

Programmation Système



Merci

- Alexandre Dang : Programmation système sous Linux et Windows (électif 2A, Rennes)
- Frédéric Tronel et Pierre Wilke : Systèmes d'exploitation (électif 2A, Rennes)
- Idir Ait Sadoune : Systèmes d'exploitation et virtualisation (Mastère spécialisé ASI, Saclay)

Plan

- Présentation
 - Introduction
 - Historique
- Programmation sous Linux
 - Fichier
 - Processus
 - Communication
 - Threads
 - Synchronisation
- Virtualisation et conteneurisation

Plan

- Présentation
 - Introduction
 - Historique
- Programmation sous Linux
 - Fichier
 - Processus
 - Communication
 - Threads
 - Synchronisation
- Virtualisation et conteneurisation

Objectifs

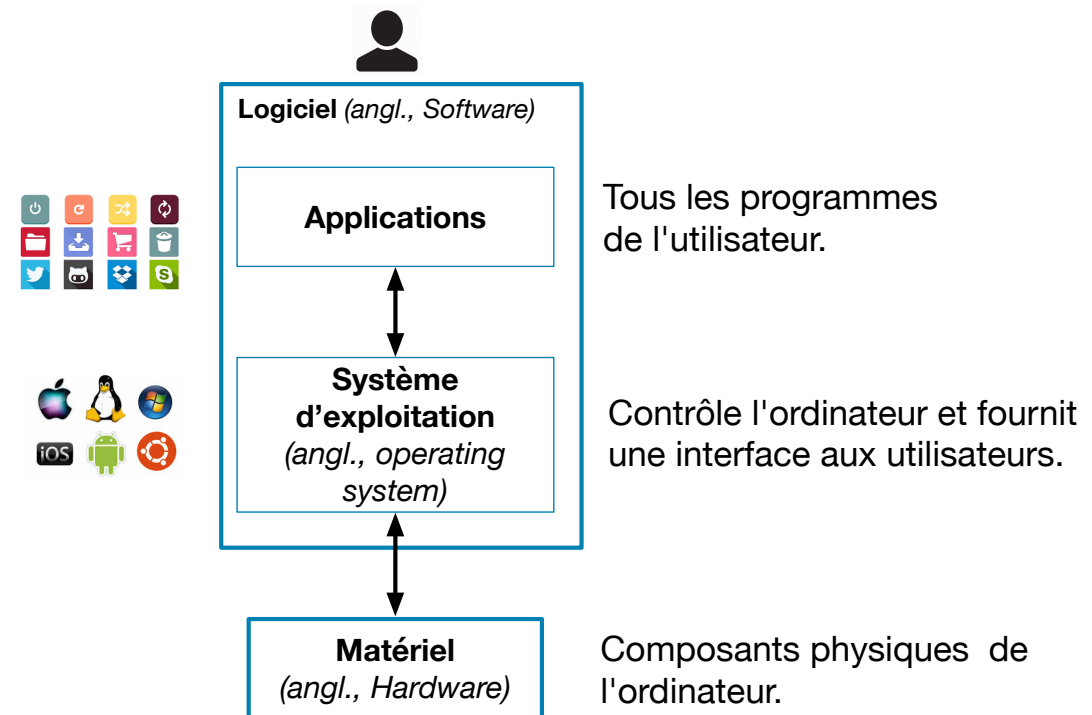
- *La programmation système est un type de programmation qui vise au développement de programmes qui font partie du système d'exploitation d'un ordinateur ou qui en réalisent les fonctions [...]*

Wikipédia

- Dans le cadre de ce cours, l'objectif est de découvrir et d'utiliser quelques services de bas niveau fournis par un système d'exploitation.
- Linux est pris comme exemple pour les services retenus.
- Connaissances supposées : les mêmes que celles du cours « Concepts des langages de programmation - Mise en œuvre en C/C++ ».

Système d'exploitation

- Un système d'exploitation (ou OS, Operating System) est une couche logicielle conceptuellement située entre le matériel et les applications.
- Il offre aux programmeurs et aux utilisateurs un ensemble d'abstractions qui permettent de ne pas avoir à gérer les spécificités du matériel.
- Les applications font des appels systèmes à des services proposés par le système d'exploitation qui, seul, gère les implications matérielles et retourne le résultat de l'appel.



Services

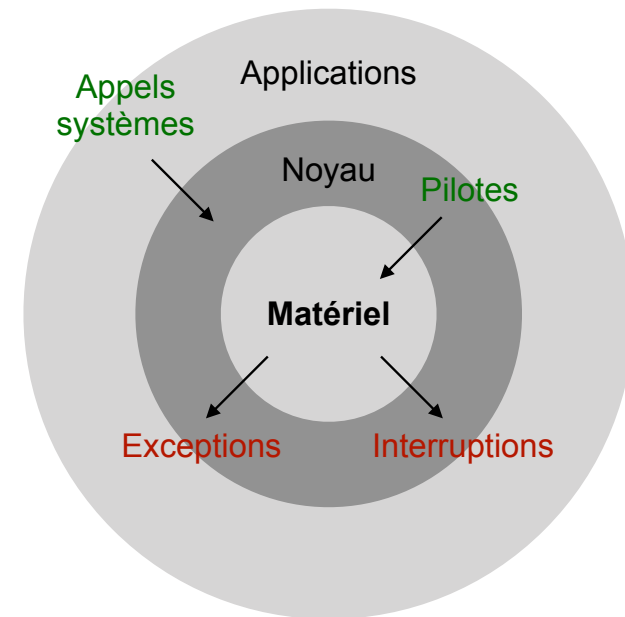
- Lancement et gestion des applications.
- Gestion de la mémoire.
- Gestion des E/S et des fichiers.
- Gestion des périphériques.
- Coopération inter-tâches.
- Communications.

Modes noyau et utilisateur

- Une application ne doit pas pouvoir accéder directement aux ressources gérées par le système d'exploitation (gestion de la mémoire, des périphériques...)
- Mais un processeur ne sait pas distinguer une instruction d'une application d'une instruction du système d'exploitation
- Un processeur possède donc (au moins) deux modes de fonctionnement (bit dans le registre d'état) : mode utilisateur (pour les applications) et mode noyau (pour le système d'exploitation)
- Certaines instructions sont interdites en mode utilisateur (déclenchement d'une exception)
- Certaines de ces instructions servent à passer du mode utilisateur au mode noyau : demande par une application d'un service au système d'exploitation
 - Intel : syscall
- Le changement de mode a un coût important (par rapport un l'appel d'un sous-programme)

Liens entre couches

- Appels systèmes : demandes de services
- Noyau (kernel) : partie du système d'exploitation rendant des services en lien avec le matériel
- Interruptions (matérielles) : événements signalés par le matériel
- Exceptions : événements générés par le processeur
- Pilotes (drivers) : composants logiciels contrôlant les périphériques



Appels systèmes en C dans Linux

- Presque 500 appels systèmes
 - Identifiés par un numéro
- Une seule fonction
 - syscall(long number, ...)
 - Écrite en assembleur et devant être adaptée à chaque processeur
- Mais les principaux appels systèmes sont accessible via des enveloppes (wrapper)
 - Par exemple, write() a pour numéro 64
- strace permet d'afficher ces appels systèmes

Démo

The GitLab logo is displayed within a dark red rectangular box. The text "GitLab" is white and centered within the box.

