le flag -U ici on propose d'utiliser Socat.

```
$ socat - UNIX-CONNECT:/tmp/test.sock  
SALUT!
```

Protocole de Base

HTTP

Requête GET

Operation

Page demandée

Standard HTTP



GET / HTTP/1.1

Host: deneb.france.paratools.com:8080

Connection: keep-alive

Upgrade-Insecure-Requests: 1

User-Agent: Mozilla/5.0

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9

Accept-Encoding: gzip, deflate

Accept-Language: en-US,en;q=0.9,fr-FR;q=0.8,fr;q=0.7

Cookie: doxygen_width=300

(OPTIONNEL)

Réponse Serveur

Operation

Page demandée

Standard HTTP

GET / HTTP/1.1

Ligne VIDE

Réponse OK

Type de contenu

Longueur du contenu

Ligne VIDE

Contenu

HTTP/1.0 200 OK

Content-Type: text/html; charset=ISO-8859-1

Content-Length: 15

<h1>Hello</h1>

```

void *client_loop( void *param )
{
    struct client_infos *info = ( struct client_infos * )param;

    char buffer[32768];

    FILE* socket = fdopen( info->client_socket, "rw" );

    if( !socket )
    {
        perror( "fdopen" );
        exit( 1 );
    }

    while( fgets( buffer, 32768, socket ) )
    {
        if( strlen( buffer ) < 6 )
        {
            continue;
        }

        char *http = strstr( buffer, " HTTP" );

        if( http )
        {
            *http = '\0';
        }

        if( buffer[0] == 'G' && buffer[4] == ' ' && buffer[8] == 'T' )
        {
            char * file_path = &buffer[4];

            printf( "GET == '%s'\n", file_path );

            if( *file_path == '/' && strlen( file_path ) == 1 )
            {
                file_path = "index.html";
            }
            else
            {
                /* Skip / */
                file_path++;
            }

            ssize_t content_size = get_file_size( file_path );

            if( 0 < content_size )
            {
                write_http_header( content_size, 200, "text/html", info->client_socket );
                sendfile( file_path, content_size, info->client_socket );
                break;
            }
            else
            {
                write_http_header( 0, 404, "text/html", info->client_socket );
            }
        }
    }

    fprintf( stderr, "Closing client socket\n" );
    /* On se déconnecte du client */
    fclose( socket );
}

```

Serveur HTTP Ultra-Basique

Généralités sur les IPC System V

Les IPC System V

Apparus dans Unix en 1983 ils permettent des communication inter-inter-processus (Inter-Process Communications, IPC)

- Files de messages
- Segment de mémoire partagée
- Sémaphores

Le noyau est chargé de la gestion des ressources associées via des commandes

Files de Messages



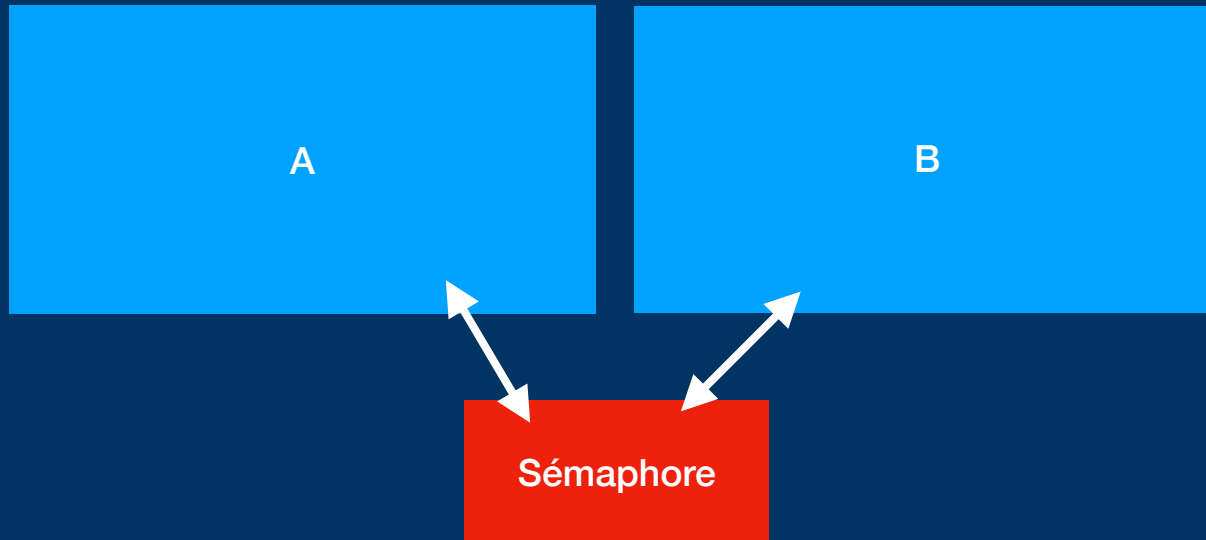
- *ftok*: **génération d'une clef IPC**
- *msgget*: **Récupère un identificateur de file de message**
- *msgrecv*: **Réception d'un message depuis une file**
- *msgsend*: **Envoi d'un message dans une file**
- *msgctl*: **Contrôle de la file de messages**

Segment de mémoire partagée



- *ftok*: génération d'une clef IPC
- *shmget*: Récupère un identificateur de segment shm
- *shmat*: Projection d'un segment SHM
- *shmdt*: Suppression d'un segment she
- *shmctl*: Contrôle du segment SHM

