

ESGI

octobre 2022

Formation Théorie des systèmes d'exploitation

Intervenant

- Valentin OUAKI
 - Co-Gérant– ANDN-Services
 - Ingénieur Réseaux, sécurité et Aéronautique – Eclipse Global Connectivity
 - Mail : valentin.ouaki@andn-services.fr





Pourquoi utilisez un OS ?

Sans système d'exploitation, un seul et unique programme dans l'ordinateur se charge de tout faire, y compris manipuler le matériel.

Dès lors, un programmeur qui souhaite se passer de système d'exploitation doit tout programmer, y compris les parties du programme en charge de la gestion des périphériques, des ports d'entrée-sortie, de la mémoire...

Vu qu'on ne gère pas les périphériques et/ou la mémoire de la même façon sur tous les ordinateurs, ce que le programmeur aura réalisé sera difficilement portable sur d'autres ordinateurs, si tant est que cela soit possible.

Programmes systèmes et applicatifs

- Mais plutôt que de créer un seul et unique programme informatique chargé de tout gérer, on peut segmenter ce programme en plusieurs programmes séparés. Ces programmes peuvent être divisés en deux types :
 - les programmes systèmes gèrent la mémoire, les périphériques, et les autres programmes applicatifs
 - les programmes applicatifs sont des programmes qui délèguent la gestion de la mémoire et des périphériques aux programmes systèmes.