GitLab

cours/progsyst/Syscall.c

Plan

- Présentation
 - Introduction
 - Historique**
- Programmation sous Linux
 - Fichier
 - Processus
 - Communication
 - Threads
 - Synchronisation
- Virtualisation et conteneurisation

Multics

- Multiplexed Information and Computing Service
- Successeur de Compatible Time-Sharing System (1961)
- Projet commun entre le MIT, les Bell Labs et General Electric (GE), démarré en 1964.
- Les Bell Labs se sont retirés en 1969.
- C'est un système d'exploitation extrêmement novateur mais qui a connu un succès commercial en demi-teinte.
- Les chercheurs des Bell Labs impliqués dans Multics s'inspireront de certaines des idées présentes dans Multics pour concevoir un système plus simple : Unix.

Unix : l'origine (1969)

- Les Bell Labs trouvent le projet Multics trop complexe et ambitieux.
- Ils se retirent progressivement du projet.
- Les derniers chercheurs à s'être retirés, reconsidèrent la construction d'un système d'exploitation multitâche de conception plus simple que Multics.
- Les concepteurs : Ken Thompson, Brian Kernighan, Dennis Ritchie
- Une première version est développée sur un PDP-7, dans l'optique de faire tourner le jeu vidéo Space Travel.

Unix : l'expansion (1)

- Les Bell Labs fournissent un soutien aux trois chercheurs, à la condition qu'ils écrivent un éditeur de texte et un système de mise en page pour le nouvel OS.
- Celui-ci est porté sur PDP-11.
- Brian Kernighan suggère le nom d'Unics (Uniplexed Information and Computing Service) qui est un jeu de mot à propos de Multics.
- Le terme UNIX remplacera rapidement celui d'Unics.
- Le logiciel de mise en page TROFF est développé et sera utilisé par la suite pour la rédaction de l'ensemble des brevets de la compagnie.
- En 1972, UNIX est réécrit en C. Ceci permet de le porter plus facilement vers d'autres processeurs.

Unix : l'expansion (2)

- En 1956, une condamnation dans un procès antitrust interdit à AT&T de se lancer dans le commerce lié aux ordinateurs.
- Les Bell Labs qui sont une division de AT&T ne peuvent donc pas commercialiser UNIX.
- Celui-ci est donc distribué de manière gratuite (avec le code source).
- UNIX devient de plus en plus utilisé et populaire dans les universités américaines notamment pour l'enseignement des systèmes d'exploitation.
- La septième version d'UNIX est produite en 1979.
- Cette version marque un changement de licence : il n'est plus possible d'étudier le code source du système d'exploitation.

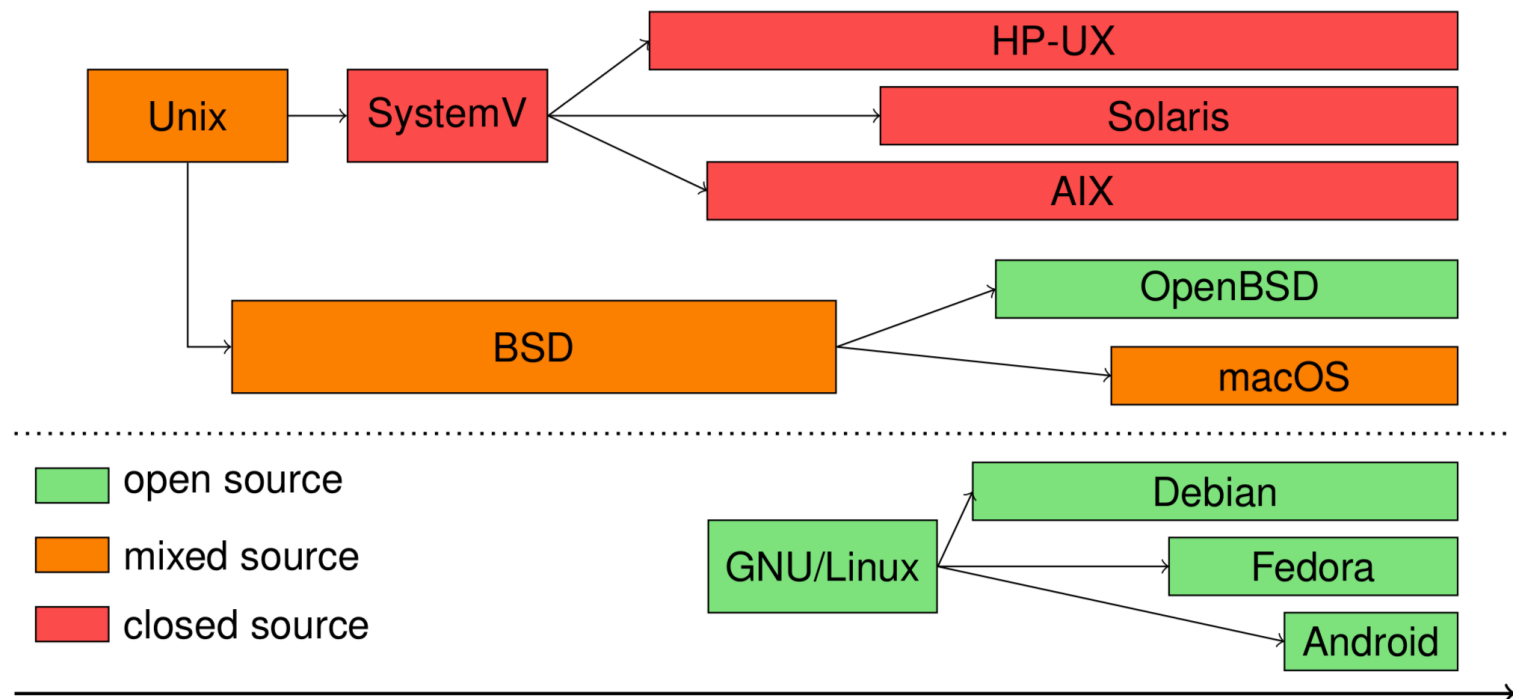
Unix : le début de la division

- En 1983, le département de la Justice américain (DOJ) entame un second procès antitrust contre AT&T.
- Ce procès a pour résultat le découpage de la compagnie, mais la relève de son interdiction de se lancer dans le commerce des ordinateurs.
- AT&T en profite pour commercialiser UNIX.
- Comme les conditions de distribution sont devenues moins favorables pour les universités, l'université de Berkeley (UCB) continue à développer sa propre version (Berkeley Software Distribution).
- La branche BSD continue d'exister de nos jours, elle s'est subdivisée en de nombreuses variantes.

