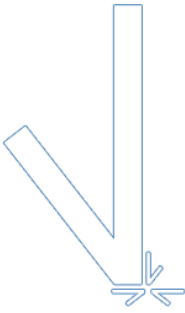


Systemes UNIX et Réseaux



Objectif:

Comprendre le fonctionnement
d'un système d'exploitation et
savoir configurer le réseau

I. Systèmes d'exploitation



Systèmes d'exploitation

I. Systèmes d'exploitation

I.1. Introduction



Introduction

I. Systèmes d'exploitation

I.1. Introduction

I.1.a Définitions

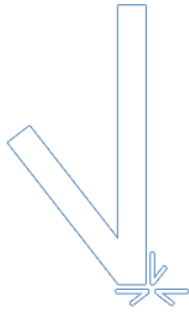


Définitions

I. Systèmes d'exploitation

I.1. Introduction

I.1.a Définitions

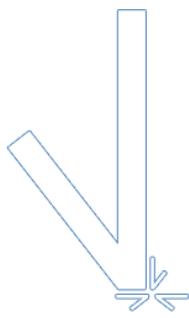


- **Système d'exploitation** : Ensemble de composants logiciels qui fournissent une **interface** entre le **matériel** et les **utilisateurs** ou les **applications utilisateur** d'un ordinateur.
- **Kernel** : Le kernel – **noyau** – est le composant logiciel du système d'exploitation qui fournit une **interface** entre le **matériel** et les **autres composants logiciels du système d'exploitation**. Il **initialise** les autres composants logiciels du système d'exploitation
- **Boot Loader** : Le boot loader – chargeur d'amorçage – est un logiciel **indépendant** du système d'exploitation qui **charge** le kernel dans la mémoire vive (RAM) de l'ordinateur. Il est stocké sur les **512 premiers octets** d'un périphérique de stockage.
- **BIOS** : **Basic Input/Output System** est un (micro)logiciel **indépendant** du système d'exploitation qui est stocké dans une mémoire morte (ROM) de l'ordinateur. Il est le **premier programme** lancé après la mise sous tension de l'ordinateur. Son rôle est d'**initialiser le matériel** autre que le microprocesseur, de **localiser le boot loader** sur les périphériques de stockage et de **charger le code du boot loader**.

I. Systèmes d'exploitation

I.1. Introduction

I.1.a Définitions



Mise sous tension

Temps

BIOS

- Stocké dans la mémoire morte;
- Initialise le matériel;
- Localise le Boot Loader;
- Charge le programme sur les **512 premiers octets** d'un périphérique de stockage : le Boot Loader.

Boot Loader

- Stocké sur les **512 premiers octets** d'un périphérique de stockage;
- Initialise un système de fichiers temporaire en mémoire vive;
- Charge le Kernel en mémoire vive.

Kernel

- Stocké sur un périphérique de stockage;
- Charge des scripts d'initialisation;
- Amorce un invite de commande ou une interface graphique.

Invite de
commande,
interface
graphique, ...

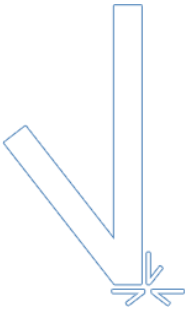
Le système
d'exploitation est
prêt à l'usage...

Amorçage du
Système d'Exploitation

I. Systèmes d'exploitation

I.1. Introduction

I.1.b Historique

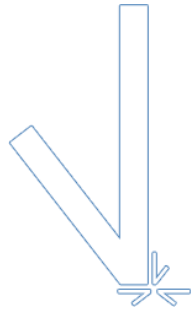


Historique

I. Systèmes d'exploitation

I.1. Introduction

I.1.b Historique

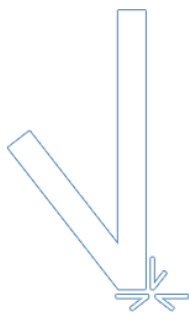


- **1956** : [GM-NAA I/O](#) (**G**eneral **M**otors – **N**orth **A**merican **A**viation **I**nut/**O**utput) est le 1^{er} système d'exploitation commercial de l'histoire pour l'[IBM 704](#). Les programmes fournis en entrée étaient exécutés les uns après les autres (*Batch Processing*).
...
- **1969** : [Unix](#) (originellement UNICS – **U**niplexed **I**nformation and **C**omputing **S**ervice) est créé au [Bell Labs](#). Il introduit les notions de système de fichiers hiérarchique, de communication inter-processus, de processus asynchrones, ...
...
- **1981** : [MS-DOS](#) (**M**icro**S**oft **D**isk **O**perating **S**ystem) est le premier système d'exploitation commercial de [Microsoft](#) pour PC compatibles avec les architectures IBM x86.
...
- **1985** : [Windows 1.0](#) est le premier système d'exploitation de Microsoft doté d'une interface graphique (**G**raphical **U**ser **I**nterface). Il est basé sur le kernel de **MS-DOS**.
...

I. Systèmes d'exploitation

I.1. Introduction

I.1.b Historique



- **1989** : [NeXTSTEP](#) est un système d'exploitation doté d'une interface graphique et basé sur un kernel libre de droits inspiré de **Unix**. Développé par *NeXT Computer*, racheté plus tard par **Apple**, il servira de base à la création de **macOS**.
- ...
- **1991** : [Linux](#) est un système d'exploitation développé par Linus Torvald qui s'inspire de l'architecture d'**Unix**. Linux est diffusé sous licence [GNU General Public License](#). C'est un logiciel libre de droits.
- ...
- **Actuellement :**
 - **Windows** n'est **plus basé** sur le **kernel** de **MS-DOS** depuis Windows NT (pour les professionnels) et Windows Millenium (pour les particuliers). Le kernel est propriétaire.
 - **MacOS** est un système d'exploitation basé sur un **kernel inspiré d'Unix**, le kernel [XNU](#) qui est libre de droits. Les autres composants du système sont généralement propriétaires. Idem pour **iOS**.
 - Le kernel **Linux** est utilisé par différents systèmes d'exploitations comme [Debian](#), [Ubuntu](#), [Android](#), etc. Ces systèmes d'exploitation peuvent être entièrement ou en partie libres de droits.

I. Systèmes d'exploitation

I.1. Introduction

I.1.c Systèmes UNIX

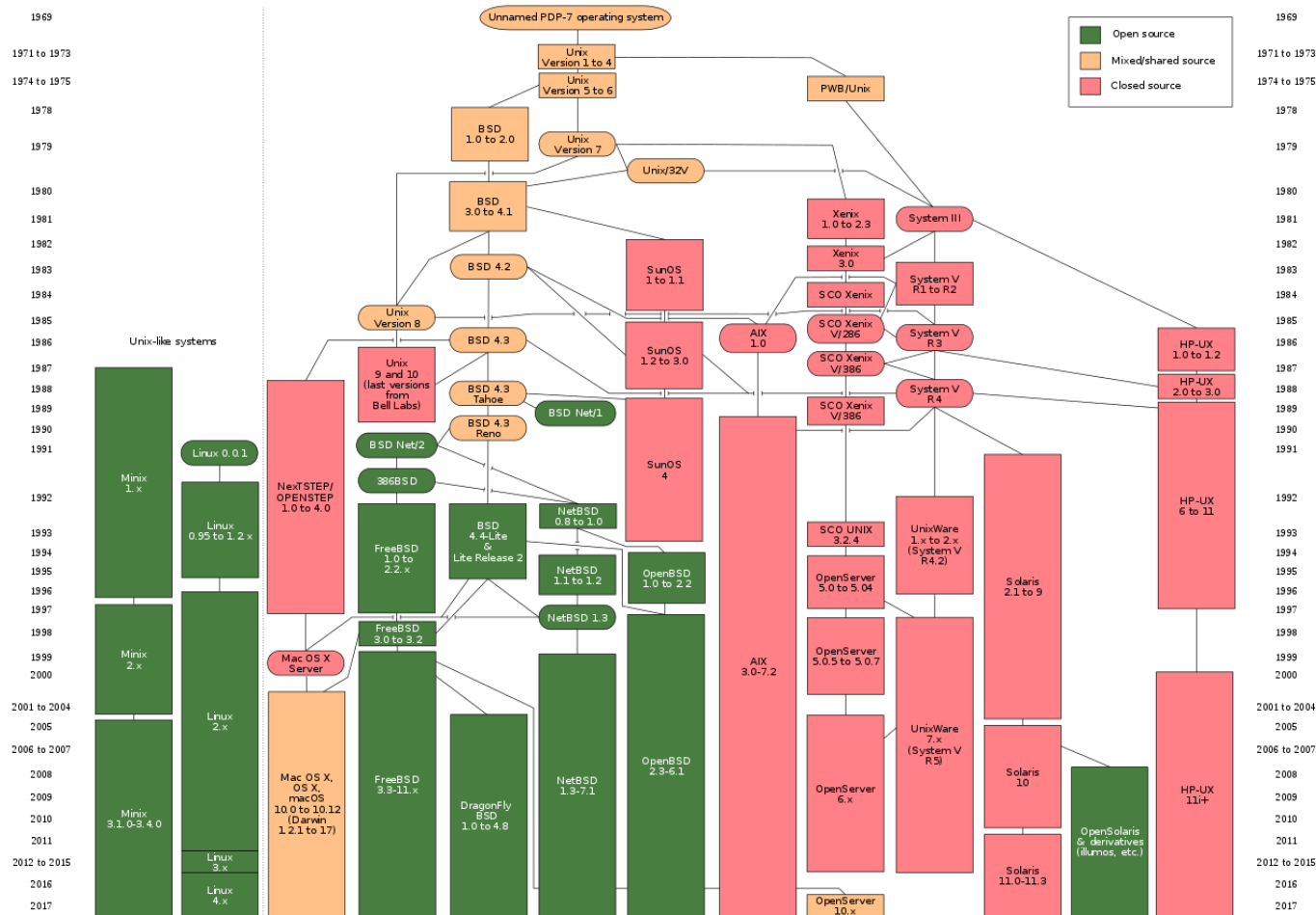
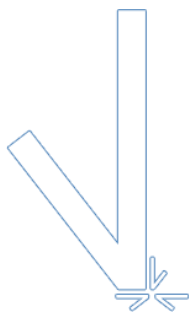


Systèmes UNIX

I. Systèmes d'exploitation

I.1. Introduction

I.1.c Systèmes UNIX



[source](#)

I. Systèmes d'exploitation

I.2. Principe de fonctionnement

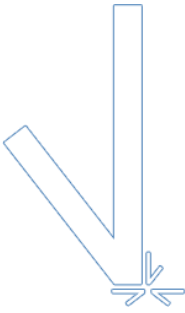


Principe de fonctionnement

I. Systèmes d'exploitation

I.2. Principe de fonctionnement

I.2.a Kernel



Kernel

I. Systèmes d'exploitation

I.2. Principe de fonctionnement

I.2.a Kernel

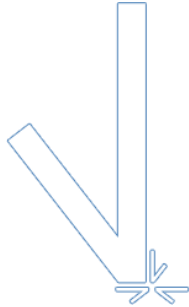


- Le **kernel** d'un système d'exploitation est **invisible** pour l'utilisateur.
- C'est le programme qui **permet** à d'autres programmes d'être exécutés.
- Il gère les **évènements** produits par la **couche matérielle**. Ces évènements sont appelés **Interruptions** – Interrupts – .
- Il gère les **évènements** produits par la **couche logicielle**. Ces évènements sont appelés **Appels Système** – System Calls –.
- Il gère les accès **aux ressources** matérielles ou logicielles.

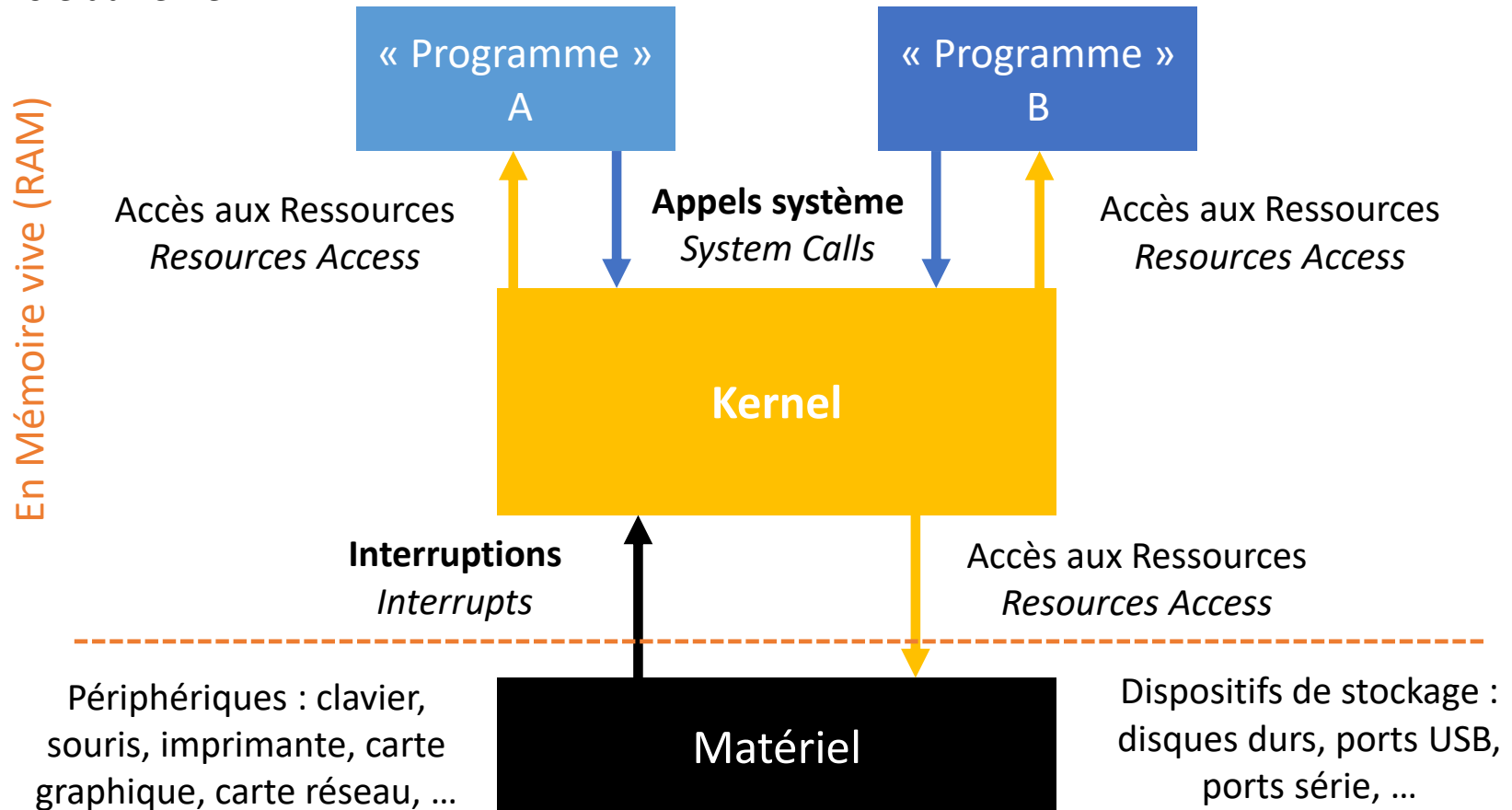
I. Systèmes d'exploitation

I.2. Principe de fonctionnement

I.2.a Kernel



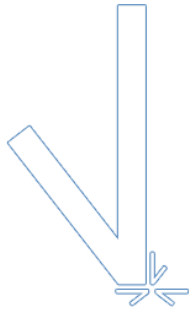
- Rôle du kernel :



I. Systèmes d'exploitation

I.2. Principe de fonctionnement

I.2.a Kernel

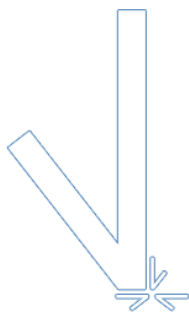


- Les **appels systèmes** doivent être programmés en **assembleur**. Pour gagner du temps on utilise plutôt des « librairies standard » – **standard librairies** – (généralement écrites en langage **C**) qui s'occupent de produire les appels systèmes tels qu'attendus par le kernel.
- Ces librairies standard sont **fournies** avec le système d'exploitation. Elles sont chargées et utilisées **dans** les programmes.
- Les systèmes (inspirés d'Unix) fournissent des librairies standard qui respectent un standard intitulé **POSIX** : **P**ortable **O**perating **S**ystem Interface. POSIX est un standard défini par l'IEEE (Institute of Electrical and Electronics Engineers) sous la référence [IEEE 1003](#).
- Le standard POSIX détaille également le fonctionnement d'**autres composants logiciels** du système d'exploitation comme l'invite de commande (le **shell**) par exemple.

