

Système d'Exploitation

MEEF "Numérique et science informatique" (MEEF-NSI)

Fabien Rico

Univ. Claude Bernard Lyon 1

20 mai 2022



- 1 Introduction
 - Interface avec le matériel
- 2 Les processus
 - État d'un processus
 - Environnement
 - Communication entre processus
 - Les processus et python
- 3 Gestion de la mémoire
 - Rôle de la mémoire
 - Organisation de la mémoire
- 4 Fichiers et système de fichiers



Plan

1 Introduction

- Interface avec le matériel

2 Les processus

- État d'un processus
- Environnement
- Communication entre processus
- Les processus et python

3 Gestion de la mémoire

- Rôle de la mémoire
- Organisation de la mémoire

4 Fichiers et système de fichiers



Introduction

Qu'est-ce qu'un système d'exploitation ?

Littéralement, *ce qui permet d'utiliser la machine*. Quatre grands rôles :

- **Interface entre applications et matériel** (e.g., gestion des périphériques)
- **Organisation** (e.g., des disques, de la mémoire, et des processus)
- **Sécurité** (e.g., des données, du matériel)
- **Interaction avec le ou les utilisateurs** (e.g., comptes, droits, installation)

Différents types de systèmes d'exploitation, pour différents usages :

- Systèmes « généralistes », multi-utilisateurs et multi-tâches : GNU-Linux (Debian, Ubuntu, Redhat...), Windows, Mac OS X...
- Pour l'embarqué (contraintes sur l'utilisation des ressources) : Windows Embedded Compact, Android...
- Pour le temps-réel (contraintes d'échéance) : RTLinux, RTAI.



Interface avec le matériel

Que se passe-t-il lorsque vous utilisez la commande Python

```
print("Message")
```

- Le message est affiché sur la *sortie standard*
- Quelle est cette sortie standard ?
 - ▶ le terminal de commande qui a lancé le programme ;
 - ▶ une fenêtre de l'environnement de programmation ;
 - ▶ un fichier ;
 - ▶ la source de donnée d'un autre programme (pipe) ;
 - ▶ le fichier de logs d'un serveur ;
 - ▶ un système de logs centralisés d'un cluster ;
 - ▶ rien.

La même action pour un code peut donner des résultats différents en fonction de ce que le système définit comme *sortie standard*.

La sortie standard est un objet virtuel proposé par le système. On peut écrire des données dessus, mais c'est via le système que l'on décide de ce que ces données vont devenir.



Utilisateur



Matériel

Utilisateur



Utilisateur



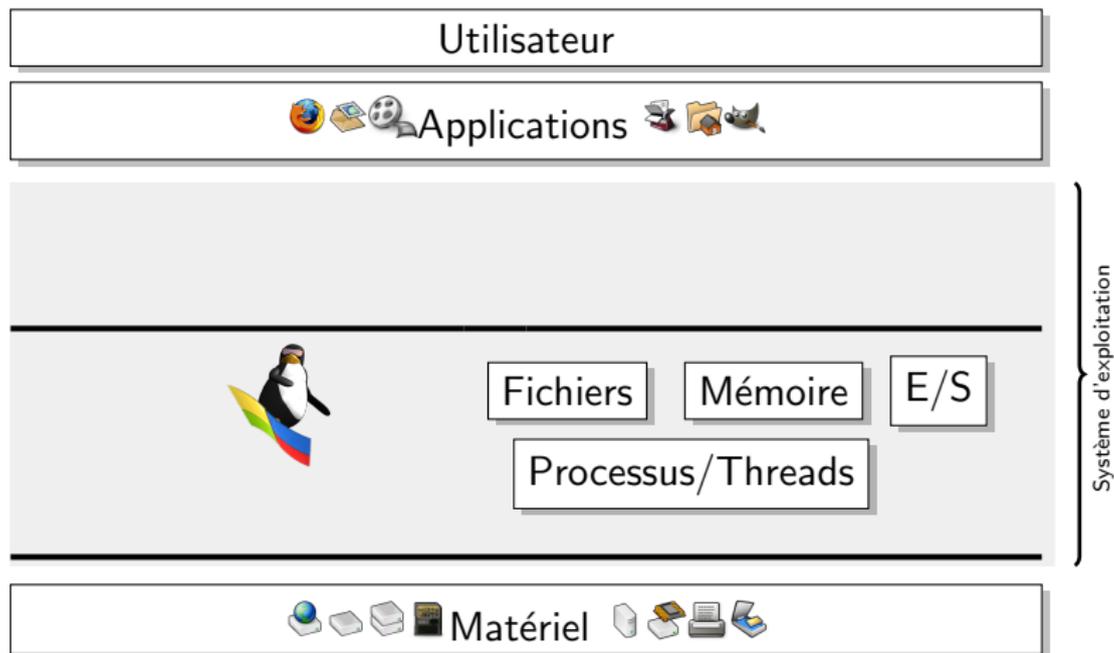
Fichiers

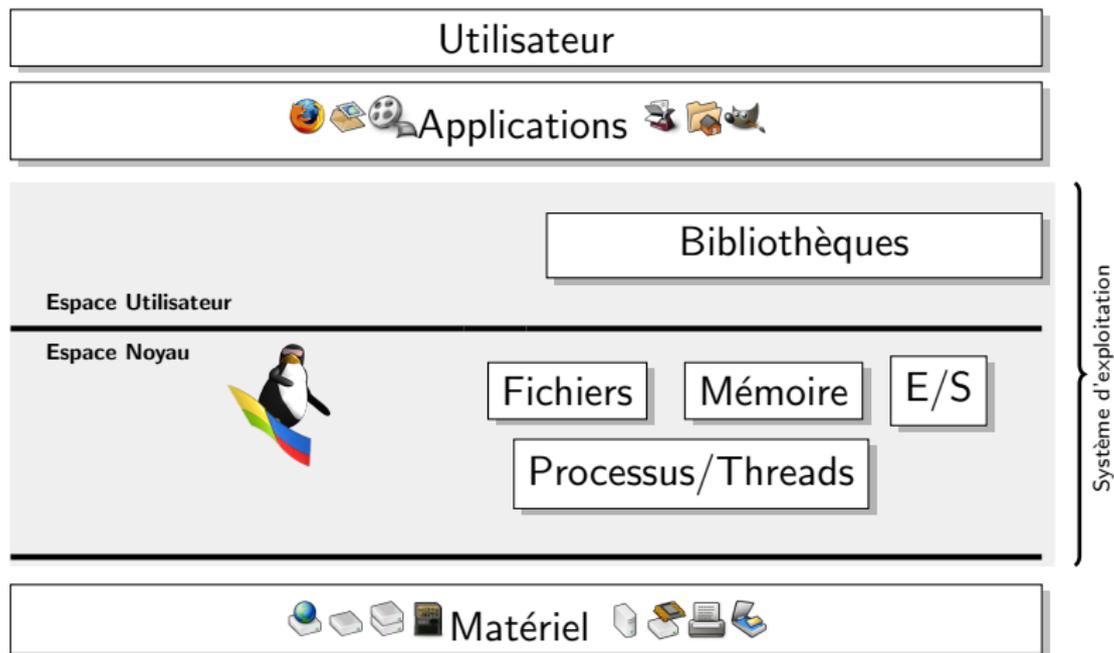
Mémoire

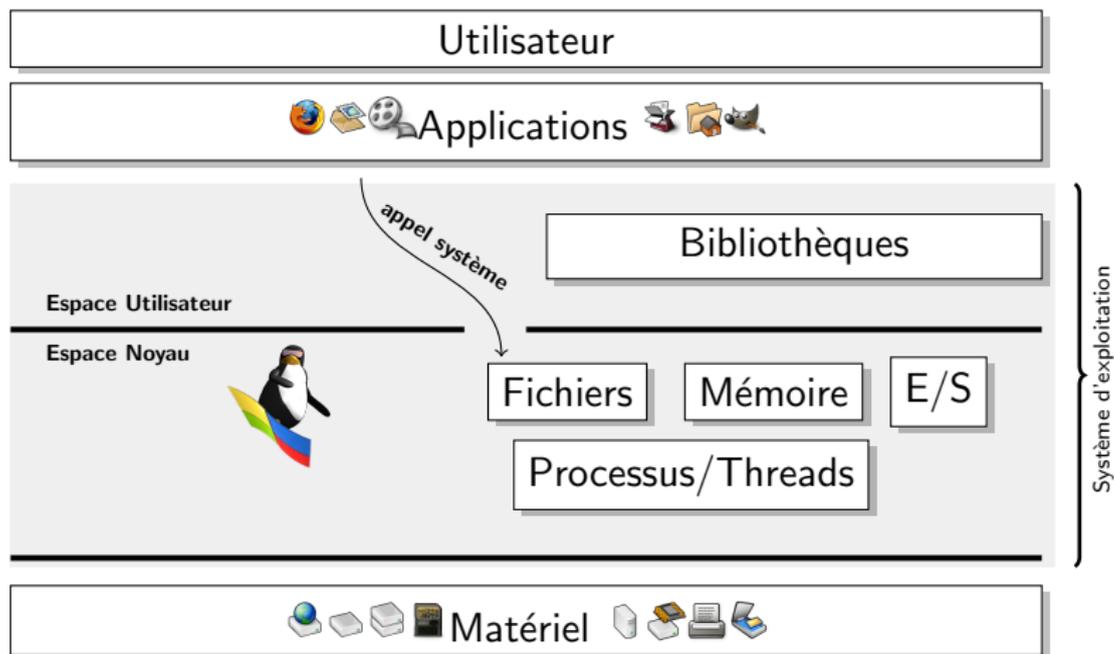
E/S

Processus/Threads









Fonctionnalités

En tant qu'interface, un système d'exploitation peut :

- tricher
 - ▶ proposer un fichier qui est en fait un objet virtuel sauvegardé ailleurs (disque réseau=*Cloud*) ;
 - ▶ enfermer le processus dans un environnement contraint (*sandbox*, *conteneur*) ;
 - ▶ proposer une machine virtuelle complète ;
- organiser
 - ▶ gérer les conflits d'accès ;
 - ▶ stopper un processus qui demande des données qui ne sont pas encore arrivées ;
 - ▶ débloquer ce processus lorsque les données arrivent ;
- assurer la sécurité
 - ▶ faire des vérifications de droits ;
 - ▶ refuser des opérations impossibles ;
 - ▶ retarder une modification qui pourrait endommager le matériel.



Notions de base

On voit apparaître dans ce qui précède certaines notions de base du système :

- **Rôle d'interface** : le système se place entre les logiciels et le matériel.
- **Objets virtuels** : le système définit des objets (fichier, processus, ...) qui sont manipulés par les logiciels.
- **Niveau de fonctionnement** : au moins deux niveaux de fonctionnement, comportant des droits différents gérés par le matériel :
 - ▶ **espace utilisateur**
 - ▶ **espace noyau**



Plan

1 Introduction

- Interface avec le matériel

2 Les processus

- État d'un processus
- Environnement
- Communication entre processus
- Les processus et python

3 Gestion de la mémoire

- Rôle de la mémoire
- Organisation de la mémoire

4 Fichiers et système de fichiers



Processus

Définition (Processus)

Un processus est une instance (une exécution) d'un programme. C'est un ensemble de données gérées par le noyau qui contient toutes les informations nécessaires afin de suivre le déroulement du programme, de le stopper et de le reprendre.