Unix : la division

- C'est aussi durant les années 80 que de nombreuses compagnies privées se mettent à produire leur propres versions d'UNIX, c'est ce qu'on a appelé la guerre des UNIX.
- Malgré des tentatives de normalisation (dont POSIX), les stratégies commerciales des différentes compagnies ont finalement nuit au succès commercial d'UNIX.
- Timeline

Unix : familles

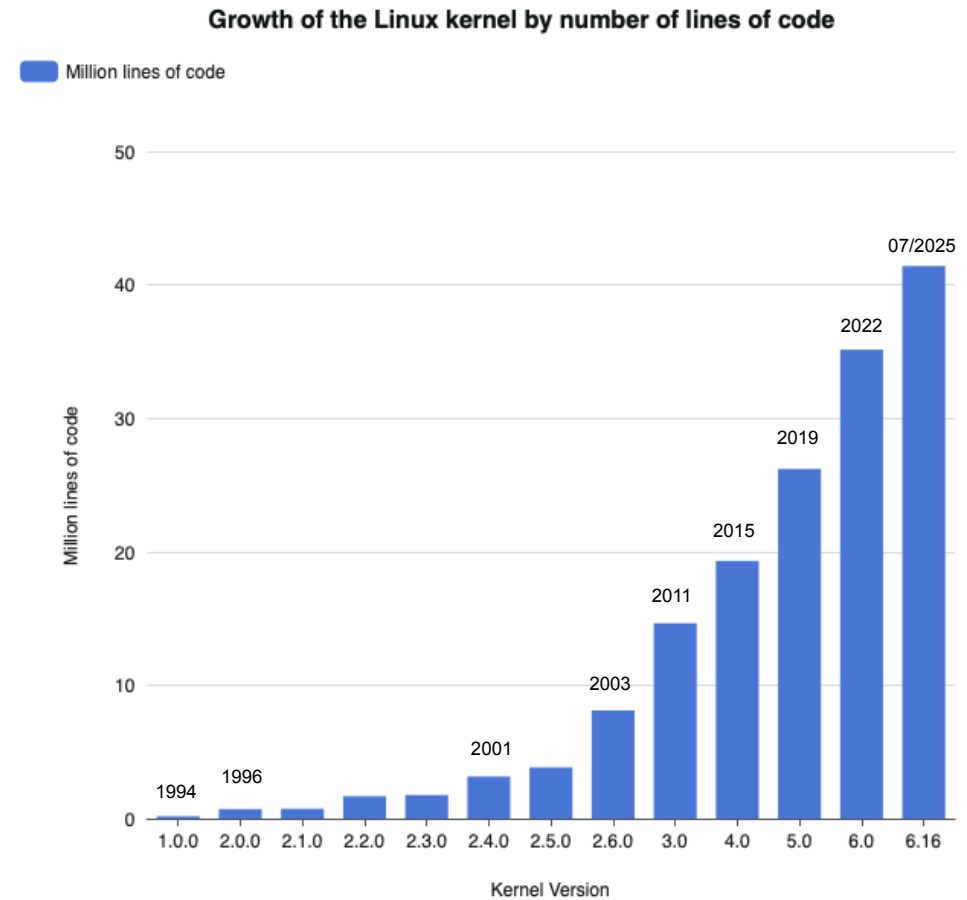


GNU/Hurd

- Richard Stallman : projet de développer un système d'exploitation libre et compatible Unix (1983).
- GNU « GNU's Not UNIX ».
- Au début des années 1990, le projet GNU possède une version utilisable de tous les éléments nécessaires à la construction d'un système d'exploitation à l'exception du plus central : le noyau.
- Projet GNU/Hurd : micro-noyau

Linux

- Linus Torvalds
- comp.os.minix
(message, minix)
- Noyau monolithique
- Historique des versions
- Les composants du noyau
- Nombreuses distributions



Crédit : https://en.wikipedia.org/wiki/Linux_kernel

De MS-DOS à MS-Windows 3

- MS-DOS 1.0 (achat par Microsoft de QDOS), sorti en 1981, était un système d'exploitation 16 bits mono-utilisateur en ligne de commande. Il occupait 8 Ko en mémoire.
- Deux versions de MS-Windows sortent respectivement en 1985 et 1987. Elles ne rencontrent pas un grand succès.
- Windows 3, sorti en 1990 pour l'Intel 386, se vend à plus d'un million d'exemplaires en 6 mois. Il s'agit avant tout d'une surcouche graphique à MS-DOS, pas d'un vrai système d'exploitation.
- Même s'ils apportent de nombreuses innovations technologiques (Multiprogrammation, gestion des processus, mémoire virtuelle), MS-Windows 95, 98 et Me sont en fait construits en interne sur la base MS-DOS.

De Windows NT 3.1 à Windows XP

- Parallèlement à Windows 3, Microsoft (après l'arrêt de sa collaboration avec IBM sur OS/2) conçoit un système d'exploitation entièrement 32 bits dédié aux serveurs, Windows NT, qui introduit l'API Win32. La première version, Windows NT 3.1, sort en 1993.
- Windows NT 4, sorti en 1996, propose la même API Win32 mais exhibe la même interface graphique que MS-Windows 95. 4 familles de processeurs sont supportés.
- Son successeur, Windows 2000, ne supportera plus de manière active que les processeurs Intel x86.
- L'arrivée de Windows XP (2001) marque un tournant important dans l'évolution des gammes d'OS Microsoft : pour la première fois, les systèmes orientés professionnels et les systèmes orientés grand-public sont basés sur le même noyau et l'API Win32.