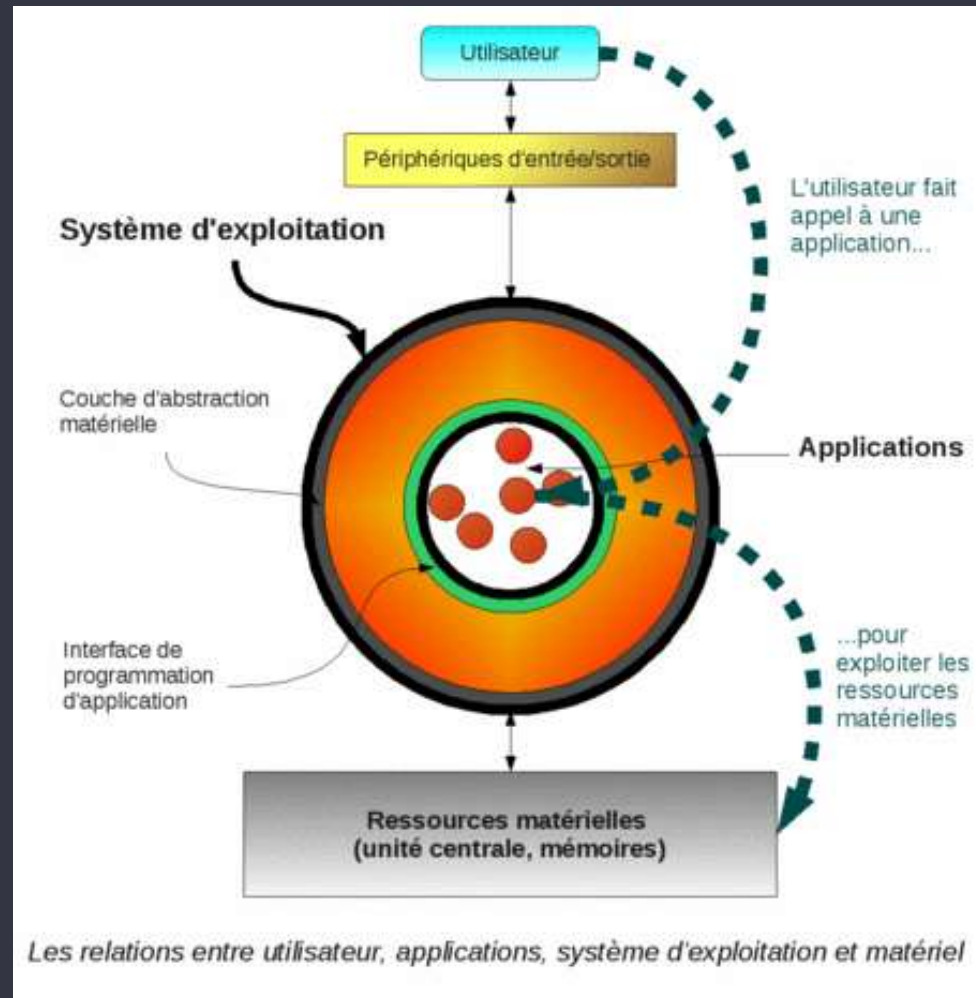
Programmes systèmes et applicatifs

- Liste de tâches délégués aux programmes systèmes
 - gérer une partie de ce qui concerne la mémoire ;
 - la gestion de l'exécution de plusieurs programmes sur un même ordinateur ;
 - permettre à plusieurs programmes d'échanger des données entre eux si besoin est ;
 - gérer tous les périphériques de l'ordinateur ;
 - la gestion des fichiers, du réseau, du son et de l'affichage ;

Programmes systèmes et applicatifs

- Généralement, les programmes systèmes sont tous regroupés dans ce qu'on appelle un système d'exploitation, aussi appelé OS (operating system en anglais).
- En plus de ces programmes systèmes, le reste de l'OS est composé de programmes applicatifs, dont certains permettent d'afficher une interface qui permet à l'utilisateur de pouvoir utiliser l'ordinateur comme il le souhaite.
- Évidemment, les programmes systèmes ne sont pas les mêmes sur tous les systèmes d'exploitation : c'est une des raisons qui font que certains programmes ne sont compatibles qu'avec (mettez ici le nom de n'importe quel OS).

Programmes systèmes et applicatifs



Programmes systèmes et applicatifs

- Cependant, un système d'exploitation ne sait pas utiliser tous les périphériques et ports d'entrée-sortie existants.
- Pour cela, on a inventé les **pilotes de périphériques**, des programmes systèmes qui permettent à un OS de communiquer avec un périphérique.
- Évidemment, la façon dont le pilote de périphérique va communiquer avec le système d'exploitation est standardisée pour faciliter le tout.