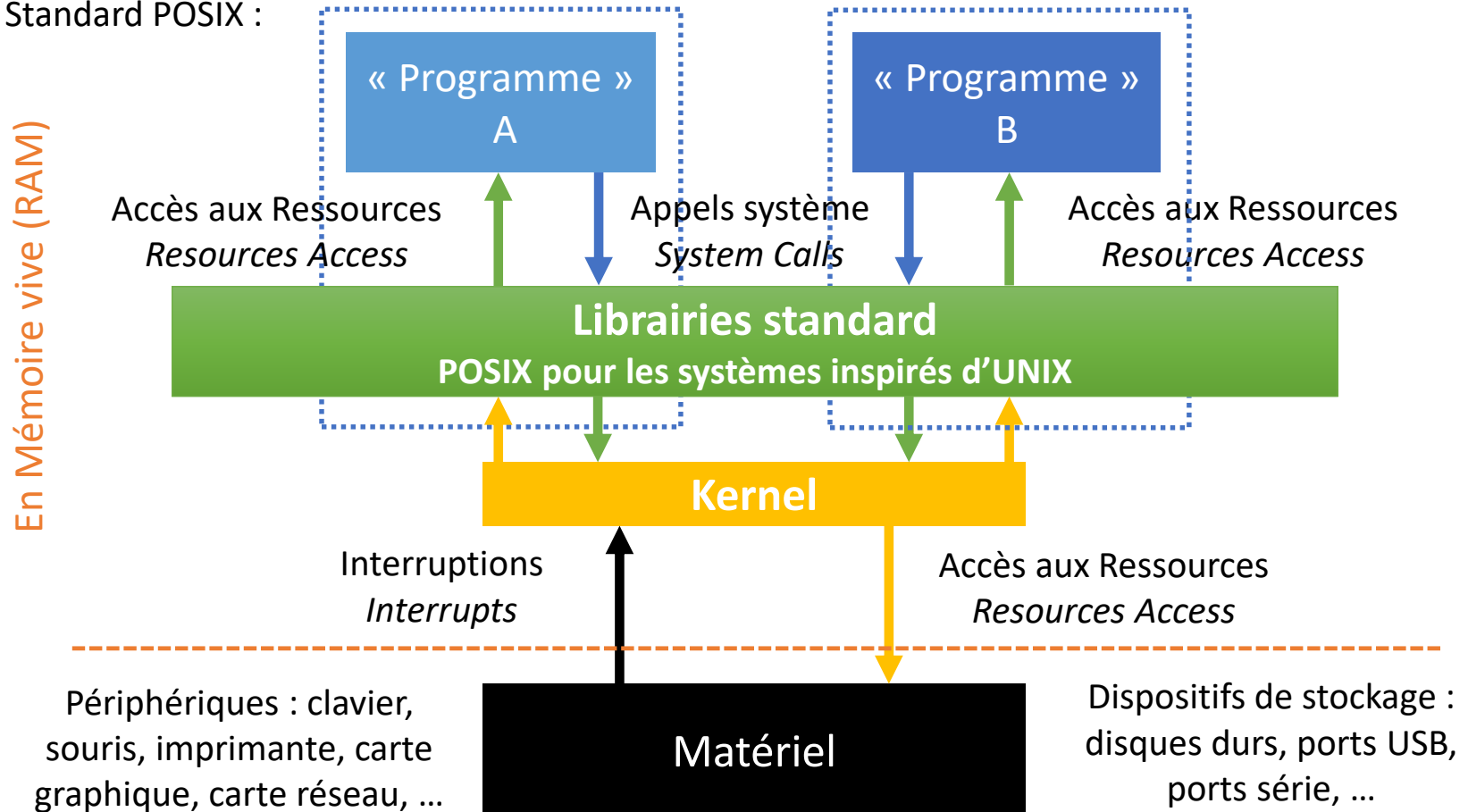
I. Systèmes d'exploitation

I.2. Principe de fonctionnement

I.2.a Kernel



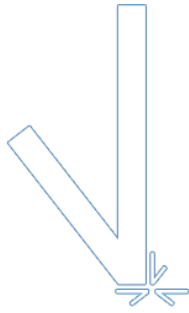
- Standard POSIX :



I. Systèmes d'exploitation

I.2. Principe de fonctionnement

I.2.a Kernel

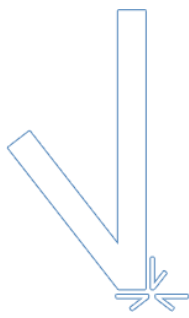


- Il existe **2 catégories** de kernel pour les systèmes UNIX.
- 1^{ère} approche : **Microkernel** :
 - Prend en charge la communication entre les programmes chargés en mémoire vive (Inter Process Communication) :
 - **IPC : communication inter processus**
 - Fournit une interface pour lire ou écrire dans la mémoire. Cette interface est appelée mémoire virtuelle, elle permet aux programmes de ne pas se soucier de l'emplacement **physique** exact où l'information sera stockée :
 - **Virtual Memory : mémoire virtuelle**
 - S'occupe d'allouer du temps d'exécution aux différents programmes chargés en mémoire. On appelle cela l'ordonnancement :
 - **Scheduling : ordonnancement**

I. Systèmes d'exploitation

I.2. Principe de fonctionnement

I.2.a Kernel

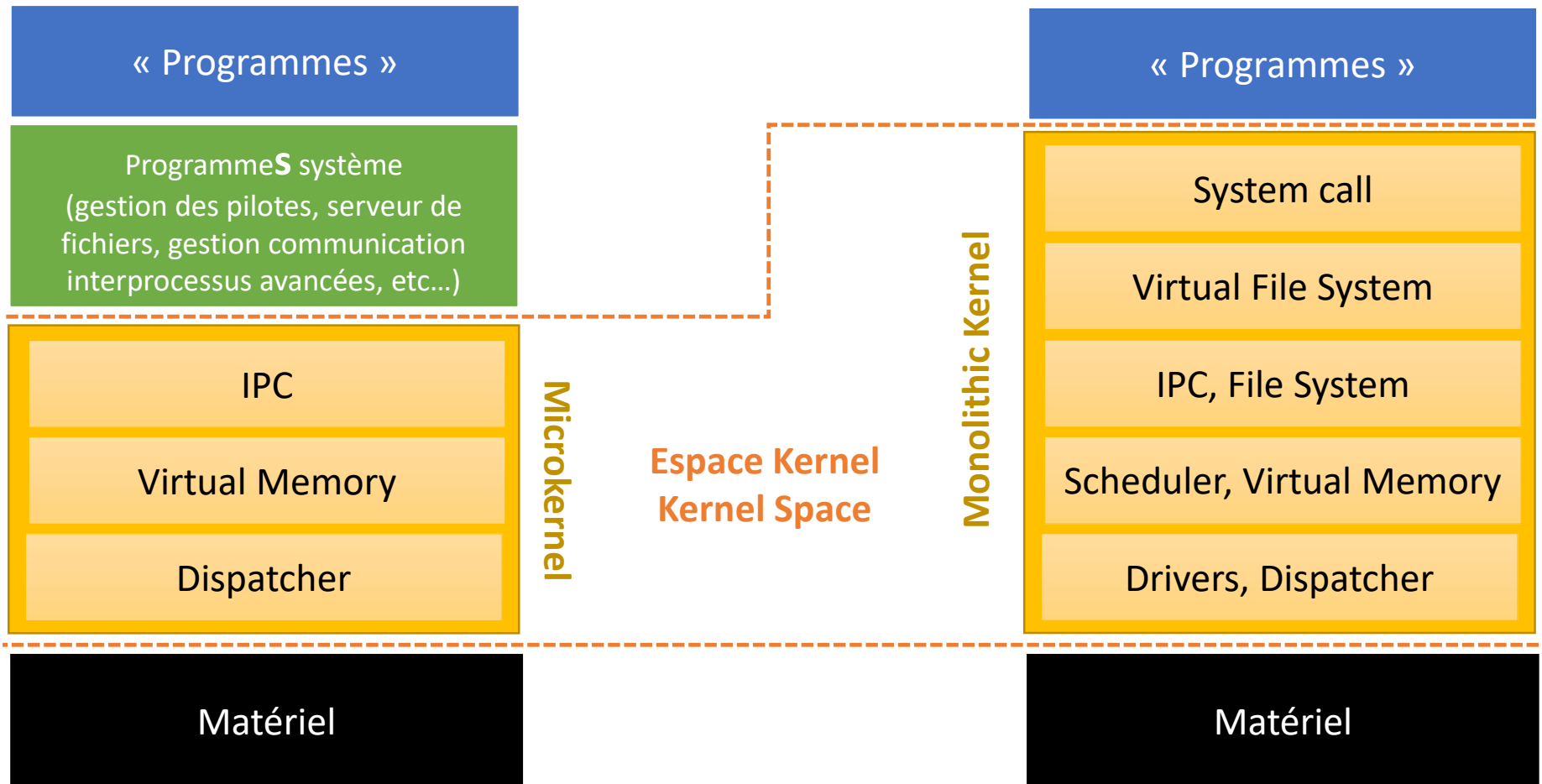
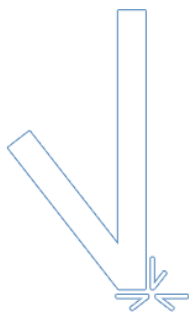


- 2^{ème} approche : **Monolithic kernel** :
 - Reprend les fonctionnalités de l'approche Microkernel :
 - **IPC** : communication inter processus, **Virtual Memory** : mémoire virtuelle, **Scheduling** : ordonnancement.
 - Contient les « pilotes » de périphériques, sous forme de modules. Il s'agit du code responsable de gérer les événements relatifs aux périphériques du système :
 - **Device Drivers** : pilotes de périphériques
 - S'occupe d'accorder du temps processeur aux différentes piles d'exécution (« sous-ensemble d'instructions ») d'un programme en cours d'exécution :
 - **Dispatcher** : distribution de temps processeur
 - Gère le stockage et fournit une interface pour accéder au stockage sous la forme d'un système de fichier :
 - **File System, Virtual File System** : système de fichier et système de fichier virtuel

I. Systèmes d'exploitation

I.2. Principe de fonctionnement

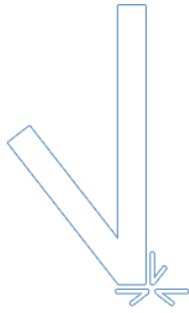
I.2.a Kernel



I. Systèmes d'exploitation

I.2. Principe de fonctionnement

I.2.a Kernel



- Exemple de systèmes d'exploitation inspirés d'UNIX à **microkernel** :
 - [Minix](#).
 - ...
- Exemple de systèmes d'exploitation inspirés d'UNIX à **kernel monolithique** :
 - **Linux et dérivés.**
 - ...
- Pour **créer un kernel**, on peut programmer en :
 - **Assembleur** (pour les plateformes x86, par exemple, pour un kernel de plateforme matérielle de type PC compatible);
 - **C/C++** en utilisant des libraires dédiées qui s'occuperont de générer le code en assembleur ([le site Internet OSDev.org recense toutes les ressources à ce sujet](#)).
- On s'intéressera **principalement** aux systèmes d'exploitation à **kernel monolithique**.

I. Systèmes d'exploitation

I.2. Principe de fonctionnement

I.2.b Drivers

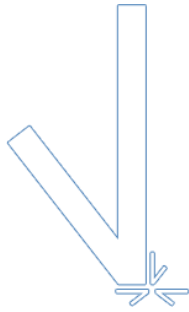


Drivers

I. Systèmes d'exploitation

I.2. Principe de fonctionnement

I.2.b Drivers

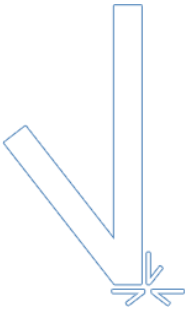


- Les pilotes – **drivers** – de périphériques sont des extraits de programmes qui :
 - **Font partie** initialement du kernel
 - Ou sont **insérés** dynamiquement dans le kernel en cours de fonctionnement sous la forme de « **modules** ».
- Les pilotes **ajoutent** au kernel :
 - La **prise en charge** de certaines **interruptions** entre certains périphériques matériels et le kernel.
 - La **prise en charge** de certains **appels systèmes** entre certaines applications logicielles et le kernel.
- Les pilotes sont généralement :
 - Fournis par le fabricant du périphérique ou développés par des membres de la communauté des contributeurs au système d'exploitation;
 - Sous licence propriétaire ou libre de droits.

I. Systèmes d'exploitation

I.2. Principe de fonctionnement

I.2.b Drivers



- Sur linux, la **liste** des « **modules** » insérés dynamiquement dans le kernel peut être consultée via l'invite de commande en utilisant l'utilitaire :

lsmod

- De **nouveaux modules** peuvent être ajoutés à l'aide de l'utilitaire :

modprobe <fichier>

(modprobe suivi du chemin vers le fichier du module)

- On peut obtenir des **informations** concernant un module à l'aide de l'utilitaire :

modinfo <nom du module>

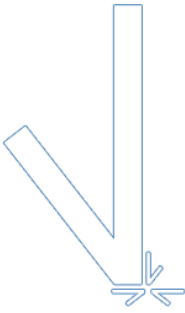
(modinfo suivi du nom du module obtenu avec lsmod)

- Il existe d'**autres** outils et fichiers qui peuvent être utilisés pour configurer les modules.

I. Systèmes d'exploitation

I.2. Principe de fonctionnement

I.2.c Gestion de la mémoire

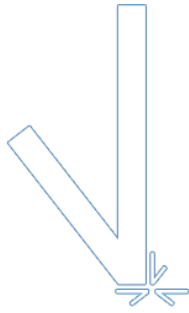


Gestion de la mémoire

I. Systèmes d'exploitation

I.2. Principe de fonctionnement

I.2.c Gestion de la mémoire

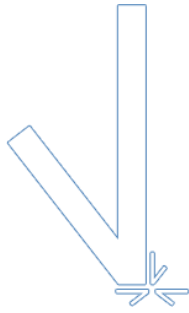


- Le kernel offre une **interface** entre la mémoire physique installée sur le matériel et les programmes. Cette interface s'appelle **mémoire virtuelle – virtual memory** –.
- Les programmes (y compris ceux faisant partie du système d'exploitation) se **réfèrent et utilisent toujours** des **adresses en mémoire virtuelle**. Le kernel « transcrit » **automatiquement** ces adresses en adresses physique réelles.
- La mémoire virtuelle présente les données de façon **contiguë** aux programmes même si, physiquement, les données peuvent être stockées sur plusieurs emplacements différents de la mémoire.
- Les données sont stockées par le kernel, sur la mémoire physique, sous forme de « **pages** » de **données** de (généralement) 4 Kilo-Octets. Les programmes sont donc chargés en mémoire physique sous la forme de 1 ou plusieurs « pages » de données.

