DIU - Système d'exploitation

Arthur Chevalier, Mathieu Faverge et Brice Goglin

2019

Plan

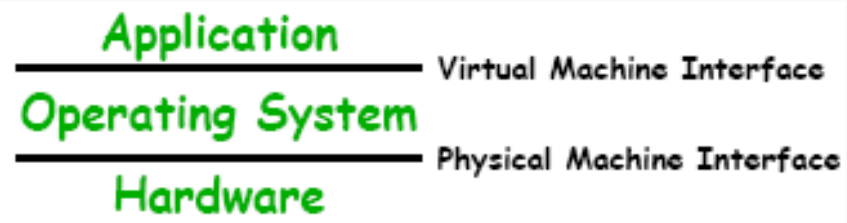
Introduction

Qu’est ce qu’un système d’exploitation?

Qu’est ce qu’un système d’exploitation?

Un système d’exploitation :

- gère et cache la complexité du matériel au programmeur
- fournit une interface virtuelle de machine
- fournit une protection entre utilisateurs (souvent)



Où sont les systèmes d'exploitation?

Où sont les systèmes d'exploitation?

Dans les centres de données (datacenter) : banques, assurances, . . .

Où sont les systèmes d'exploitation?

Dans les systèmes embarqués : internet box, équipements multimédia

Où sont les systèmes d'exploitation?

Dans les équipements mobiles : téléphones android, symbian, dans les kits mains libres, . . .

Où sont les systèmes d'exploitation?

Avec le cloud computing $\rightarrow$ plus de localisation

- Le programme Google : navigateur (Chrome OS)
- L'ordinateur Google : des milliers d'ordinateurs google dans les centres de données et de calcul répartis dans le monde
- Services : système de fichier (web), mail/chat (communication réseau), ...

Cacher la complexité du matériel

Les machines sont très différentes

- Type et nombre de processeurs/cœurs
- Quantité de mémoire
- Type de disque dur ou carte réseau

On veut qu'un même programme fonctionne partout sans devoir être modifié

- Peu importe la quantité de mémoire disponible, je peux lancer mon programme
- J'ai une carte réseau, peu importe ce qu'elle sait faire, je l'utilise de la même façon
- Peu importe si mon disque dur est un SSD ou un NAS connecté par le réseau, j'utilise mes fichiers de la même façon

Qu’est ce qu’un système d’exploitation?

Services proposés par l’OS

- Gestionnaire de ressources matérielles
  - cache la complexité du matériel
  - arbitre les requêtes d’accès aux ressources, évite les conflits, permet un usage équitable
  - gère les erreurs
  - empêche les usages impropres de la machine
- Facilite l’utilisation de la machine
  - ensemble de bibliothèques standard (fenêtrage par ex.)
  - rend la programmation plus facile, plus simple

Qu’est ce qu’un système d’exploitation?

Quelles ressources, quels services?

- Temps CPU :
  - Ordonnancement des tâches sur le processeur (scheduling)
    - Une seule tâche par processeur/cœur à la fois!
- Mémoire :
  - Allocation
  - Protection mémoire
- Entrées/sorties :
  - Système de fichiers et droits,
  - Gestion des diverses cartes et protocoles réseau,
  - Fenêtrage
- Consommation d’énergie : réglage fréquence, . . .
- Utilisateurs :
  - Avec des droits différents

Qu’est ce qu’un système d’exploitation?

Structurer les donnés

- Multiplexer le réseau
  - Un seul flux de bits qui contient des informations de différents programmes vers différents serveurs
  - Comme un tuyau qui contiendrait plusieurs produits qu’on peut séparer à la sortie
- Organiser le disque dur
  - Un immense tableau de blocs où on veut stocker une arborescence de dossiers et fichiers
  - Comme un grande feuille de papier sur laquelle on écrirait à plusieurs en même temps

Où sont les systèmes d'exploitation?

Certains systèmes critiques n'utilisent pas d'OS

- Systèmes qui n'exécutent qu'un seul programme
  - et ce programme n'a pas besoin de s'exécuter ailleurs
- Systèmes temps réels :
  - OS spécifique, juste pour un ensemble de tâches et d'événements (centrale nucléaire)
  - pas d'OS (Airbus A320-380)
- Langages de programmation, compilateurs spécifiques temps réel
  - essayer de garantir certaines propriétés (temps de réaction, pas de crash, . . .)
  - Lustre, Esterel, Oasis, Signal

Un peu d'histoire

Années 50-60

- Un seul programme à la fois, sur un seul processeur

Années 70

- Un autre programme peut s'exécuter quand le premier est bloqué
  - attente de la fin d'une lecture sur le disque dur

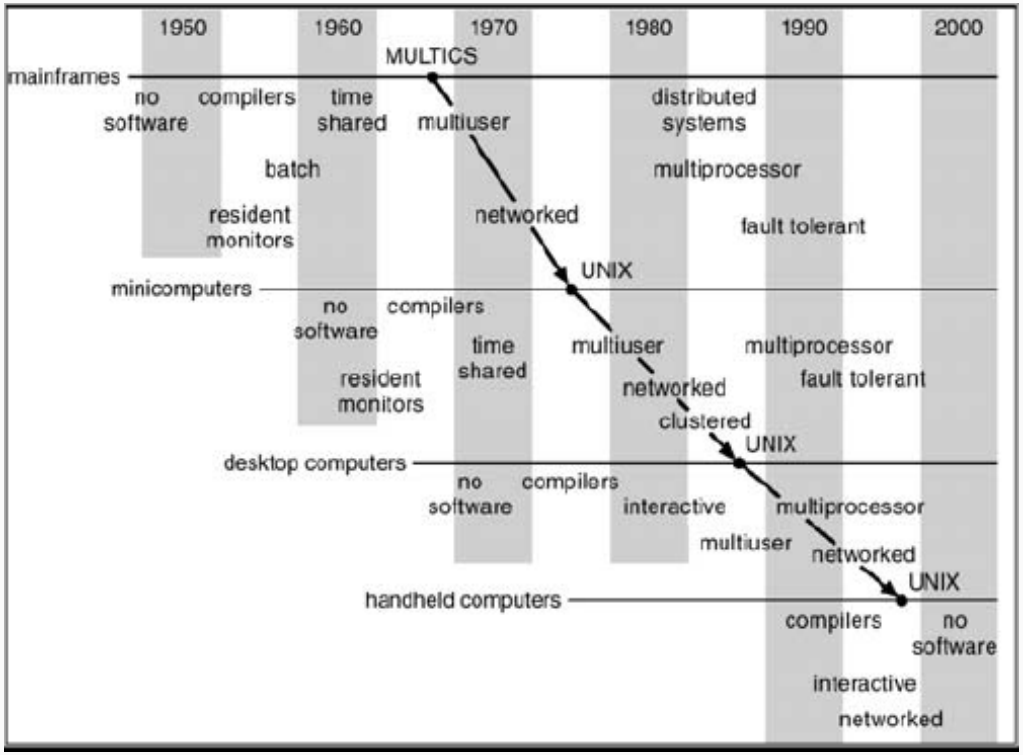
Années 80

- Interactivité
  - Les programmes réagissent aux actions de l'utilisateur
  - L'utilisateur peut alterner entre plusieurs programmes

Années 90

- Plusieurs processeurs puis plusieurs cœurs par processeur
  - Plusieurs programmes s'exécutent en même temps

Structure



Complexité $\rightarrow$ Besoin d'OS

Structure

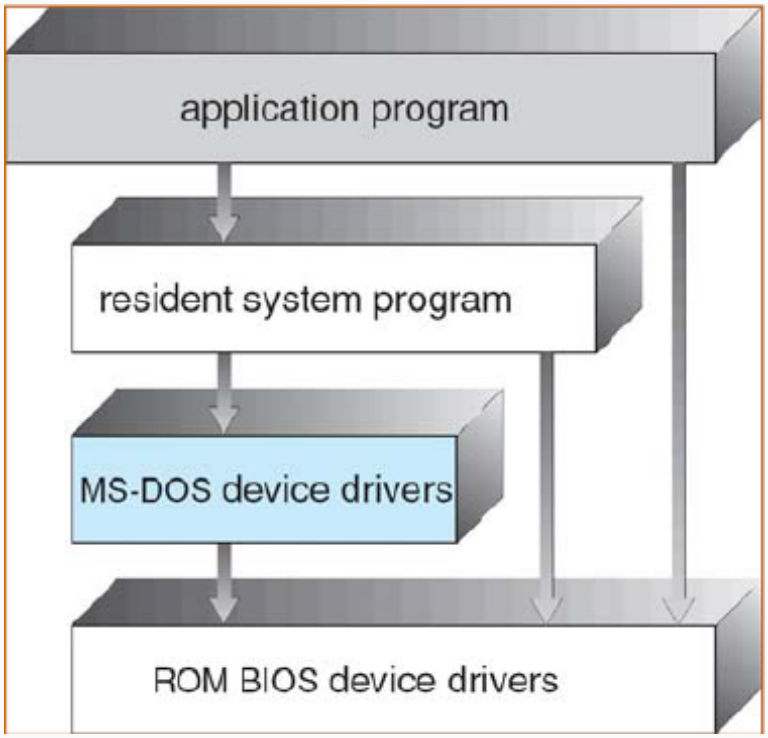
Au début, un OS=quelques bibliothèques

Hypothèses simplificatrices :