Sémaphore IPC



- *ftok*: **génération d'une clef IPC**
- *semget*: **Récupère un identificateur de sémaphore**
- *semop*: **Fait une opération sur le sémaphore**
- *semctl*: **Contrôle du sémaphore**

Les IPC System V

\$ ipcs

----- Files de messages -----

clef	msqid	propriétaire	perms	octets utilisés	messages
------	-------	--------------	-------	-----------------	----------

----- Segment de mémoire partagée -----

clef	shmid	propriétaire	perms	octets	nattch	états
0x00000000	42729472	jbbesnard	600	1048576	2	dest
0x00000000	39616513	jbbesnard	600	524288	2	dest

----- Tableaux de sémaphores -----

clef	semid	propriétaire	perms	nsems
------	-------	--------------	-------	-------

Les IPC System V

```
$ ipcrm -h
```

Utilisation :

```
ipcrm [options]  
ipcrm shm|msg|sem <id> ...
```

Supprimer certaines ressources IPC.

Options :

```
-m, --shm-id <ident.>    retirer le segment de mémoire partagée par ident.  
-M, --shm-key <clef>     retirer le segment de mémoire partagée par clef  
-q, --queue-id <ident.>  retirer la file de messages par identifiant  
-Q, --queue-key <clef>   retirer la file de messages par clef  
-s, --sem-id <id.>       retirer le sémaphore par identifiant  
-S, --sem-key <clef>     retirer le sémaphore par clef  
-a, --all[=shm|msg|sem]  tout retirer (dans la catégorie indiquée)  
-v, --verbose            expliquer les actions en cours  
  
-h, --help              afficher cette aide et quitter  
-V, --version            afficher les informations de version et quitter
```

Consultez `ipcrm(1)` pour obtenir des précisions complémentaires.

Les IPC System V

```
$ ipcmk -h
```

Utilisation :

```
ipcmk [options]
```

Créer diverses ressources IPC.

Options :

-M, --shmem <taille>	créer un segment de mémoire partagée de taille <taille>
-S, --semaphore <nsems>	créer un tableau de sémaphores à <nsems> éléments
-Q, --queue	créer une file de messages
-p, --mode <mode>	droits de la ressource (0644 par défaut)
-h, --help	afficher cette aide et quitter
-V, --version	afficher les informations de version et quitter

Consultez `ipcmk(1)` pour obtenir des précisions complémentaires.

Resources

Les ressources IPC sont indépendante des processus

- Il est possible de laisser des scories si l'on ne fait pas attention
- Un processus peut se « rater » à un segment lors de son redémarrage par exemple
- Les processus partagent des segments avec un mécanisme de clef qui est un secret « a priori » pour la sécurité

Clefs pour les IPCs System V

La Clef

Un IPC (de tout type) est partagé par une clef:

- C'est un entier qui doit être le même entre tous les processus partageant la resource;
- On peut la connaître a priori avec risque de conflit (un peu comme un port TCP);
- Une clef spéciale `IPC_PRIVATE` crée une file limitée à un processus et l'ensemble de ses descendants;
- On peut la créer avec une fonction « `ftok` » qui repose sur un fichier et un nom de projet.

Ftok

SYNOPSIS

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

```
key_t ftok(const char *pathname, int proj_id);
```

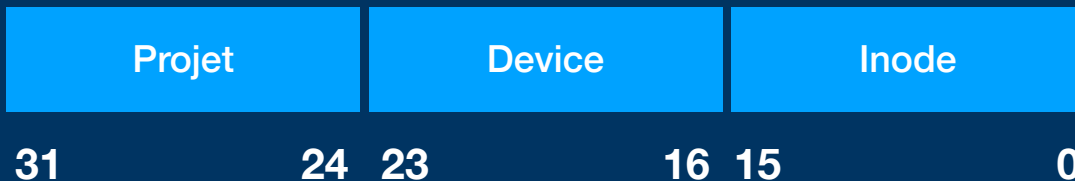
DESCRIPTION

The `ftok()` function uses the identity of the file named by the given `pathname` (which must refer to an existing, accessible file) and the least significant 8 bits of `proj_id` (which must be nonzero) to generate a `key_t` type System V IPC key, suitable for use with `msgget(2)`, `semget(2)`, or `shmget(2)`.

The resulting value is the same for all `pathnames` that name the same file, when the same value of `proj_id` is used. The value returned should be different when the (simultaneously existing) files or the project IDs differ.

RETURN VALUE

On success, the generated `key_t` value is returned. On failure `-1` is returned, with `errno` indicating the error as for the `stat(2)` system call.



Création / Récupération de ressources

Une fois que l'on a une clef de type *key_t* on peut retrouver/créer une resource:

- File de message : *msgget*
- Segment de mémoire partagée: *shmget*
- Sémaphore: *semget*

Les Files de Messages IPC SYSTEM V

Files de Messages pour une Communication entre Processus sur un Même Noeud.

Le message sera toujours de la forme:

```
Struct XXX {  
    long id; // Toujours > 0 !  
    ... DATA ...  
    // Taille max sans le long MSGMAX (8192 Octets)  
};
```

Lors de l'envoi et de la réception d'un message la taille et TOUJOURS sans le long qui définit le type de message. Cette même valeur (ici id) doit TOUJOURS être supérieure à 0.

En pratique on crée une struct statique sur la pile car l'allocation d'un objet avec piggybacking demande plus de code.