Processus

Définition (Processus)

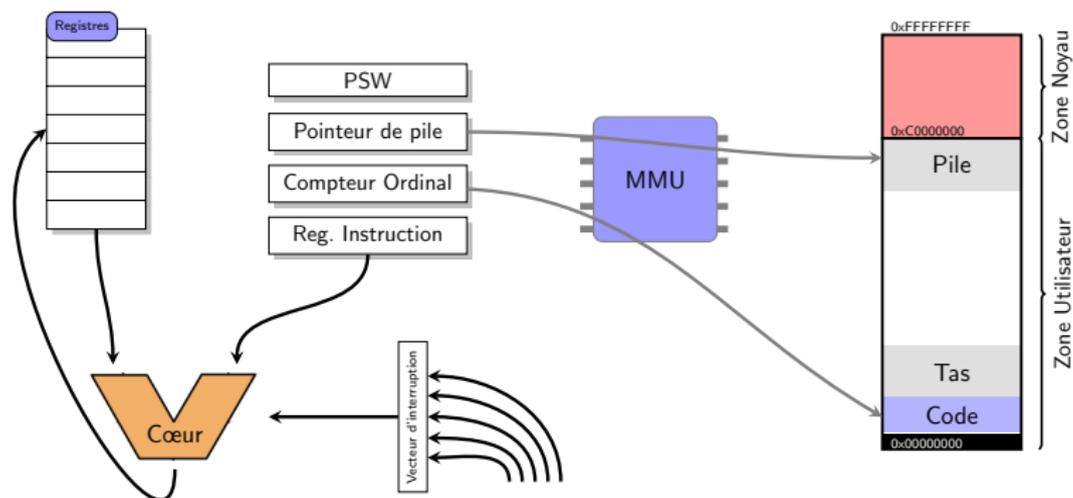
Un processus est une instance (une exécution) d'un programme. C'est un ensemble de données gérées par le noyau qui contient toutes les informations nécessaires afin de suivre le déroulement du programme, de le stopper et de le reprendre.

Son rôle est :

- d'identifier le programme ;
- de vérifier que le programme ne fait que ce qu'il a le droit de faire ;
- de lui permettre d'accéder aux fichiers et aux ressources ;
- d'isoler les programmes les uns des autres ;
- de libérer les ressources à la fin de l'exécution (fermer les fichiers, libérer toute la mémoire allouée, etc.) ;
- de permettre le passage d'une tâche à l'autre.



Processeur



Constituant d'un processus

- Position en cours dans le code du programme (Compteur ordinal)
- Contenu des registres du processeur qui l'exécute
- Informations sur la mémoire
- État (en cours, prêt, bloqué, ...)
- Ressources (fichiers, sockets, ...)
- Environnement (répertoire courant, id utilisateur, ...)
- Informations de gestion (id, pid, groupe, ...)
- ...

Sous linux ces informations peuvent se trouver dans le répertoire virtuel
`/proc/<pid processus>`



État d'un processus

Rappel : Le système doit

- gérer l'accès au processeur
- gérer l'occupation de la mémoire

Définition (état d'un processus)

Le système doit donc gérer plusieurs listes de processus,

- ceux qui peuvent s'exécuter
- ceux qui sont en mémoire
- ceux qui sont bloqués car demandant une ressource occupée
- ...