I. Systèmes d'exploitation

I.2. Principe de fonctionnement

I.2.c Gestion de la mémoire



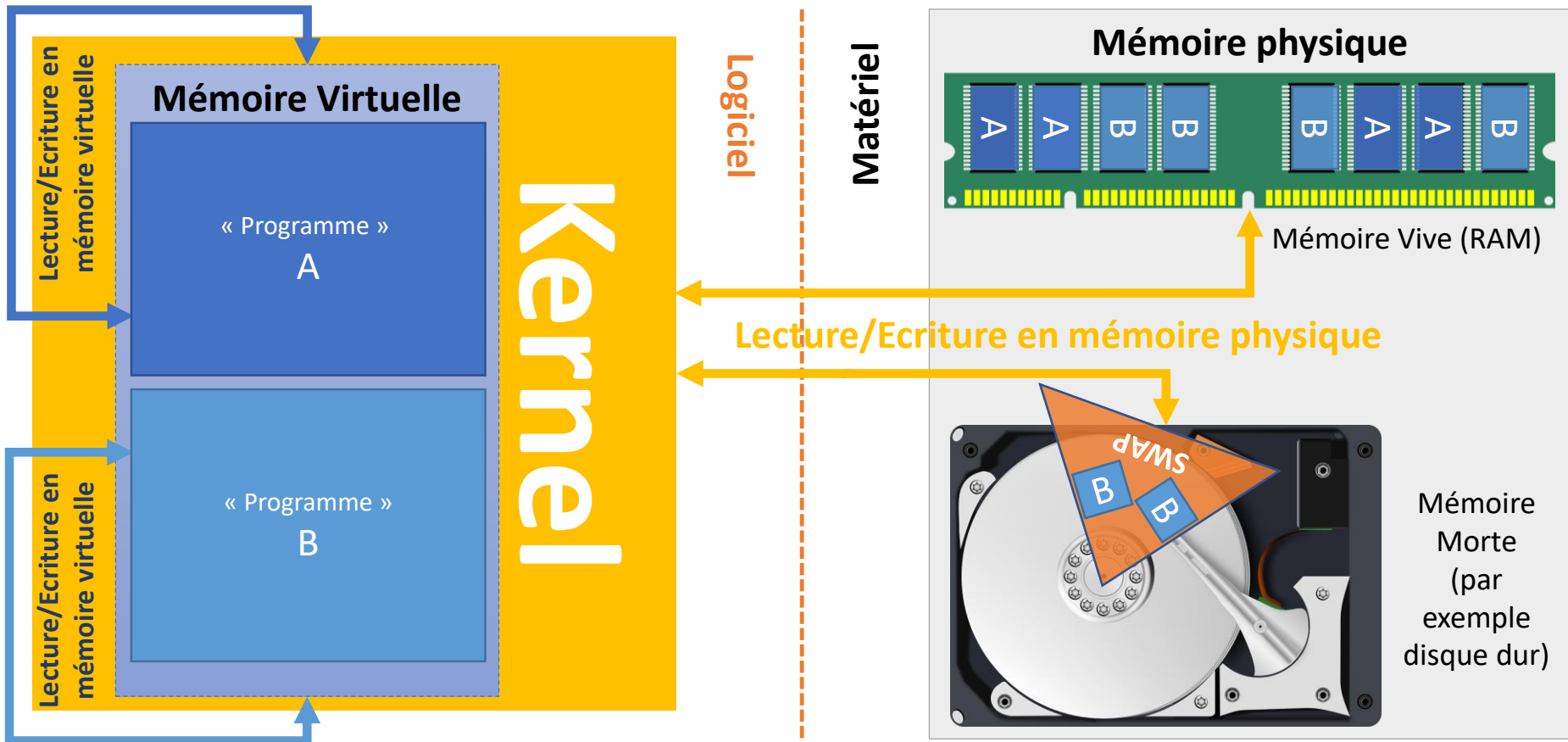
- Le kernel stocke les « pages » de données dans la **mémoire vive** (Random Access Memory – **RAM**).
 - Les accès en lecture/écriture à la mémoire vive sont très rapides mais les données ne sont pas conservées si l'ordinateur est mis hors tension.
- Si le kernel n'a plus d'espace à sa disposition pour stocker les « pages » de données en mémoire vive, il les stocke en **mémoire morte** (disque dur – hard drive) dans un espace de stockage réservé appelé **swap**.
 - Les accès en lecture/écriture à la mémoire morte sont très lents mais les données sont conservées si l'ordinateur est mis hors tension.
- Si le kernel n'a **plus d'espace** à sa disposition pour stocker les « pages » de données en mémoire vive ou morte, alors :
 - Il est **impossible de charger** de nouveaux programmes en mémoire;
 - Les programmes déjà chargés en mémoire **ne peuvent plus utiliser de mémoire** et sont donc susceptibles de **s'arrêter inopinément**.

I. Systèmes d'exploitation

I.2. Principe de fonctionnement

I.2.c Gestion de la mémoire

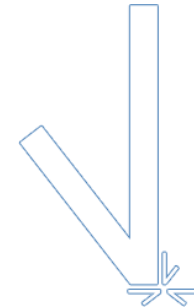
- Mémoire virtuelle et mémoire physique :



I. Systèmes d'exploitation

I.2. Principe de fonctionnement

I.2.c Gestion de la mémoire



- Sur linux, on peut obtenir des informations générales concernant l'**utilisation** de mémoire vive (RAM) et morte (SWAP) physique à l'aide de l'utilitaire :

free -h

(free suivi de l'argument -h)

- Pour obtenir des informations détaillées concernant la **mémoire physique installée**, on peut se servir de l'utilitaire :

dmidecode

- Pour obtenir des informations concernant la **mémoire virtuelle**, on peut se servir de l'utilitaire :

vmstat

I. Systèmes d'exploitation

I.2 Principe de fonctionnement

I.2.d Programmes et Processus

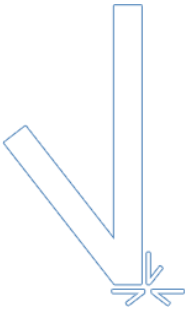


Programmes et Processus

I. Systèmes d'exploitation

I.2. Principe de fonctionnement

I.2.d Programmes et Processus

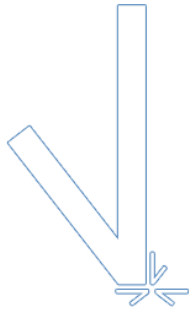


Notion de Processus

I. Systèmes d'exploitation

I.2. Principe de fonctionnement

I.2.d Programmes et Processus

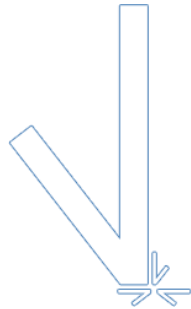


- Un **programme** est un **ensemble d'instructions** stockées en mémoire vive ou morte qui sont **inactives**.
- Un **processus** est un programme chargé en mémoire [virtuelle] qui est **en cours d'exécution**.
- C'est le **système d'exploitation**, en particulier le kernel, qui **charge les programmes** en mémoire [virtuelle] **et les exécute**. On peut alors parler de **processus en mémoire**.
- Un seul et même processus donner lieu à la création de plusieurs processus :
 - En demandant l'**exécution d'autres programmes**;
 - Ou **en se dupliquant en mémoire**.
- On peut alors parler de **processus enfant – child process –** .
- L'opération qui consiste à dupliquer un processus en mémoire s'appelle **fork**. Il s'agit d'un **appel système** qui peut être effectué par un processus pour demander sa duplication.

I. Systèmes d'exploitation

I.2. Principe de fonctionnement

I.2.d Programmes et Processus

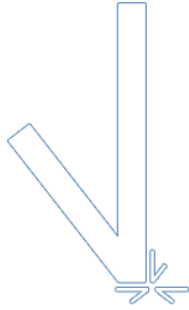


- L'**espace mémoire alloué à un processus** se compose de différentes parties : des **segments mémoire**. Ces segments mémoire sont :
 - Le **code** : Correspond à un **espace de mémoire fixé au départ**. Contient le [code du] programme, est consulté en lecture uniquement par le système d'exploitation.
 - Les **segments de données – data segment** – : Correspond à un **espace de mémoire fixé au départ**. Contient les variables globales initialisées (les **data**) et les variables globales non initialisées (les **bss – block started by symbol** –).
 - Le segment de **mémoire dynamique – heap** – : Correspond à un **espace de mémoire alloué dynamiquement** et accessible **globalement** (partout dans le code en cours d'exécution). Contient des variables dont la taille est variable et dont **la taille maximale dépend uniquement des limitations matérielles**.
 - Le segment de mémoire pour la **pile d'exécution – stack** – : Correspond à un **espace de mémoire alloué dynamiquement** et accessible **localement** (uniquement à partir de la fonction du code en cours d'exécution). Contient des variables dont la taille est variable et dont **la taille maximale est limitée par le système d'exploitation**.

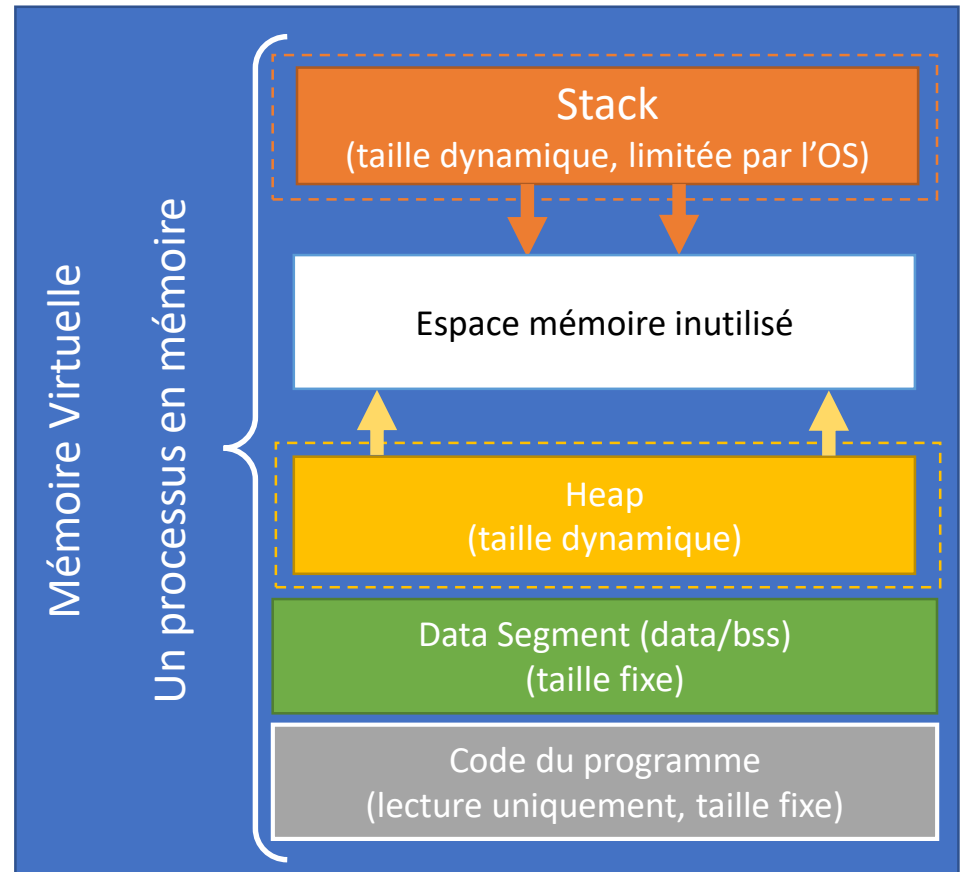
I. Systèmes d'exploitation

I.2. Principe de fonctionnement

I.2.d Programmes et Processus



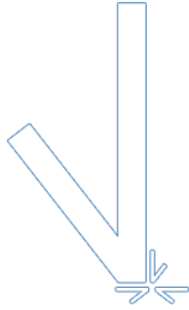
- Structure d'un processus en mémoire :
- **Un processus n'a pas accès aux espaces mémoire alloués à d'autres processus.**
- Les variables déclarées lors de l'exécution d'une fonction sont créées dans la **stack**.
- En sortie de fonction, toutes les variables créés dans la stack sont supprimées automatiquement.
- **Les fonctions d'un même processus n'ont donc pas accès aux variables créés par d'autres fonctions du même process.**
- Les variables créées dans la **heap** ou dans le **data segment** sont accessibles à tout moment par n'importe quelle fonction. Les variables de la **heap** doivent être supprimées manuellement.



I. Systèmes d'exploitation

I.2. Principe de fonctionnement

I.2.d Programmes et Processus



- Structure d'un processus en mémoire :
- La taille totale de la **stack** est limitée par le système d'exploitation. Par défaut sur linux elle est de 8192 kilo-octets. Pour connaître la limite de taille on peut utiliser l'utilitaire :

ulimit -s

(ulimit suivi de l'argument -s)

- La taille totale disponible pour les autres espaces mémoire d'un processus est limitée par la couche matérielle :

espace total inutilisé en mémoire virtuelle (ram + swap)

-

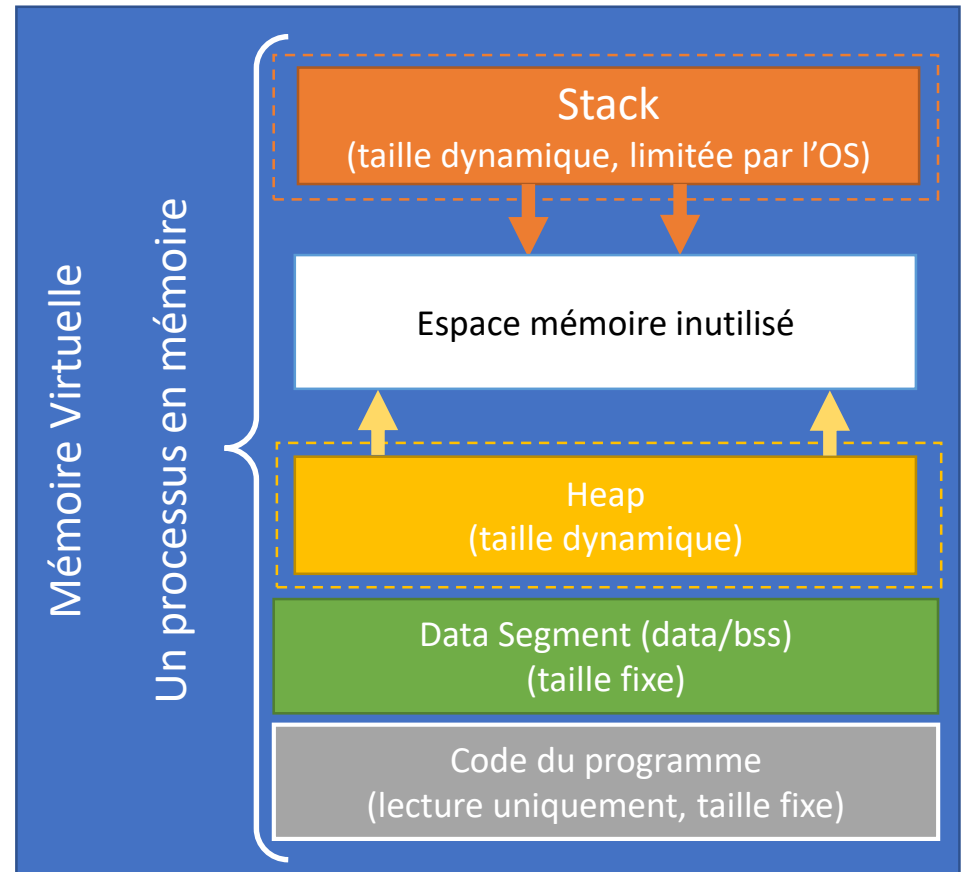
espace déjà alloué en mémoire virtuelle (ram + swap)

-

8192 kilo-octets (pour la stack)

=

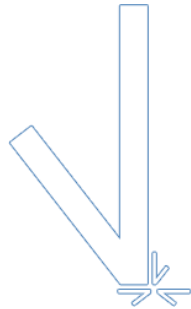
espace disponible pour les espaces mémoire (hors stack)
nécessaires à l'exécution d'un processus



I. Systèmes d'exploitation

I.2. Principe de fonctionnement

I.2.d Programmes et Processus



- Lorsque la limite de taille de la stack est atteinte, le système d'exploitation émet une erreur de type **stack overflow** et le processus est interrompu.
 - L'écriture de fonctions récursives avec une condition de sortie erronée est souvent la cause de ce type d'erreur.
- La liste des processus en mémoire peut être consultée à l'aide de l'utilitaire :

ps -aux

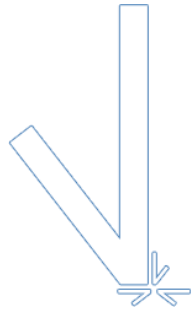
(ps suivi des arguments -aux)

- Informations relatives à un processus :
 - **USER** : l'utilisateur à l'origine du processus;
 - **PID** : l'identifiant unique du processus;
 - **%CPU** et **%MEM** : consommation en ressources CPU et Mémoire.
 - **VSZ – Virtual Memory Size –** : Espace occupé en mémoire virtuelle.
 - **RSS – Resident Set Size –** : Espace occupé en mémoire vive uniquement.
 - **TTY – TeleTYpewriter –** : Terminal à partir duquel le processus a été démarré.
 - **STAT** : [Codes d'état du processus](#).
 - **START** : Date de démarrage du processus.
 - **TIME** : Temps CPU consommé.
 - **COMMAND** : Programme à l'origine du processus.