- pas de mauvais programmeurs/utilisateurs
- un programme à la fois

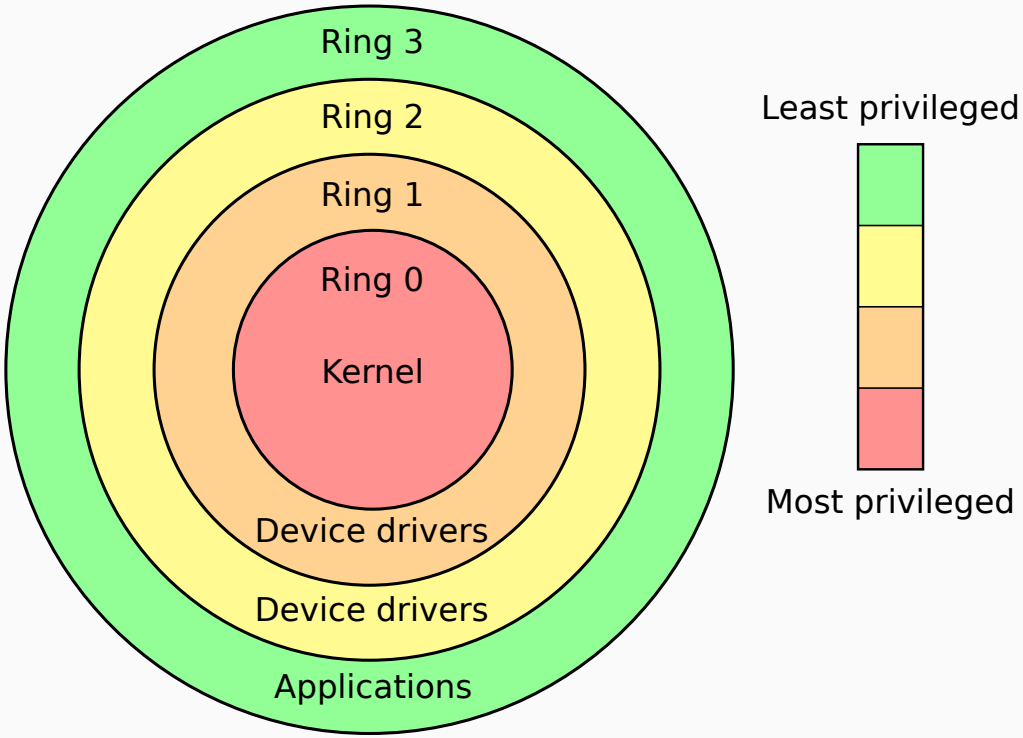
⇒ Mauvaise utilisation :

- du temps machine
- du temps de l'utilisateur



Un peu d'histoire

Un structure en oignon



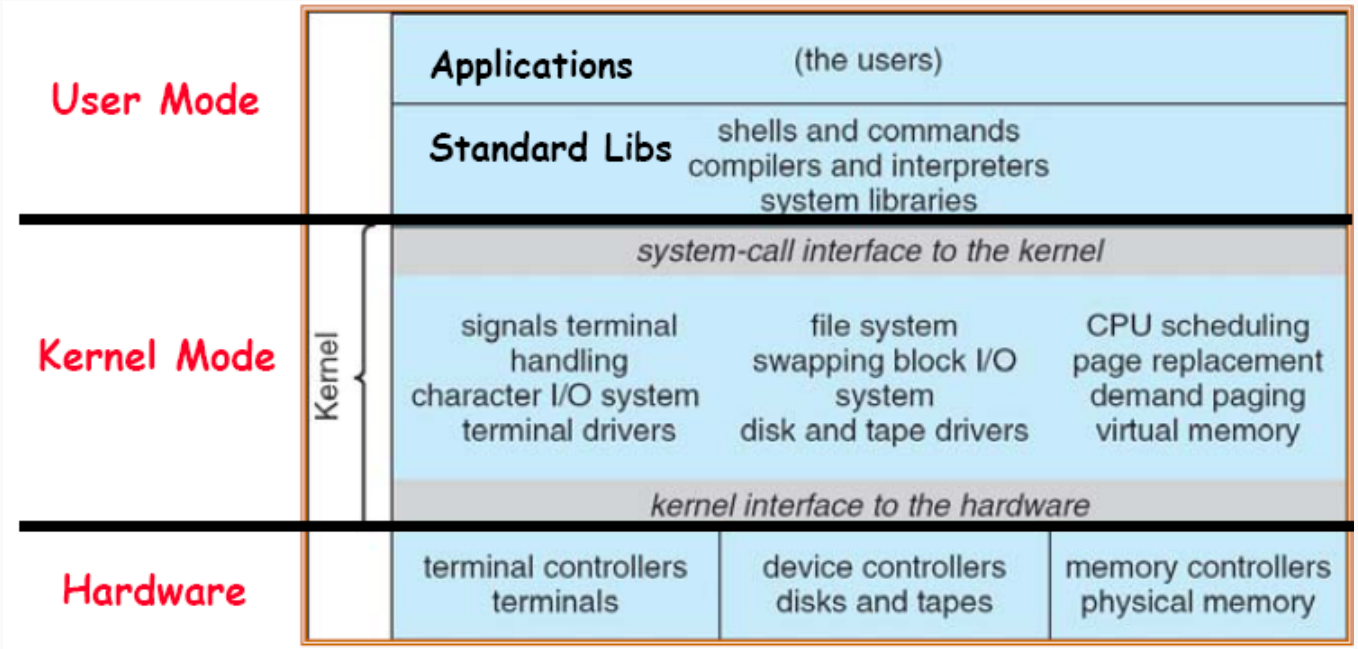
https://blog.xmco.fr/info-minix-un-systeme-dexploitation-cache-dans-vos-processeurs-intel/