Créer Une File de Messages

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
```

```
int msgget(key_t key, int msgflg);
```

- Key : Une clef, soit manuelle, soit via ftok ou bien IPC_PRIVATE
- msgflg: mode de création de la file et ses droits UNIX
 - ➡ IPC_CREAT crée une file s'il y en a aucune associée à cette clef
 - ➡ IPC_EXCL échoue s'il existe déjà une file sur la clef indiqué (toujours combiné avec IPC_CREAT!)
 - ➡ 0600 droit UNIX en octal (important car si omis 0000 et la file est moins pratique !)

Créer Une File de Messages

- Créer une file pour un processus et ses fils

➡ `file = msgget(IPC_PRIVATE, 0600);`

- Créer une file pour accéder à une file potentiellement existante:

➡ `file = msgget(key , IPC_CREAT | 0600);`

- Pour être sûr de créer une nouvelle file en lecture écriture pour soi et en lecture seule pour les autres utilisateurs:

➡ `file = msgget(key, IPC_CREAT | IPC_EXCL | 0622);`

- Utiliser uniquement une file existante précédemment créée par un serveur:

➡ `file = msgget(key, 0);`

Envoyer un Message

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
```

```
int msgsnd(int msqid, const void *msgp, size_t msgsz, int msgflg);
```

- msqid : file de message à utiliser, créée avec msgget
- msgp : pointeur vers les données à envoyer (comprend forcément un long qui est l'ID de message)
- size : taille du message **SANS** le long qui est l'ID du message
- msgflg: mode d'envoi du message
 - ➡ IPC_NOWAIT ne pas bloquer si la file est pleine (renvoie EAGAIN dans errno)
 - ➡ 0 en général

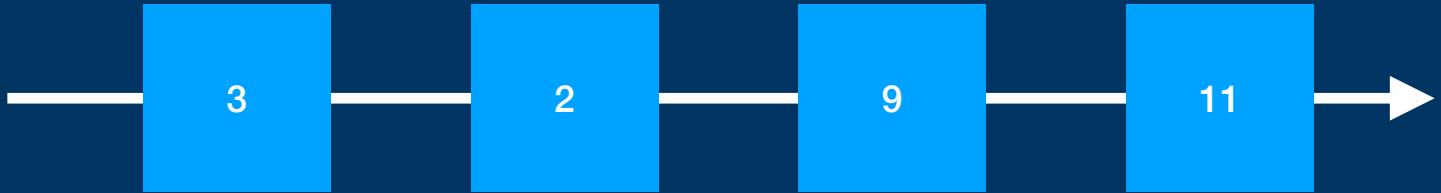
Recevoir un Message

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
```

```
ssize_t msgrcv(int msqid,
               void *msgp, size_t msgsz, long msgtyp, int msgflg);
```

- msqid : file de message à utiliser, créée avec msgget
- msgp : pointeur vers les données à envoyer (comprend forcément un long qui est l'ID de message)
- size : taille du message **SANS** le long qui est l'ID du message
- msgtyp : type de message à recevoir:
 - ➡ 0 : prochain message de la file
 - ➡ 0 < TYP prochain message avec l'ID donné
 - ➡ TYP < 0 prochain message avec un ID inférieur ou égal à TYP, utilisé pour gérer des priorités de messages
- msgflg: mode de réception du message:
 - ➡ IPC_NOWAIT ne pas bloquer si pas de message du TYP donné (renvoie ENOMSG dans errno)
 - ➡ MSG_EXCEPT renvoie un message d'un TYP différent de celui donné (seulement pour TYP > 0)
 - ➡ MSG_NO_ERROR permettre au message d'être tronqué à la réception (à la différence du comportement de base)

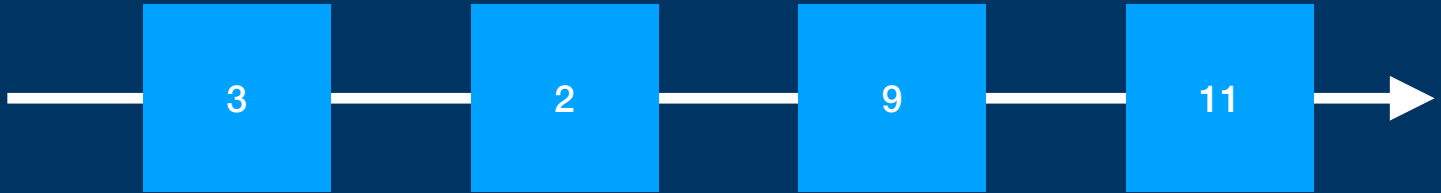
Recevoir un Message



```
msgrcv(file, &msg, sizeof(int), 2, 0);
```

Quel message ??

Recevoir un Message

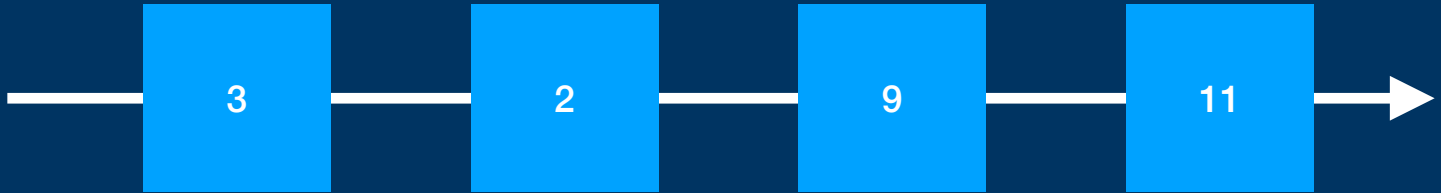