I. Systèmes d'exploitation

I.2. Principe de fonctionnement

I.2.d Programmes et Processus



- Pour communiquer avec les processus on peut utiliser :

kill -<signal> <pid>

(kill suivi du code signal à envoyer au processus et du PID du processus)

- Pour connaître la liste des signaux possibles :

kill -l

(kill suivi de l'argument -l)

- Les signaux envoyés aux processus peuvent être **pris en charge par le processus** pour entraîner un **comportement particulier** du processus. Si il ne sont **pas pris en charge** par le processus, ils entraînent un **comportement par défaut**.
- Pour arrêter un processus on peut utiliser :

kill -9 <pid>

(kill suivi de l'argument -9 pour SIGKILL – tuer le processus – et du PID du processus)

I. Systèmes d'exploitation

I.2. Principe de fonctionnement

I.2.d Programmes et Processus

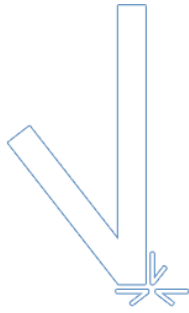


Notion de Threads

I. Systèmes d'exploitation

I.2. Principe de fonctionnement

I.2.d Programmes et Processus



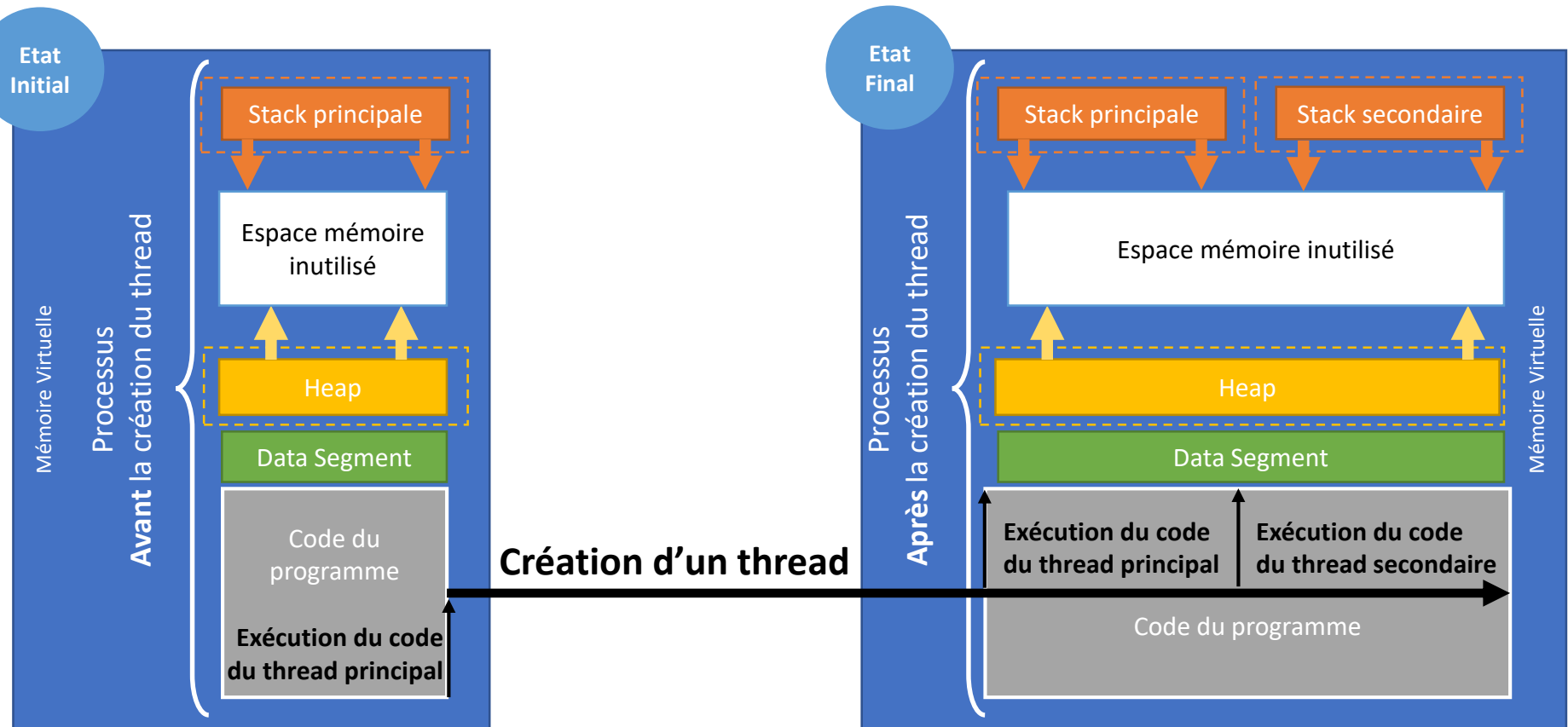
- Un **code en cours d'exécution** au sein d'un processus peut demander l'exécution du même code « *en parallèle* » de sa propre exécution.
- Le **code en cours d'exécution** d'un processus est appelé **thread principal**. Le **thread principal** peut demander l'exécution d'un ou plusieurs **threads secondaires** qui seront exécutés en « parallèle »
- Le **Dispatcher** du kernel s'occupe d'accorder un peu de temps CPU à chaque **thread**. D'un point de vue matériel, le nombre de cœur du CPU détermine le nombre de threads pour lesquels le code sera réellement exécuté en parallèle.
- Lorsqu'un nouveau thread est créé, la **stack** utilisée par le thread créateur est **dupliquée** et **utilisée** par le nouveau thread.
- Le reste des données, relatives au processus, sont **partagées** par les différents threads démarrés par le processus. D'où le problème à résoudre des **accès concurrents**.

I. Systèmes d'exploitation

I.2. Principe de fonctionnement

I.2.d Programmes et Processus

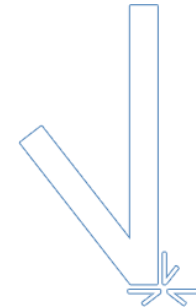
- Création d'un thread :



I. Systèmes d'exploitation

I.2. Principe de fonctionnement

I.2.d Programmes et Processus



- Contrairement à la stack du thread principal d'un processus dont la taille maximale est fixée par défaut à 8192 kilo-octets (8Mo), la taille maximale de la stack d'un thread secondaire d'un processus est fixée par défaut à 2048 kilo-octets (2Mo).
- Le **nombre maximal de threads** qui peuvent être créés par un processus est fixé par le système d'exploitation. Pour obtenir cette information on peut consulter le contenu du fichier :

```
cat /proc/sys/kernel/threads-max
```

(utilitaire `cat` suivi du fichier dont le contenu doit être affiché)

- On peut visualiser les threads démarrés par un processus à l'aide de l'utilitaire :

```
ps -Tp <PID>
```

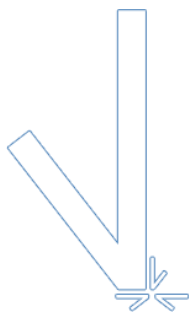
(`ps` suivi des arguments `-Tp` et du PID du processus)

- Informations relatives à un thread :
 - **PID** : l'identifiant unique du processus.
 - **SPID** : l'identifiant unique du thread.
 - **Time** : Temps CPU consommé.
 - **CMD** : Programme à l'origine du processus et des threads.

I. Systèmes d'exploitation

I.2. Principe de fonctionnement

I.2.d Programmes et Processus



- Les threads **concurrents** ont accès en lecture et en écriture aux variables globales (dans le **data segment**) et à taille dynamique (dans le **heap**).
- Il est nécessaire, lorsque le processus est « **multithread** », de **synchroniser** l'accès aux variables pour empêcher un thread d'accéder en lecture ou écriture à une variable pendant qu'un autre thread y accède en lecture ou en écriture.
- La section de code pour laquelle la synchronisation est nécessaire s'appelle **section critique** – **critical section** –.
- Parmi les algorithmes de synchronisation qui sont utilisés pour éviter les accès concurrents, on citera :
 - La technique de **MutEX** (EXclusion Mutuelle) ou encore **sémaphores d'exclusion mutuelle** qui permet de **protéger l'accès à une ressource** lorsque celle-ci est utilisée par un thread;
 - La technique du **Sémaphore** ou encore **sémaphores numériques** qui permet de **mettre en attente** des threads avant une section critique et, à un thread qui n'est pas en attente, de **signaler**, en sortie de section critique, aux autres threads qu'ils peuvent reprendre leur exécution.

I. Systèmes d'exploitation

I.2. Principe de fonctionnement

I.2.d Programmes et Processus



Notion de processus enfants

I. Systèmes d'exploitation

I.2. Principe de fonctionnement

I.2.d Programmes et Processus



- Un processus peut charger et démarrer un autre processus :
 - Si c'est un processus totalement différent, on parle de **processus enfant – child process –** .
 - Si c'est le même processus qui est dupliqué et qui doit reprendre l'exécution du code à partir de *l'appel système* de clonage, on parle de **fork** (processus enfant qui est un clone du processus original mais qui reprend au moment où *l'appel système* de clonage a eu lieu).
- Le processus père peut conserver le PID de ses processus enfant.
- Les processus **ne partagent pas** leurs **espaces mémoire**.
- On peut visualiser les processus enfant des processus démarrés à l'aide de l'utilitaire :

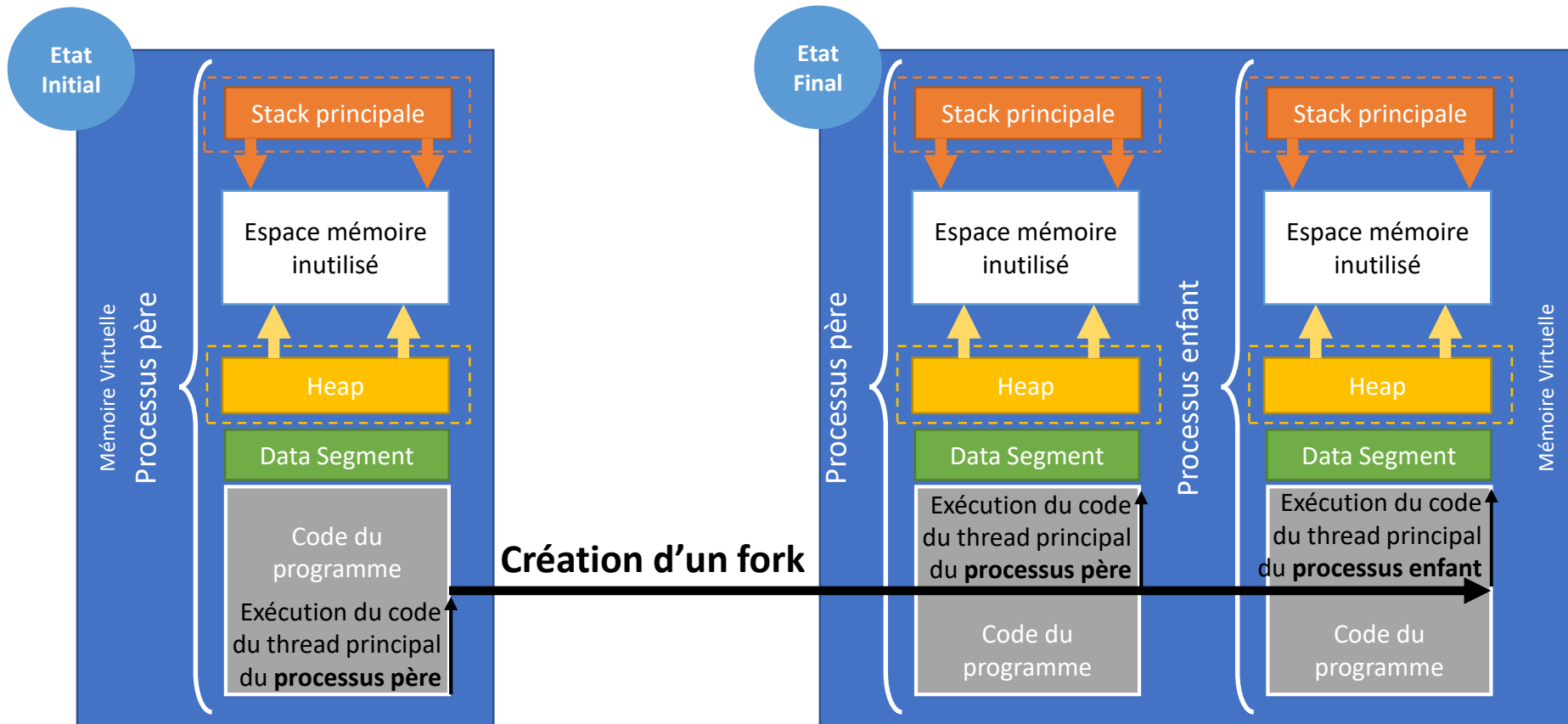
ps tree

I. Systèmes d'exploitation

I.2. Principe de fonctionnement

I.2.d Programmes et Processus

- Création d'un processus enfant de type **fork** :



I. Systèmes d'exploitation

I.2. Principe de fonctionnement

I.2.d Programmes et Processus

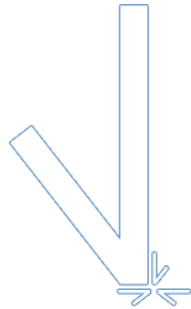


Communication Inter Processus - IPC

I. Systèmes d'exploitation

I.2. Principe de fonctionnement

I.2.d Programmes et Processus

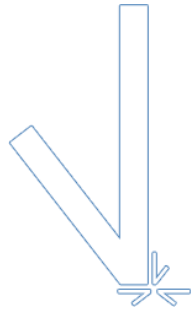


- Bien que les processus **ne partagent pas** leurs espaces mémoire, le système d'exploitation propose plusieurs mécanismes (les [IPC](#)) pour échanger des données entre les processus :
 - Les signaux – **Signals** – : les signaux sont des codes numériques qui peuvent être émis par des processus via des *appels système*. Les processus peuvent déclencher du code de façon asynchrone en cas de réception d'un signal via l'utilisation des fonctions standard;
 - Les messages et file d'attente de messages – **Messages and Message Queues** – : les messages sont un mécanisme qui permet d'échanger des données entre 2 processus. Le processus expéditeur envoie son message au kernel au moyen d'un *appel système*. Ce dernier place le message dans une file d'attente. Le processus récepteur peut effectuer un *appel système* pour récupérer le message dans la file d'attente. Un message est limité par défaut à 8192 octets (8 Ko).
 - Les tubes – **Pipes** – : les « pipes » sont un **canal de communication unidirectionnel** entre les processus. Un processus peut écrire des données dans un « pipe ». Un autre processus peut lire les données reçues dans un « pipe ». Le volume de données qui peut être échangé en une fois à l'aide d'un pipe est limité à 65536 octets (65 Ko);
 - La mémoire dynamique partagée – **Shared Heap Memory (SHM)** – : Il s'agit de segments de mémoire qui peuvent être partagés entre les processus. Leur utilisation nécessite de définir des permissions d'accès pour que seuls les processus autorisés puissent accéder aux segments mémoire qui les concernent.