De Windows Vista à Windows 11

- Windows Vista (2007), plus de 5 ans après Windows XP, a reçu beaucoup de critiques.
- Windows 7 (2009) a été lui très bien accueilli
- Windows RT (2012) à destination des processeurs ARM (Surface RT)
- Windows 10 en 2015, Windows 11 en 2021
- Timeline

macOS

- Mac OS Classic de 1984 à 2001 (Système 1 à Mac OS 9)
- macOS (Mac OS X jusqu'en 2012, OS X jusqu'en 2015)
Basé sur un noyau XNU et l'implémentation BSD d'Unix (Darwin)
Interface graphique propriétaire issue de Next
- OS dérivés : iOS, iPadOS, watchOS, tvOS, visionOS, audioOS

Plan

- Présentation
 - Introduction
 - Historique
- Programmation sous Linux
 - Fichier
 - Processus
 - Communication
 - Threads
 - Synchronisation
- Virtualisation et conteneurisation

Linux : tout est fichier

- fichier normal
- répertoire
- lien symbolique : une référence vers un autre fichier
- named pipe ou FIFO : un outil de communication unidirectionnel entre processus locaux
- socket : un outil de communication bidirectionnel entre processus distants
- périphériques (écran, imprimante, clavier, disque, clef USB...)
- processus (Linux)

Descripteur de fichier

- Tout ces fichiers sont manipulables par des descripteurs de fichier
- En C ils sont représentés par des entiers non négatifs (**int**)
 - `STDIN_FILENO` (0) : entrée standard
 - `STDOUT_FILENO` (1) : sortie standard
 - `STDERR_FILENO` (2) : sortie d'erreurs standard
- Avantage/désavantage:
 - + API commune pour pouvoir gérer toutes les ressources avec des descripteurs de fichiers
 - `open()`, `creat()`, `read()`, `write()`, `fcntl()` et `close()`
 - Fonctions génériques et de bas niveau, ne permettant pas une utilisation simple de ressources spécifiques

Flux de données

- Surcouche aux descripteurs de fichiers pour les *fichiers normaux*
- Dans la bibliothèque standard C (`#include <stdio.h>`)
 - Représentation : `FILE *`
 - `fopen()`, `fclose()`, `fread()`, `fwrite()`,
`fscanf()`, `fprintf()`...
- Avantages
 - + Portabilité, appartient au standard C
 - + Performance, gestion plus optimisée des lectures/écritures
 - + Simplicité, fonctions spécialisées dans la gestion de fichiers normaux
- Trois flux standards pour chaque processus
 - `stdin`, `stdout` et `stderr`

Plan

- Présentation
 - Introduction
 - Historique
- Programmation sous Linux
 - Fichier
 - Processus**
 - Communication
 - Threads
 - Synchronisation
- Virtualisation et conteneurisation

Programme et processus

- Un **programme** est l'ensemble des instructions et données décrivant l'apparence et le comportement d'une application
 - Un programme est stocké dans un fichier
 - Un programme est statique
 - Un programme est exécutable
- Un **processus** est l'ensemble des informations décrivant l'état d'un programme en cours d'exécution
 - Un processus n'existe qu'en mémoire
 - Un processus est dynamique
 - Plusieurs processus peuvent être associés à un même programme

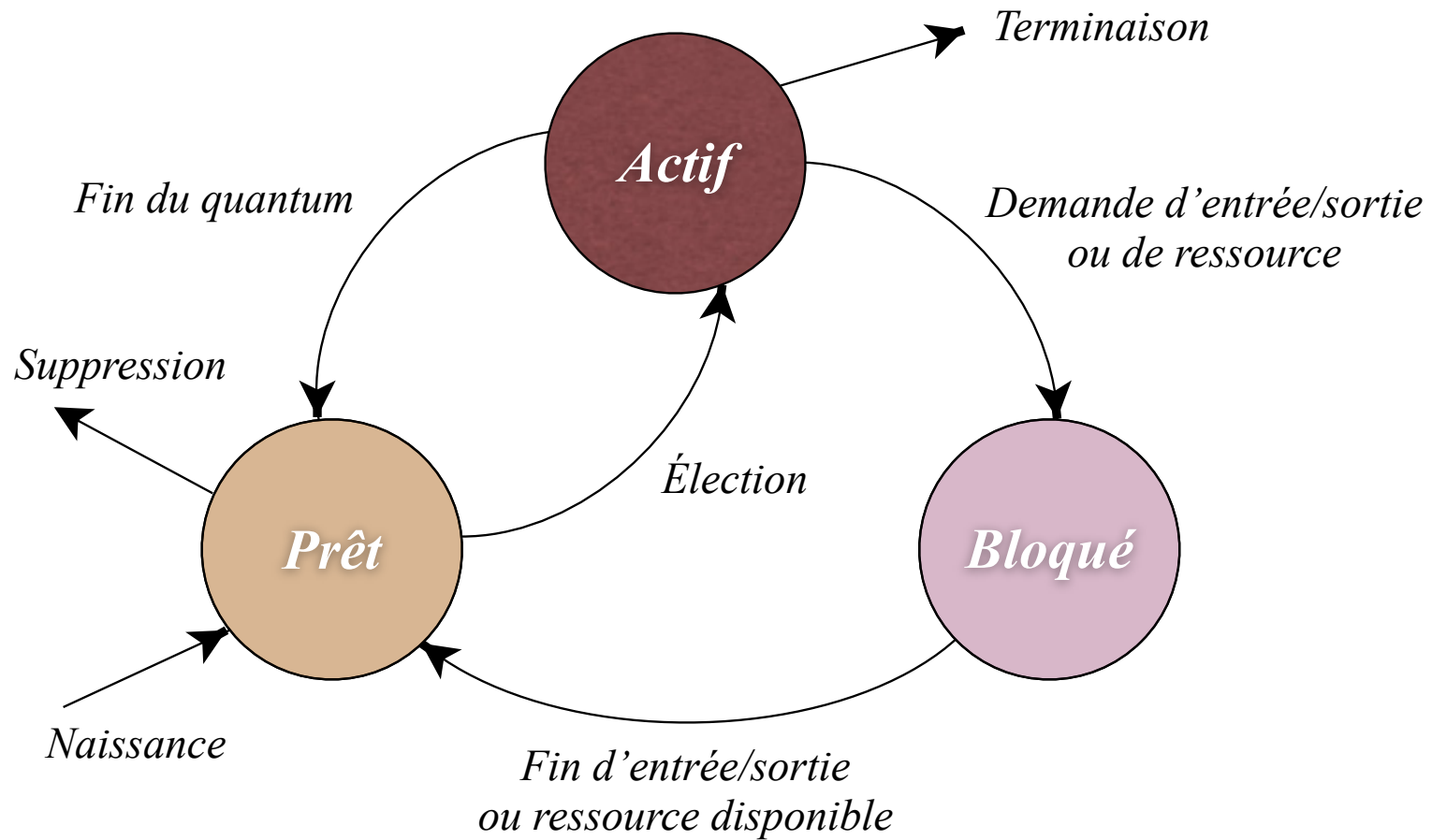
Analogie culinaire classique

- Recette = programme
- Cuisinier = processeur
- Plat en cours de préparation = processus
- Un cuisinier peut préparer simultanément plusieurs plats (certains pouvant correspondre à la même recette)