```
msgrcv(file, &msg, sizeof(int), 2, 0);
```

Quel message ??

2

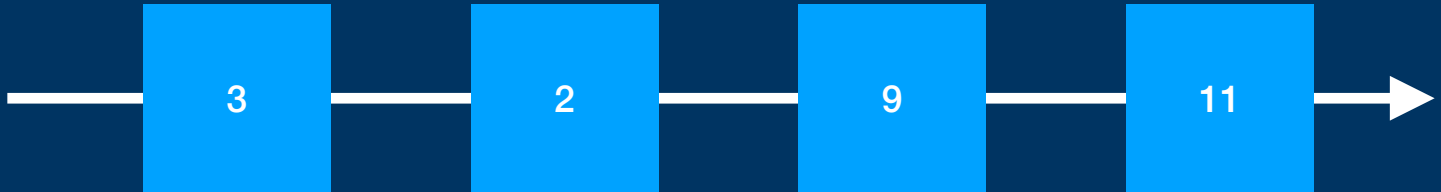
Recevoir un Message



```
msgrcv(file, &msg, sizeof(int), -10, 0);
```

Quel message ??

Recevoir un Message

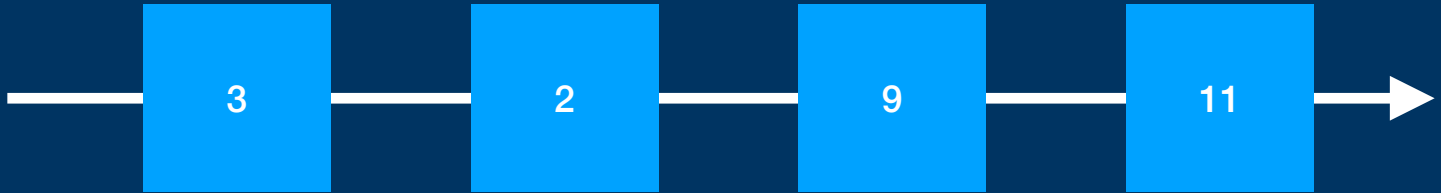


```
msgrcv(file, &msg, sizeof(int), -10, 0);
```

Quel message ??

9

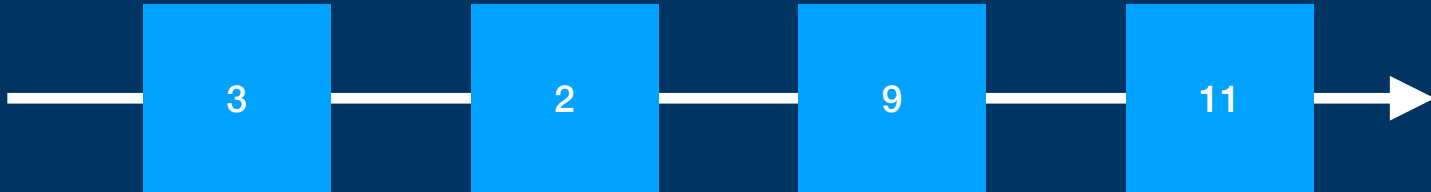
Recevoir un Message



```
msgrcv(file, &msg, sizeof(int), 99, 0);
```

Quel message ??

Recevoir un Message

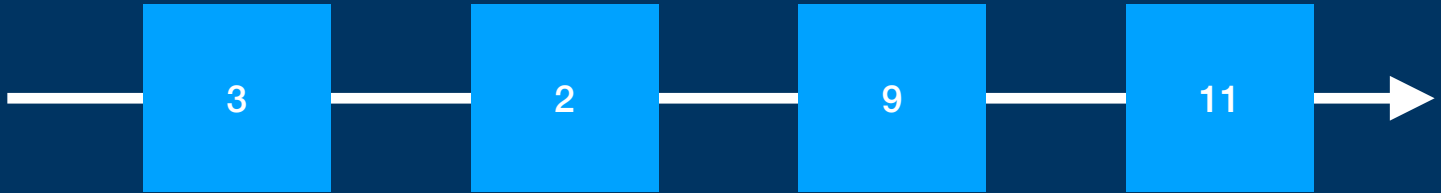


```
msgrcv(file, &msg, sizeof(int), 99, 0);
```

Quel message ??

L'appel reste bloqué indéfiniment si un message 99 n'est jamais posté.

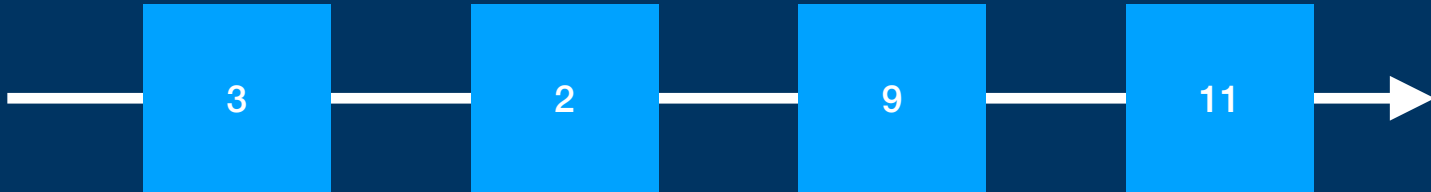
Recevoir un Message



```
msgrcv(file, &msg, sizeof(int), 99, IPC_NOWAIT);
```

Quel message ??

Recevoir un Message

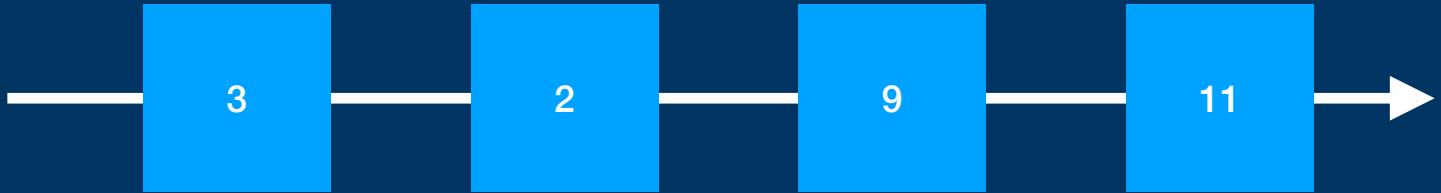