I. Systèmes d'exploitation

I.2. Principe de fonctionnement

I.2.d Programmes et Processus



- Les *appels système* pour utiliser les messages, les « pipes », les signaux, et la SHM peuvent effectués par les programmes via des *appels systèmes* se basant sur l'API Standard POSIX mise à disposition par le système d'exploitation.
- L'exploitation de la SHM nécessite de gérer les **accès concurrents entre les processus**. C'est pourquoi le kernel propose un mécanisme permettant d'échanger des valeurs de sémaphore.
- Pour afficher la liste des **messages, segments en SHM et sémaphores**, on peut utiliser l'utilitaire :

ipcs

- Pour utiliser les « pipes » entre des processus lancés à partir de l'invite de commande (*shell*), on peut utiliser le symbole `|` comme par exemple avec les utilitaires `ls` et `wc` :

ls | wc

utilitaire `ls` : liste les fichiers

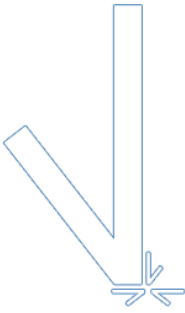
liste qui est envoyée à l'aide d'un « pipe » | dans

utilitaire `wc` : compte le nombre de lignes, de mots et d'octets dans les données fournies (ici la liste de fichiers)

I. Systèmes d'exploitation

I.2 Principe de fonctionnement

I.2.e Ordonnancement

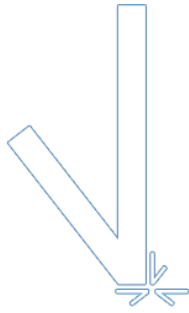


Ordonnancement

I. Systèmes d'exploitation

I.2. Principe de fonctionnement

I.2.e Ordonnancement



- L'ordonnancement est l'opération par laquelle le kernel alloue du temps d'exécution CPU à chaque processus chargé en mémoire. Le composant du kernel responsable de cette opération s'appelle le **Scheduler**.
- Le **Scheduler** du kernel de Linux ([CFS – Completely Fair Scheduler](#)) est basé sur un algorithme de type **Round Robin**.
- L'algorithme Round Robin **alloue un temps fixé de CPU** pour l'exécution tour à tour du code de chaque processus. Un mécanisme de **priorité** des processus permet d'allouer plus ou moins de temps CPU à un processus en particulier.
- On peut assigner ou modifier la priorité d'un processus à l'aide, respectivement, des utilitaires suivants :

nice -n 5 <chemin vers le programme à charger>

(`nice` suivi des arguments `-n` et de la priorité, puis du chemin vers le programme à démarrer)

renice -n 5 -p <pid>

(`renice` suivi des arguments `-n` et de la priorité, `-p` et du PID du processus)

I. Systèmes d'exploitation

I.3 Système de fichiers

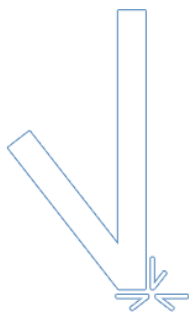


Système de fichiers

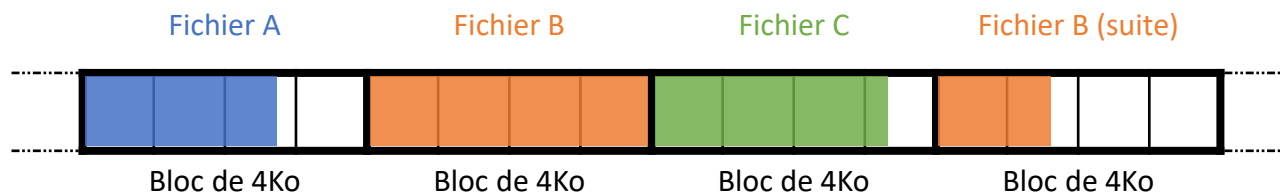
I. Systèmes d'exploitation

I.3. Système de fichiers

I.3.a Fichiers



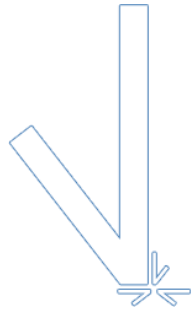
- La mémoire morte (disque dur, clé USB, ...) contient de l'information sous la forme d'**octets**.
- Ces octets représentent des **données** (texte, son, image, ...) et des **métadonnées** (données à propos des données) qui identifient la **nature des données**.
- Les **métadonnées et données** associées sont appelées **fichier – file –**.
- L'espace disponible sur un support de stockage est généralement découpé en blocs de 4096 octets (4 Kilo-octets). Ces blocs sont appelés **allocation unit** (unité d'allocation) ou **cluster** (groupe).
- Les fichiers sont stockés sur **1 ou plusieurs blocs non contigus** :



I. Systèmes d'exploitation

I.3. Système de fichiers

I.3.b inodes

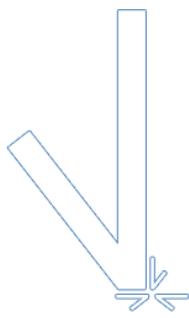


- Pour reconstituer l'intégralité d'un fichier sur le support physique, il faut un système de « sommaire ». Ce système de « sommaire » doit contenir à minima :
 - Un **nom** qui permet d'identifier le fichier;
 - La **liste des blocs** auxquels il faut accéder pour reconstituer la donnée.
- la **liste des blocs** constituant un fichier est stockée sur le support physique dans une structure de données appelée **nœud d'index** – **index node** – ou encore **inode**. Un inode contient donc les informations relatives à l'emplacement d'un fichier sur le support physique.
- Les noms des fichiers sont stockés dans des fichiers qui **associent 1 nom avec le numéro d'1 inode**. Ces fichiers sont appelés **dossiers** – **directories** –. Chaque nom dans un dossier est associé à un numéro d'inode. **Un dossier est donc un fichier**.
- Chaque **nom**, dans un dossier, est donc associé à un numéro d'**inode**. Un **inode** peut :
 - Pointer vers des blocs : on parle alors de lien dur – **hard link** –.
 - Pointer vers un autre inode : on parle alors de lien symbolique – **symbolic link** –.

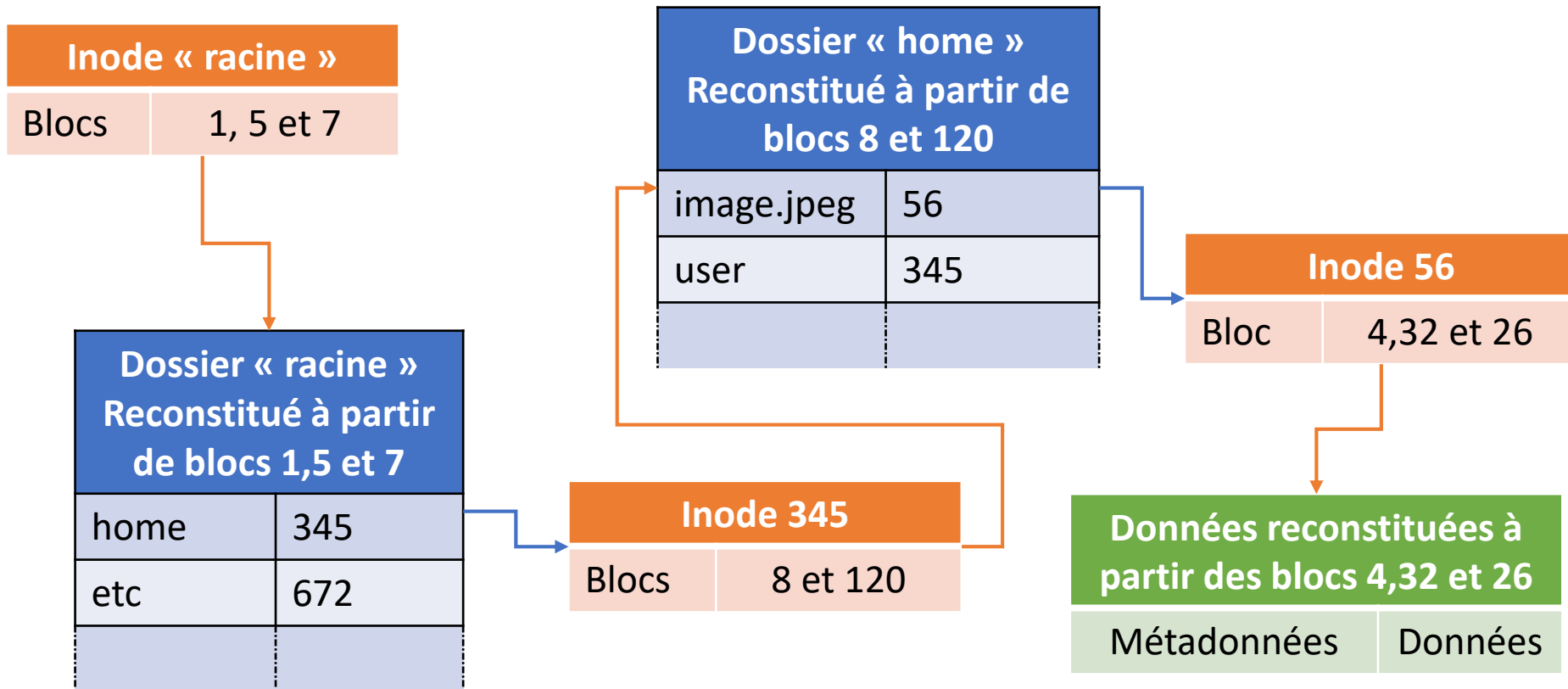
I. Systèmes d'exploitation

I.3. Système de fichiers

I.3.b inodes



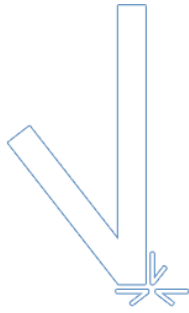
- Principe de fonctionnement des **inodes** (exemple) :



I. Systèmes d'exploitation

I.3. Système de fichiers

I.3.b inodes



- On peut consulter la **liste de noms associés à des inodes** d'un dossier à l'aide de l'utilitaire :

ls -i <chemin vers le dossier>

(ls suivi de l'argument -i et du chemin vers le dossier)

- On peut consulter le **contenu de l'inode** correspondant à un fichier à l'aide de l'utilitaire :

stat <chemin vers le fichier>

(stat suivi du nom du fichier, ou dossier, dont on souhaite consulter l'inode)

- On peut créer un **hardlink** vers un fichier à l'aide de l'utilitaire :

ln <chemin vers le fichier cible> <nom>

(ln suivi du chemin vers le fichier et du nouveau nom de fichier)

- On peut créer un **symbolic link** ou **symlink** vers un fichier à l'aide de l'utilitaire :

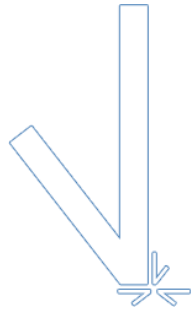
ln -s <chemin vers le fichier cible> <nom>

(ln suivi de l'argument -s, du chemin vers le fichier et du nom de lien symbolique)

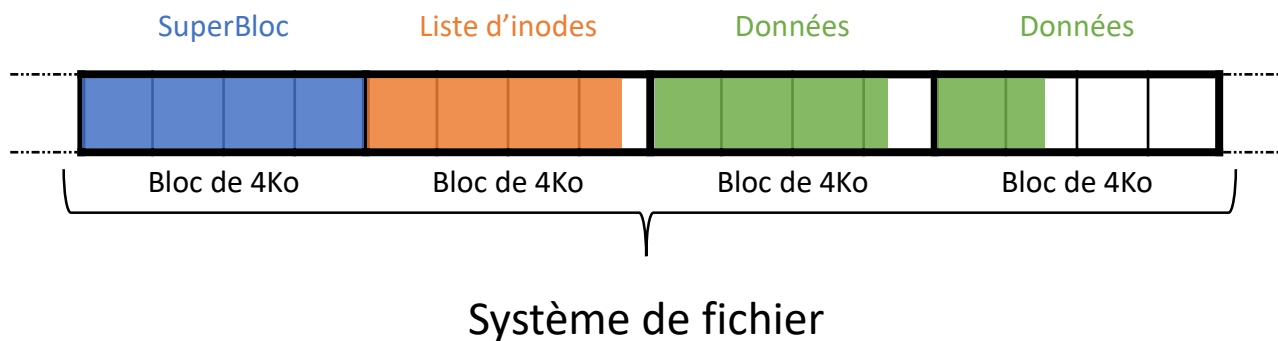
I. Systèmes d'exploitation

I.3. Système de fichiers

I.3.c Partitions



- Un bloc situé un peu après le début du support physique appelé communément **SuperBloc** – **SuperBlock** – contient les informations nécessaires pour localiser l'inode « racine », le nombre maximal d'inode, la taille des blocs, le nombre de blocs vers lesquels peut pointer un inode, etc...
- Le **SuperBloc**, les **Inodes** et les **Données** constituent ce qu'on appelle **système de fichier**.

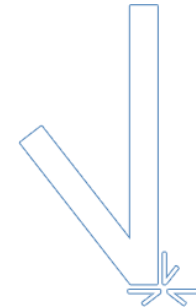


- Un support physique peut contenir **plusieurs systèmes de fichiers**. On parle alors de **partitions**.

I. Systèmes d'exploitation

I.3. Système de fichiers

I.3.c Format

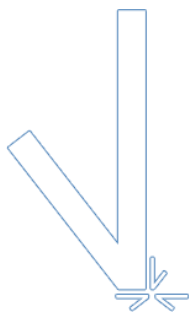


- Chaque système de fichier de chaque **partition** d'un support physique peut avoir un **format** différent des autres. Les formats les plus connus sont :
 - **Ext4 (pour Linux/Unix)**
 - **Ext3 (pour Linux/Unix)**
 - **NTFS (de Microsoft)**
 - **FAT32 (de Microsoft)**
 - **APFS (de Apple)**
 - ...
- Du format dépendra :
 - le nombre total de blocs vers lesquels 1 inode peut pointer (et donc la taille maximale d'un fichier);
 - le nombre total de blocs vers lesquels on peut pointer (et donc la taille maximale du système de fichiers);
 - le nombre total d'inode qu'on peut créer (donc le nombre maximal de fichiers qu'on peut créer); ...
- La Wikipédia propose une [liste exhaustive des formats de systèmes de fichier et de leurs caractéristiques respectives](#).

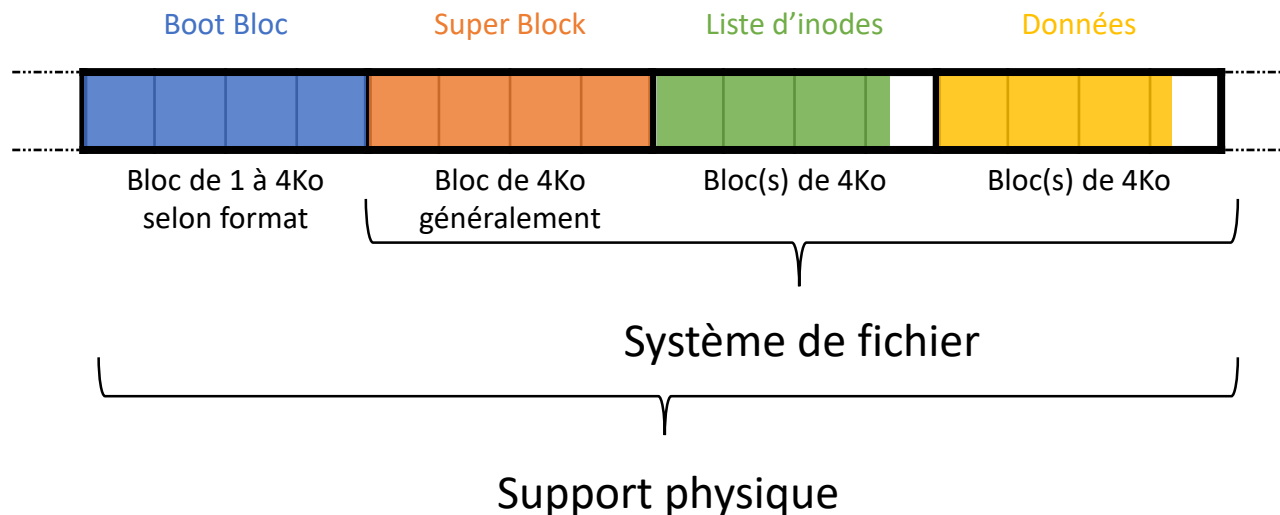
I. Systèmes d'exploitation

I.3. Système de fichiers

I.3.c Format



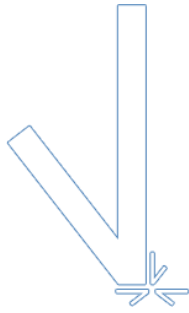
- Le premier bloc sur support physique est appelé **bloc d'amorçage – boot block** –. Ce bloc contient :
 - le **boot loader** du kernel si il s'agit du support physique principal. On qualifie alors le boot block de **MBR – Master Boot Record** –.
 - Et la position des **SuperBlocks** des différentes **partitions** du support physique.



I. Systèmes d'exploitation

I.3. Système de fichiers

I.3.c Format



- On peut consulter la **liste des partitions** sur les différents supports physiques à l'aide de l'utilitaire :

**lsblk ou
fdisk -l**

(`fdisk` suivi de l'argument `-l`)

- Pour connaître le **format** des partitions, on peut utiliser l'utilitaire :

blkid

- Pour avoir des **informations générales** concernant les partitions :

df -T

(`df` suivi de l'argument `-T`)

- Et pour avoir des informations concernant le nombre d'**inodes** utilisés ou disponibles :

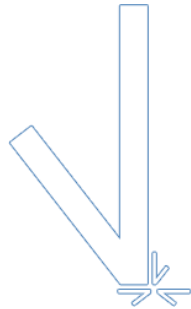
df -i

(`df` suivi de l'argument `-i`)

I. Systèmes d'exploitation

I.3. Système de fichiers

I.3.d Montage



- L'opération qui consiste à charger une partition (et donc un système de fichier) dans le **VFS** s'appelle le « **montage** ».
- Cette opération peut être effectuée **automatiquement** par le kernel si la configuration adéquate existe. Cette configuration de « montage » statique est inscrite dans le fichier `/etc/fstab` :

cat /etc/fstab

(`cat` suivi du chemin vers le fichier `fstab`)

- Cette opération peut être effectuée **manuellement** en utilisant la commande :

mount <chemin de la partition> <dossier>

(`mount` suivi du chemin vers la partition et du dossier sous lequel sera montée la partition)

- Cette opération peut être effectuée **manuellement** en utilisant la commande :

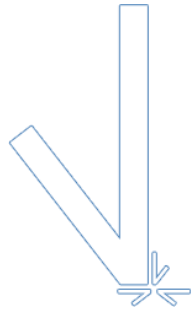
umount <dossier>

(`umount` suivi du dossier sous lequel est montée la partition)

I. Systèmes d'exploitation

I.3. Système de fichiers

I.3.e Système de fichiers virtuel

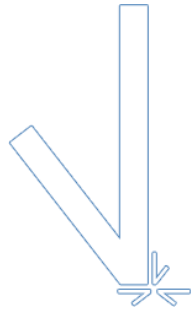


- Toutes les commandes, concernant le système de fichiers, saisies jusqu'à présent, ont effectué des *appels système* au **système de fichiers virtuel – Virtual Filesystem switch, VFS** – du kernel.
- Le **VFS** est la couche d'abstraction qui permet aux utilisateurs et aux applications d'accéder à **différents systèmes de fichiers** à l'aide d'une **interface unique**.
- Il propose une API pour **monter** un système de fichier à partir d'un support physique (**mémoire morte** ou **mémoire vive**), **créer** un système de fichier sur un support physique, **écrire, lire, modifier** des fichiers, ...
- Le dossier `/proc` par exemple est le point de montage d'un système de fichier qui correspond à l'ensemble des processus chargés en mémoire vive.
- Il présente **tous les systèmes de fichiers** sous la forme d'une **seule et unique arborescence** de fichiers et de dossiers.