Structure

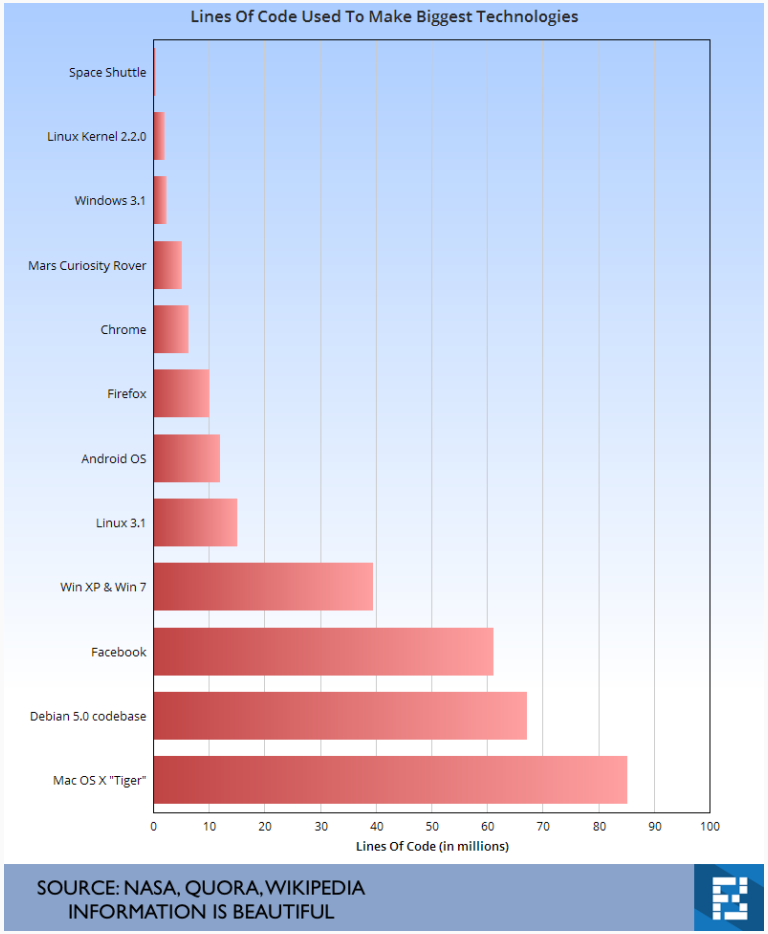
Différents composants logiciels

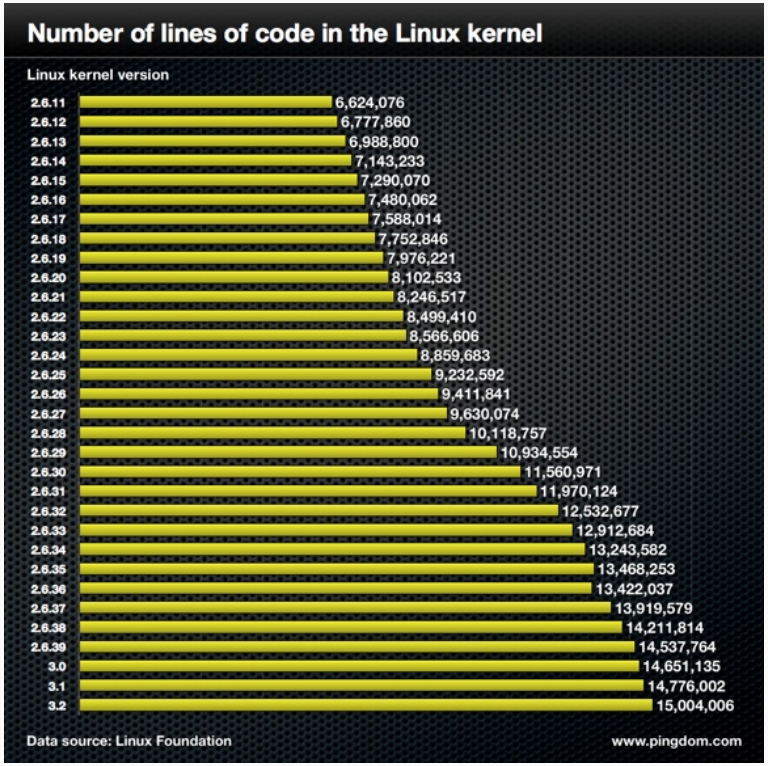
- Un noyau qui s'exécute un mode privilégié et peut tout faire
  - Modifier la mémoire, parler aux périphériques, etc.
  - Ce n'est pas le super-utilisateur/administrateur/root
    - Il a juste un peu plus de droits que les autres
- Des programmes qui s'exécutent en mode non-privilégié
  - Ils peuvent uniquement utiliser/modifier leur mémoire
    - Sauf s'ils font un appel système (voir plus loin)
  - Par exemple, des outils en ligne de commande (cp, mv, ...)
  - Plein d'autres programmes!
    - Par exemple un compilateur est juste un programme qui lit un fichier source et écrit un fichier binaire

Structure d'un OS moderne



Un logiciel très complexe



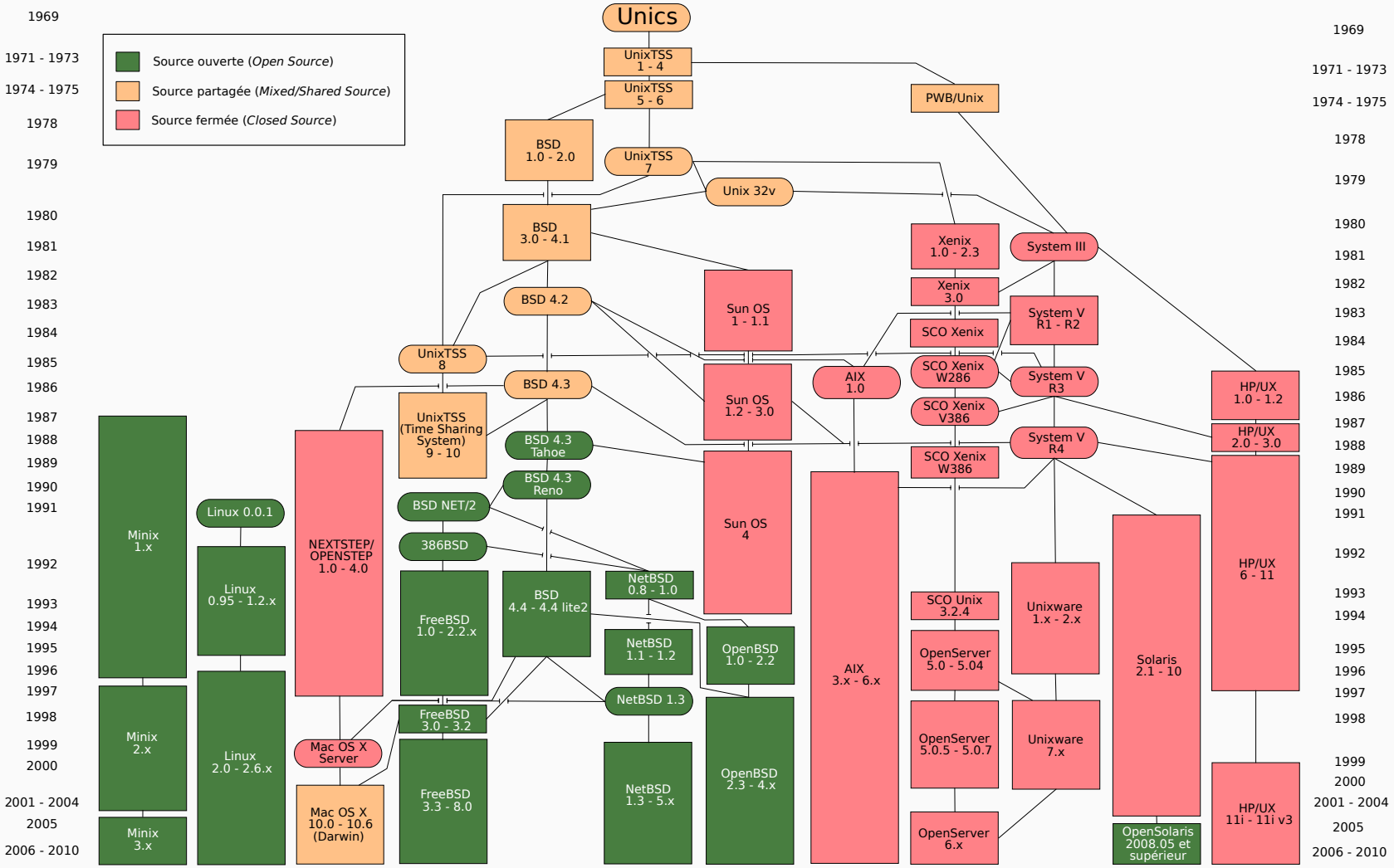


- supporter de nombreuses plates-formes différentes
- efficacement, pour satisfaire les utilisateurs

Exemples

- Linux, sur de nombreuses architectures
- De nombreux autres Unix (Solaris, BSD, etc)
- MacOS X qui dérive de BSD
- Android qui dérive de Linux
- Windows
- GNU/Hurd, basé sur un micro-noyau, avec une organisation assez différente
- De nombreux autres OS spécifiques (Plan9, Symbian, VxWorks, etc)

Généalogie


1969
1971 - 1973
1974 - 1975
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001 - 2004
2005
2006 - 2010
Source ouverte (Open Source)
Source partagée (Mixed/Shared Source)
Source fermée (Closed Source)
Unics
UnixTSS 1 - 4
UnixTSS 5 - 6
PWB/Unix
BSD 1.0 - 2.0
UnixTSS 7
Unix 32v
BSD 3.0 - 4.1
Xenix 1.0 - 2.3
System III
Xenix 3.0
BSD 4.2
Sun OS 1 - 1.1
System V R1 - R2
SCO Xenix
UnixTSS 8
AIX 1.0
SCO Xenix W286
System V R3
HP/UX 1.0 - 1.2
BSD 4.3
Sun OS 1.2 - 3.0
SCO Xenix V386
HP/UX 2.0 - 3.0
UnixTSS (Time Sharing System) 9 - 10
BSD 4.3 Tahoe
SCO Xenix W386
System V R4
BSD 4.3 Reno
BSD NET/2
Sun OS 4
Minix 1.x
Linux 0.0.1
NEXTSTEP/ OPENSTEP 1.0 - 4.0
386BSD
AIX 3.x - 6.x
HP/UX 6 - 11
Linux 0.95 - 1.2.x
FreeBSD 1.0 - 2.2.x
BSD 4.4 - 4.4 lite2
NetBSD 0.8 - 1.0
SCO Unix 3.2.4
Unixware 1.x - 2.x
Solaris 2.1 - 10
NetBSD 1.1 - 1.2
OpenBSD 1.0 - 2.2
OpenServer 5.0 - 5.04
Minix 2.x
Linux 2.0 - 2.6.x
FreeBSD 3.0 - 3.2
NetBSD 1.3
OpenServer 5.0.5 - 5.0.7
Unixware 7.x
Mac OS X Server
FreeBSD 3.3 - 8.0
NetBSD 1.3 - 5.x
OpenBSD 2.3 - 4.x
HP/UX 11i - 11i v3
Minix 3.x
Mac OS X 10.0 - 10.6 (Darwin)
OpenServer 6.x
OpenSolaris 2008.05 et supérieur

Références

- Livre
  - Operating System Concepts, Silbershatz, Galvin, Gagne (8^{eme} édition)
- Cours en ligne
  - J. Kubiatowicz, CS 162, Berkeley University
  - D. Maziere, CS 140, Stanford University
  - A. Cohen, INF570, école Polytechnique
- Autres ressources en ligne
  - `http://www.top500.org`
  - `http://www.wikipedia.org`
  - `http://www.linuxmanpages.com` ou `http://linux.die.net`
  - `http://www.osdata.com`

Interfaces, shell et système de fichiers

Comment utiliser un système d'exploitation ?