```
msgrcv(file, &msg, sizeof(int), 99, IPC_NOWAIT);
```

Quel message ??

L'appel renvoie -1 et met errno à ENOMSG

Recevoir un Message



```
msgrcv(file, &msg, sizeof(int), 11, MSG_EXCEPT);
```

Quel message ??

Recevoir un Message



```
msgrcv(file, &msg, sizeof(int), 11, MSG_EXCEPT);
```

Quel message ??

9

Contrôler une File

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
```

```
int msgctl(int msqid, int cmd, struct msqid_ds *buf);
```

- msqid : ID de la file à contrôler
- cmd: commande à appliquer à la file
 - ➡ IPC_STAT récupère les informations sur la file dans la *struct msqid_ds* (voir man)
 - ➡ IPC_SET permet de régler certains attributs en passant une *struct msqid_ds*
 - ➡ **IPC_RMID supprime la file toute les opérations courantes ou future échouent (avec la possibilité non gérée qu'une nouvelle file soit créée avec la même clef). La synchronisation et à la charge du programmeur.**
 - ➡ ... il existe d'autres flags voir man

PENSEZ à SUPPRIMER VOS FILES !!!

```

#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <unistd.h>
#include <sys/wait.h>

#include <sys/time.h>

double get_time(){
    struct timeval val;
    gettimeofday(&val, NULL);
    return (double)val.tv_sec + 1e-6 * val.tv_usec;
}

#define SIZE 16

struct msg_t{
    long type;
    int data[SIZE];
};

#define NUM_MSG 65536

int main( int argc, char ** argv ){
    int file = msgget(IPC_PRIVATE, IPC_CREAT | 0600);

    if( file < 0 ){
        perror("msgget");
        return 1;
    }

    int i;
    struct msg_t m;
    m.type = 1;

    int pid = fork();

    if( pid == 0 )
    {
        int stop = 0;

        while(!stop)
        {
            msgrcv(file, &m, SIZE*sizeof(int), 0, 0);
            /* Notify end */
            if( m.data[0] == 0 )
                stop = 1;
            m.type = 1;
            msgsnd(file, &m, SIZE*sizeof(int), 0);
        }
    }
}

```

```

else
{
    double total_time = 0.0;

    for( i = 2 ; i <= NUM_MSG ; i++)
    {
        m.data[0] = i;
        m.type = i;

        double start = get_time();
        int ret = msgsnd(file, &m, SIZE*sizeof(int), 0);

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

        double end = get_time();
        total_time += end - start;

        msgrcv(file, &m, SIZE*sizeof(int), 1, 0);
    }

    m.data[0] = 0;
    msgsnd(file, &m, SIZE*sizeof(int), 0);

    wait( NULL );

    msgctl( file, IPC_RMID, NULL);

    fprintf(stderr, "Pingpong takes %g usec Bandwidth is %g MB/s »
        total_time/NUM_MSG*1e6,
        (double)(SIZE*NUM_MSG*sizeof(int))/
            (total_time*1024.0*1024.0));

}

return 0;
}

```

Les Segments SHM

IPC SYSTEM V

Partager une Zone Mémoire entre Deux Processus

SHM = SHared Memory

Les avantages:

- Communication directe sans recopie mémoire;
- Pas de passage par l'espace noyau à la différence des files messages (context switch et recopie);
- Latences plus faible (même mémoire)

Les inconvénients:

- Il faut manuellement synchroniser les communications (lock ou sémaphore)
 - ➡ *Comprenez qu'il est possible de mettre un lock dans cette zone mémoire, un spin lock directement, un mutex avec le bon attribut (PTHREAD_PROCESS_SHARED). Ou bien un sémaphore des IPC.*
- La structuration des données est à la charge du programme

Créer le Segment SHM

```
#include <sys/ipc.h>
#include <sys/shm.h>
```

```
int shmget(key_t key, size_t size, int shmflg);
```

- key : Une clef, soit manuelle, soit via ftok ou bien IPC_PRIVATE
- Size: taille du segment SHM en octet (arrondie à la page supérieure).
Donc mapper un int est un gros gâchis de mémoire (une page fait 4 KB).
- shmflg: mode de création de la file et ses droits UNIX
 - ➡ IPC_CREAT crée une file s'il y en a aucune associée à cette clef
 - ➡ IPC_EXCL échoue s'il existe déjà une file sur la clef indiquée (toujours combiné avec IPC_CREAT!)
 - ➡ 0600 droit UNIX en octal (important car si omis 0000 et la file est moins pratique !)