On parle *d'état du processus*

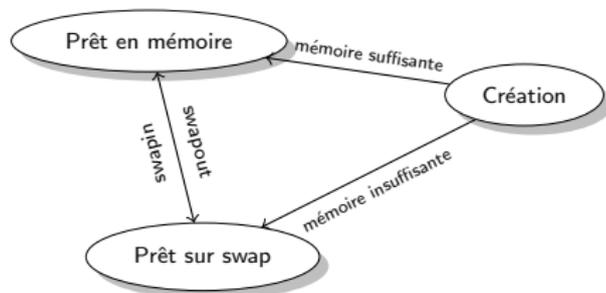


État d'un processus

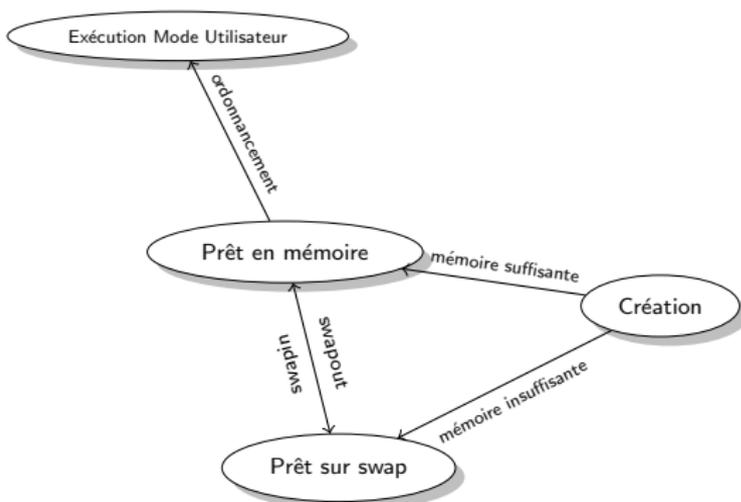


Création

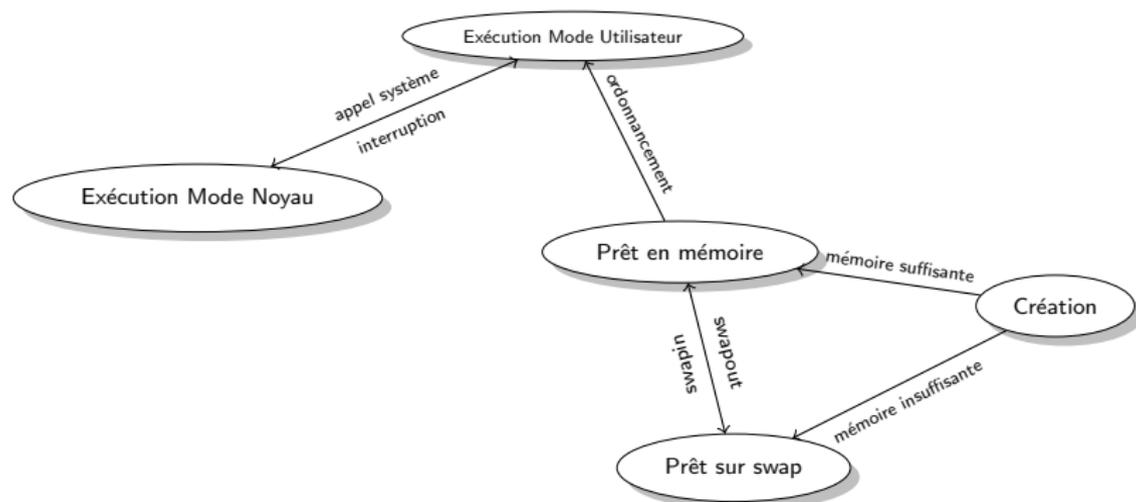
État d'un processus



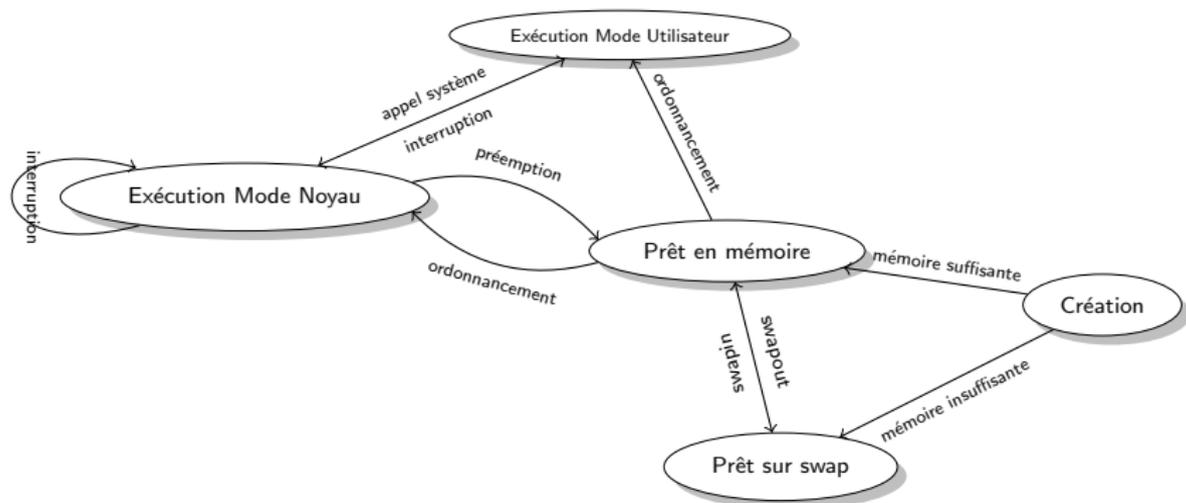
État d'un processus



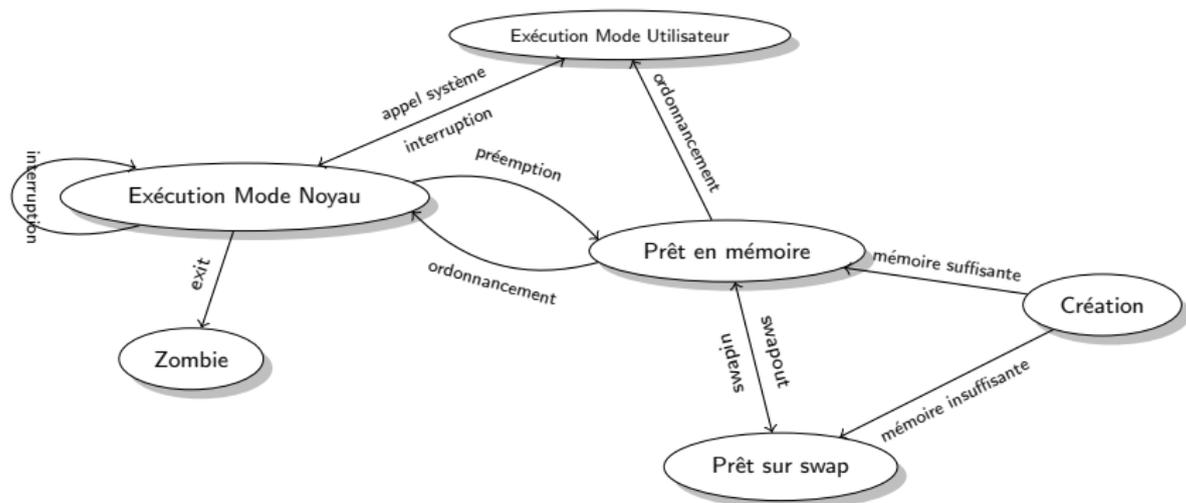
État d'un processus



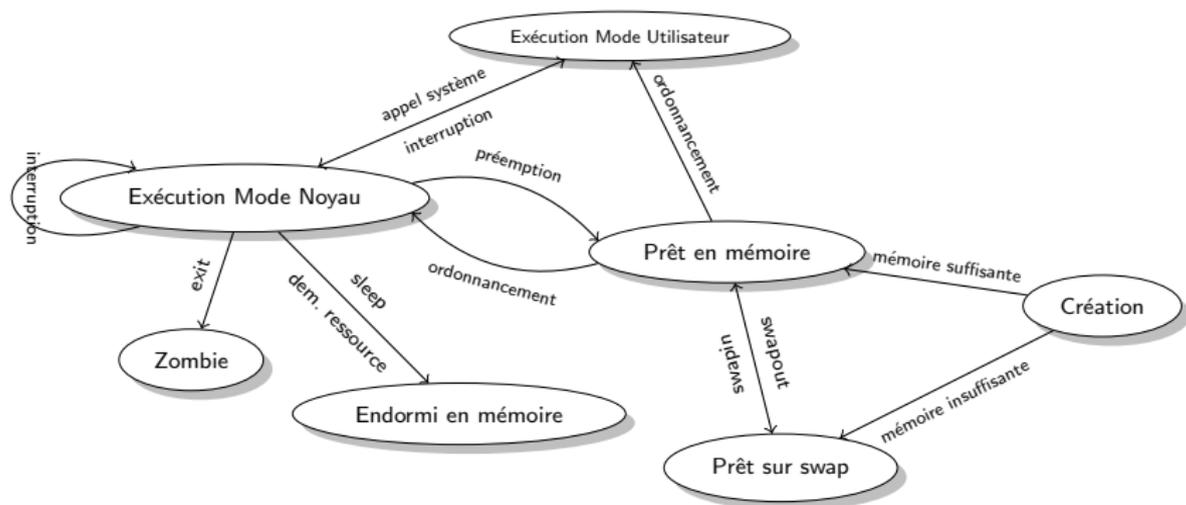
État d'un processus



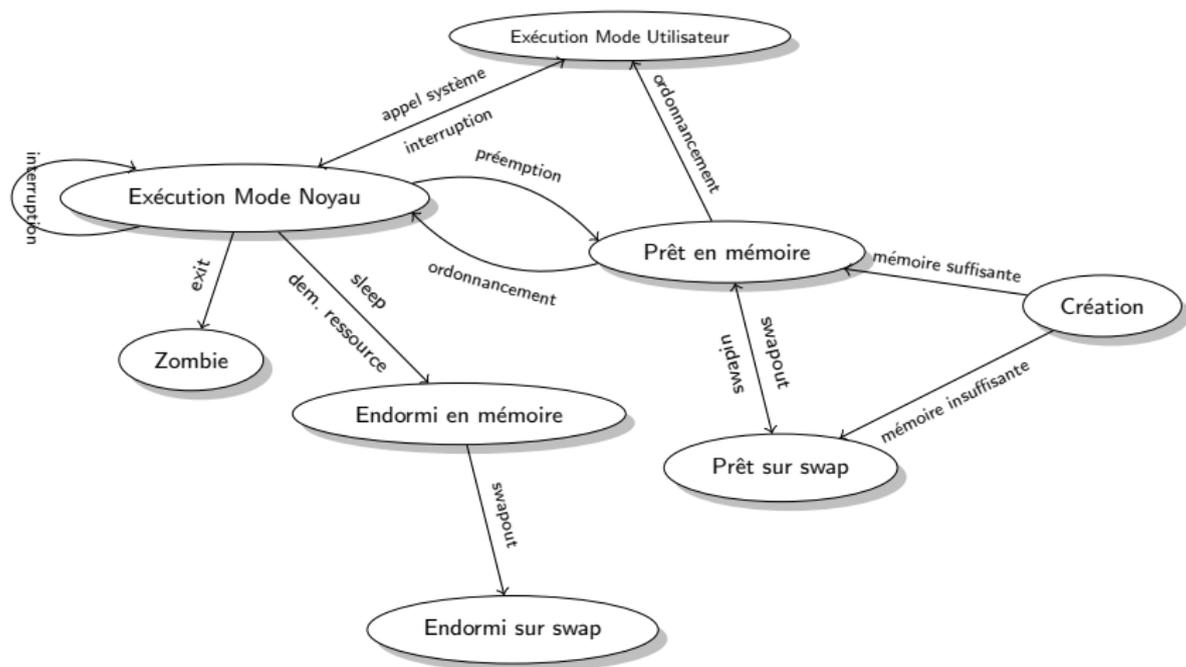
État d'un processus



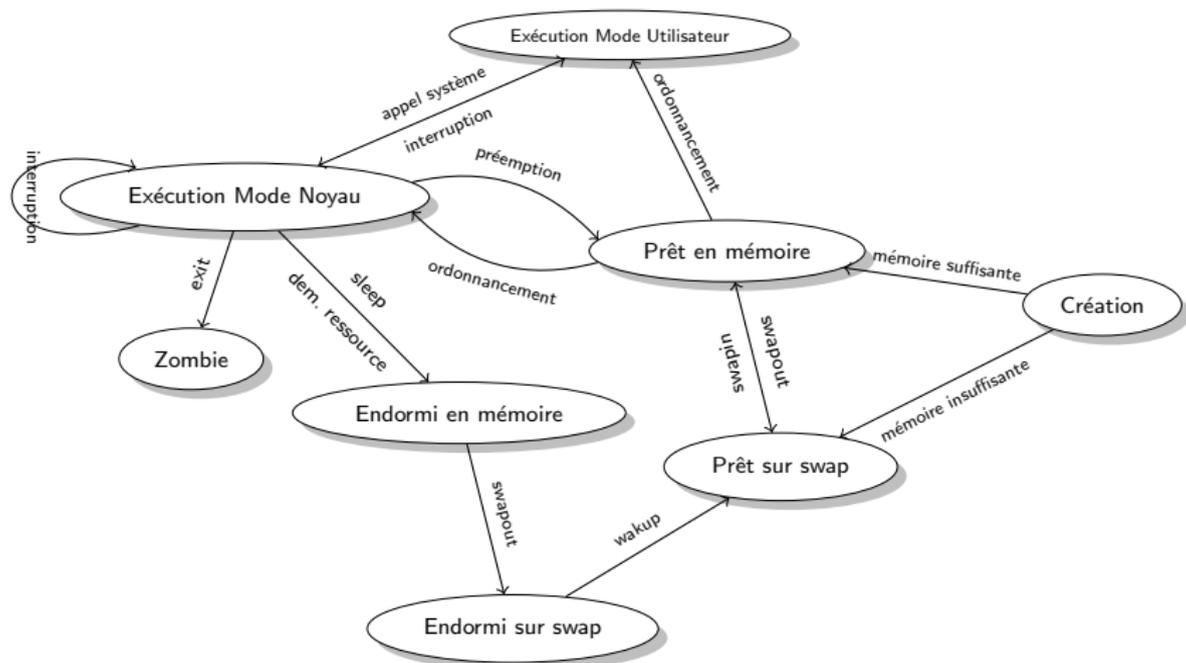
État d'un processus



État d'un processus



État d'un processus



Environnement

Les processus fonctionnent dans un environnement :

- Ils sont exécutés depuis un répertoire le *Working Directory* et toutes les actions en rapport avec le système de fichiers se font à partir de là.
- Ils sont la propriété d'un utilisateur et héritent de ses droits (avec possibilité d'escalade de droits).
- Ils ont une définition de *l'environnement linguistique* (langue, affichage des dates, ...)
- ...

La plupart des informations sur l'environnement sont transmises par des variables héritées du processus qui les lance via des *variables d'environnement*



Variables d'environnement

- `PATH` : liste des répertoires où sont recherchés les exécutables
- `LD_LIBRARY_PATH` : liste des répertoires où sont recherchés les fichiers de bibliothèque au lancement des applications.
- `USER` : nom de l'utilisateur.
- `HOME` : répertoire de l'utilisateur sous Unix.
- `HOME_DRIVE` et `HOME_PATH` : disque et répertoire de l'utilisateur sous Windows.
- `PYTHONPATH`, `PYTHONHOME` pour la gestion de python (les *virtual env*).
- ...



Variables d'environnement (suite)

Ces variables peuvent être lues et modifiées sous Linux (bash)

```
# lire une variable
echo $VARIABLE
# lire toutes les variables
env
# modifier seulement pour une commande
VARIABLE=valeur commande
# exporter une variable dans l'environnement d'un
# processus pour affecter tous ses descendants
export VARIABLE=valeur
# ajouter l'export dans un script pour avoir un effet permanent
# - pour un utilisateur
echo "export VARIABLE=valeur" >> $HOME/.bashrc
# - pour tous les utilisateurs
echo "export VARIABLE=valeur" >> /etc/profile.d/mesvars.sh
```

Attention, modifier de manière permanente et erronée certaines variables comme PATH peut rendre le système inutilisable !



Variables d'environnement (suite)

Ces variables peuvent être lues et modifiées sous Windows

```
# lire une variable
echo %VARIABLE%
# lire toutes les variables
set
# exporter une variable dans l'environnement d'un
# processus pour affecter tous ses descendants
set VARIABLE valeur
# ajouter de manière permanente à l'utilisateur
setx VARIABLE valeur
# ajouter de manière permanente au système
setx /M VARIABLE valeur
```



Paramètres

Les processus ont des paramètres, ils permettent par exemple de spécifier les détails d'une commande.

- En ligne de commandes, les paramètres sont les mots passés après le nom de la commande. Par exemple :

```
cp fichierdepart /repertoire/arrive
```

- Selon la commande, certains de ces paramètres sont obligatoires d'autre sont des options :

- ▶ Pour `cp` « `-r` » est l'option « copier récursivement » c'est à dire le répertoire et tous les sous répertoires.

```
cp -r ./repertoire/depart /repertoire/arrive
```

- ▶ les options sont souvent reconnues grâce au caractère `-` sous linux et / sous windows.
 - ▶ elles peuvent aussi utiliser `--` lorsque l'option est un mot de plusieurs caractères
 - ▶ les options peuvent prendre des paramètres `ls --sort size`
- Il existe des bibliothèques pour simplifier la lecture des paramètres. En python `argparse`.

Pour connaitre les fonctionnalités d'une commande, il existe la commande `man NOMDELACOMMANDE`



Communication entre processus

Les processus sont relativement isolés. Leurs codes ne peuvent pas se perturber car ils n'utilisent pas les mêmes espaces en mémoire. Mais il est parfois nécessaire de les faire communiquer.

- Pour prévenir d'un événement (une alarme) il y a les signaux.
- Pour échanger des données il y a des objets spéciaux proches des fichiers : *tube (pipe)*, *socket*, ...



Signaux

Définition (signaux)

Les signaux fonctionnent comme des alarmes, ils « dérangent » le processus pour lui faire exécuter une action prévue à l'avance. Il y en a un nombre fixe, prédéfinis.

On peut modifier l'action prévue mais ce n'est pas au programme.

Vous pouvez lancer un signal à un processus via la commande ou la fonction `kill`. Par exemple

```
# tue le processus de numéro 3452 en lui envoyant le signal 9
kill -9 3452
# stoppe tous les processus dont la commande est emacs en leur
# envoyant le signal SIGSTOP
killall -SIGSTOP emacs
```



Échange de données

L'échange de données est plus complexe, car il permet de communiquer toute forme de données entre processus sur des machines qui peuvent être distantes.

En TP vous utiliserez :

- Les *pipe*, uniquement locaux qui permettent de transmettre des données d'un processus à l'autre souvent via les entrées et sortie standards.

```
ls -l | grep toto
```

Le pipe `<< | >>` permet d'envoyer la sortie de la commande `ls` sur l'entrée de la commande `grep`

- Les *socket*, outils réseaux qui permettent à un processus donné de se connecter à un serveur distant pour lui transmettre des informations.
`nc -l 8080` ouvre un serveur en écoute sur votre ordinateur, ce serveur accepte un client qui se connecte sur le port 8080
`nc www.google.fr 80` cette commande se connecte sur le port 80 du serveur `www.google.fr` (c'est le serveur web).

La commande `nc`, en plus d'ouvrir la socket en mode TCP, permet d'envoyer tout ce qui est écrit sur son entrée standard et affiche tout ce qui est reçu via le réseau.