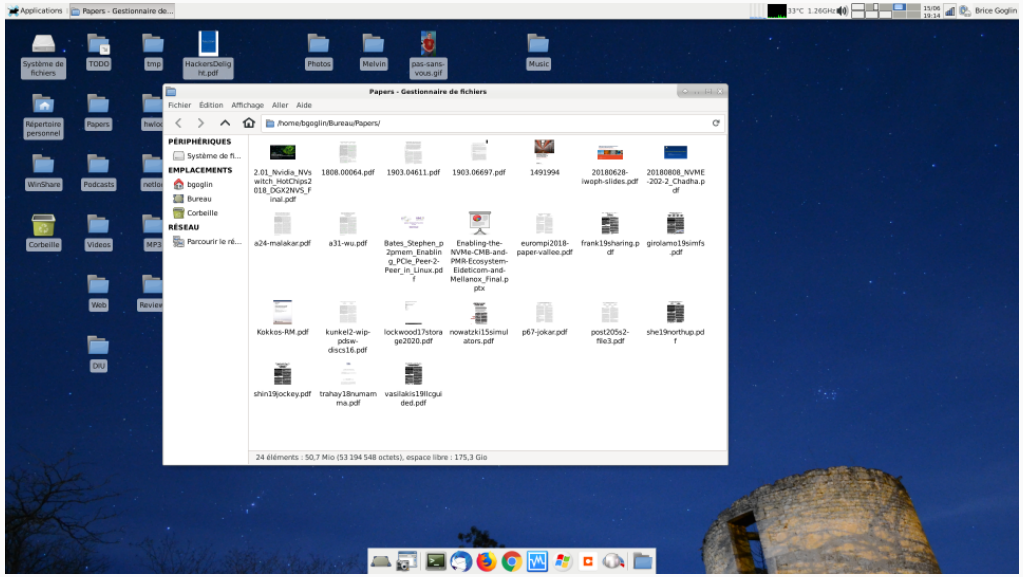
En théorie, on utilise les appels-système

- Mais ce sont des instructions très particulières (voir plus loin)
- Il faudrait écrire un programme et le compiler pour chaque opération
- Pas du tout pratique pour lancer les programmes, ouvrir des fichiers, les modifier, etc.

En pratique, on utilise des interfaces utilisateur

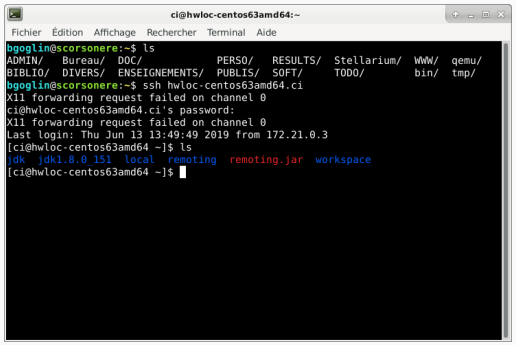
- Plusieurs existent, avec leurs avantages et inconvénients

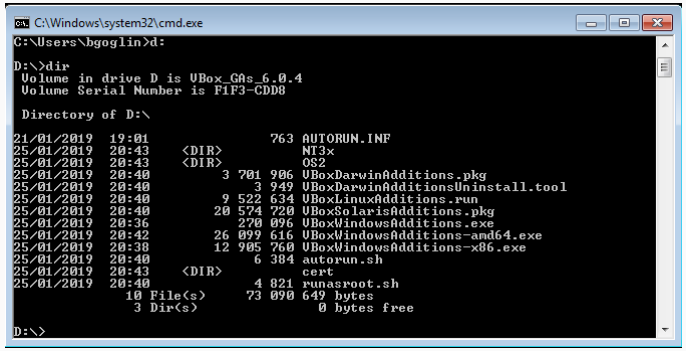
Interfaces graphiques



- Très pratique pour les opérations simples
- Pas adapté aux opérations nombreuses ou complexes
  - Renommer tous les fichiers dont le nom contient `toto` dans un dossier ou ses sous-dossiers

Ligne de commande (terminal/shell, cmd, etc.)





- On utilise des programmes dédiés à certaines opérations de base
  - Déplacer des fichiers, renommer un fichier, changer de dossier, changer des droits, etc.
- Pas très pratique au premier abord
- Extrêmement puissant après un peu de pratique
  - Utilisation de *Joker* pour sélectionner certains fichiers

Exemples en shell Unix (sh)

Entrer dans le sous-dossier `divers` du sous-dossier `autres` :

```
$ cd autres/divers
```

Savoir dans quel dossier on se trouve :

```
$ pwd
/home/login/autres
```

Lister le contenu du dossier courant :

```
$ ls
dossier1 dossier2 fichier.txt
```

Lister le dossier courant avec les détails :

```
$ ls -la
```

Exemples en shell Unix (sh)

Voir le contenu d'un fichier texte :

```
$ cat monfichier
```

Ouvrir un fichier dans un éditeur :

```
$ emacs monfichier
```

Renommer le fichier `toto` en `tata` :

```
$ mv toto tata
```

Le caractère spécial `*` permet de sélectionner un ensemble de fichiers dont le nom correspond à un schéma. Le shell remplace les noms contenant ce caractère par tous les fichiers correspondants puis appelle le programme demandé :

```
$ cp dossier*/toto*titi autredossier/
```

Arborescence des fichiers

- Fichiers sont organisés dans des dossiers, sous-dossiers, etc.
- Chemin de fichier composé des noms de ses dossiers parents, séparés par `/` sous Unix
  - S'il commence par `/`, il fait référence à la racine de l'arborescence
  - Sinon c'est un chemin relatif au dossier courant
- `..` permet de référencer le dossier parent
- Sous Windows
  - Chemin commence par un identifiant de disque (`D:\`)
  - Noms des dossiers séparés par `\`

Droits des fichiers

Toute opération sur un fichier est soumise à un ensemble de droits d'accès qui sont de 3 types.

r : droit de lecture

- Si répertoire, consultation de ses entrées (c.-à-.d, `ls` autorisé)
- Sinon, consultation du contenu du fichier

w : droit d'écriture

- Si répertoire, droit de création, de renommage et de suppression d'une entrée dans le répertoire
- Sinon, droit de modification du contenu du fichier

x :

- si répertoire, droit de traverser (c.-à-.d., `cd` autorisé)
- sinon, droit d'exécution

Droits des fichiers

- 3 catégories d'utilisateurs :
  - Propriétaire (`u`)
  - Groupe propriétaire (`g`)
  - Tous les autres (`o`)
- Chaque catégorie possède ses types d'accès `r w x`

Droits des fichiers - consultation

- `ls -ld` → donne les droits des fichiers
- Format de sortie de `ls -l`
- `- --- --- ---`
- `-` → Type du fichier
  - `-`: fichier régulier
  - `d`: répertoire
  - `l`: lien symbolique
- `---` → Droits du propriétaire (rwx)
- `---` → Droits du groupe (rwx)
- `---` → Droits des autres (rwx)

Droits des fichiers - modification

- Modification des droits sur un fichier/répertoire existant
  - `chmod droit fichier`
- Droits à appliquer:
  - Catégorie: u (=user), g (=group), o(=other), ou a (=all)
  - Opérations: + (ajout), - (retrait), = (affectation)
  - Example: u+rx,o=r
- Alternativement, on peut aussi les donner en octal: 1 (exécution), 2 (lecture) et 4 (écriture)
  Exemple: 641 pour rw-r—-x, 752 pour rwxr-x-w-

Processus et gestion mémoire

Contexte

Plusieurs dizaines de processus s'exécutent simultanément sur une machine qui possède un nombre limité de ressources pour les exécuter.

Objectifs

- Savoir observer les processus s'exécutant sur une machine
- Comprendre comment les processus se partagent les ressources (ordonnancement, virtualisation)
- Comprendre comment les processus sont créés, leur état (ordonnancement, arborescence de processeurs)

Notion de processus

- Un processus est associé à un contexte qui lui est propre
  - Il a une vision de la mémoire qui lui est propre
  - Il ne peut pas voir la mémoire des autres processus
- Il est isolé sur la machine et a l'impression d'être seul
  - Les autres processus sont cachés, de même que le système.

Notion de processus

Processus
Un programme en cours d'exécution ainsi que son contexte (mémoire, fichiers ouverts, où il en est dans son code, ...)

Caractéristiques statiques

- *PID: Process Identifier*, identifiant unique d'un processus
- *PPID: Parent Process Identifier*, identifiant de l'unique du parent dans l'arborescence des processus
- Utilisateur propriétaire
- Droits d'accès aux ressources (fichiers, . . .)