Projeter le Segment SHM

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

```
void *shmat(int shmid, const void *shmaddr, int shmflg);
```

- shmid : le descripteur du segment SHM
- shmaddr: une adresse où mapper le segment, alignée sur une frontière de page. **NULL si indifférent.**
- shmflg: options relative à la projection du segment
 - ➡ SHM_RND arrondit l'adresse passée par *shmaddr* à une frontière de page
 - ➡ SHM_RDONLY partager le segment en lecture seule
 - ➡ ... il existe d'autres flags voir man

Retirer le Segment SHM

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

int shmdt(const void *shmaddr);
```

- shmaddr: adresse renvoyée par shmat

Tous les processus doivent retirer le segment de leur mémoire autrement la suppression avec shmctl n'est pas effective. Si un processus se termine il détache la mémoire mais cela ne marque pas le segment pour suppression.

Supprimer le Segment SHM

```
#include <sys/ipc.h>
#include <sys/shm.h>
```

```
int shmctl(int shmid, int cmd, struct shmid_ds *buf);
```

- shmid : ID du segment à contrôler
- cmd: commande à appliquer à la file
 - ➡ IPC_STAT récupères les informations sur la file dans la *struct shmid_ds* (voir man)
 - ➡ IPC_SET permet de régler certains attributs en passant une *struct shmid_ds*
 - ➡ **IPC_RMID marque le segment SHM pour destruction cela ne se produira que quand tout les processus l'ayant projeté se seront détachés**
 - ➡ ... il existe d'autre flags voir man particulièrement IPC_INFO et SHM_INFO utiles pour connaitre les limites sur le système cible

PENSEZ à SUPPRIMER VOS Segments !!!

Totalement arbitraire

```
#include <sys/ipc.h>
#include <sys/shm.h>
```

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

```
int main(int argc, char **argv)
{
    int shm = shmget(19999, 2 * sizeof(int),
                    IPC_CREAT | IPC_EXCL | 0600);
```

```
    if( shm < 0)
    {
        perror("shmget");
        return 1;
    }
```

```
    int *val = (int*) shmat(shm, NULL, 0);
```

```
    if( !val )
    {
        perror("shmat");
        return 1;
    }
```

```
    /* valeur de départ */
    val[0] = 1;
    val[1] = 0;
```

```
    while(val[0])
    {
        sleep(1);
        val[1]++;
    }
```

```
    /* Unmap segment */
    shmdt(val);
    /* Server marks the segment for deletion */
    shmctl(shm, IPC_RMID, NULL);
```

```
    return 0;
}
```

Serveur

```
#include <sys/ipc.h>
#include <sys/shm.h>
```

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

```
int main(int argc, char **argv)
{
    int shm = shmget(19999, 2 * sizeof(int), 0);
```

```
    if( shm < 0)
    {
        perror("shmget");
        return 1;
    }
```

```
    int *val = (int*) shmat(shm, NULL, 0);
```

```
    if( !val )
    {
        perror("shmat");
        return 1;
    }
```

```
    /* valeur de départ */
    int last_val = -1;
    while(1)
    {
        if( val[1] != last_val ){
            printf("Val is %d max is 60\n", val[1]);
            last_val = val[1];
```

```
        /* Stop condition */
        if( 60 <= val[1] )
        {
            val[0] = 0;
            break;
        }
        else
        {
            usleep(100);
        }
    }
```

```
}
```

```
/* Unmap segment */
shmdt(val);
```

```
return 0;
}
```

Client

```
$ ./serveur &
```

```
$ ipcs -m
```

```
----- Segment de mémoire partagée -----
```

clef	shmid	propriétaire	perms	octets	nattch	états
0x00004e1f	42827778	jbbesnard	600	8	1	