Notions définies par le système

En ce qui concerne les processus, les notions importantes sont :

- *Les processus* : un programme en train de tourner.
- *L'état du processus* : son status à un instant donné.
- *L'environnement* : la description du lien entre le processus et le système.
- *Les paramètres* : informations données au lancement du processus.
- *Les canaux de communications* : outils du système pour permettre l'échange de données.
- *Les signaux* : moyen pour un processus ou le système de prévenir et dérouter un autre processus.
- *Interblocage* : risque pour plusieurs processus qui interagissent de se bloquer l'un et l'autre.
- *Assynchronisme* : fait de ne plus voir un programme comme une suite d'opérations qui se suivent mais comme des réponses à des événements.



Module python en rapport avec les processus

Plusieurs modules sont utiles pour la gestion des processus par python.

Mais il faut savoir que toute interaction avec le système est un risque pour la portabilité.

- Toute opération nécessitant une action du système doit passer par un appel système.
- Au départ chaque système avait sa propre liste d'appel, avec des paramètres spécifiques ou des résultats divers.
- \implies il y a donc un souci pour faire des appels système dans un langage comme python qui doit fonctionner sur des systèmes différents.
- Il y a eu une tentative de normalisation : les normes POSIX utilisées par les unixes.

Mais tous les systèmes n'ont pas choisis de l'implémenter totalement ou d'utiliser la même version de la norme.



Module python en rapport avec les processus (digression)

Il y a plusieurs grandes familles notamment : les unices (linux, bsd, macos...) et les MS windows.

Par exemple pour créer un processus, la norme POSIX définie dans les années 70 imposait 2 appels système :

- un pour créer une copie du processus courant `fork` ;
- un autre pour remplacer son code par un nouveau `exec`.

Cela était le choix efficace à cause du peu de mémoire disponible.

Windows défini dans les années 80 a choisi de le faire en une seule étape pour les deux opérations `createprocess`.

Mais comment faire un module de langage qui implémente le `fork` ?



Module python en rapport avec les processus (fin)

PYTHON va donc proposer plusieurs modules pour la gestion des processus.

- Le module `os` propose `fork` mais ne fonctionnera que sous unix.
- Le module `win32process` propose `createprocess` mais ne fonctionnera que sous windows.

La solution est de définir des modules plus spécialisés avec les primitives réellement utiles au programmeurs et faisables quel que soit le système :

- `multiprocessing` va permettre de gérer les processus dans le contexte du calcul parallèle
 - ▶ lancer un groupe de processus ;
 - ▶ gérer l'échange de données entre eux.
- `subprocess` va permettre de gérer les processus dans le cas de l'appel d'une commande
 - ▶ lancer une commande avec ses paramètres ;
 - ▶ gérer les entrées sortie et code d'erreur.



Plan

1 Introduction

- Interface avec le matériel

2 Les processus

- État d'un processus
- Environnement
- Communication entre processus
- Les processus et python

3 Gestion de la mémoire

- Rôle de la mémoire
- Organisation de la mémoire

4 Fichiers et système de fichiers



À quoi sert la mémoire ?

- Stocker le code du programme



À quoi sert la mémoire ?

- Stocker le code du programme
- Stocker le code des fonctions partagées



À quoi sert la mémoire ?

- Stocker le code du programme
- Stocker le code des fonctions partagées
- Stocker les variables globales (Tas)



À quoi sert la mémoire ?

- Stocker le code du programme
- Stocker le code des fonctions partagées
- Stocker les variables globales (Tas)
- Stocker les variables locales (Pile)

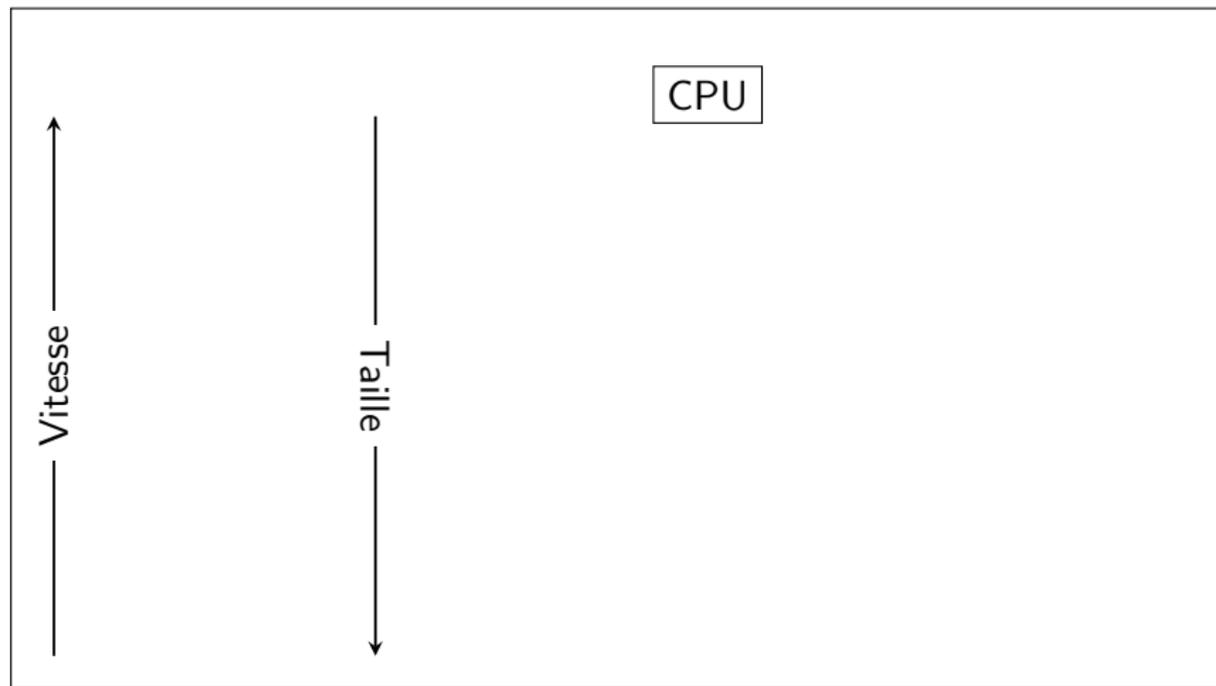


À quoi sert la mémoire ?

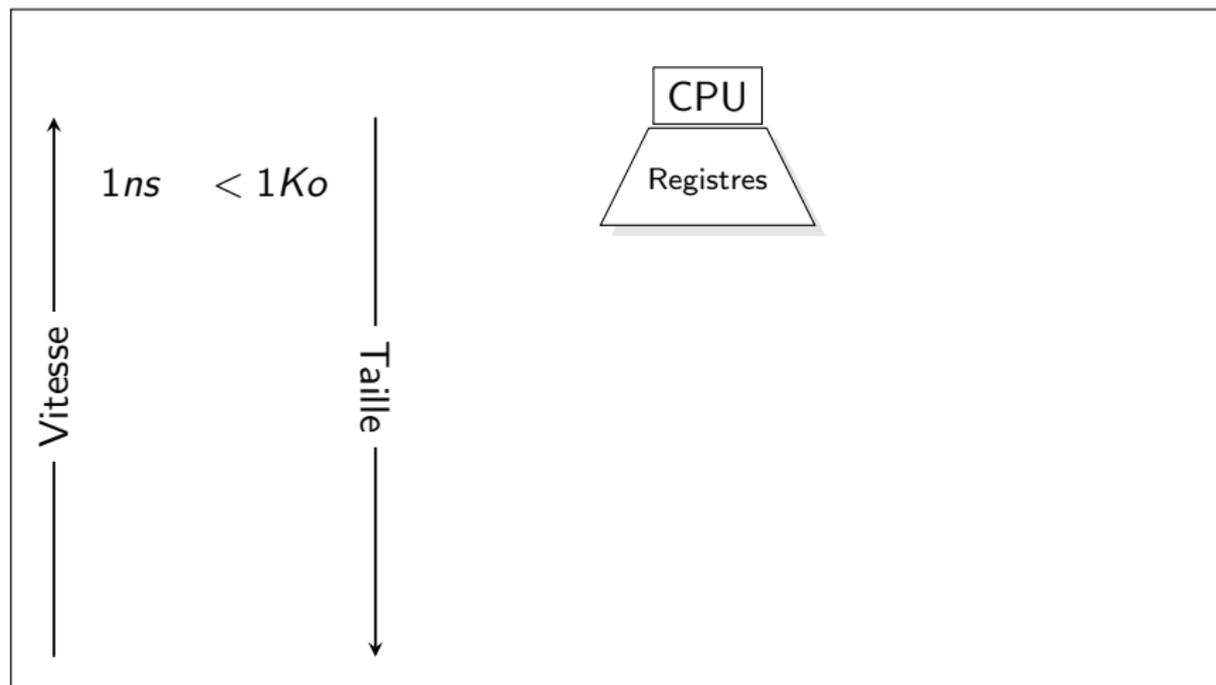
- Stocker le code du programme
- Stocker le code des fonctions partagées
- Stocker les variables globales (Tas)
- Stocker les variables locales (Pile)
- Une donnée pour pouvoir être traitée doit être dans un registre du processeur.