Caractéristiques dynamique

- Priorité, état des registres, . . .
- Données statistiques : temps CPU consommé, temps utilisateur, temps système, . . .
- Liste des fichiers ouverts

Modes d'exécution (processus/processeur)

Un processus peut s'exécuter au travers de deux modes d'exécution du processeur.

Mode utilisateur (non-privilégié)

- Par défaut le processeur exécute le code en mode non privilégié
- Les programmes peuvent juste faire des additions, lectures, écritures, . . . dans leur mémoire
- Quand ils exécutent ces instructions basiques, le processeur ne touche que les données privées du processus (cf mémoire virtuelle plus loin)

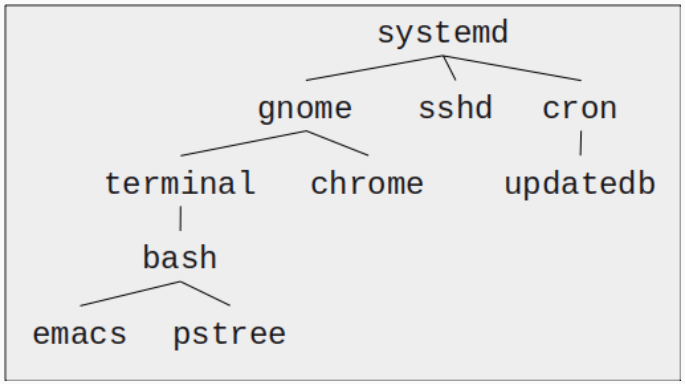
Modes d'exécution (processus/processeur)

Un processus peut s'exécuter au travers de deux modes d'exécution du processeur.

Mode noyau (privilégié)

- Si un processus veut faire autre chose que les opérations précédentes, il doit demander l'autorisation à l'OS par une instruction spéciale : un appel système (*syscall*)
- Le matériel passe en mode privilégié et s'assure via l'OS que l'opération demandée par le processus est possible :
  - ouvrir un fichier, le modifier, le renommer, allouer de la mémoire, créer une tâche, terminer une tâche, ...
  - tout ce qui est potentiellement sensible et qui pourrait nuire aux autres tâches et utilisateurs

Arborescence de processus



- Chaque processus possède un processus parent
- Sauf le premier processus (systemd ou init, PID=1)

- Deux types de processus :
  - Processus utilisateurs (attachés à un terminal)
  - Daemons : processus qui assurent un service (détachés de tout terminal)

Observer les processus

```
$ ps -l
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000 22995  1403  0  80   0 -  6285 -      pts/1    00:00:00 bash
0 S  1000 29526 22995  0  80   0 - 128631 -     pts/1    00:00:05 emacs
0 S  1000 29826 22995  0  80   0 - 51571 -      pts/1    00:00:00
oosplash
0 S  1000 29843 29826  1  80   0 - 275029 -     pts/1    00:00:48
soffice.bin
0 R  1000 30323 22995  0  80   0 -  2790 -      pts/1    00:00:00 ps
```

- `ps` permet d'afficher la liste des processus s'exécutant à un instant *t*.

Observer les processus

```
$ pstree -pA
systemd(1)-+-ModemManager(535)-+-{gdbus}(675)
           |                   `-{gmain}(580)
           |-NetworkManager(552)-+-dhclient(27331)
           |                     |-{NetworkManager}(673)
           |                     |-{gdbus}(756)
           |                     `-{gmain}(733)
           |-acpid(692)
           |-konsole(1403)-+-bash(22995)-+-emacs(29526)-+-{dconf worker}(29529)
                           |             |              |-{gdbus}(29528)
                           |             |              `-{gmain}(29527)
                           |             `-pstree(30412)
                           `-{QProcessManager}(1411)
```

- `pstree` permet d'afficher l'arborescence des processus s'exécutant à un instant *t*.

Observer les processus

```
$ top
top - 15:52:18 up 5 days,  2:04,  3 users,  load average: 0,19, 0,12, 0,13
Tasks: 176 total,   1 running, 175 sleeping,   0 stopped,   0 zombie
%Cpu(s):  6,0 us,  1,3 sy,  0,1 ni, 92,5 id,  0,1 wa,  0,0 hi,  0,0 si,  0,0 st
KiB Mem:   8099392 total,  5840956 used,  2258436 free,   494524 buffers
KiB Swap: 10157052 total,        0 used, 10157052 free.  3114404 cached Mem
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
  866 root      20   0  731892 377196 346672 S   6,4  4,7  21:01.97 Xorg
 1375 trahay     9 -11  651480  11108   8052 S   6,4  0,1  23:23.48 pulseaudio
    1 root      20   0  176840   5420   3144 S   0,0  0,1   0:02.57 systemd
    2 root      20   0       0      0      0 S   0,0  0,0   0:00.01 kthreadd
    3 root      20   0       0      0      0 S   0,0  0,0   0:04.34 ksoftirqd/0
    5 root       0 -20       0      0      0 S   0,0  0,0   0:00.00 kworker/0:0H
    7 root      20   0       0      0      0 S   0,0  0,0   0:30.37 rcu_sched
```

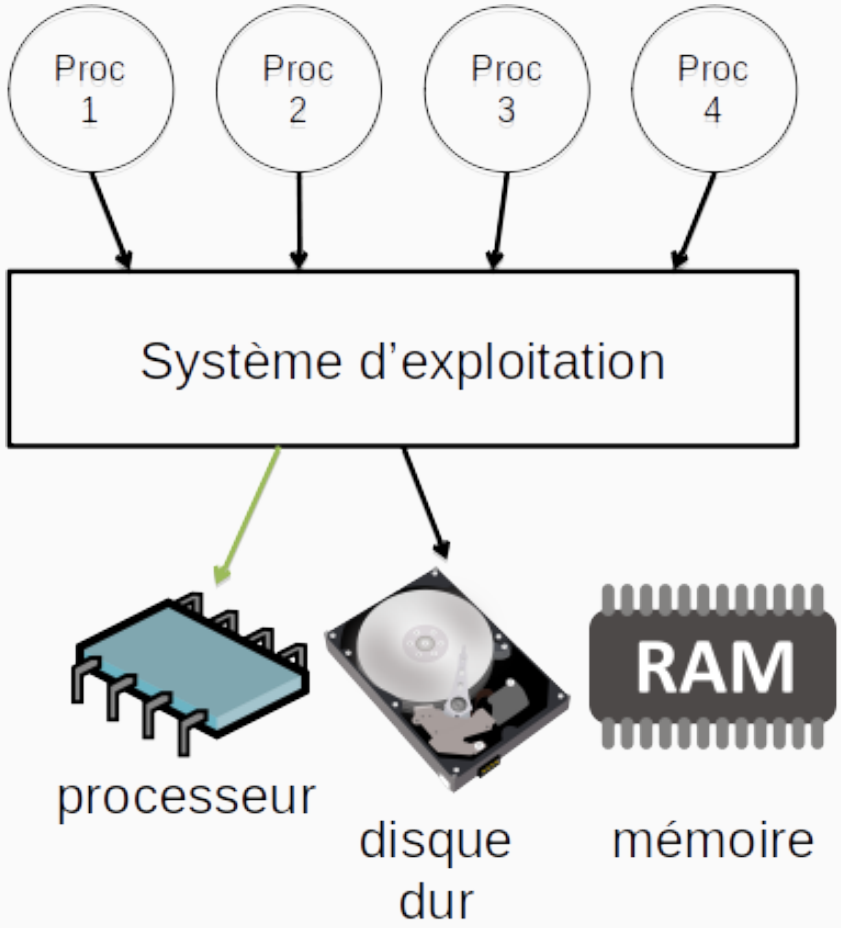
- `top` permet d'afficher dynamiquement les processus avec leur occupation des ressources.

Processus et gestion mémoire

Ordonnancement

Comment partager les ressources?