```
$ ./client
```

```
Val is 0 max is 60
```

```
Val is 1 max is 60
```

```
(...)
```

```
Val is 7 max is 60
```

```
Val is 8 max is 60
```

```
Val is 60 max is 60
```

```
[2]+  Fini
```

```
./server
```

```
$ ipcs -m
```

```
----- Segment de mémoire partagée -----
```

clef	shmid	propriétaire	perms	octets	nattch	états
------	-------	--------------	-------	--------	--------	-------

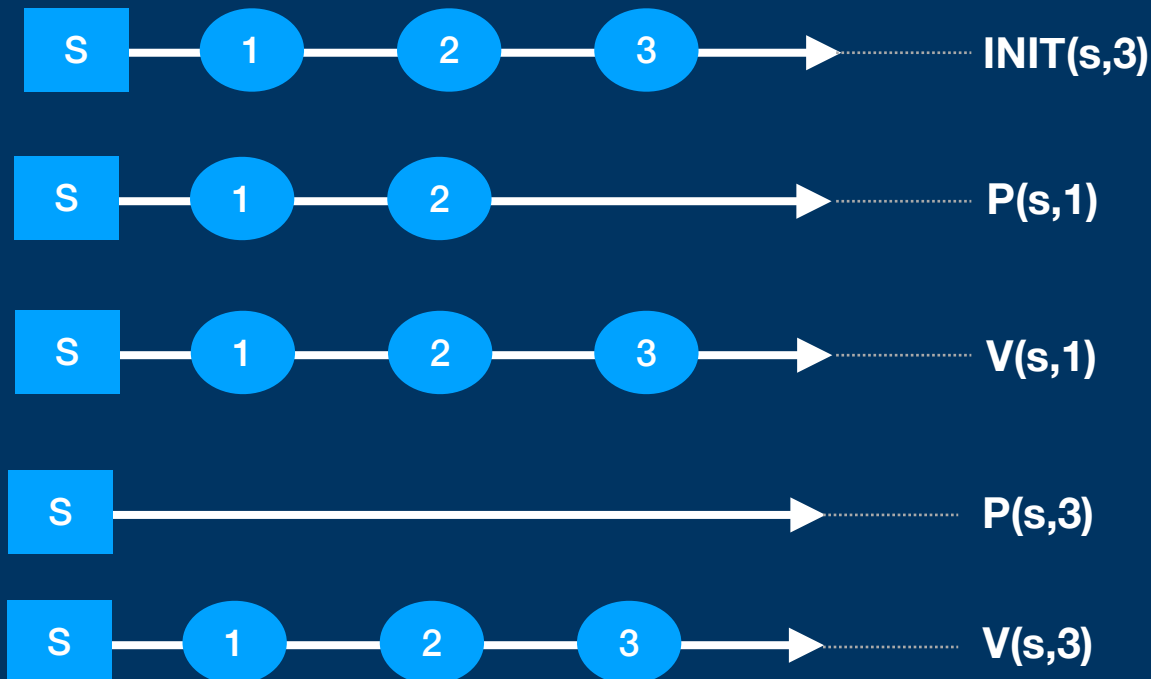
Les Sémaphores

IPC SYSTEM V

Notion de Sémaphore

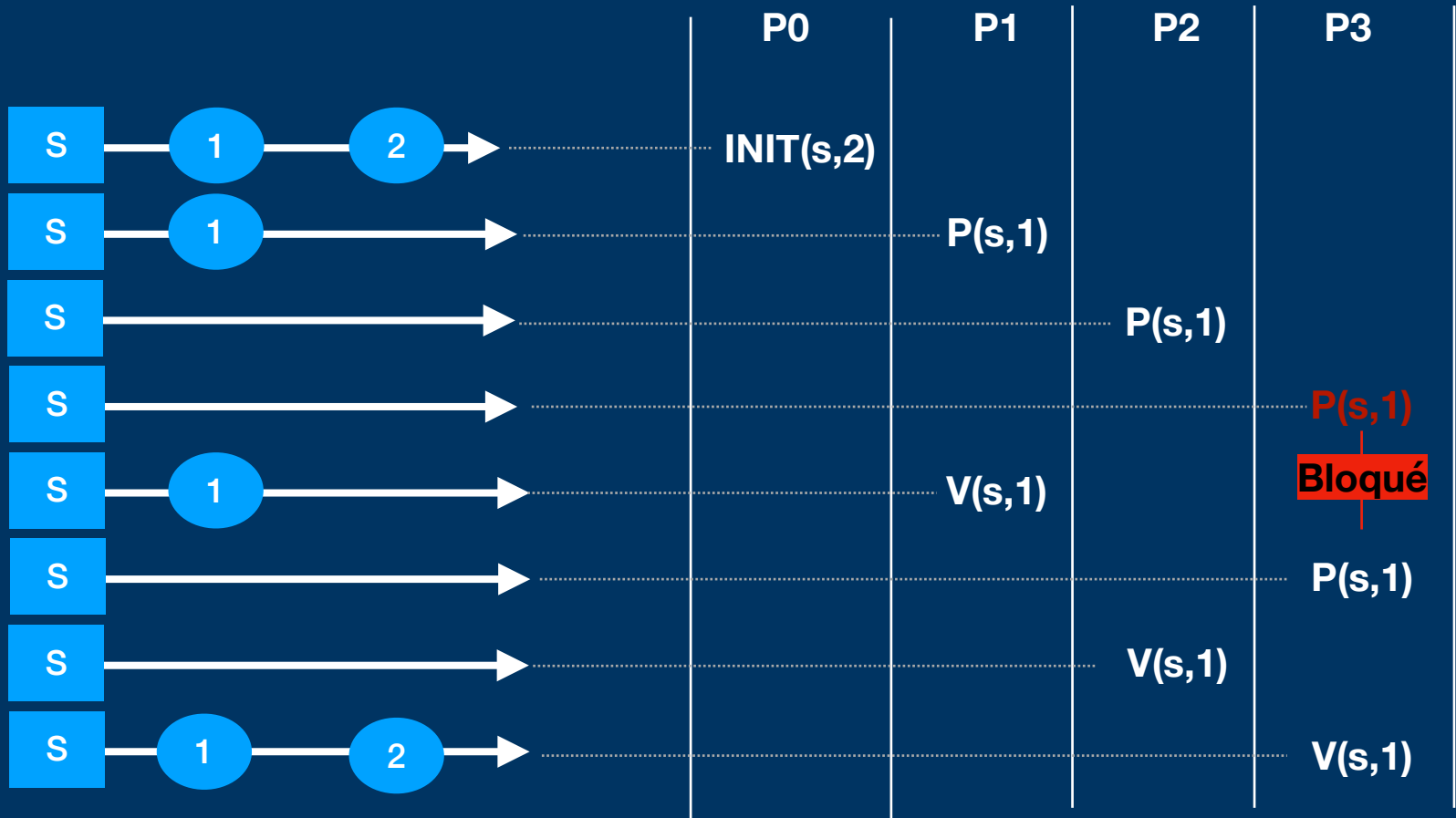
Un sémaphore est un élément de synchronisation qui permet de partager un ensemble de ressources. Il existe des sémaphores pour la programmation en mémoire partagée. Ici les sémaphore System V sont inter-processus. On définit classiquement deux opérations:

- $P(s,n)$: « Tester » (de l'allemand *passering* du fait de Dijkstra)
- $V(s,n)$: « Relâcher » (de l'allemand *vrijgave* du fait de Dijkstra)



Synchronisation avec des Sémaphores

- $P(s,n)$: « Tester »
- $V(s,n)$: « Relâcher »



Créer des Sémaphores

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
```

```
int semget(key_t key, int nsems, int semflg);
```

- key : Une clef, soit manuelle, soit via ftok ou bien IPC_PRIVATE
- nsem: nombre de sémaphores à créer
- shmflg: mode de création de la file et ses droits UNIX
 - ➡ IPC_CREAT crée une file s'il y en a aucune associée à cette clef
 - ➡ IPC_EXCL échoue s'il existe déjà une file sur la clef indiqué (toujours combiné avec IPC_CREAT!)
 - ➡ 0600 droit UNIX en octal (important car si omis 0000 et la file est moins pratique !)