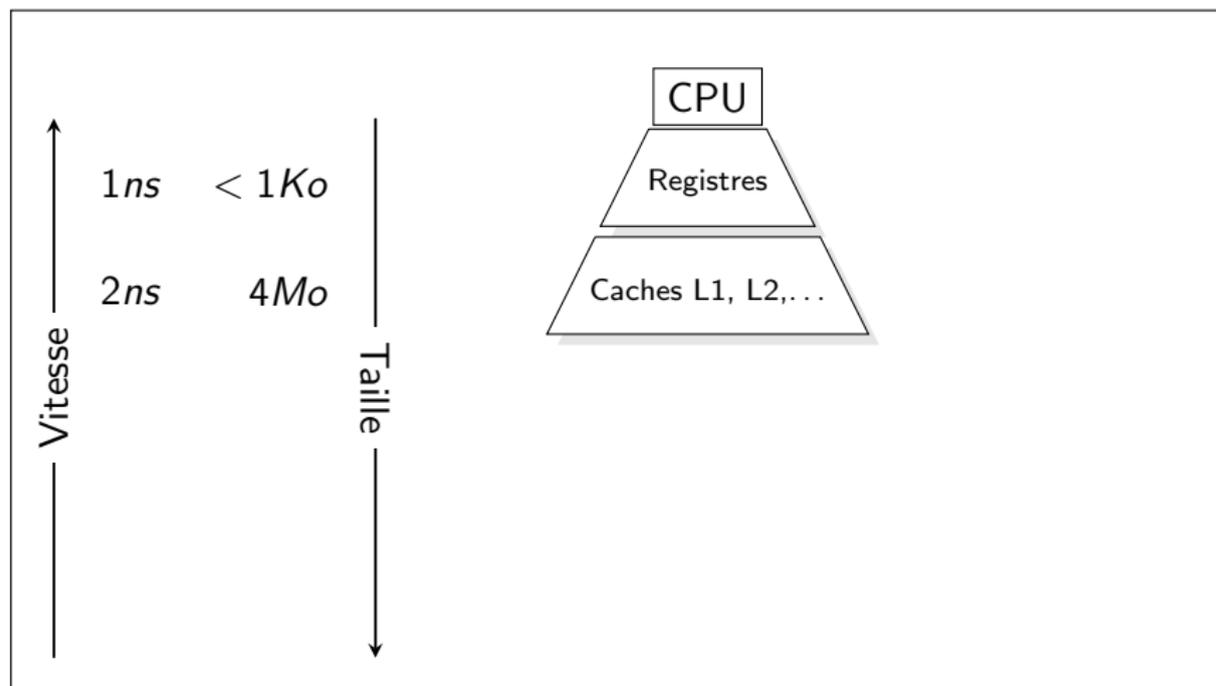
Hiérarchie des mémoires



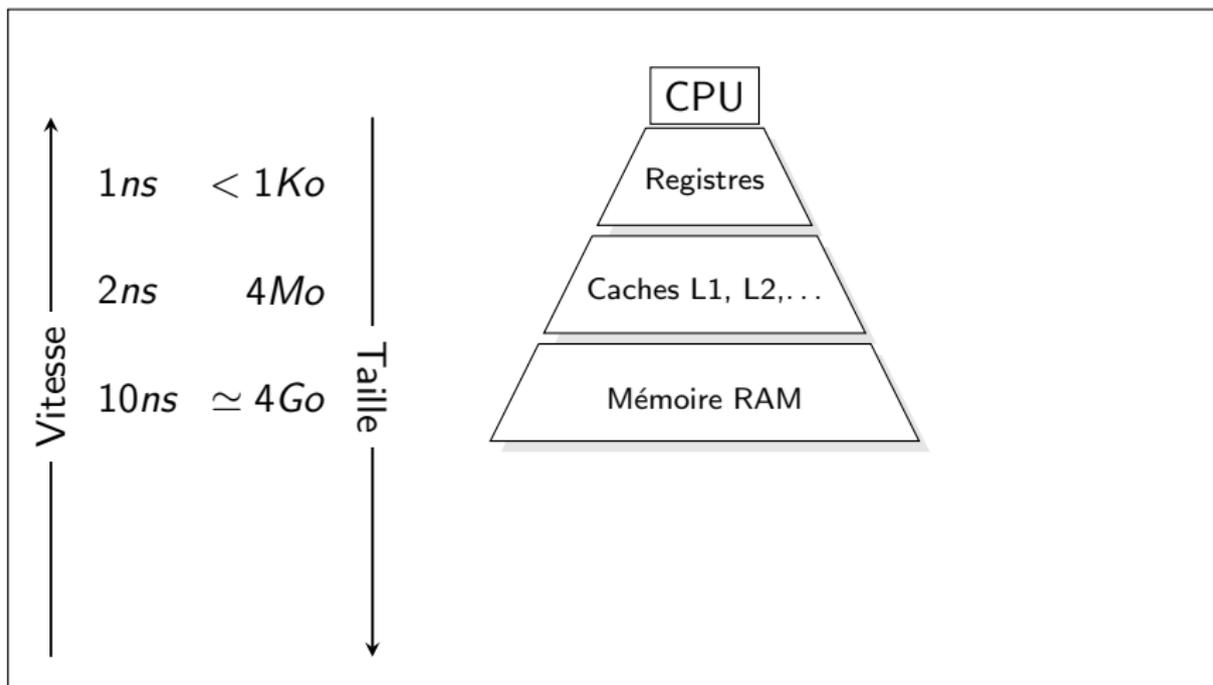
Hiérarchie des mémoires



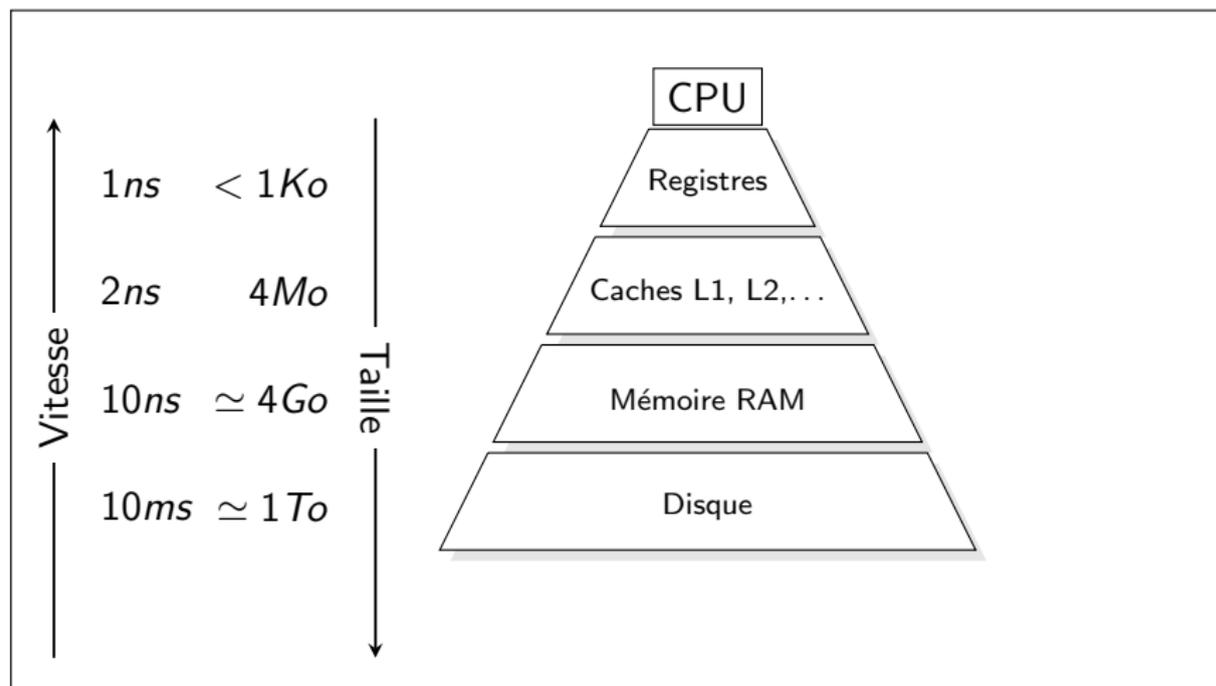
Hiérarchie des mémoires



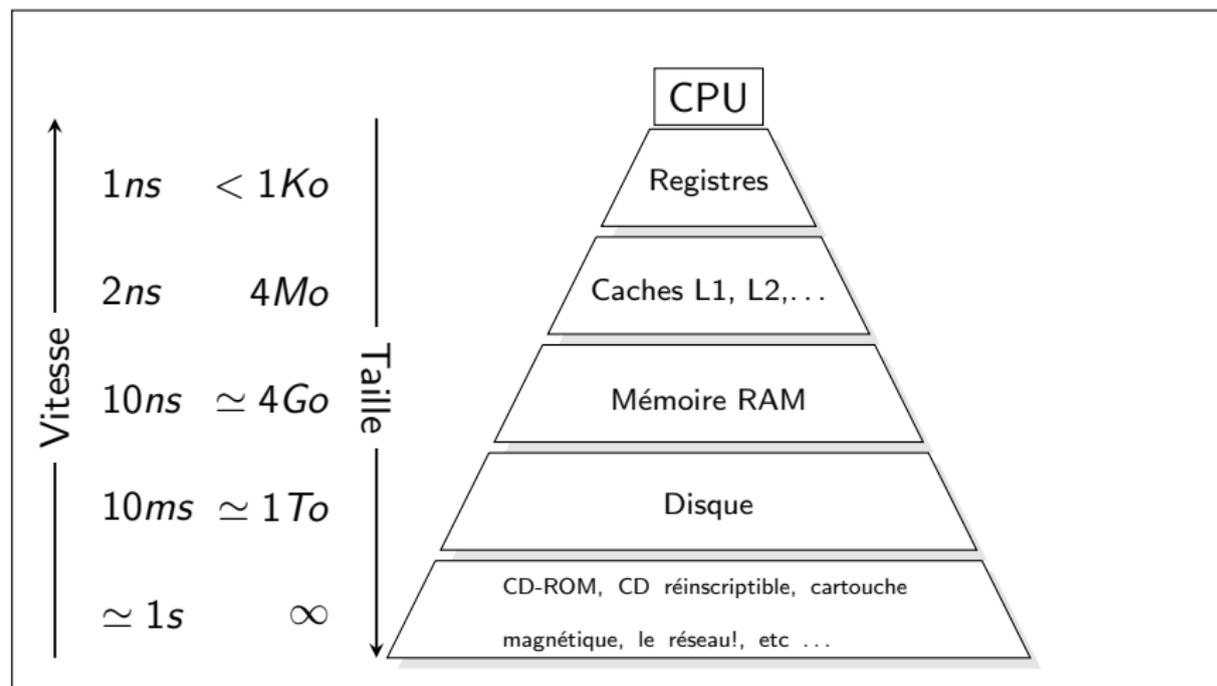
Hiérarchie des mémoires



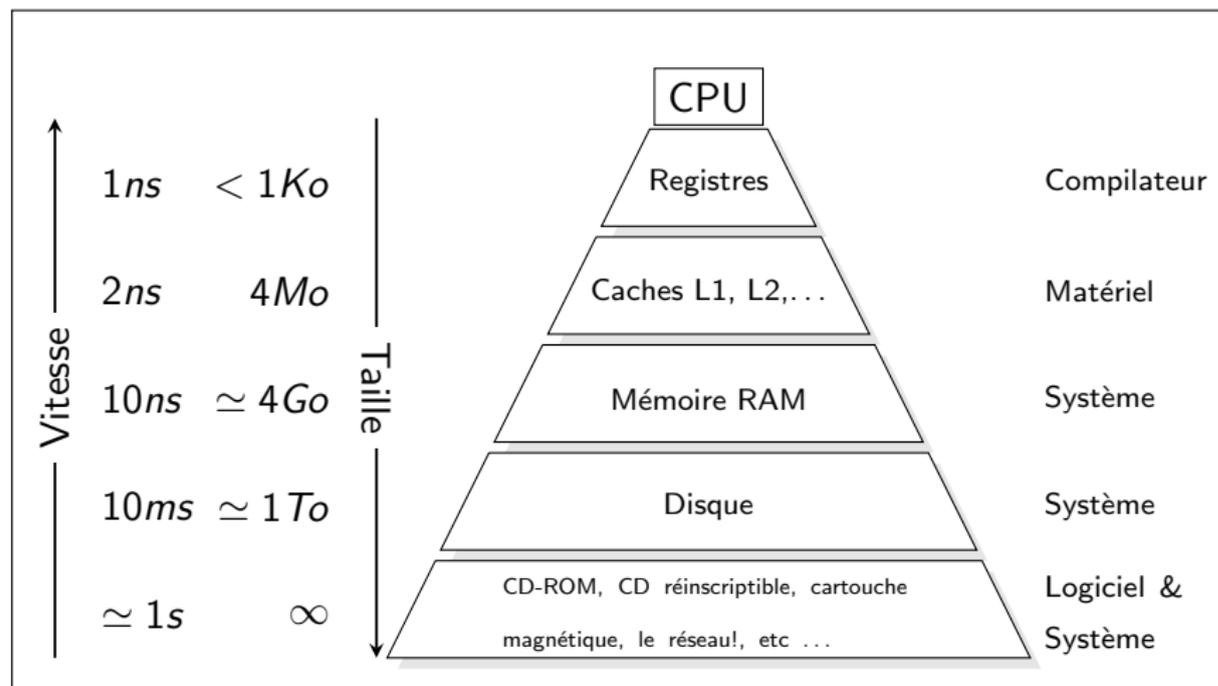
Hiérarchie des mémoires



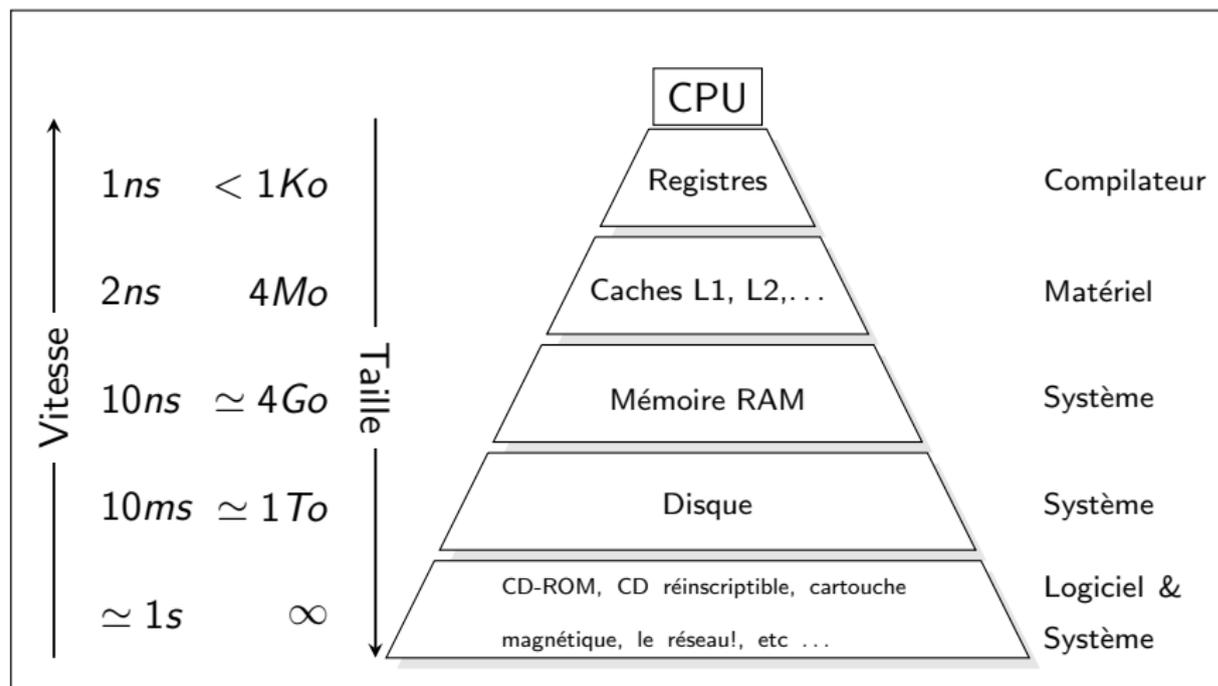
Hiérarchie des mémoires



Hiérarchie des mémoires



Hiérarchie des mémoires



Principe de localité : Les programmes tendent à utiliser des instructions et des données accédées dans le passé (localité temporelle) ou proches de celle-ci (localité spatiale)



Cercle *vertueux* ?

- Les programmeurs suivent naturellement le principe de localité.
- Donc ceux qui conçoivent les systèmes s'en aperçoivent et l'utilisent cette propriété (mécanismes de pages, overlay...).
- Donc pour garder de bonnes performances les programmeurs suivent ce principe.
- Donc ceux qui conçoivent les processeurs l'utilisent (pipeline, cache mémoire...)



Cercle *vertueux* ?

- Les programmeurs suivent naturellement le principe de localité.
- Donc ceux qui conçoivent les systèmes s'en aperçoivent et l'utilisent cette propriété (mécanismes de pages, overlay...).
- Donc pour garder de bonnes performances les programmeurs suivent ce principe.
- Donc ceux qui conçoivent les processeurs l'utilisent (pipeline, cache mémoire...)
- Donc optimiser un code revient souvent à améliorer sa localité
- ...
- Ce principe prend de plus en plus d'importance.



Le problème

- Chaque accès à une variable ou une fonction est un accès à la mémoire.
- Le compilateur doit faire la correspondance entre un nom qui est dans le code et une adresse demandée par les opérations du processeur.
- Mais le programme doit être portable, c'est à dire fonctionner sur des ordinateurs différents et en concurrence avec d'autres processus.
- Le compilateur ne peut donc pas donner l'adresse de la case mémoire où se trouve réellement une variable.

Historiquement, il y a plusieurs façon de régler le problème :

- translation d'adresse ;
- notions de segment ;
- mémoire virtuelle.

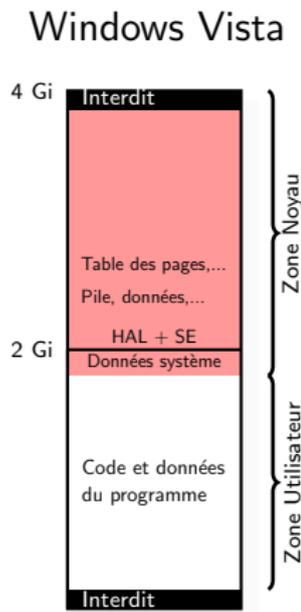
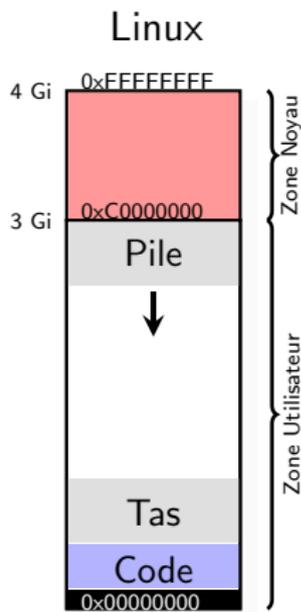


Mémoire virtuelle

- Idée : le processeur travaille avec des adresses mémoires différentes des adresses physiques.
- *Capacité d'adressage* : l'ensemble des adresses que l'on peut coder, par ex. 4Go sur un processeur 32 bits (0x00000000 - 0xFFFFFFFF), 16Eo pour un 64 bits.
- *Espace d'adressage* : partie utilisable par le processus.
- L'espace d'adressage est partitionné en segments dépendants du système d'exploitation.
- À chaque lecture mémoire, l'adresse physique est calculée.
- Ce calcul permet de faire des vérifications supplémentaires :
 - ▶ présence effective de l'adresse en mémoire ;
 - ▶ droits.
- Il faut un matériel dédié la *Memory Management Unit*.



Espace d'adressage d'un processus



Pagination

Comment faire la correspondance par exemple sur un processeur 32 bits

- La MMU utilise une table de correspondance $@ \text{logique} \mapsto @ \text{physique}$
- La table est stockée en mémoire
- Si on stocke tout, 2^{32} @ à conserver
- \Rightarrow On rassemble les adresses en *pages* (par ex. 4096 octets)
- \Rightarrow 1 Mo à stocker (par processus), la plupart du temps les cases de la table sont vides.
- Donc on utilise plusieurs niveaux

Ob 0100 1110 0001 0101 1001 1001 0101 1001



Pagination

Comment faire la correspondance par exemple sur un processeur 32 bits

- La MMU utilise une table de correspondance $@ \textit{logique} \mapsto @ \textit{physique}$
- La table est stockée en mémoire
- Si on stocke tout, 2^{32} @ à conserver
- \Rightarrow On rassemble les adresses en *pages* (par ex. 4096 octets)
- \Rightarrow 1 Mo à stocker (par processus), la plupart du temps les cases de la table sont vides.
- Donc on utilise plusieurs niveaux

Ob 0100 1110 :0001 0101 1001 :1001 0101 1001



Pagination

Comment faire la correspondance par exemple sur un processeur 32 bits

- La MMU utilise une table de correspondance $@ \text{logique} \mapsto @ \text{physique}$
- La table est stockée en mémoire
- Si on stocke tout, 2^{32} @ à conserver
- \Rightarrow On rassemble les adresses en *pages* (par ex. 4096 octets)
- \Rightarrow 1 Mo à stocker (par processus), la plupart du temps les cases de la table sont vides.
- Donc on utilise plusieurs niveaux

Ob 0100 1110 :0001 0101 1001 :1001 0101 1001

Ox 4e : 159 : 959



Pagination

Comment faire la correspondance par exemple sur un processeur 32 bits

- La MMU utilise une table de correspondance $@ \text{logique} \mapsto @ \text{physique}$
- La table est stockée en mémoire