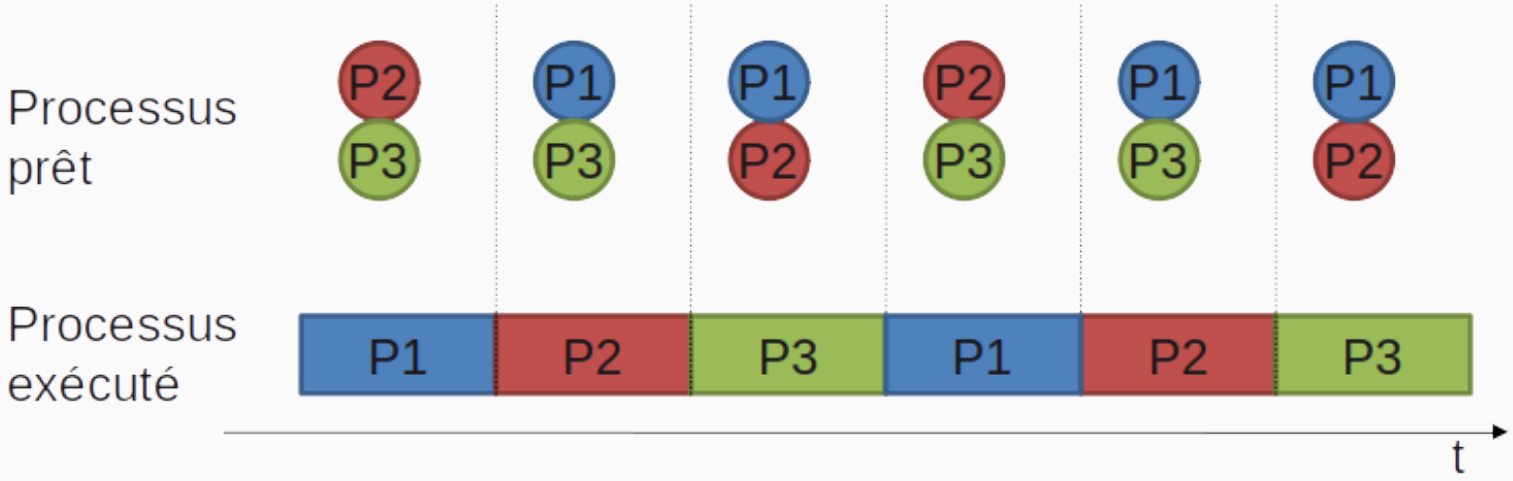
- Un coeur/processeur ne peut exécuter qu'une tâche à la fois
- Le système doit partager les coeurs disponibles entre les tâches prêtes
- Il faut éviter de consommer des ressources pour des tâches en attente



Voir dans top/ps

Partage des ressources CPUs

- À un instant donné, 1 seule tâche s'exécute par cœur
  - Les autres processus attendent
- Besoin d'ordonnancer les processus pour partager les ressources



Partage des ressources CPUs

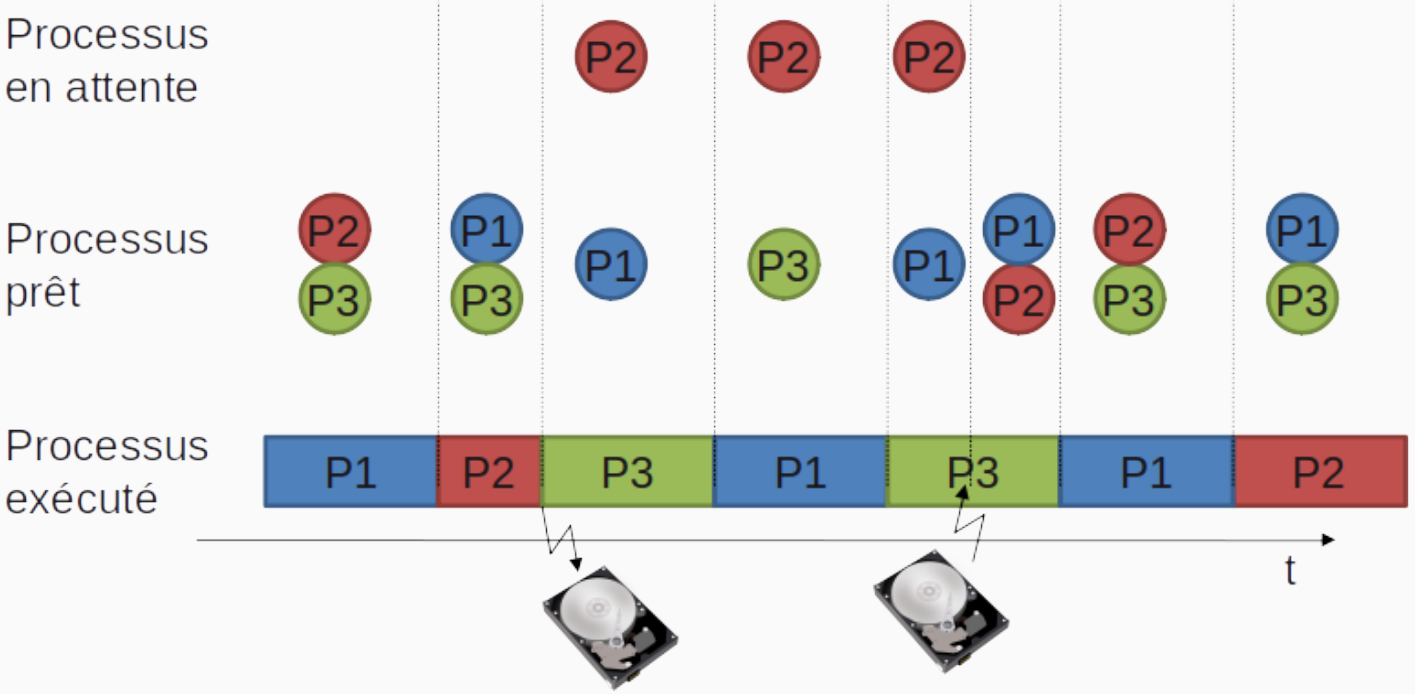
- L'idée est d'alterner entre les processus avec une faible granularité temporelle
  - Cela suffit pour que le processus ne s'en rende pas compte vu qu'il ne s'exécute pas pour le constater.
  - Cela suffit pour que l'utilisateur ait l'impression que son processus s'exécute assez régulièrement
- Certaines tâches peuvent avoir une priorité plus (ou moins) grande pour avoir plus (ou moins) de temps d'exécution.

Comment décider de donner la main à une autre tâche?

- Les processus peuvent rendre la main eux-mêmes quand
  - ils ont terminé
  - ils s'endorment volontairement en attendant un évènement (attente passive)
    - un clic, une frappe clavier,
    - arrivée d'un message réseau,
    - lecture/écriture disque
    - ...
- Mais le système peut aussi leur prendre de force (*Preemption*)
  - Garantie que tous les processus pourront s'exécuter un peu de temps en temps
  - Même si certains sont méchants et ne rendent jamais la main
    - Boucle infinie.

Mise en attente des processus

- Un processus dit à l'OS qu'il souhaite attendre un évènement
  - Ex: attente d'un clic, ou de la terminaison d'accès disque
- L'OS déplace la tâche sur une file d'attente spéciale, il ne peut plus être exécuté
- Quand l'évènement arrive (interruption matérielle), l'OS enlève le processus de la file d'attente pour qu'il puisse à nouveau s'exécuter.

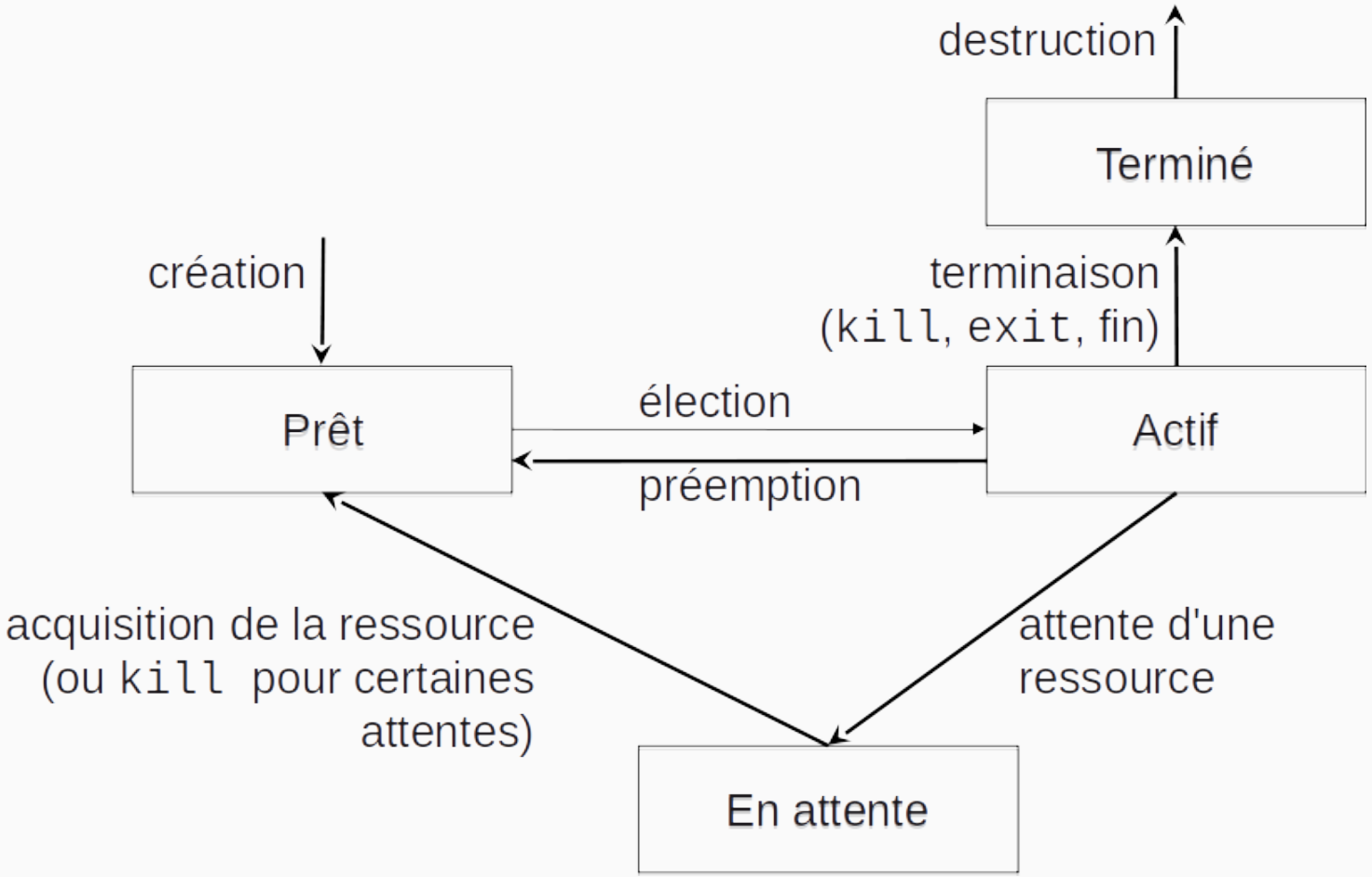


Cas de l'attente active

Les processus peuvent également attendre un événement de manière *active* (boucle while).