Opération sur des Sémaphores

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
```

```
int semop(int semid, struct sembuf *sops, size_t nsops);
```

- *semid* : identifiant du sémaphore
- *sembuf*: opération(s) à effectuer via un tableau

```
struct sembuf {
    unsigned short sem_num; /* semaphore number */
    short          sem_op;  /* semaphore operation */
    short          sem_flg; /* operation flags */
};
```

➡ *sem_num*: numéro du sémaphore

➡ *sem_op*: opération à effectuer

▸ *sem_op* > 0 : V(s)

▸ *sem_op* < 0 : P(s)

▸ *sem_op* == 0 : attente de la valeur 0 -> utile pour synchroniser les processus

➡ Drapeau à utiliser :

▸ *IPC_NOWAIT*: non-bloquant et renvoie EAGAIN si l'opération avait dû bloquer

▸ *IPC_UNDO*: demande au noyau d'annuler l'opération si le processus se termine en cas d'arrêt intempestif

- *nsops*: nombre d'opérations à effectuer (elle sont faites de manière atomique)


Contrôle du Sémaphore

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
```

```
int semctl(int semid, int semnum, int cmd, ...);
```

- *semid* : identifiant du sémaphore
- *semnum*: identifiant du sémaphore
- *cmd*: commande à appliquer au sémaphore

Non défini dans les headers !



```
union semun {
    int          val; /* Value for SETVAL */
    struct semid_ds *buf; /* Buffer for IPC_STAT, IPC_SET */
    unsigned short *array; /* Array for GETALL, SETALL */
    struct seminfo *__buf; /* Buffer for IPC_INFO
                           (Linux-specific) */
};
```

- ➡ IPC_STAT récupère les informations sur le sémaphore
- ➡ SETALL définit la valeur du sémaphore (prend un tableau de unsigned short int en paramètre additionnel)
- ➡ **IPC_RMID supprime immédiatement le sémaphore et débloque les processus en attente**
- ➡ ... il existe **BEAUCOUP** d'autres flags voir man


```

#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <errno.h>
#include <unistd.h>
#include <sys/wait.h>

int main( int argc, char ** argv ){
    int sem = semget(IPC_PRIVATE, 1, IPC_CREAT | 0600);

    if( sem < 0 ){
        perror("msgget");
        return 1;
    }

    unsigned short val = 1;
    if( semctl(sem, 0, SETALL, &val) < 0 ){
        perror("semctl");
        return 1;
    }

    int pid = fork();
    struct sembuf p;

    p.sem_num = 0;
    p.sem_op = -1;
    p.sem_flg = SEM_UNDO;

    struct sembuf v;

    v.sem_num = 0;
    v.sem_op = 1;
    v.sem_flg = SEM_UNDO;

    if( pid == 0 ) { /* Child */
        while(1){
            if( semop(sem, &p, 1) < 0 ){
                printf("Child: SEM deleted\n");
                return 0;
            }

            printf("CHILD holding the sem\n");
            sleep(1);
            semop(sem, &v, 1);
        }
    }
}

```

Suite ...

```

    else
    {
        /* Parent */
        int i = 0;
        while(i < 5)
        {
            semop(sem, &p, 1);

            printf("PARENT holding the sem\n");
            sleep(1);
            semop(sem, &v, 1);
            i++;
        }

        /* Parent delete the sem and unlock the child */
        semctl(sem, 0, IPC_RMID);

        wait( NULL );
    }

    return 0;
}

```

Sortie du Programme

```
$ ./a.out  
PARENT holding the sem  
CHILD holding the sem  
PARENT holding the sem  
CHILD holding the sem  
PARENT holding the sem  
CHILD holding the sem  
PARENT holding the sem  
CHILD holding the sem  
PARENT holding the sem  
CHILD holding the sem  
Child: SEM deleted
```

IPCs POSIX

IPC POSIX

Le standard POSIX plus récent propose également les même mécanismes:

- Files de messages
 - Segment de mémoire partagée
 - Sémaphores
-
- Il sont plus fiables en termes de libération et de partage de la ressource;
 - Enfin l'ensemble de l'interface est thread-safe;
 - Les objets sont demandés par nom et non avec une valeur donnée;
 - Ces appels sont un peu moins portable et sont à attendre plus sur des LINUX que des UNIX au sens large;
 - On les décrit généralement comme plus simples à utiliser.

Files de Message POSIX

À vous de jouer avec le man:

- mq_open
- mq_close
- mq_send
- mq_receive
- mq_unlink

Portez l'exemple SYS-V

Que pensez-vous de *mq_notify* ?

Segment SHM POSIX

À vous de jouer avec le man:

- shm_open
- shm_unlink
- mmap

Portez l'exemple SYS-V

Sémaphore IPC POSIX

Aussi « sémaphore nommé » à ne pas confondre avec les sémaphore « anonymes » de la NPTL (libpthread) qui sont dans le même header.

Rappel (ou pas) pour un sémaphore «anonyme »:

- sem_init
- sem_destroy
- **sem_post**
- **sem_wait**

À vous de jouer avec le man pour un sémaphore nommé:

- sem_open
- sem_close
- **sem_post**
- **sem_wait**
- sem_unlink

Portez l'exemple SYS-V

Peut-on l'implémenter avec un sémaphore anonyme et pourquoi ?