- Si on stocke tout, 2^{32} @ à conserver
- \Rightarrow On rassemble les adresses en *pages* (par ex. 4096 octets)
- \Rightarrow 1 Mo à stocker (par processus), la plupart du temps les cases de la table sont vides.
- Donc on utilise plusieurs niveaux

Ob 0100 1110 :0001 0101 1001 :1001 0101 1001

Ox	4e	:	159	:	959
	rep	:	page	:	offset

rep index dans le répertoire des pages, *page* index dans la table des pages, *offset* position dans la page.



Ex. de pagination (suite)

La MMU du pentium permet une table de page avec 2 indirections :

- Pour lire l'adresse `0x4e159959`



Ex. de pagination (suite)

La MMU du pentium permet une table de page avec 2 indirections :

- Pour lire l'adresse `0x4e159959`
- La MMU contient un registre donnant l'adresse de la *table des répertoires de pages*.



Ex. de pagination (suite)

La MMU du pentium permet une table de page avec 2 indirections :

- Pour lire l'adresse $0x4e159959$
- La MMU contient un registre donnant l'adresse de la *table des répertoires de pages*.
- On trouve la table des pages à l'index $0x4e = 78$ de la table des répertoires de page



Ex. de pagination (suite)

La MMU du pentium permet une table de page avec 2 indirections :

- Pour lire l'adresse $0x4e159959$
- La MMU contient un registre donnant l'adresse de la *table des répertoires de pages*.
- On trouve la table des pages à l'index $0x4e = 78$ de la table des répertoires de page
- On trouve la page à l'index $0x159 = 345$ de la table des pages



Ex. de pagination (suite)

La MMU du pentium permet une table de page avec 2 indirections :

- Pour lire l'adresse $0x4e159959$
- La MMU contient un registre donnant l'adresse de la *table des répertoires de pages*.
- On trouve la table des pages à l'index $0x4e = 78$ de la table des répertoires de page
- On trouve la page à l'index $0x159 = 345$ de la table des pages
- La page se trouve en mémoire physique à une adresse $0x?????000$ le mot mémoire demandé est à l'adresse physique $0x?????959$



Ex. de pagination (suite)

La MMU du pentium permet une table de page avec 2 indirections :

- Pour lire l'adresse $0x4e159959$
- La MMU contient un registre donnant l'adresse de la *table des répertoires de pages*.
- On trouve la table des pages à l'index $0x4e = 78$ de la table des répertoires de page
- On trouve la page à l'index $0x159 = 345$ de la table des pages
- La page se trouve en mémoire physique à une adresse $0x?????000$ le mot mémoire demandé est à l'adresse physique $0x?????959$
- Si on ne trouve pas la page, il y a interruption



Ex. de pagination (suite)

La MMU du pentium permet une table de page avec 2 indirections :

- Pour lire l'adresse $0x4e159959$
- La MMU contient un registre donnant l'adresse de la *table des répertoires de pages*.
- On trouve la table des pages à l'index $0x4e = 78$ de la table des répertoires de page
- On trouve la page à l'index $0x159 = 345$ de la table des pages
- La page se trouve en mémoire physique à une adresse $0x?????000$ le mot mémoire demandé est à l'adresse physique $0x?????959$
- Si on ne trouve pas la page, il y a interruption
 - ▶ Si la page existe mais n'est pas en mémoire *Défaut de page*



Ex. de pagination (suite)

La MMU du pentium permet une table de page avec 2 indirections :

- Pour lire l'adresse $0x4e159959$
- La MMU contient un registre donnant l'adresse de la *table des répertoires de pages*.
- On trouve la table des pages à l'index $0x4e = 78$ de la table des répertoires de page
- On trouve la page à l'index $0x159 = 345$ de la table des pages
- La page se trouve en mémoire physique à une adresse $0x?????000$ le mot mémoire demandé est à l'adresse physique $0x?????959$
- Si on ne trouve pas la page, il y a interruption
 - ▶ Si la page existe mais n'est pas en mémoire *Défaut de page*
 - ▶ Si la page n'existe pas ou est interdite le processus est signalé (SIGSEGV)



Ex. de pagination (suite)

La MMU du pentium permet une table de page avec 2 indirections :

- Pour lire l'adresse $0x4e159959$
- La MMU contient un registre donnant l'adresse de la *table des répertoires de pages*.
- On trouve la table des pages à l'index $0x4e = 78$ de la table des répertoires de page
- On trouve la page à l'index $0x159 = 345$ de la table des pages
- La page se trouve en mémoire physique à une adresse $0x?????000$ le mot mémoire demandé est à l'adresse physique $0x?????959$
- Si on ne trouve pas la page, il y a interruption
 - ▶ Si la page existe mais n'est pas en mémoire *Défaut de page*
 - ▶ Si la page n'existe pas ou est interdite le processus est signalé (SIGSEGV)
- Pour changer de contexte, il faut changer le registre d'adresse du répertoire de page.