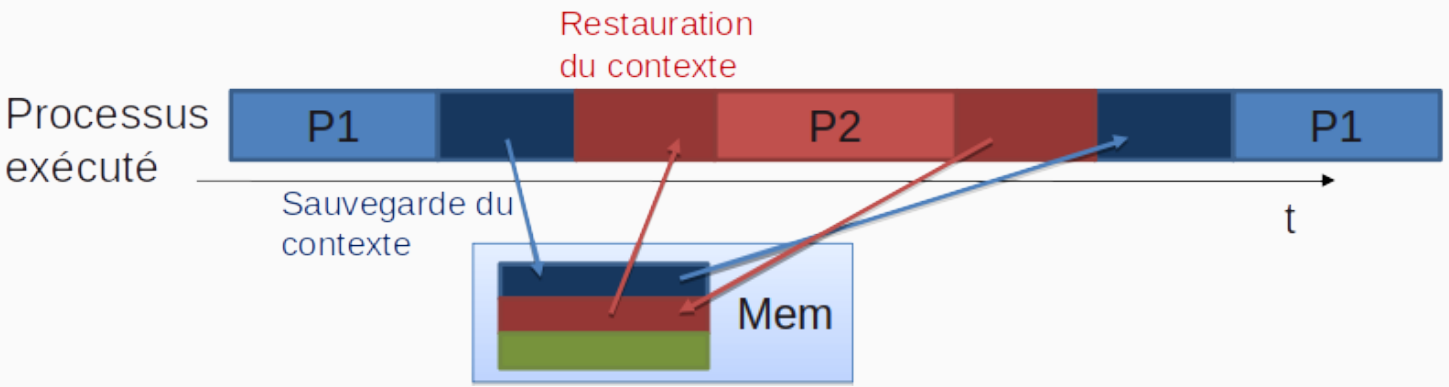
- Cela consomme plein de temps CPU, le coeur ne fait rien d'autre
- Gros gaspillage de ressources si on attend longtemps
- Pas bon pour le réseau (on ne sait pas quand le paquet va arriver, voire il peut ne jamais arriver)
- Pas bon pour le disque (les disques prennent plusieurs millisecondes à lire/écrire, ca fait des millions d'instructions gaspillées à attendre)
- Besoin de la préemption pour empêcher la tâche de consommer des ressources indéfiniment.

Diagramme d'état des processus



Comment changer de processus?

- Changement de contexte (*context switch*) :
  - Sauvegarde du contexte du processus évincé
  - Chargement du contexte du processus élu
- Contexte : ensemble des infos associées au processus
  - Valeur des registres
  - Informations mémoire (emplacement, etc.)
- Le temps du *context switch* doit être réduit au minimum



Processus et gestion mémoire

Création

Création d'un processus

Sous Windows

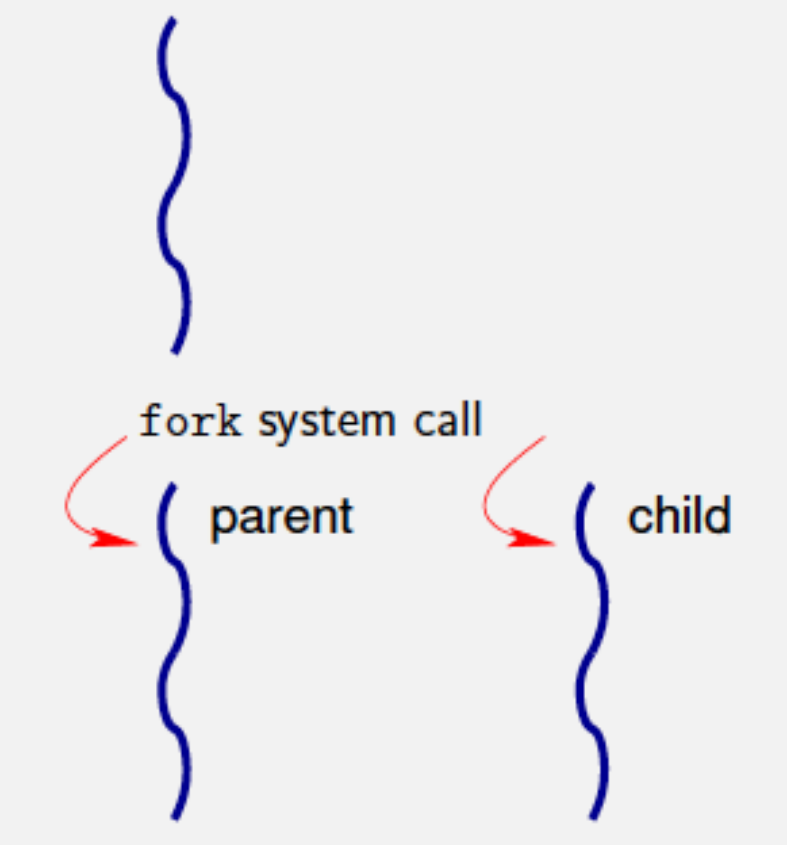
- On lance un nouveau processus avec `CreateProcess()` en disant quel programme on veut lancer avec quelles options
- Quand on clique sur l'icône d'un programme, c'est cette fonction qui est appelée.

Sous Unix

1. On crée un clone du processus courant avec `fork()`
2. Éventuellement, on change quelques paramètres
3. On appelle `exec()` pour transformer le processus en un programme différent
4. Les icônes et le shell Unix utilisent ces commandes.

Création de processus par clonage

- Fonctions `fork()` ou `clone()`
- Le processus fils :
  - exécute le même programme
  - possède une copie de l'espace virtuel du père
- Sous linux, seule méthode de création !



Processus et gestion mémoire

Gestion mémoire par l'OS

Partage de la mémoire

Deux processus peuvent s'exécuter en même temps sur deux coeurs sans se marcher dessus avec les mêmes adresses mémoire.

TP avec python ou C ou on affiche les adresses et leur contenu

Comment c'est possible?

Partage de la mémoire

- On pourrait lancer les processus avec des adresses différentes
  $\rightarrow$ Ca serait compliqué à implémenter, et fork ne marcherait plus
- le matériel nous aide à chaque accès mémoire, il traduit depuis des adresses virtuelles spécifiques au processus vers des adresses physiques globales
- Cela permet d'éviter de se marcher sur les pieds, et d'éviter des gros problèmes de sécurité

Mémoire virtuelle / fonctionnement

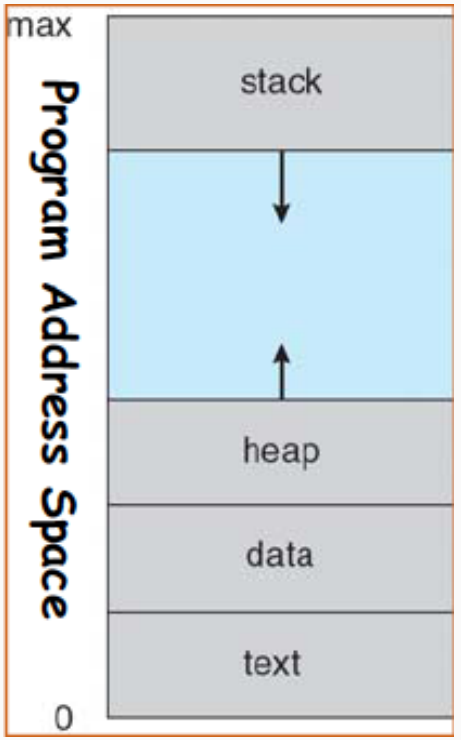
- c'est fait page par page (pagination, 4ko sur nos machines)
- l'OS rempli une table de pages décrivant quelle pages virtuelles de chaque processus correspond à quelle page physique, et le CPU la lit pour chaque accès mémoire
- c'est super optimisé dans le cpu parce que ca doit aller vite

Conclusion
Quand on lance un programme ou quand on alloue de la mémoire, on prend n'importe quelle page de mémoire physique disponible, et on l'associe a une page de mémoire virtuelle du processus qui a fait la demande.