I. Systèmes d'exploitation

I.3. Système de fichiers

I.3.f Organisation

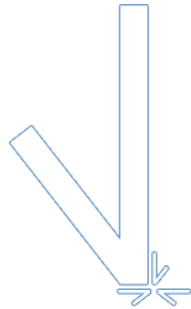


- L'organisation du système de fichier d'**un point de vue utilisateur** est une arborescence qui contient, par défaut, les dossier suivant :
 - `/` : **dossier racine – root directory** –, tout est situé sous le dossier `/`
 - `/bin` : dossier des **binaires – binaries** – (programmes) pour tous les utilisateurs.
 - `/sbin` : dossier des **binaires système – system binaries** – pour les super-utilisateurs.
 - `/usr` : dossier des binaires **utilisateur – user binaries** – pour tous ou certains des utilisateurs.
 - `/lib` : dossier des **librairies – libraries** – indispensables au fonctionnement des binaires.
 - `/boot` : dossier des fichiers de démarrage – **boot** –. **Boot Loader** et **Kernel**.
 - `/dev` : dossier périphériques – **devices** –. Chaque fichier dans ce dossiers correspond soit à un fichier d'entrées/sortie vers un périphérique soit à un pseudo-périphérique (un fichier qui ne correspond pas réellement à un périphérique mais qui produit un comportement particulier en entrée/sortie).
 - `/etc` : dossier des **configurations**. Ce dossier ne contient que des fichiers de configuration pour des programmes utilisateur ou relatifs à des comportements du système d'exploitation.
 - `/home` : dossier des dossiers utilisateur. Contient un dossier par utilisateur du système (à l'exception du dossier du superadministrateur – **root user** –).
 - `/root` : dossier des fichiers du superadministrateur – **root user** – .

I. Systèmes d'exploitation

I.3. Système de fichiers

I.3.f Organisation



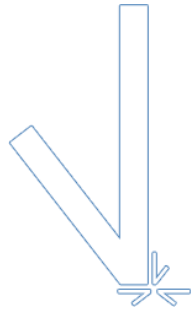
- `/tmp` : dossier des fichiers **temporaires**.
 - `/media` : dossier des points de montage des **médias amovibles**.
 - `/proc` : dossier des **processus** en cours d'exécution.
 - `/sys` : dossier des **variables système**. Contient des informations concernant le système d'exploitation et les périphériques matériels.
 - `/var` : dossier des **fichiers variables**. Contient des fichiers tels que les journaux d'activités, les e-mails, et autres qui sont amenés à changer pendant l'exécution du système d'exploitation.
 - ...
- Dans le système de fichier hiérarchique, le symbole `/` initial représente donc le dossier racine et entre le nom de chaque sous dossier on inscrit un `/`.
 - Lorsqu'on veut indiquer le chemin qui mène à un sous dossier ou un sous fichier on peut écrire un **chemin absolu** (à partir du dossier racine) :

`/chemin/vers/le/dossier/fichier`

I. Systèmes d'exploitation

I.3. Système de fichiers

I.3.f Organisation



- Lorsqu'on veut indiquer le chemin qui mène à un sous dossier ou fichier, on peut écrire un **chemin relatif** au dossier courant comme par exemple :

vers/le/dossier/fichier

- Pour lister le contenu d'un dossier, on peut utiliser la commande :

ls </chemin/vers/le/dossier>

(ls suivi du chemin vers le dossier)

- Pour se positionner sous un dossier, on peut utiliser la commande :

cd </chemin/vers/le/dossier>

(cd suivi du chemin vers le dossier)

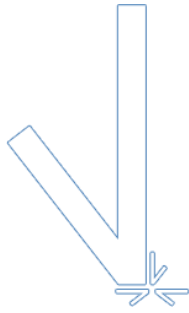
- Pour savoir quel est le dossier sous lequel on se trouve :

pwd

I. Systèmes d'exploitation

I.3. Système de fichiers

I.3.f Organisation



- Le système de fichier propose des « **alias** » qui sont les chemins absolus vers :
 - Le dossier courant, il s'agit du `.`
 - Le dossier parent du dossier courant, il s'agit du `..`
 - Le dossier de l'utilisateur courant, il s'agit du `~`
- Pour **créer** un **dossier**, on peut utiliser :
`mkdir <chemin vers le nouveau dossier>`
- Pour **supprimer** un **dossier**, on peut utiliser :
`rmdir <chemin vers le dossier>`
- Pour **créer** un **fichier**, on peut utiliser :
`touch <chemin vers le nouveau fichier>`
- Pour **supprimer** un **fichier** ou un **dossier** on peut utiliser :
`rm <chemin vers le fichier>`
- Pour **copier** un **dossier** ou un **fichier**, on peut utiliser :
`cp <chemin vers le fichier ou le dossier> <destination>`
- Pour **déplacer** (ou *renommer*) un **dossier** ou un **fichier**, on peut utiliser :
`mv <chemin vers le fichier ou le dossier source> <destination>`

I. Systèmes d'exploitation

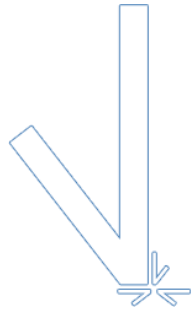
I.4 Notion de session



Notion de session

I. Systèmes d'exploitation

I.4. Notion de Sessions



- Linux est un système **multi-utilisateur**. Un **utilisateur n'est pas forcément un humain...** Un programme faire l'objet d'une ouverture de session avec un compte dédié.
- Une session commence lorsqu'un utilisateur est authentifié auprès du système d'exploitation. C'est la connexion – **login** –.
- Une session se termine lorsqu'un utilisateur n'est plus authentifié auprès du système d'exploitation. C'est la déconnexion – **logout** –.
- Pendant une session, l'utilisateur peut effectuer, en fonction d'un **système de permissions**, certains *appels système*, utiliser certains utilitaires systèmes, démarrer certains programmes, accéder à certaines parties du système de fichier, ...
- Le mécanisme de **gestion des session** repose sur :
 - La gestion de **comptes utilisateur**;
 - La gestion de **groupes utilisateur**;
 - La gestion de **permissions** et de **droits** sur les composantes du système de fichiers.

I. Systèmes d'exploitation

I.4. Notion de Sessions



- Un compte super utilisateur ou administrateur, généralement appelé **root**, est créé, à l'installation, sur chaque système. Ce compte utilisateur possède des droits étendus.

- Chaque utilisateur a un **UID** (**ID**entifiant **U**tilisateur). Pour **créer** un utilisateur

adduser <nom d'utilisateur>

- Chaque utilisateur appartient à **un** ou **plusieurs groupes** d'utilisateur.

- Chaque groupe à un **GID** (**ID**entifiant de **G**roupe). Pour **créer** un groupe :

addgroup <nom de groupe>

- Pour **ajouter** un utilisateur à un groupe existant :

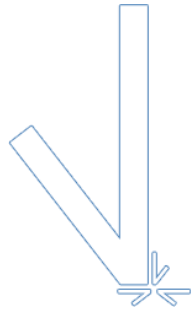
usermod -aG <nom de groupe> <nom d'utilisateur>

- Pour ouvrir une nouvelle session avec un autre utilisateur que l'utilisateur courant :

su <nom d'utilisateur>

I. Systèmes d'exploitation

I.4. Notion de Sessions



- Sur certaines distributions, les utilisateurs n'ont pas la permission d'exécuter certains utilitaires comme `su`. On peut les assigner au groupe `sudoers`. Cela leur donne la permission d'utiliser l'utilitaire `sudo`.
- L'utilitaire `sudo` permet d'exécuter certaines commandes en s'identifiant momentanément à l'aide du compte super utilisateur. Par exemple :

```
sudo su <nom d'utilisateur>
```

- Pour **supprimer** un utilisateur

```
deluser <nom d'utilisateur>
```

- Pour **supprimer** un groupe

```
delgroup <nom du groupe>
```

I. Systèmes d'exploitation

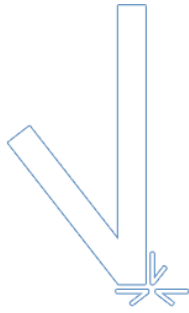
I.5 Permissions et Droits



Permissions et Droits

I. Systèmes d'exploitation

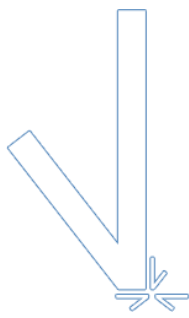
I.5. Permissions et Droits



- Les fichiers (et dossiers) du système de fichiers possèdent tous un **propriétaire**. Le propriétaire est l'utilisateur (**u**) qui a créé le fichier.
- Un fichier appartient aussi à un groupe (**g**). Le groupe est généralement le groupe principal de l'utilisateur qui a créé le fichier.
- Seul le propriétaire du fichier ou le super utilisateur peuvent changer le propriétaire ou le groupe d'appartenance d'un fichier.
- On peut définir des **permissions** pour le propriétaire (**u**) d'un fichier, pour le groupe (**g**) d'un fichier et, pour les autres utilisateurs (**o**)

I. Systèmes d'exploitation

I.5. Permissions et Droits



- Les permissions qui peuvent être attribuées à un fichier sont :
 - La permissions de **lire** : **R**
 - La permission d'**écrire** : **W**
 - La permission d'**exécuter** : **X**
 - Pour modifier le propriétaire et/ou le groupe d'un fichier on peut utiliser :
- chown <nouveau propriétaire>:<nouveau groupe>**
- Pour modifier les permissions pour l'utilisateur (**u**), le groupe (**g**) ou les autres (**o**), on peut écrire par exemple :

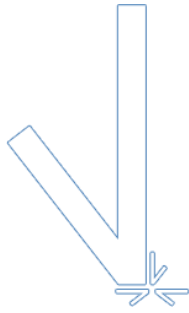
chmod u+rx,g+rx,o+rx <chemin vers le fichier>

OU

chmod u+rx,g+rw-x,o+r-wx <chemin vers un autre fichier>

I. Systèmes d'exploitation

I.5. Permissions et Droits



- Les permissions sont stockées dans l'inode correspondant au fichier sous forme binaire sous la forme de 3 groupes de 3 bits. Le premier groupe de 3 bits représente les permissions pour l'utilisateur (**u**), le second pour le groupe (**g**) et le troisième pour les autres (**o**).
- Par exemple :
 - **rwX** pour l'utilisateur (**u**) serait **111** soit **7** en décimal,
 - **r** uniquement pour le groupe (**g**) serait **100** soit **4** en décimal,
 - Et rien pour les autres (**o**) donnerait **000** soit **0** en décimal.
- C'est pourquoi, par exemple :

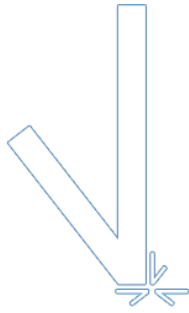
chmod u+rwX,g+r-wX,o-rwX <chemin vers le fichier>

- Peut être écrit :

chmod 740 <chemin vers le fichier>

I. Systèmes d'exploitation

I.5. Permissions et Droits



- La permission d'écrire, autorise la modification du contenu du fichier (modifier les octets correspondant à du texte, ou autre, ...) ou du dossier (modifier, ajouter, ou supprimer des liens physiques ou symboliques).
- La permission de lire, autorise la lecture du contenu du fichier (lire les octets correspondant à du texte ou autre) ou du dossier (lister le contenu du dossier)
- La permission d'exécuter autorise l'exécution du fichier (en tant que programme) ou du dossier (pour se positionner en dessous au niveau de l'arborescence).

I. Systèmes d'exploitation

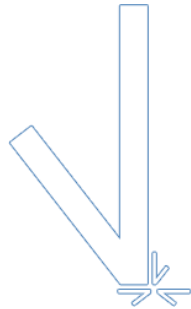
I.6 Notion de Terminal



Notion de Terminal

I. Systèmes d'exploitation

I.6. Notion de Terminal



- Le terminal des systèmes UNIX s'appelle `shell`. Sur linux la version du `shell` s'appelle `BASH`.
- Il s'agit d'un invite de commande qui sert d'**interface** entre l'**utilisateur** et le **système d'exploitation**.
- L'accès au `shell` nécessite de s'authentifier pour ouvrir une nouvelle session utilisateur.
- Pour savoir qui est l'utilisateur courant du shell :

whoami

- Pour savoir qui sont les utilisateurs qui ont ouvert un `shell` pour communiquer avec le système d'exploitation :

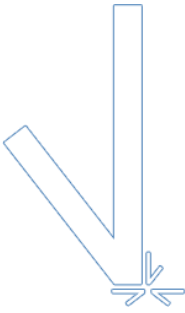
who

- Pour quitter le `shell` :

exit

I. Systèmes d'exploitation

I.7 Utilitaires Système



Utilitaires Système

I. Systèmes d'exploitation

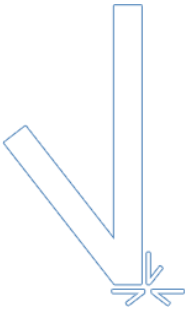
I.7. Utilitaires Système



- **cat** : lire un fichier
- **cut** : filtrer un fichier
- **sort** : trier un fichier
- **more** : lire un fichier par page
- **less** : lire un fichier par page
- **tail** : lire les dernières lignes d'un fichier
- **head** : lire les premières lignes d'une fichier
- **echo** : renvoyer un texte
- **man**: accéder au manuel

I. Systèmes d'exploitation

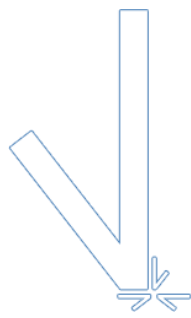
I.8 Variables d'Environnement



Variables d'Environnement

I. Systèmes d'exploitation

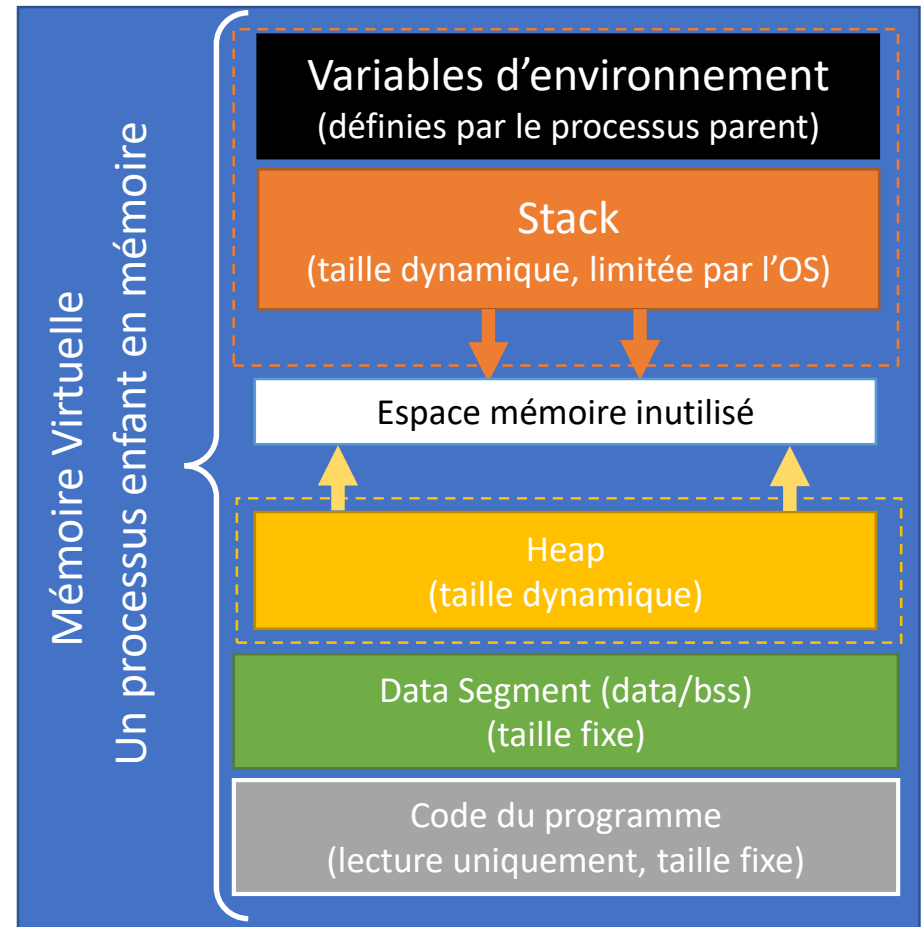
I.8. Variables d'Environnement



- Les variables d'environnement sont des variables **définies** par un **processus parent** dans l'**espace mémoire** de ses **processus enfant**. On peut dire qu'un processus enfant *hérite* des variables d'environnement définies par son processus parent.
- Les variables d'environnement sont une **alternative** au passage d'**arguments en ligne de commande** entre les processus.
- Les variables d'environnement se présentent sous la forme de **couples clé=valeur**
- La **liste complète des variables d'environnement** définies au sein d'un processus est appelée **environnement du processus**.
- Elle peut être lue dans le pseudo-fichier :