Remarque

- En réalité, plusieurs accès mémoire par accès demandé



Remarque

- En réalité, plusieurs accès mémoire par accès demandé
 - ▶ Il faut un cache de traduction pour améliorer le **TLB** « Translation Lookaside Buffer »



Remarque

- En réalité, plusieurs accès mémoire par accès demandé
 - ▶ Il faut un cache de traduction pour améliorer le *TLB* « Translation Lookaside Buffer »
 - ▶ Équilibre taille des tables/nombre d'indirections



Remarque

- En réalité, plusieurs accès mémoire par accès demandé
 - ▶ Il faut un cache de traduction pour améliorer le *TLB* « Translation Lookaside Buffer »
 - ▶ Équilibre taille des tables/nombre d'indirections
- L'adresse d'une page n'utilise qu'une partie d'un mot mémoire (20 bits/32), les bits restant servent à stocker des informations :



Remarque

- En réalité, plusieurs accès mémoire par accès demandé
 - ▶ Il faut un cache de traduction pour améliorer le **TLB** « Translation Lookaside Buffer »
 - ▶ Équilibre taille des tables/nombre d'indirections
- L'adresse d'une page n'utilise qu'une partie d'un mot mémoire (20 bits/32), les bits restant servent à stocker des informations :
 - ▶ La page peut-elle être mise en cache ?



Remarque

- En réalité, plusieurs accès mémoire par accès demandé
 - ▶ Il faut un cache de traduction pour améliorer le **TLB** « Translation Lookaside Buffer »
 - ▶ Équilibre taille des tables/nombre d'indirections
- L'adresse d'une page n'utilise qu'une partie d'un mot mémoire (20 bits/32), les bits restant servent à stocker des informations :
 - ▶ La page peut-elle être mise en cache ?
 - ▶ La page existe ?



Remarque

- En réalité, plusieurs accès mémoire par accès demandé
 - ▶ Il faut un cache de traduction pour améliorer le **TLB** « Translation Lookaside Buffer »
 - ▶ Équilibre taille des tables/nombre d'indirections
- L'adresse d'une page n'utilise qu'une partie d'un mot mémoire (20 bits/32), les bits restant servent à stocker des informations :
 - ▶ La page peut-elle être mise en cache ?
 - ▶ La page existe ?
 - ▶ La page est-elle en accès lecture/écriture/exécution ?



Remarque

- En réalité, plusieurs accès mémoire par accès demandé
 - ▶ Il faut un cache de traduction pour améliorer le **TLB** « Translation Lookaside Buffer »
 - ▶ Équilibre taille des tables/nombre d'indirections
- L'adresse d'une page n'utilise qu'une partie d'un mot mémoire (20 bits/32), les bits restant servent à stocker des informations :
 - ▶ La page peut-elle être mise en cache ?
 - ▶ La page existe ?
 - ▶ La page est-elle en accès lecture/écriture/exécution ?
 - ▶ La page est modifiée par rapport au disque (Dirty bit) ?



Remarque

- En réalité, plusieurs accès mémoire par accès demandé
 - ▶ Il faut un cache de traduction pour améliorer le **TLB** « Translation Lookaside Buffer »
 - ▶ Équilibre taille des tables/nombre d'indirections
- L'adresse d'une page n'utilise qu'une partie d'un mot mémoire (20 bits/32), les bits restant servent à stocker des informations :
 - ▶ La page peut-elle être mise en cache ?
 - ▶ La page existe ?
 - ▶ La page est-elle en accès lecture/écriture/exécution ?
 - ▶ La page est modifiée par rapport au disque (Dirty bit) ?
 - ▶ La page a-t-elle été accédée récemment ?



Remarque

- En réalité, plusieurs accès mémoire par accès demandé
 - ▶ Il faut un cache de traduction pour améliorer le **TLB** « Translation Lookaside Buffer »
 - ▶ Équilibre taille des tables/nombre d'indirections
- L'adresse d'une page n'utilise qu'une partie d'un mot mémoire (20 bits/32), les bits restant servent à stocker des informations :
 - ▶ La page peut-elle être mise en cache ?
 - ▶ La page existe ?
 - ▶ La page est-elle en accès lecture/écriture/exécution ?
 - ▶ La page est modifiée par rapport au disque (Dirty bit) ?
 - ▶ La page a-t-elle été accédée récemment ?
 - ▶ ...



Remarque

- En réalité, plusieurs accès mémoire par accès demandé
 - ▶ Il faut un cache de traduction pour améliorer le **TLB** « Translation Lookaside Buffer »
 - ▶ Équilibre taille des tables/nombre d'indirections
- L'adresse d'une page n'utilise qu'une partie d'un mot mémoire (20 bits/32), les bits restant servent à stocker des informations :
 - ▶ La page peut-elle être mise en cache ?
 - ▶ La page existe ?
 - ▶ La page est-elle en accès lecture/écriture/exécution ?
 - ▶ La page est modifiée par rapport au disque (Dirty bit) ?
 - ▶ La page a-t-elle été accédée récemment ?
 - ▶ ...
- Les pages ne peuvent pas toutes être gérées de cette manière



Remarque

- En réalité, plusieurs accès mémoire par accès demandé
 - ▶ Il faut un cache de traduction pour améliorer le **TLB** « Translation Lookaside Buffer »
 - ▶ Équilibre taille des tables/nombre d'indirections
- L'adresse d'une page n'utilise qu'une partie d'un mot mémoire (20 bits/32), les bits restant servent à stocker des informations :
 - ▶ La page peut-elle être mise en cache ?
 - ▶ La page existe ?
 - ▶ La page est-elle en accès lecture/écriture/exécution ?
 - ▶ La page est modifiée par rapport au disque (Dirty bit) ?
 - ▶ La page a-t'elle été accédée récemment ?
 - ▶ ...
- Les pages ne peuvent pas toutes être gérées de cette manière
 - ▶ besoin de pages contigue en mémoire (driver) ;



Remarque

- En réalité, plusieurs accès mémoire par accès demandé
 - ▶ Il faut un cache de traduction pour améliorer le **TLB** « Translation Lookaside Buffer »
 - ▶ Équilibre taille des tables/nombre d'indirections
- L'adresse d'une page n'utilise qu'une partie d'un mot mémoire (20 bits/32), les bits restant servent à stocker des informations :
 - ▶ La page peut-elle être mise en cache ?
 - ▶ La page existe ?
 - ▶ La page est-elle en accès lecture/écriture/exécution ?
 - ▶ La page est modifiée par rapport au disque (Dirty bit) ?
 - ▶ La page a-t-elle été accédée récemment ?
 - ▶ ...
- Les pages ne peuvent pas toutes être gérées de cette manière
 - ▶ besoin de pages contigue en mémoire (driver) ;
 - ▶ accès directe pour le système ;



Remarque

- En réalité, plusieurs accès mémoire par accès demandé
 - ▶ Il faut un cache de traduction pour améliorer le **TLB** « Translation Lookaside Buffer »
 - ▶ Équilibre taille des tables/nombre d'indirections
- L'adresse d'une page n'utilise qu'une partie d'un mot mémoire (20 bits/32), les bits restant servent à stocker des informations :
 - ▶ La page peut-elle être mise en cache ?
 - ▶ La page existe ?
 - ▶ La page est-elle en accès lecture/écriture/exécution ?
 - ▶ La page est modifiée par rapport au disque (Dirty bit) ?
 - ▶ La page a-t'elle été accédée récemment ?
 - ▶ ...
- Les pages ne peuvent pas toutes être gérées de cette manière
 - ▶ besoin de pages contigue en mémoire (driver) ;
 - ▶ accès directe pour le système ;
 - ▶ ...



Avantage de la pagination

- Les processus ont tous un même espace mémoire.



Avantage de la pagination

- Les processus ont tous un même espace mémoire.
- Permet la séparation des processus.



Avantage de la pagination

- Les processus ont tous un même espace mémoire.
- Permet la séparation des processus.
- Permet le partage de mémoire :



Avantage de la pagination

- Les processus ont tous un même espace mémoire.
- Permet la séparation des processus.
- Permet le partage de mémoire :
 - ▶ Entre processus pour créer un moyen de communication.



Avantage de la pagination

- Les processus ont tous un même espace mémoire.
- Permet la séparation des processus.
- Permet le partage de mémoire :
 - ▶ Entre processus pour créer un moyen de communication.
 - ▶ Partage de zones mémoires contenant le code des bibliothèques partagées.



Avantage de la pagination

- Les processus ont tous un même espace mémoire.
- Permet la séparation des processus.
- Permet le partage de mémoire :
 - ▶ Entre processus pour créer un moyen de communication.
 - ▶ Partage de zones mémoires contenant le code des bibliothèques partagées.
- Copie à la demande de la mémoire d'un processus lors d'un `fork`.



Avantage de la pagination

- Les processus ont tous un même espace mémoire.
- Permet la séparation des processus.
- Permet le partage de mémoire :
 - ▶ Entre processus pour créer un moyen de communication.
 - ▶ Partage de zones mémoires contenant le code des bibliothèques partagées.
- Copie à la demande de la mémoire d'un processus lors d'un `fork`.
 - ▶ Le processus fils recopie la table des pages du processus père en marquant chaque page en lecture seule.



Avantage de la pagination

- Les processus ont tous un même espace mémoire.
- Permet la séparation des processus.
- Permet le partage de mémoire :
 - ▶ Entre processus pour créer un moyen de communication.
 - ▶ Partage de zones mémoires contenant le code des bibliothèques partagées.
- Copie à la demande de la mémoire d'un processus lors d'un `fork`.
 - ▶ Le processus fils recopie la table des pages du processus père en marquant chaque page en lecture seule.
 - ▶ C'est l'écriture qui déclenche réellement la copie.



Avantage de la pagination

- Les processus ont tous un même espace mémoire.
- Permet la séparation des processus.
- Permet le partage de mémoire :
 - ▶ Entre processus pour créer un moyen de communication.
 - ▶ Partage de zones mémoires contenant le code des bibliothèques partagées.
- Copie à la demande de la mémoire d'un processus lors d'un `fork`.
 - ▶ Le processus fils recopie la table des pages du processus père en marquant chaque page en lecture seule.
 - ▶ C'est l'écriture qui déclenche réellement la copie.
- Initialisation à zéro sur le même principe.



Multiprogrammation et mémoire

- En général, les cadres libres sont gérés de manières globales pour tous les processus.
- Une processus qui demande plus de mémoire aura plus de pages qui lui sont affectées.
- Il existe cependant des limites d'utilisation (ulimit, cgroups, ...)
- Comment être sûr qu'un processus a suffisamment de mémoire ?

Quelle est la définition de « suffisamment de mémoire » ?