La configuration de la mémoire virtuelle n'est modifiable qu'en mode privilégié

Mémoire virtuelle / espace d'adressage

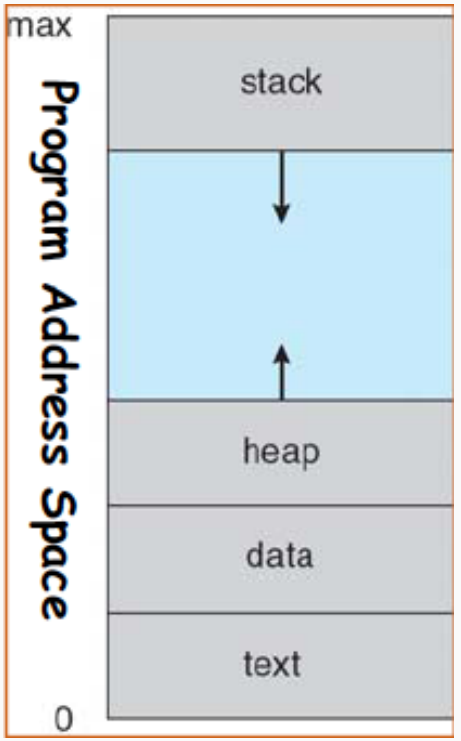
- Les processeurs modernes sont 64bits
- Cela veut dire que les adresses mémoires peuvent aller de 0 à 0xffffffffffffffff $= 2^{64} =$ 16 milliards de Go (adresse sur n bits $\rightarrow 2^n$ adresse possibles)
- En pratique, les CPUs et l'OS n'autorisent pas tant, plutôt $256\,To = 2^{48}$ actuellement
- C'est largement suffisant pour les programmes actuels
- Enfin, chaque processus ne voit que son espace d'adressage



Mémoire virtuelle / zones mémoires

- La mémoire est initialisée pour chaque processus en différentes zones ou segments
- Stack : pile où sont stockées les variables locales
- Heap (tas) : allocation dynamique, par malloc
- Data : zone de données globales ou statiques
- Texte : zone de code, des instructions du programme
- code des bibliothèques, piles des threads, . . .

Voir /proc/pid/maps



Mémoire virtuelle : Pagination à le demande

- Un programme peut avoir virtuellement plein de trucs en mémoire virtuelle, même si la RAM physique n'est que 8Go sur la machine.
- A un instant t, un programme n'utilise qu'un petit bout de mémoire, il n'a pas besoin de teraoctets. L'OS va charger en RAM ce dont le programme a besoin le reste reste sur le disque dur en attendant.

Pagination à la demande : pourquoi ca marche?

- Ca marche bien grâce au principe de localité
- Les données utilisées à t sont souvent les mêmes que celles utilisées à $t-1$ ou $t+1$.
- L'OS peut anticiper en chargeant de manière anticipée les données suivantes et en supprimant les données qui n'ont pas été utilisées depuis longtemps.
- L'OS doit être bon pour faire ça de manière anticipée car les chargement/déchargements sur disque sont lent parce que charger/décharger implique le disque donc c'est lent
- Ex: quand on démarre une nouvelle application comme firefox, seul le code de démarrage est chargé, pas le reste. Puis quand on utilise un plugin, ce plugin est chargé à ce moment là et si la RAM est pleine, l'OS va virer le code de démarrage de firefox parce qu'il n'a pas servi depuis longtemps.

Partage de mémoire physique entre processus

Problèmes

- Si les processus sont créé par clonage, on va rapidement se retrouver à cours de mémoire physique : duplication des bibliothèques, des zones de code, . . .
- La copie de toute ces pages mémoires peut-être coûteuse.

Partage de mémoire physique entre processus

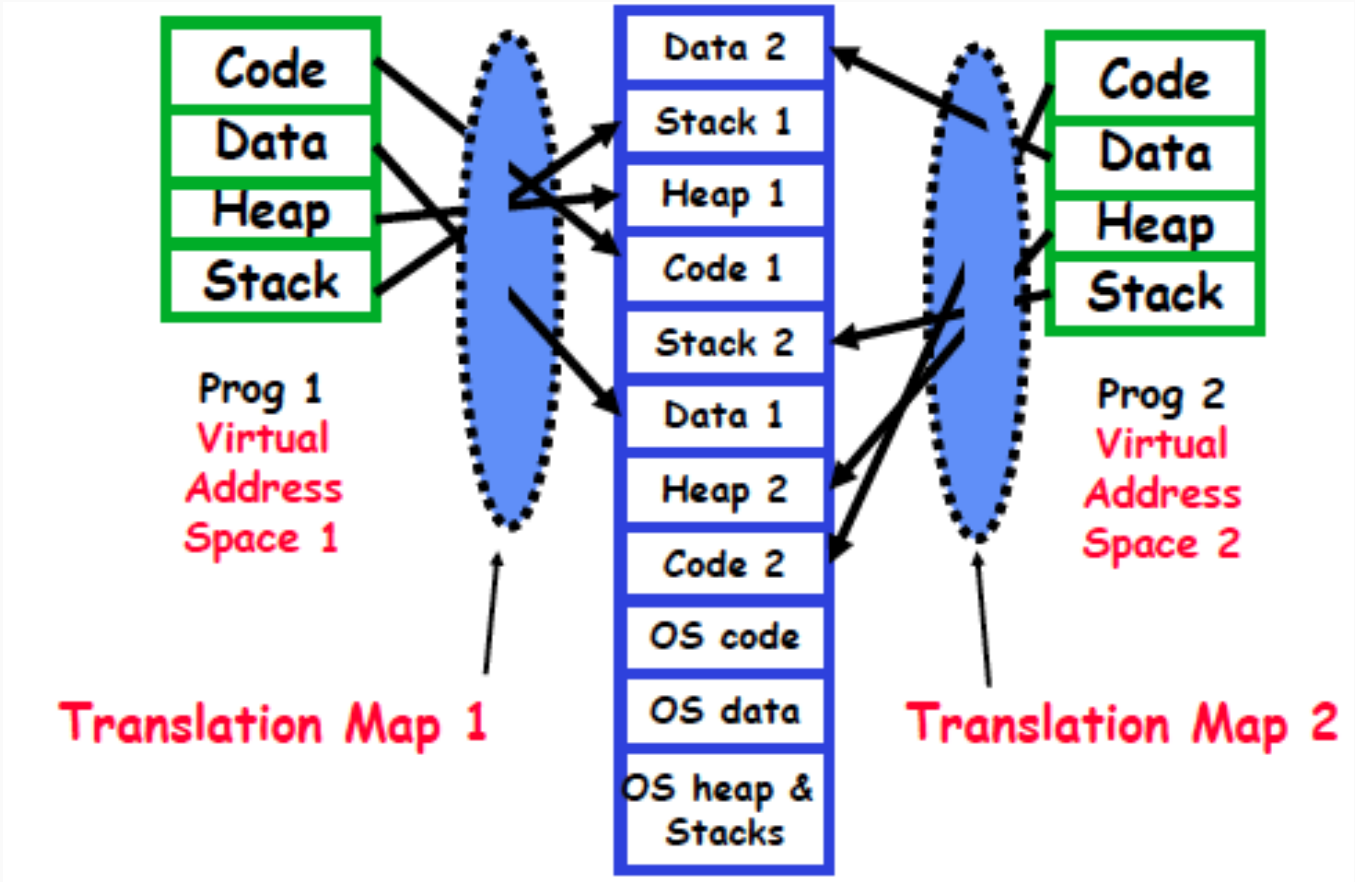
Problèmes

- Si les processus sont créé par clonage, on va rapidement se retrouver à cours de mémoire physique : duplication des bibliothèques, des zones de code, . . .
- La copie de toute ces pages mémoires peut-être coûteuse.

Solution

- Création de processus paresseuse :
  - Création d'un nouveau Process Control Block (pas cher)
  - Seule la table des pages virtuelles est copiée. La copie mémoire n'est faite que si un des processus modifie la donnée physique.
- Permet une gestion plus économique de la mémoire
- Permet une création plus rapide des processus

Lien entre mémoire virtuelle et mémoire réelle



Les processus légers : thread

- Avec les architectures multicœurs actuelles, les processus contiennent eux-mêmes plusieurs threads par processus.
- Les threads partagent la même mémoire virtuelle. Cela permet un partage simplifié des données, mais peut aussi introduire des problèmes d'accès concurrents aux données.
- Ils ont des exécutions différentes.
- Souvent utilisé pour paralléliser les applications, pour recouvrir des opérations bloquantes (1 thread attend l'opération bloquante, pendant qu'1 thread continue le calcul), ...