Gestion des processus

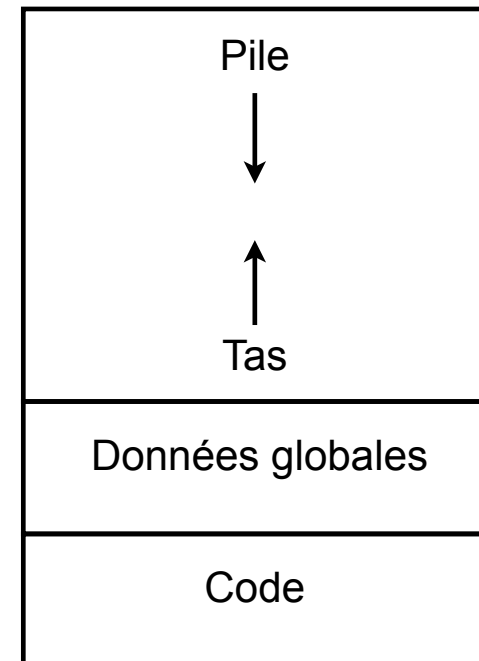
- Le système d'exploitation a en charge :
 - La création de processus
 - La répartition de l'utilisation du processeur par les différents processus (**ordonnanceur**, **scheduler**)
 - La communication entre processus (copier/coller, mémoire partagée...)
 - La gestion de l'utilisation des ressources par les processus
 - mémoire
 - entrées/sorties

États d'un processus



Mémoire vue d'un processus

- Chaque processus pense être le seul à s'exécuter sur l'ordinateur (hors aspects liés aux communications).
- Un processus voit un espace mémoire *illimité* (de l'adresse 0 à l'adresse ...).
- Les adresses manipulées par un processus sont dites **logiques**.



Démo

The GitLab logo is displayed within a dark red rectangular box. The text "GitLab" is white and centered within the box.