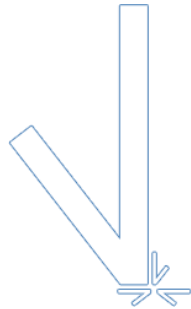
/proc/[PID]/environ

(pid est l'identifiant du processus)



I. Systèmes d'exploitation

I.8. Variables d'Environnement



- Sur un système UNIX, le **premier processus** (PID 1) démarré par le kernel est celui qui contient les **scripts d'initialisation** des différents **services (daemon)** du système d'exploitation (*gestionnaire de session utilisateur, de réseau, gui, serveurs, ...*) .
- Ce premier processus est démarré à partir du fichier :

`/sbin/init`

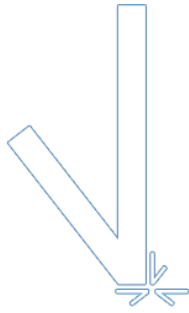
- Tous les processus sont donc des **enfants**, ou des **petits enfants** de ce premier processus.
- Sur les distributions récentes de Linux, ce fichier est un **lien symbolique** vers le programme **systemd** qui est le système d'initialisation fourni avec le système :

`ls -la /sbin/init`
indique
`/lib/systemd/systemd`

- Sur les distributions récentes de Linux, tous les processus sont donc des **enfants**, ou des **petits enfants** de **systemd**.

I. Systèmes d'exploitation

I.8. Variables d'Environnement



- Pour **définir** les **variables d'environnement** des **enfants** de **systemd**, on édite le fichier :

`/etc/environment`

- On peut également créer des fichiers qui déclarent des variables d'environnement sous le dossier :

`/etc/environment.d/*.conf`

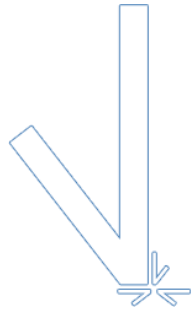
- Elles doivent être définies, dans ces fichiers, sous la forme (clé=valeur) :

KEY=VALUE

- Ces variables d'environnement sont disponibles dans tous les processus enfant de **systemd**. On les qualifie parfois de « *variables d'environnement globales* ».
- **Attention**, cependant, à bien noter que les « *variables d'environnement globales* », sont créées au sein de chaque processus enfant et sont donc **locales** du point de vue d'un processus .

I. Systèmes d'exploitation

I.8. Variables d'Environnement



- On dispose également de fichiers de configuration pour définir les variables d'environnement qui seront disponible lors d'**une connexion utilisateur via un terminal (création d'une nouvelle session utilisateur)** :

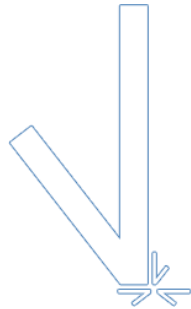
`/etc/profile`
et
`/etc/profile.d/*.conf`

- On peut également configurer des variables de configuration qui seront disponible à l'**ouverture d'un terminal sans connexion** (ouverture d'un terminal alors qu'on est déjà connecté, pas de nouvelle session utilisateur) dans le fichier :

`~/.bashrc`
fichier `.bashrc` dans le dossier de l'utilisateur connecté
ou
`~/.bash_profile`
fichier `.bash_profile` dans le dossier de l'utilisateur connecté
ou
`~/.profile`
fichier `.profile` dans le dossier de l'utilisateur connecté

I. Systèmes d'exploitation

I.8. Variables d'Environnement



- Pour mettre à jour les variables d'environnement au sein du terminal, il faut se reconnecter au terminal ou utiliser la commande **source** pour re-exécuter le code du fichier de configuration. Par exemple :

```
source ~/.bashrc
```

- Pour définir une variable d'environnement dans ces fichiers, il faut utiliser la commande **export**. Par exemple ajouter dans un de ces fichiers de configuration la ligne :

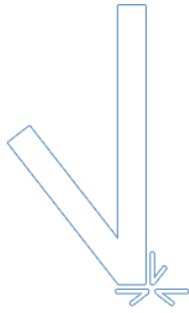
```
export MAISON="habitants"
```

- On peut également utiliser directement cette commande au niveau du terminal pour définir **temporairement** une variable d'environnement.
- Pour supprimer une variable d'environnement, il faut utiliser la commande **unset**. Par exemple :

```
unset MAISON
```

I. Systèmes d'exploitation

I.8. Variables d'Environnement



- Pour afficher toutes les variables d'environnement déclarées, on peut utiliser les commandes :

printenv

ou

env

- Pour afficher spécifiquement une variable d'environnement du terminal, on peut utiliser **echo** :

echo \$MAISON

- Pour démarrer un programme, à partir du terminal, avec des variables d'environnement spécifiques, on peut utiliser la commande **env** :

env MAISON="habitants" /chemin/vers/le/programme

I. Systèmes d'exploitation

I.9 Programmation

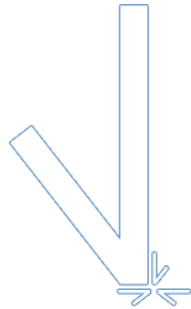


Programmation

I. Systèmes d'exploitation

I.9. Programmation

I.9.a Shell/BASH

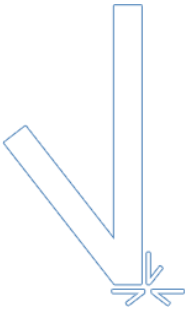


- « Shell » est le nom des terminaux sur les systèmes UNIX mais aussi un **interpréteur de lignes de commandes**.
- Le Shell original d'UNIX, s'appelle *Bourne Shell*, du nom de son créateur *Stephen Bourne*.
- Le programme Shell est situé dans **/bin/sh**. On le retrouve sur tous les systèmes UNIX.
- Deux variantes connues du Shell sont :
 - **/bin/bash** qui est un Shell incluant des fonctionnalités supplémentaires et qu'on retrouve généralement, par défaut, sur les distributions **Linux**.
 - **/bin/zsh** qui est un Shell incluant des fonctionnalités supplémentaires et qu'on retrouve généralement, par défaut, sur **MacOS**.

I. Systèmes d'exploitation

I.9. Programmation

I.9.a Shell/BASH

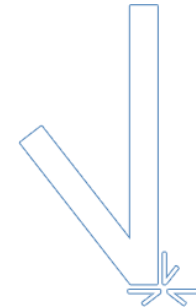


- L'interpréteur de lignes de commandes Shell (ou ses évolutions) peut interpréter :
 - Des **commandes saisies par l'utilisateur** via son terminal;
 - Des **fichiers contenant des commandes Shell**. On parle alors de **scripts Shell**.
- Le langage de commande Shell comprend la prise en charge :
 - Des variables;
 - Des conditions;
 - Des boucles;
 - Des fonctions;
 - Des substitutions;
 - Des commentaires;
 - ...

I. Systèmes d'exploitation

I.9. Programmation

I.9.a Shell/BASH



- Pour interpréter et donc exécuter un script Shell, on peut procéder de **2 façons**.
- Soit :
 - On **démarre l'interpréteur avec en argument le fichier à interpréter**. Par exemple :

`/bin/bash /chemin/vers/mon/script`

- Soit :
 - Le script Shell doit être **exécutable** (le droit `x` doit être positionné sur le script pour l'utilisateur exécutant le script);
 - Le script Shell **doit commencer** par une ligne **shebang** C'est une séquence de caractères commençant par `#!` qui indique au système d'exploitation qu'un fichier doit être **interprété en temps que programme exécutable** avec un interpréteur en particulier.
 - Exemple de ligne **shebang** :

`#!/chemin/absolu/vers/interpréteur`

par exemple :

`#!/bin/zsh`

I. Systèmes d'exploitation

I.9. Programmation

I.9.a Shell/BASH



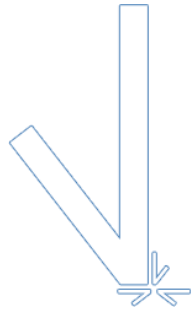
- Le langage Shell est un langage **impératif**.
- Pour les subtilités de la programmation Shell, je vous invite à consulter :

[https://fr.wikibooks.org/wiki/Programmation Bash](https://fr.wikibooks.org/wiki/Programmation_Bash)

I. Systèmes d'exploitation

I.9. Programmation

I.9.b Autres



- On peut installer d'autres interpréteurs sur un système UNIX. Auquel cas, les scripts écrits dans les langages correspondants peuvent être exécutés selon les mêmes modalités que les scripts Shell.
- Soit :
 - On **démarre l'interpréteur avec en argument le fichier à interpréter**. Par exemple, avec Node installé dans `/bin` :

```
/bin/node /chemin/vers/mon/script
```

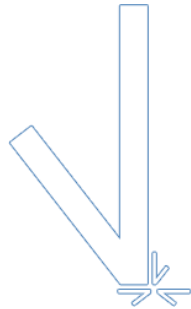
- Soit :
 - Le script doit être **exécutable** (le droit `x` doit être positionné sur le script pour l'utilisateur exécutant le script);
 - Le script **doit commencer par** une ligne **shebang**. Par exemple, toujours avec Node installé dans `/bin` :

```
#!/bin/node
```

I. Systèmes d'exploitation

I.9. Programmation

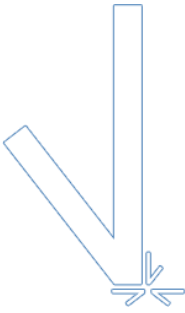
I.9.b Autres



- Si le programme est écrit en langage C ou C++, le fichier source (qui contient le code source) doit être **compilé**. C'est-à-dire transformé en *binaire*.
- Un *binaire* est un fichier dont le **contenu** est constitué de code **en assembleur**. Ce code en assembleur interagit avec le kernel via des *appels système*. On qualifie souvent ce code en assembleur de *langage machine*.
- Ce binaire doit être **exécutable** par un utilisateur (droit \times positionné pour l'utilisateur).
- Pour compiler un programme en C ou C++, Linux propose la **suite logicielle GCC** (GNU Compiler Collection) qui s'utilise en ligne de commande via l'utilitaire `gcc`.
- Pour les programmes nécessitant de compiler de multiples fichiers et dans un certain ordre, Linux propose l'**utilitaire** `make` et son **fichier de configuration** `makefile`.
- La configuration de `make` permet de définir l'ordre des fichiers à compiler, les options à passer à `gcc`, les variables d'environnement à déclarer, les dossier ou fichier à créer, etc.

I. Systèmes d'exploitation

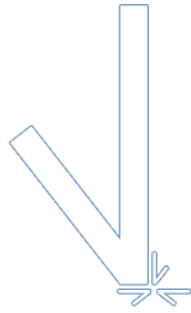
I.10. Gestion des Paquets



Gestion des Paquets

I. Systèmes d'exploitation

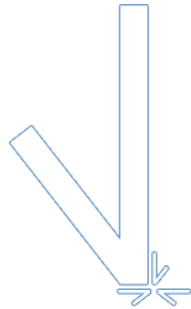
I.10. Gestion des Paquets



- Un **paquet** qualifie un type d'archive pour les systèmes UNIX.
- Un paquet contient des **fichiers** et des **dossiers** sous une certaine forme, des **métadonnées** sur son contenu et des **programmes exécutables** ou des **fichiers de code source**.
- On citera, parmi les **formats** de paquets :
 - APK : Format utilisé par Android par exemple;
 - RPM : Format utilisé par la distribution RedHat de Linux;
 - DEB : Format utilisé par la distribution Debian de Linux et ses dérivés comme Ubuntu;
 - SNAP : Format utilisé par Ubuntu;
 - PKG : Format utilisé par MacOS (Attention à ne pas confondre avec le format DMG qui est une image disque et qui généralement contient elle-même un fichier PKG).
 - ...
- Les **métadonnées** du paquet indiquent **où** et **comment** son contenu doit être installé.

I. Systèmes d'exploitation

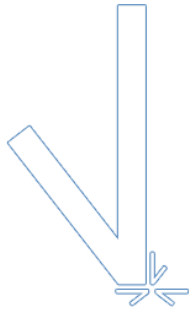
I.10. Gestion des Paquets



- Pour traiter les format de paquets, on utilise un **gestionnaire de paquets**.
- Le gestionnaire de paquets permet de :
 - **Installer, désinstaller** et **mettre à jour** un paquet;
 - Utiliser des paquets provenant **de sources différentes** (locale, distantes);
 - Vérifier la **présence des dépendances** (paquets dont dépend le paquet);
 - Vérifier l'**intégrité du paquet** (valide que le contenu du paquet est conforme à une somme de contrôle).
- Exemples de gestionnaire de paquets :
 - APT pour le format DEB
 - Ou DPKG pour le format DEB;
 - PKG pour le format PKG;
 - RPM pour le format RPM;
 - ...
- Ces gestionnaires de paquets peuvent avoir des interfaces graphiques comme Aptitude ou Synaptic pour APT ou DPKG.

I. Systèmes d'exploitation

I.10. Gestion des Paquets

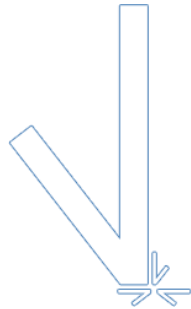


- En prenant pour exemple `APT`, voici le type de fonctionnalités proposées :

Installer un paquet	<code>apt install <paquet></code>
Supprimer un paquet	<code>apt remove <paquet></code>
Supprimer les fichiers de configuration d'un paquet	<code>apt purge <paquet></code>
Supprimer les dépendances non utilisées	<code>apt autoremove</code>
Lister les paquets installés	<code>apt list --installed</code>
Chercher un paquet installable	<code>apt-cache search <paquet></code>
Mettre à jour la liste des paquets installables	<code>apt update</code>
Mettre à jour un paquet	<code>apt upgrade <paquet></code>
Mettre à jour tous les paquets	<code>apt upgrade</code>

I. Systèmes d'exploitation

I.10. Gestion des Paquets



- L'origine des paquets installables par APT peut être configurée :
 - Dans le fichier : `/etc/apt/sources.list`
 - Sous le dossier : `/etc/apt/sources.list.d/*.list`
- Les bases de données de paquets installables sont appelée **dépôts** (ou *repositories*).
- Les fichiers de configurations des gestionnaires de paquets comme APT contiennent des URL de **dépôts distant** – *remote repository* – (chemin réseau par exemple) ou des chemins vers des **dépôts locaux** (CD-ROM par exemple).