Ensemble de travail

Définition

Par le principe de localité, un processus sur une période courte utilise un sous ensemble limité de ses pages. C'est l'*ensemble de travail* (*working set*).

- Un processus qui accède à la mémoire va faire de nombreux défauts de page jusqu'à ce qu'il reconstitue son *espace de travail*.

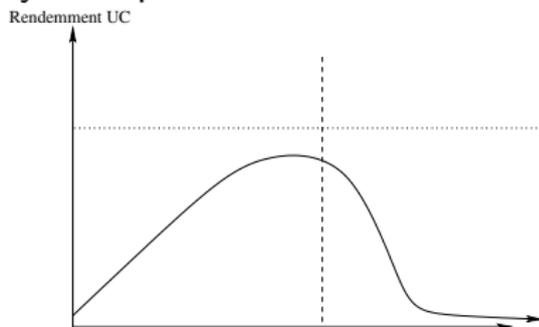


Ensemble de travail

Définition

Par le principe de localité, un processus sur une période courte utilise un sous ensemble limité de ses pages. C'est l'*ensemble de travail* (*working set*).

- Un processus qui accède à la mémoire va faire de nombreux défauts de page jusqu'à ce qu'il reconstitue son *espace de travail*.
- Si le nombre de pages des ensembles de travail des processus prêts est supérieur à la mémoire physique, le système génère de nombreux défauts de pages. Il y a risque d'**écroulement** du système



Notions définies par le système

Les notions importantes sont

- **Principe de localité** : le fait qu'une donnée accédée est souvent proche dans l'espace ou le temps d'une données déjà accédée. C'est un principe à considérer lors de la programmation.
- **Mémoire** : mémoire disponible pour les processus.
- **Espace d'adressage** : ensemble des adresses que peut utiliser un processus.
- **Segment** : partie de l'espace d'adressage effectivement utilisé par un processus et ayant un rôle (système, code, données, ...).
- **Droit sur les pages** : on attribue un rôle aux pages, si le processeur tente une utilisation en dehors du rôle, cela génère une exception traitée par le système.
- **Pages mémoire** : découpage de la mémoire en petits morceaux (les pages) qui sont affectées aux processus.
- **Swapping** : fait de déplacer une page de la mémoire au disque dur pour libérer de l'espace ou inversement pour donner des moyens de travail au processus.
- **Écroulement** : Lorsque le nombre de pages effectivement utilisées par l'ensemble des processus dépassent le nombre de cadres disponibles, le système va consacrer son temps à *swapper*.



Plan

1 Introduction

- Interface avec le matériel

2 Les processus

- État d'un processus
- Environnement
- Communication entre processus
- Les processus et python

3 Gestion de la mémoire

- Rôle de la mémoire
- Organisation de la mémoire

4 Fichiers et système de fichiers



Les fichiers

L'une des tâches importantes du système est le stockage de l'information. Les fichiers permettent ce stockage.

- Ils ont une structure interne en relation avec le logiciel qui doit les utiliser.
- Ils ont souvent un type qui permet au système de savoir ce qu'il doit faire du fichier :
 - ▶ lancer un logiciel précis pour le lire ;
 - ▶ exécuter le fichier comme une commande ;
 - ▶ utiliser le fichier comme une bibliothèque de fonction ;
 - ▶ ...
- Ils sont classés dans une structure arborescente de répertoires.
- Un fichier est découpé en blocs de données qui sont les éléments que l'on peut lire ou écrire sur le disque.
- Le « disque » qui contient les fichiers est lui même organisé de manière à pouvoir retrouver le contenu du fichier à partir d'une structure de description (par exemple l'inode)



Type de fichier Unix

Traditionnellement on distingue 3 grands types de fichiers. Ils classe les fichiers selon la façon dont on peut les lire :

- Les *fichiers réguliers* ou *normaux* ceux qui contiennent des données, lisibles par blocks.
- Les *catalogues* ou *répertoires*, les fichiers qui contiennent une liste de fichiers et servent à stocker la structure arborescente.
- Les *fichiers spéciaux*, ce sont des fichiers permettant d'accéder aux données gérées par le système sous la forme d'un fichier. Ils correspondent aux périphériques, socket, tubes. . .

Sous unix « tout est fichier », le système permet donc la lecture ou la modifications des objets du système via les appels systèmes de lecture et écriture dans les fichiers (`open`, `read`, . . .). Les droits sont aussi gérés de la même manière.

Bien sur, le terme *fichiers réguliers* regroupent un grand nombre de choses différentes.



Typage des fichiers réguliers

Les *fichiers réguliers* ont différentes utilisations (exécutable, fichier texte, image, ...); le rôle du système est de savoir quoi faire lorsque l'utilisateur demande une action sur le fichier.

Il y a 2 écoles :

- *Typage fort* : le fichier a un type défini par son nom (*extension*). C'est le cas par exemple sous DOS ou Windows.
- *Typage déduit* : le type du fichier dépend de son contenu ou de ses propriétés. C'est le cas sous Unix/Linux.
- *Type MIME* ou *Content-Type* : typage des données sur internet.
 - ▶ Le nom (donc l'extension) disparaît lors d'un envoi des données sur le réseau.
 - ▶ Les pages web ou les emails (pièces jointes) utilisent le type MIME écrit dans un entête.
 - ▶ Le navigateur ou le logiciel de lecture choisit le logiciel à appeler.



Typage fort

Tous les fichiers ont une extension qui peut être cachée mais est forcément présente.

- Un fichier exécutable doit se terminer par `.exe`, `.com` ou `.bin`.
- un script doit se terminer par `.bat`
- Pour le reste, le système reconnaît le logiciel permettant d'ouvrir le fichier en fonction de l'extension (tout est dans la base de registre `HKEY_CLASSES_ROOT`).



Typage déduit

Ce qu'il faut faire du fichier dépend de propriétés ou de sont contenu :

- Un fichier est présumé exécutable s'il a le droit d'exécution.
- On utilise souvent un code ou des directives placées en début de fichier :
 - ▶ le *nombre magique* au début du fichier **voir ici** ;
 - ▶ la première ligne des scripts comme `#!/usr/bin/env python3`
- Voir la commande `file` sous Unix.

En réalité, pour des raisons de compatibilités entre systèmes et pour éviter les problèmes, le typage est souvent mixte et la façon dont il est géré évolue souvent.



Certains fichiers sont en fait compatibles avec différents logiciels :

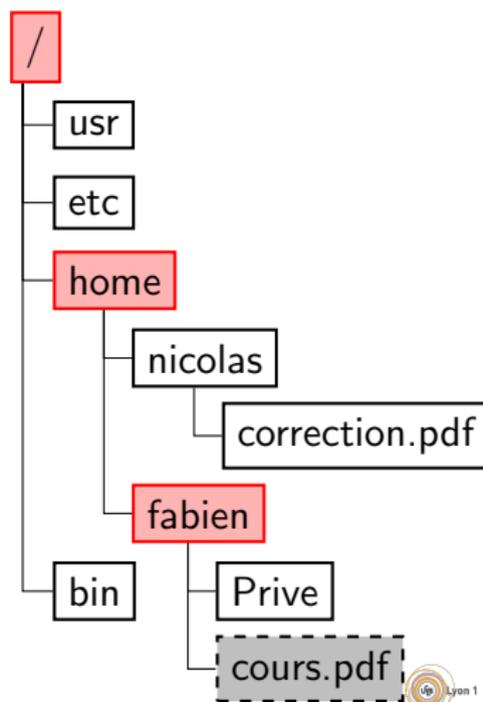
- un script `python` peut être à la fois un exécutable et un fichier de code ;
- un fichier `docx` ou `odt` est une archive ZIP contenant des fichiers de descriptions `xml` et les documents intégrés ;
- un fichier `jar` (exécutable java) est une archive contenant les fichiers `.class` et une description pour le lancement ;
- ...



L'organisation des répertoire

Les répertoires sont des fichiers spéciaux qui contiennent des fichiers ou d'autres répertoires

- Un répertoire ne fait partie que d'un autre répertoire (structure d'arbre).
- Chaque répertoire contient au moins deux autres : `.` lui même et `..` son parent.
- Tout part de la racine :
 - ▶ Sous windows, il y a une racine par partition : `C:`, `D:`, ...
 - ▶ Sous unix, il y a une racine unique : `/`.
- Pour accéder à un fichier on a besoin d'un chemin :
 - ▶ Chemin absolu `C:\User\fabien\cours.pdf`
 - ▶ Chemin relatif (par rapport au répertoire courant du processus qui utilise le nom) : `../nicolas/correction.pdf` pour un processus se trouvant dans `/home/fabien/`



Les fichiers sur le disque

Un disque n'est pas lu octet par octet mais par morceaux qu'on appelle *bloc*. Les fichiers sont alors découpés en blocs et écrits sur le disque.

Définition (Système de fichiers)

Le *système de fichiers* est la structure de données qui définit la façon de retrouver tous les blocs d'un fichier.

- Par exemple NTFS, EXT, FAT, APFS
- Chaque *partition* peut être formatée avec un système différent
- Les systèmes ont des propriétés différentes :
 - ▶ taille maximum des fichiers ;
 - ▶ vitesse d'accès selon les modèles d'utilisation ;
 - ▶ possibilité d'opération en ligne
 - ▶ ...



Processus et fichiers

- Les processus peuvent utiliser les fichiers :
 - ▶ les ouvrir/créer (`open`), lire ou écrire (`write`, `read`), les fermer `close` ;
 - ▶ explorer le système de fichiers avec les bibliothèques (`os` et `os.path`) ;
 - ▶ stocker des objets directement via des formats comme JSON.
- Ils sont toujours exécutés dans un *répertoire de travail* (parfois difficile à retrouver si on utilise un IDE).
- Ils ont toujours 3 fichiers spéciaux ouverts :
 - ▶ l'entrée standard `sys.stdin` ;
 - ▶ la sortie standard `sys.stdout` ;
 - ▶ la sortie d'erreur `sys.stderr`.



Notion de sérialisation

Le stockage de données dans un fichier pose souvent la question de sérialisation. C'est à dire « Comment représenter dans un ensemble d'octets qui se suivent une structure de données présente en mémoire avec son organisation ? ».

Plusieurs possibilités sont utilisées pour résoudre le problème :

- méthode *ad-hoc* définie par le programmeur
- méthodes de sérialisation adaptées au langage (`serialize` en php, `pickle` en python)
 - + capable de sérialiser facilement la plupart des objets ;
 - dépendant du langage voir plus.
- méthodes de sérialisation généralistes :
 - ▶ XML : +très expressive, +lecture de flux, -gros surcout de données
 - ▶ json, yaml : -juste l'essentiel (tableau, listes type primitifs), -il faut parser la totalité du fichier, +très compact.

Comment choisir ?



Résumé des notions sur les fichiers

- *Les fichiers* : ensemble de données que l'on peut lire et modifier de manière séquentielle ; associés à des flux dans les langage de prog.
- *Les répertoires* : fichiers spéciaux qui contiennent les noms et points d'accès à d'autres fichiers.
- *Les partitions* : disque ou partie de disque organisée de manière à pouvoir déposer les fichiers.
- *Le système de fichiers* : organisation des données sur le disque définie par le *formatage* de la partition. Il en existe plusieurs types en fonction de l'os ou des besoins des programmes.
- *Espace de nommage* : façons de représenter les fichiers sous une forme d'arbre (répertoire, sous repertoire...) afin de les retrouver.
- *Type de fichiers* : description du contenu qui permet d'associer à chaque fichier un programme ou une action.
- *Droits sur les fichiers* : liste d'actions autorisées ou refusées à un utilisateur ou groupe d'utilisateurs.