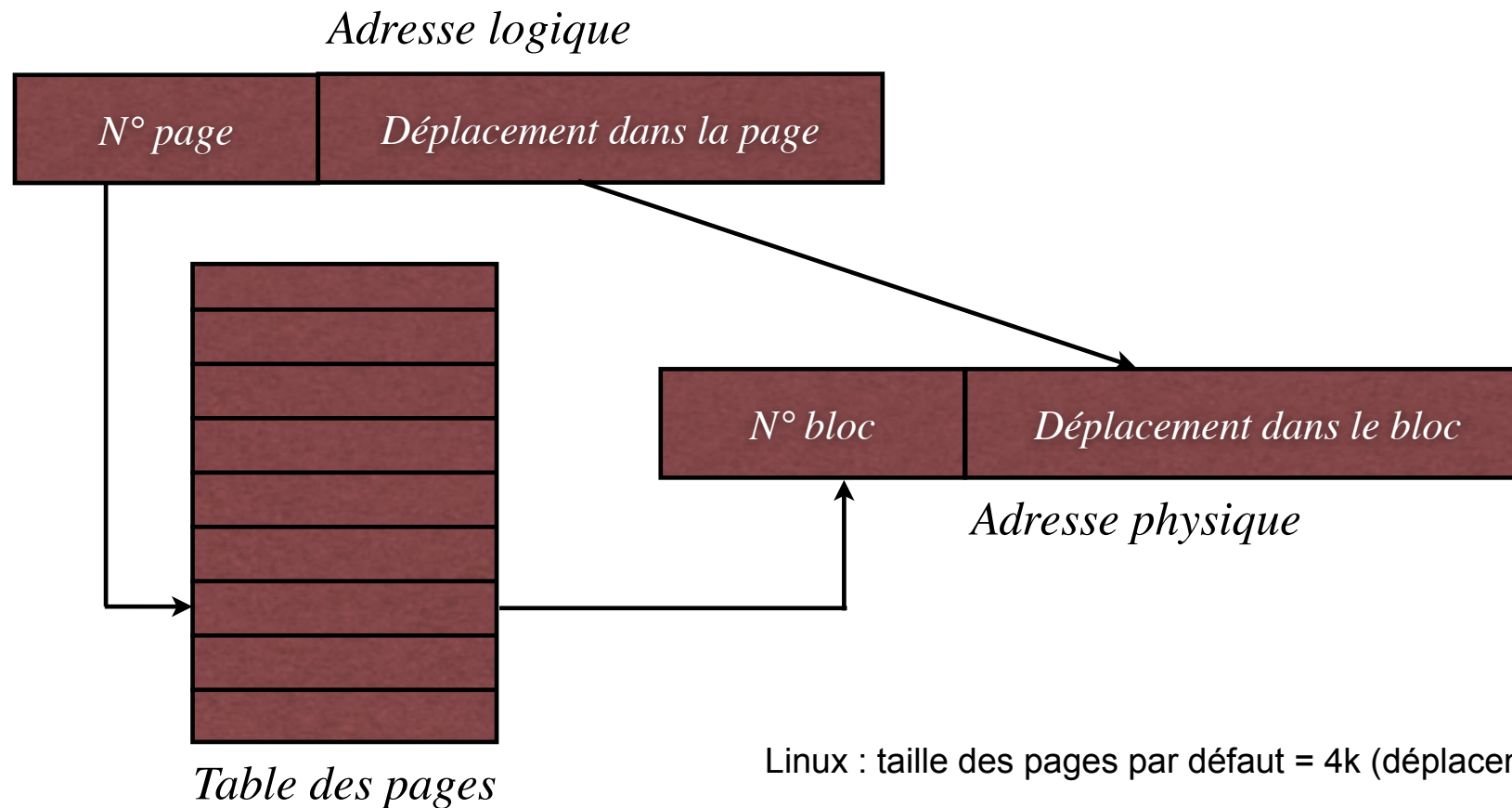
GitLab

cours/progsyst/Memory.cpp

Mémoire vue du système d'exploitation

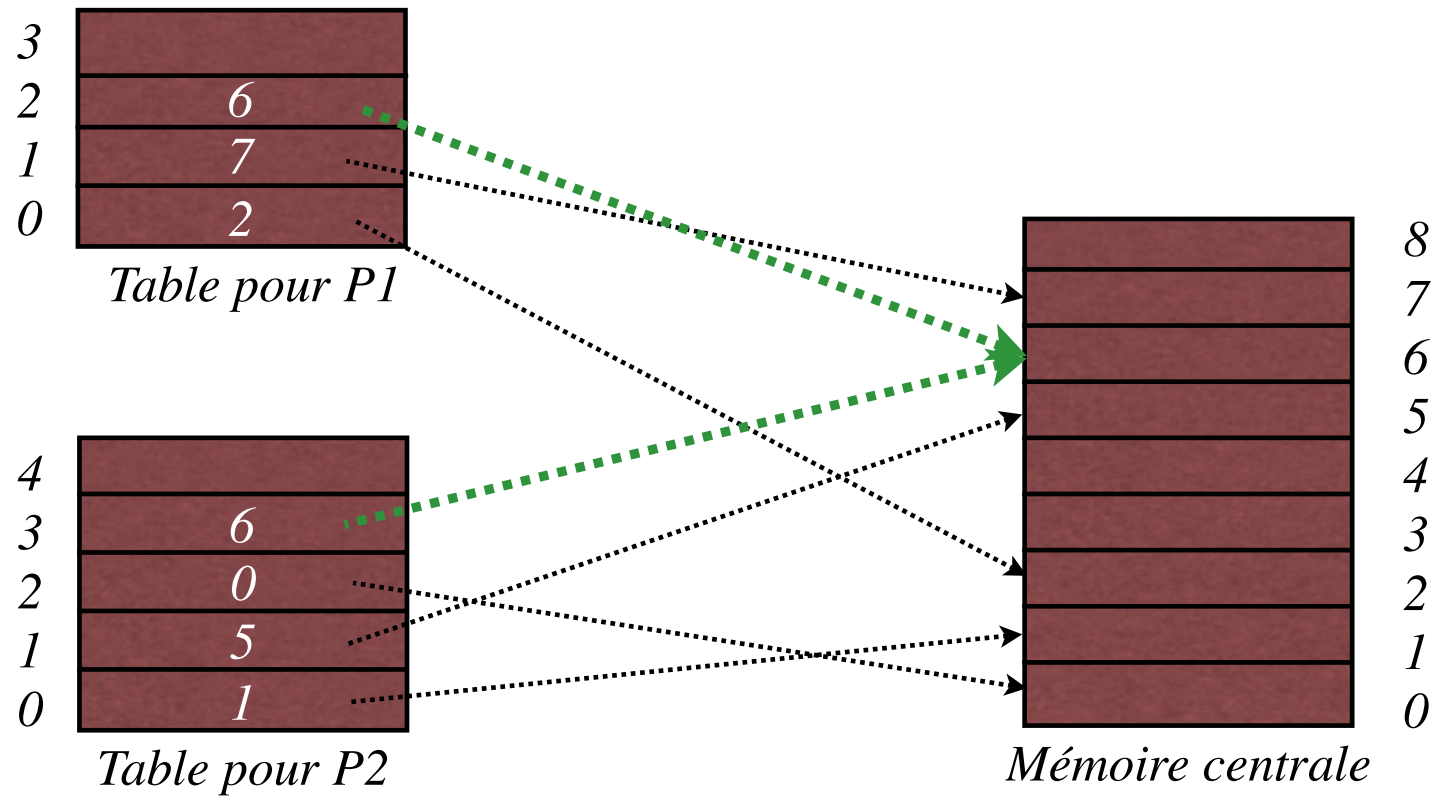
- Le système d'exploitation a accès à la mémoire réelle, le noyau utilise des adresses **physiques**.
- Il alloue à chaque processus une partie de cette mémoire réelle.
- Il configure le composant matériel (MMU) chargé de convertir les adresses logiques du processus actif en adresses physiques.
- Les ordinateurs actuels (processeur et système d'exploitation) utilisent le mécanisme de **pagination** pour cela (ils offrent aussi la **mémoire virtuelle**).

Principe de la pagination



Linux : taille des pages par défaut = 4k (déplacement sur 12 bits)

Mémoire partagée via la pagination



Bloc de contexte

- Lors du changement de processus actif, l'ordonnateur doit sauvegarder l'état du processus courant :
 - registres du processeur,
 - ressources utilisées :
 - espace mémoire utilisé (configuration de la pagination),
 - fichiers ouverts,
 - canaux de communications utilisés,
 - ...
- et restaurer celui du processus qui va (re)devenir actif

Types de processus

- Différents types de processus
 - Processus contrôlé par le shell
`./a.out`
 - Processus en tâche de fond
`gedit &`
 - Services (daemon)
`httpd, sshd...`

Création d'un processus

- On peut créer un processus en dupliquant le processus courant avec `fork()`

```
pid_t fork();
```

- retourne 0 si on se trouve dans le processus fils
 - retourne le PID du fils si on se trouve dans le processus père
- C'est maintenant un *wrapper* de l'appel système `clone()`
- Un processus possède un identifiant unique appelé PID

```
pid_t getpid();
```

- Tous les processus ont un ancêtre commun : le processus lancé au démarrage (PID = 1).

Changement de programme

- Pour changer le programme d'un processus (en général le fils), il faut appeler une fonction de la famille `exec()` :

```
int execl( const char * path,  
           const char * arg, ..., NULL );
```

```
int execv( const char * path,  
           char * const argv[] );
```

Démo

The GitLab logo is displayed within a dark red rectangular box. The text "GitLab" is white and centered within the box.

GitLab

`cours/progsyst/Fork.c`

Terminaison d'un processus

- Terminaison normale

`return` depuis la fonction `main()`

`void exit( int status );`

- Terminaison anormale

`void abort();`

`void assert( expression );`

- Attente de la terminaison d'un processus fils

`pid_t wait( int * status );`

Gestion de la mémoire

- Gestion de bas niveau

```
void * mmap(  
    void * addr, size_t length, int prot,  
    int flags, int fd, off_t offset );
```

```
int munmap( void * addr, size_t length );
```

- Gestion via libc

```
void * malloc( size_t size );
```

```
void free( void * ptr );
```

```
void * calloc( size_t nmemb, size_t size );
```

```
void * realloc( void * ptr, size_t size );
```

Plan

- Présentation
 - Introduction
 - Historique
- Programmation sous Linux
 - Fichier
 - Processus
 - Communication
 - Threads
 - Synchronisation
- Virtualisation et conteneurisation

IPC

- Douglas McIlroy (1978) et Peter H. Salus (1994) :

Philosophie Unix

- Write programs that do one thing and do it well.
 - Write programs to work together.
 - Write programs to handle text streams, because that is a universal interface.
-
- Les outils de communication entre processus sont communément appelés IPC (Inter-Process Communication)

Différents IPC

- Signaux
- *Pipes* ou tubes
- Sockets
- Fichiers partagés
- Mémoire partagée
- (Files d'attente de message)

Signaux

- Message asynchrone d'un processus à un autre
- Envoi des signaux avec

```
int kill( pid_t pid, int sig );
```

- Action sur réception d'un signal :