Réseaux IP

II. Réseaux IP

II.1 Modèle TCP/IP



Modèle TCP/IP

II. Réseaux IP

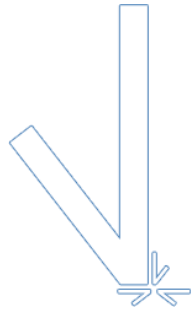
II.1. Modèle TCP/IP



- Par l'IETF : Internet Engineering Task Force
- Modèle TCP/IP :
 - S'appuie sur une famille de protocoles :
 - **TCP** : Transmission Control Protocol
 - Et **IP** : Internet Protocol
 - Propose de résoudre les problématiques liées à l'**émission** et à la **réception** de données sur un réseau informatique avec un maximum de **4 niveaux d'abstraction** appelés **couches**.
 - Document de référence : [RFC1122](#)

II. Réseaux IP

II.1. Modèle TCP/IP



Couches	Description	Exemple d'application	Exemple de protocole
APPLICATION	Point d'accès aux réseaux pour les utilisateurs ou les applications	Programme qui ouvre ce qui est reçu ou crée ce qui doit être envoyé.	HTTP, SMTP, ...
	Formatage des données.	Chiffrement et déchiffrement.	ASCII, JPEG, ...
	Gère la connexion, déconnexion et la synchronisation entre 2 processus.	Synchronisation par échange de jetons.	RPC, NFS, SQL, ...
TRANSPORT	S'assure que les messages sont reçus sans erreur, dans l'ordre et sans perte ou doublons.	Contrôle du flux entre 2 hôtes.	TCP, UDP, ...
INTERNET	Gère les sous réseaux et l'acheminement des paquets sur les sous réseaux.	Création de paquets.	IPv4, IPv6, ICMP, ...
LIAISON	S'occupe de la création, du contrôle, de la transmission et la réception des trames .	Création de trames.	Ethernet, IEEE 802, ...
	S'occupe de la transmission et de la réception du flux de bit à travers un support physique.	Structures physiques : câbles, hubs, ...	Hub

II. Réseaux IP

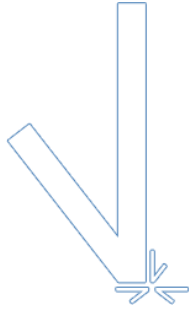
II.1. Modèle TCP/IP







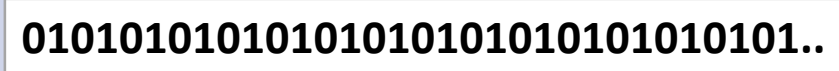
- Chaque couche produit une unité de données : **PDU** (**P**rotocol **D**ata **U**nit) selon les règles imposées par un protocole.
- Chaque PDU est l'**encapsulation** correspondant à une couche.
- Chaque PDU peut être constitué d'un en-tête de données (**Data Header – DH**), d'une unité de données (**Data Unit – DU**) et de données complémentaires (**Data Trailer – DT**).

II. Réseaux IP

II.1. Modèle TCP/IP

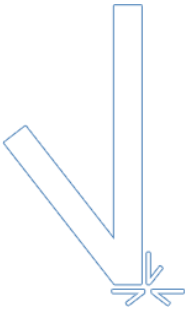


- On peut représenter les PDUs par couche comme suit :

Couches	PDU (Protocol Data Unit)	Structure (Data Header : DH, Data Unit : DU, DT : Data Trailer)
APPLICATION	APDU : Application PDU	
TRANSPORT	TPDU : Transport PDU	
INTERNET	Packet (Paquet)	
LIAISON	Frame (Trame)	
	Bits	

II. Réseaux IP

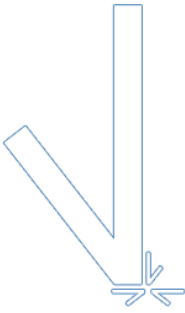
II.1. Modèle TCP/IP



- Le modèle TCP/IP est le modèle conceptuel du réseau Internet.
- Nous allons étudier différents protocoles pour chaque couche de ce modèle.
- L'objectif est de comprendre :
 - Les règles de l'encapsulation de données pour leur transmission à la couche **inférieure**;
 - Les règles de la décapsulation de données pour leur transmission à la couche **supérieure**.

II. Réseaux IP

II.2 Adressage IP

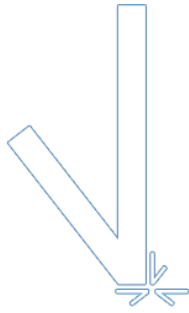


Adressage IP

II. Réseaux IP

II.2. Adressage IP

II.2.a Adresses MAC

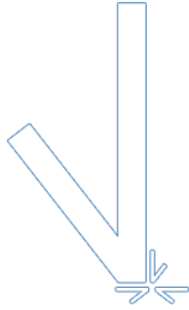


- La couche **liaison** est responsable de la création et de la réception des **trames** qui vont être transmises à travers le support physique du réseau.
- Généralement, une trame contient au moins 3 informations :
 - Les adresses **MAC (Media Access Control)** de l'expéditeur et du destinataire dans l'**en-tête** (data header);
 - Des données (**payload** – data unit), généralement un paquet basé sur **IP**;
 - Un contrôle de redondance cyclique (CRC – **Cyclic Redundancy Check**) pour contrôler l'intégrité des données transmises dans les **données complémentaires** (data trailer).
- Une adresse **MAC (Media Access Control)** est un identifiant unique (au monde !) d'interface réseau.

II. Réseaux IP

II.2. Adressage IP

II.2.a Adresses MAC



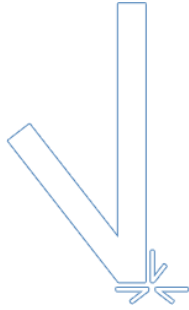
- Une adresse MAC est constitué d'après le standard actuel, le [IEEE EUI-48](#), de 48 bits.
- Les 24 premiers bits d'une adresse MAC sont soit :
 - Un identifiant unique d'organisation (**OUI**, **O**rganization **U**nique **I**dentifier)
 - Ou un identifiant unique d'entreprise (**CID** – **C**ompany **I**D)
- Exemple d'adresse MAC :

MAC	Identifiant d'entreprise (CID)			Identifiant d'interface réseau		
Correspondance	Hon Hai Precision Ind. Co., Ltd					
Hexadécimal (base 16)	C4	8E	8F	F3	54	7F
Bits (base 2, binaire)	11000100	10001110	10001111	11110011	01010100	01111111

II. Réseaux IP

II.2. Adressage IP

II.2.a Adresses MAC



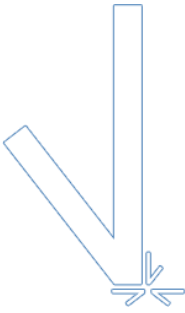
- Exemple : structure d'une trame IEEE 802.3 (Ethernet type II) :

Adresse Mac de Destination	Adresse Mac Source	Type d'Ethernet (Ici type II)	Données (Payload)	CRC
C48E8FAAAAAA	C48E8FAAAAAB	0800	3A6E5F...4FD5F1	571E4F56
En-Tête 14 Octets			Données 46 à 1500 Octets	Contrôle 4 Octets
Trame Ethernet Type II				

II. Réseaux IP

II.2. Adressage IP

II.2.b Adresses IP

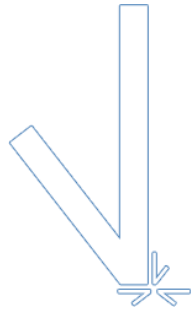


- Les données transmises ou reçues dans les trames sont ce qu'on appelle des **paquets IP**.
- La couche **internet** est responsable de la création et de la réception des **paquets IP**.
- Les protocoles **IP** (Internet **P**rotocol) version 4 (**IPv4**) et version 6 (**IPv6**) qui définissent des schémas d'adressage et la structure des paquets.
- Une adresse IP comporte **2 parties** :
 - La première partie identifie le **réseau**.
 - La deuxième partie identifie l'**hôte**.
- L'**IPv4** offre un espace d'adressage sur **32 bits**, soit 2^{32} adresses.
- L'**IPv6** offre un espace d'adressage sur **128 bits**, soit 2^{128} adresses.

II. Réseaux IP

II.2. Adressage IP

II.2.b Adresses IP



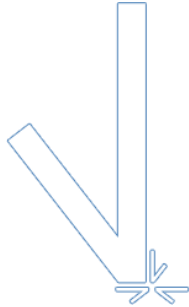
- Une adresse IP peut être suivie d'un « slash » et d'un **nombre décimal** appelé **CIDR** (Classless Inter-Domain Routing) qui exprime le nombre de **bits** identifiant le réseau. Le reste identifiant l'hôte.
- Exemple d'adresse IPv4 avec CIDR :

Adresse IPv4 32 bits (4 Octets)								CIDR (Décimal) Compris entre 0 et 32
8 bits (1 octets)		8 bits (1 octets)		8 bits (1 octets)		8 bits (1 octets)		Nombre entier décimal
00001110	.	00010111	.	00100000	.	00101001	/	16
14	.	23	.	32	.	41	/	16
Identifiant du réseau								
00001110	.	00010111	.	00000000	.	00000000		
Identifiant de l'hôte								
00000000	.	00000000	.	00100000	.	00101001		

II. Réseaux IP

II.2. Adressage IP

II.2.b Adresses IP



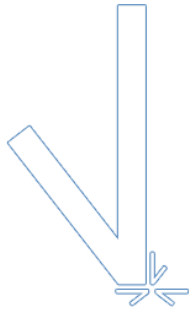
- Une adresse IP peut être suivie d'un masque de sous-réseau représenté sur 32 bits. Un **ET logique** entre le masque de sous-réseau et l'adresse de sous-réseau permet d'identifier le sous-réseau, le reste identifiant l'hôte.
- Exemple d'adresse IPv4 avec masque :

Adresse IPv4							
00001110	.	00010111	.	00100000	.	00101001	
14	.	23	.	32	.	41	
Masque de sous-réseau							
11111111	.	11111111	.	00000000	.	00000000	
255	.	255	.	0	.	0	
Identifiant du réseau							
00001110	.	00010111	.	00000000	.	00000000	
Identifiant de l'hôte							
00000000	.	00000000	.	00100000	.	00101001	

II. Réseaux IP

II.2. Adressage IP

II.2.b Adresses IP



- Certaines plages d'adresses IPv4 sont **réservées**. Par exemple :
 - 10.0.0.0/8 ;
 - 172.16.0.0/12 ;
 - **et** 192.168.0.0/16 ;

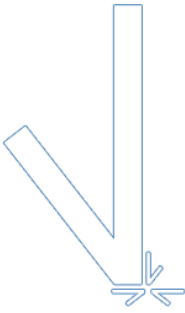
sont des adresses réservées aux **réseaux privés**.

- La plage d'adresse 127.0.0.0/8 est réservée à la **boucle locale - loopback** (adresse d'un hôte pour lui-même).
- La **dernière adresse IP** d'un réseau ou d'un sous réseau IPv4 est l'adresse IPv4 de **broadcast** (adresse réseau pour envoyer un paquet IP à tous les hôtes d'un réseau).

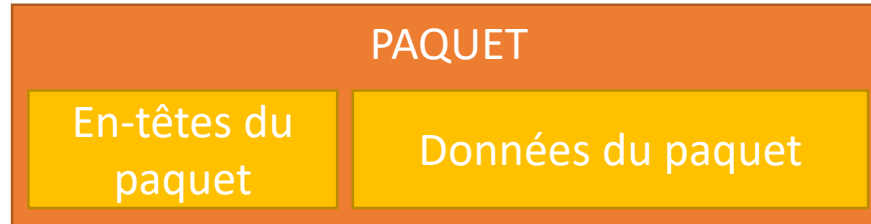
II. Réseaux IP

II.2. Adressage IP

II.2.b Adresses IP



- Le protocole IPv4 définit la structure d'un paquet comme des en-têtes suivies de données :



- Les en-têtes contiennent l'adresse IP de l'**expéditeur** et du **destinataire**.
- Pour configurer une interface réseau et lui assigner une adresse IP, on peut s'appuyer sur l'utilitaire (anciennement) :

```
ifconfig -a
    pour lister les interfaces réseau puis
ifconfig <interface> A.B.C.D/CIDR
    ou alors
ifconfig <interface> A.B.C.D W.X.Y.Z
    Avec A.B.C.D comme adresse IP et W.X.Y.Z comme masque
```

- Ou (plus récemment) :

```
ip a
    pour lister les interfaces réseau puis
ip net addr A.B.C.D/CIDR
```

II. Systèmes d'exploitation

II.2. Adressage IP

II.2.c Notion de Port

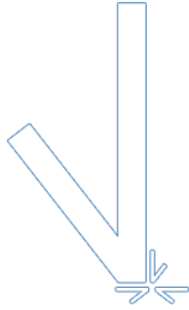


- Les protocoles de la couche transport sont responsables de la **fiabilité des échanges** et de l'**ordre d'arrivée** des données.
- Les protocoles de la couche transport font le **lien** entre les **hôtes** et les **applications**.
- La couche transport s'appuie principalement sur 2 protocoles :
 - Le protocole **UDP** (**U**ser **D**atagram **P**rotocol) : non-fiable et « non-connecté ».
 - Le protocole **TCP** (**T**ransmission **C**ontrol **P**rotocol) : fiable et « connecté ».

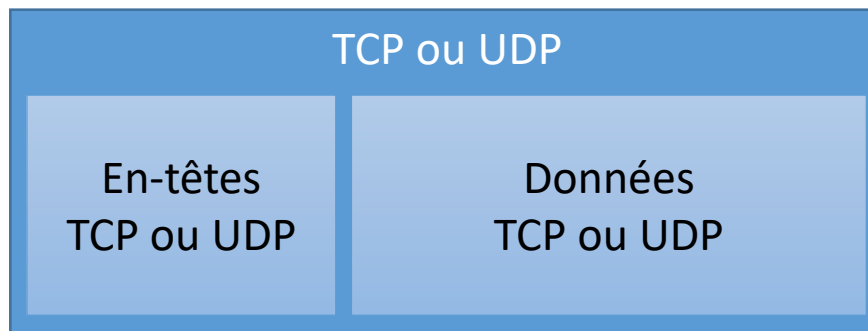
II. Systèmes d'exploitation

II.2. Adressage IP

II.2.c Notion de Port



- Ces protocoles introduisent la notion de **port logiciel**.
 - Un **port** est l'identifiant unique d'une application de la couche application.
 - Un port peut avoir une valeur comprise en **1** et **65535**.
- Ces protocoles produisent des données constituées d'**en-têtes** et de **données**.



II. Réseaux IP

II.2. Adressage IP

II.2.d Protocoles Applicatifs

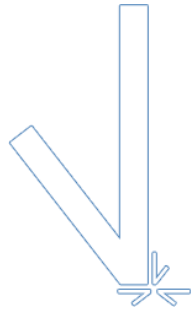


- Les protocoles de la couche application s'appuient sur les notions introduites par les couches inférieures.
- Les protocoles de la couche application reçoivent et envoient des données contenant des en-têtes suivies de données.
- Le **HTTP**, par exemple, est un protocole de la couche application.
- La gestion des encapsulation et décapsulation successives (trames <-> paquets IP <-> paquet TCP ou UDP) est assurée par le kernel. Le kernel met à disposition des processus une couche d'abstraction du réseau appelée **socket**.
- Un socket représente, du point de vue applicatif, le **point d'accès au réseau**. Chaque processus démarré peut créer un ou plusieurs socket(s) pour communiquer avec le réseau.

II. Réseaux IP

II.2. Adressage IP

II.2.d Protocoles Applicatifs



- Dans le cas d'un système qui initie l'échange (un logiciel client par exemple), pour créer un **socket**, le kernel à besoin :
 - De **l'adresse IP de l'hôte à joindre**;
 - du **numéro de port du logiciel concerné sur l'hôte à joindre**.
- Dans le cas d'un système passif qui attend qu'on échange avec lui (un logiciel serveur par exemple), pour créer un **socket**, le kernel à impérativement besoin :
 - du **numéro de port du logiciel à assigner au système**.
- Un socket est :
 - Une **couche d'abstraction logiciel** du réseau depuis un processus;
 - Un moyen d'**échanger des données entre des processus** à travers le réseau.
- Le principe des socket est également utilisé pour les IPC via la création de socket locaux. On les appelle **socket UNIX**.

II. Réseaux IP

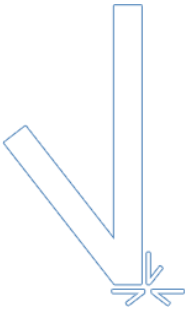
II.4 Utilitaires Réseau



Utilitaires Réseau

II. Réseaux IP

II.4. Utilitaire Réseau



- `tcpdump` : affiche en temps réel les trames reçues et émises par une interface réseau.
- `ping` : utilise le protocole IP ICMP pour tester la connectivité entre 2 hôtes.
- `nc` : client ou serveur selon l'usage. Permet de tester la création d'un socket entre 2 hôtes.
- `wget` : permet d'envoyer une requête HTTP à un URL à travers un socket TCP
- `nslookup` : permet de vérifier la résolution DNS d'un nom de domaine (correspondance entre nom de domaine et adresse IP)