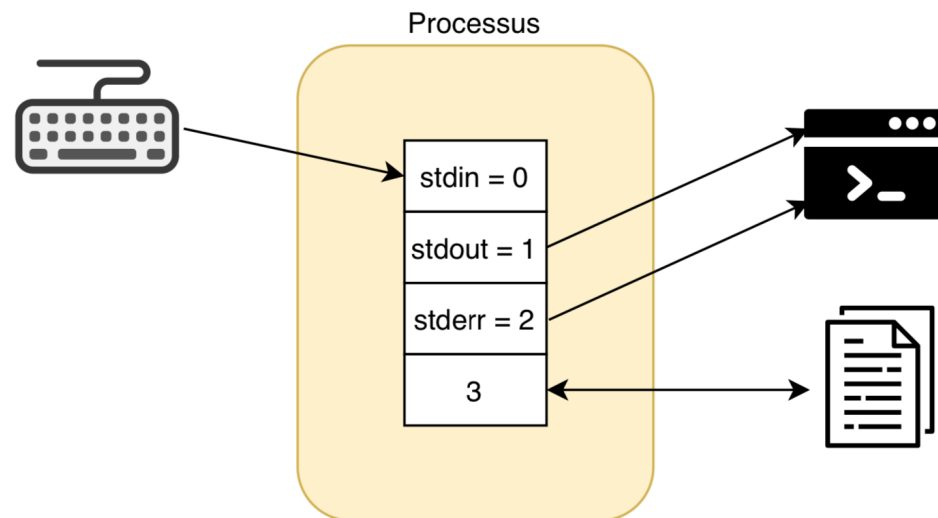
```
int sigaction( int signum,  
               const struct sigaction * act,  
               struct sigaction * oldact);
```

- Signaux standards

SIGINT (Ctrl+C), SIGQUIT (Ctrl+D) ...

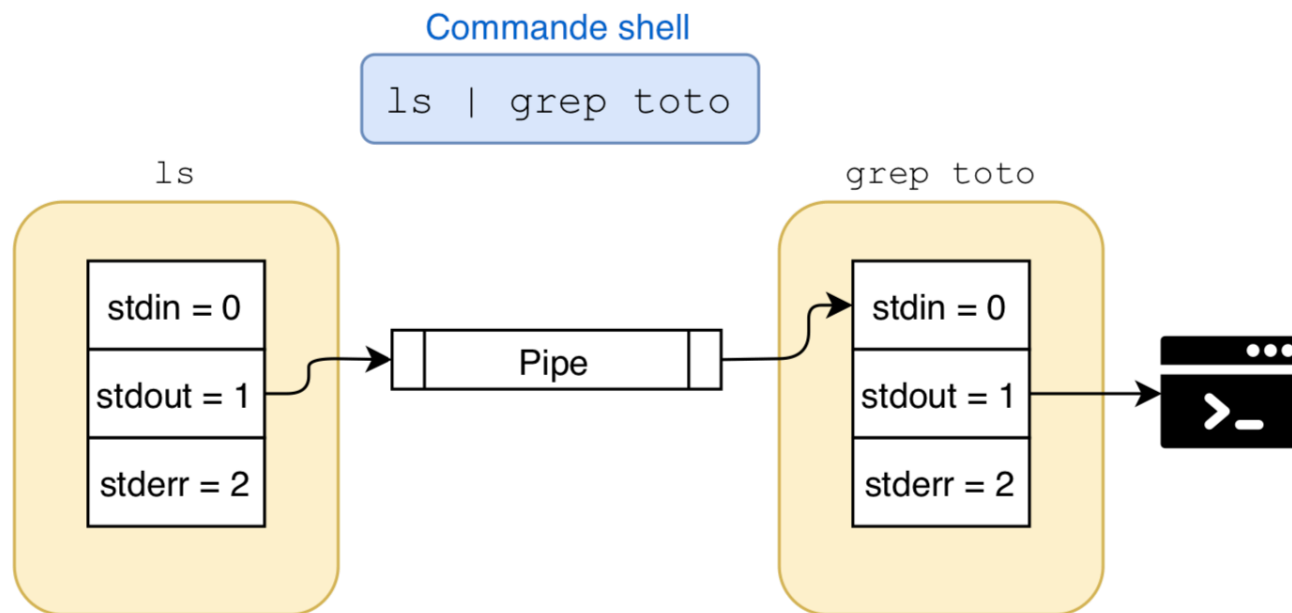
Table des descripteurs

- Le noyau alloue des ressources I/O pour chaque processus sous forme de descripteurs de fichiers. La plupart des IPC sont représentés par des descripteurs de fichiers.



Pipes

- Les pipes sont des IPC unidirectionnels



Exemple pipe (1)

- On veut un pipe dans lequel un processus fils peut écrire des messages au processus père

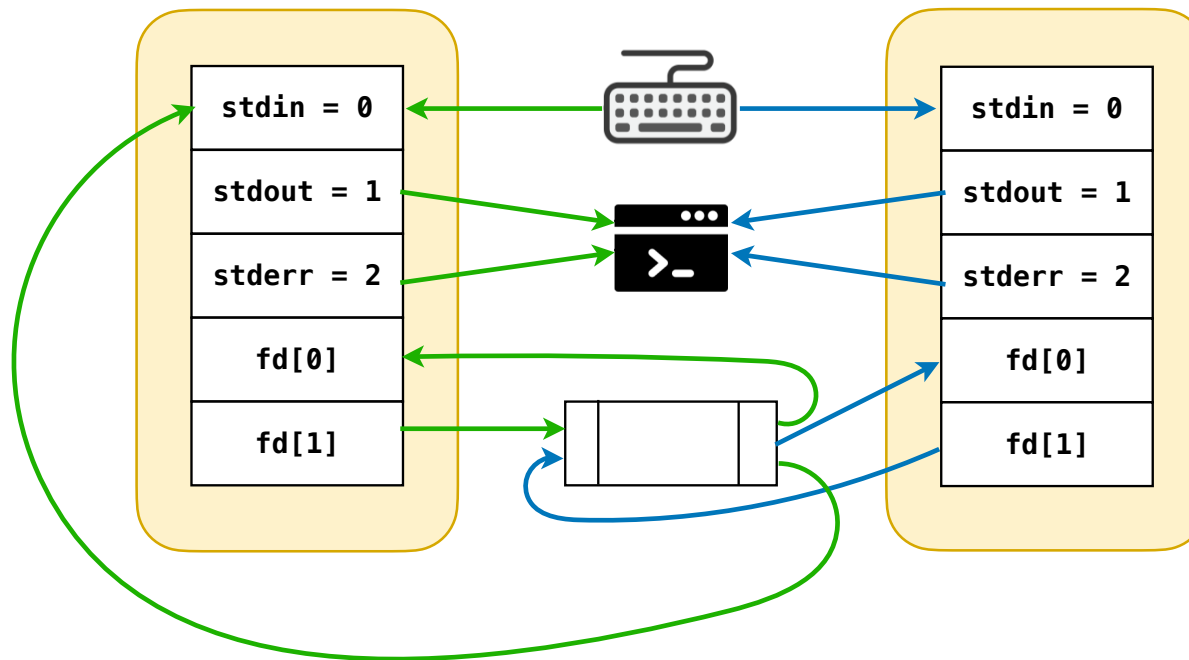
```
// Tableau pour 2 descripteurs de fichier
int fds[2];
// Création d'un pipe, les bouts sont stockés dans fds
pipe( fds );
pid_t pid = fork();
if( pid == 0 ) {           // Processus fils
    close( fds[0] );      // Fermeture du côté lecture
    // ...
} else {                  // Processus père
    close( fds[1] );      // Fermeture du côté écriture
    // On raccroche stdin au côté lecture du pipe
    dup2( fds[0], STDIN_FILENO );
    // ...
}
```

Exemple pipe (2)

`dup2( fd[0], stdin )`

`fork()`

`close( fd[0] )`



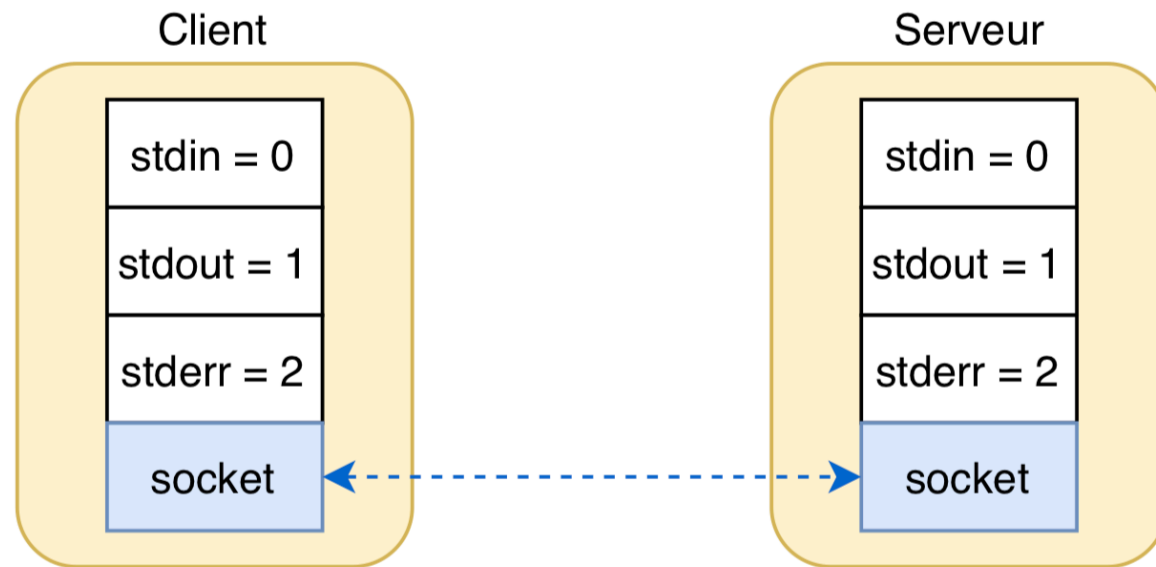
Démo

GitLab

cours/progsyst/Pipe.c

Sockets

- Les sockets sont des IPC bidirectionnels pouvant utiliser les protocoles réseaux (UDP, TCP...)



Fonctions sockets (1)

- Création d'un socket

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

- Liaison de la socket à une adresse IP et un port

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

- Description de l'adresse

```
struct sockaddr_in address;  
address.sin_family = AF_INET;  
address.sin_addr.s_addr = INADDR_ANY; //localhost  
address.sin_port = htons( PORT );
```

Fonctions sockets (2)

- Attente de connexion par le serveur

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

- Ouverture de la connexion par le client

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

- Acceptation par le serveur de la connexion

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

Ordre des octets

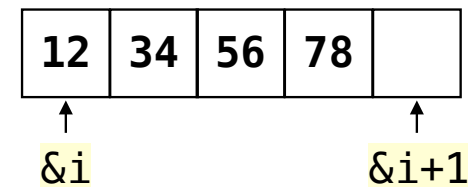
- Le nombre sur 32 bits

```
int i = 0x12345678;
```

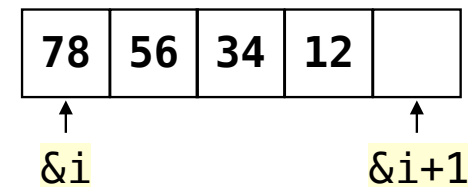
<https://history.rfc-editor.org/ien/ien137.txt>

https://fr.wikipedia.org/wiki/Les_Voyages_de_Gulliver

- peut être stocké en *gros-boutiste* (big-endian)



- ou en *petit-boutiste* (little-endian)



- Des ordinateurs qui échangent des données doivent se mettre d'accord sur l'ordre des octets utilisé : fonctions `htonl()`, `htons()`, `ntohl()`, `ntohs()`

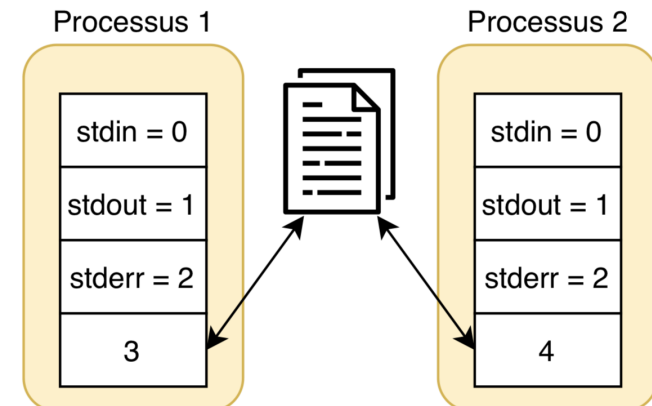
Fichiers partagés

- Ouverture du même fichier par plusieurs processus

```
int open( const char * name, int flags, mode_t mode );
```

- Mappage possible en mémoire

```
void * mmap(  
  void * addr, size_t length, int prot,  
  int flags, int fd, off_t offset );
```



Mémoire partagée

- Une mémoire partagée est nommée, comme un fichier

```
int shm_open( const char * name, int oflag, mode_t mode );
```

- Mappage en mémoire