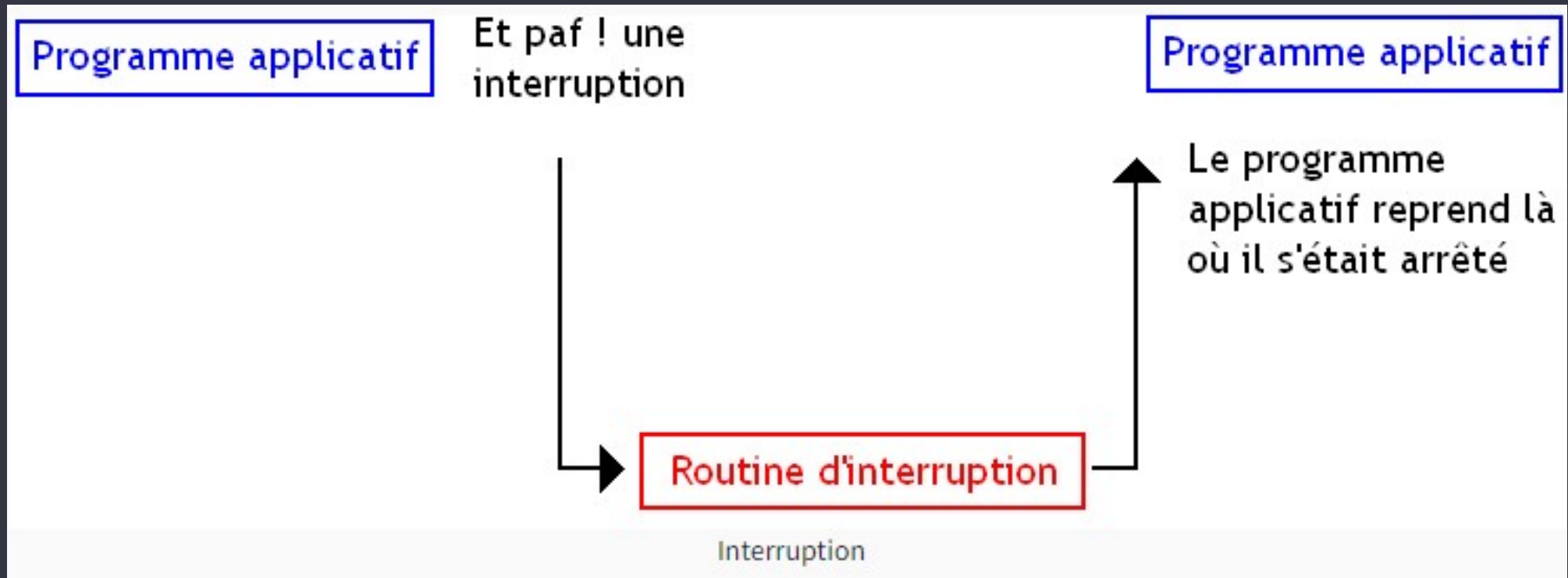
Appels système

- L'ensemble des opérations qui permettent à notre programme applicatif d'exécuter des programmes systèmes au besoin s'appelle un **appel système**.
- Pour en effectuer, les programmes applicatifs vont utiliser des fonctionnalités du processeur qu'on appelle des **interruptions**.

Appels système : Interruptions

- C'est une fonctionnalité de notre processeur qui va permettre d'arrêter temporairement l'exécution d'un programme pour en exécuter un autre.
- Ces interruptions ont pour but d'interrompre un programme, effectuer un traitement et de rendre la main au programme stoppé.
- L'interruption va exécuter un petit programme auquel on a donné le nom technique de **routine d'interruption**.

Appels système : Interruptions



Appels système : Vecteurs Interruptions

- Devant la multiplicité des périphériques, on se doute bien qu'il existe plusieurs routines d'interruption :
- un programme envoyant un ordre au disque dur sera différent d'un programme agissant sur une carte graphique.

Appels système : Vecteurs Interruptions

- Pour retrouver la routine en mémoire, certains ordinateurs utilisent un tableau qui stocke les adresses de chaque routine d'interruption appelé le **vecteur d'interruption**.
- Ce vecteur d'interruption est initialisé par le BIOS au démarrage de l'ordinateur, mais les pilotes et l'OS vont fournir leurs propres routines.

Appels système : Vecteurs Interruptions

- En clair, le vecteur d'interruption ne contiendra plus l'adresse servant à localiser la routine du BIOS, mais renverra vers l'adresse localisant la routine de l'OS.

Appels système : Comment profiter de ces interruptions ?

- Pour déclencher une interruption, les programmes applicatifs exécutent une instruction machine spéciale.
- Sur les processeurs x86, on utilise l'instruction machine **int**.
- Cette instruction machine a besoin de quelques petites informations pour faire ce qui est demandé et notamment de savoir quelle interruption exécuter.
- Pour cela, elle a besoin du numéro ou de l'adresse de l'interruption dans le vecteur d'interruption.

Appels système : Comment profiter de ces interruptions ?

- Une grande partie de ces routines a besoin qu'on leur fournisse des paramètres, des informations pour qu'elles fassent leur boulot.
- Par exemple, une routine devant afficher une lettre à l'écran aura besoin qu'on lui passe en entrée la lettre à afficher.
- Pour chaque routine, il suffira de copier ces paramètres (ou un pointeur vers ceux-ci) dans des petites mémoires ultra-rapides intégrées dans le processeur qu'on appelle les **registres**.

Noyau d'un OS : Espace Noyau et espace utilisateur

- Tout programme qui s'exécute avec le niveau de privilège de l'espace noyau va pouvoir faire tout ce qu'il souhaite : accéder aux périphériques et aux ports d'entrée-sortie, manipuler l'intégralité de la mémoire, etc.
- Tous les programmes de notre système d'exploitation placés dans l'espace noyau sont ce qu'on appelle **le noyau du système d'exploitation**.
- La moindre erreur de programmation d'un programme en espace noyau a des conséquences graves : tous vos écrans bleus ou vos *kernel panic* (l'équivalent chez les OS UNIX) sont dus à une erreur en espace noyau (généralement par un pilote de carte 3D ou un problème matériel).

Noyau d'un OS

- Les programmes en espace utilisateur ne peuvent pas écrire ou lire dans la mémoire des autres programmes ou communiquer avec un périphérique, il doivent déléguer cette tâche à un programme système via une interruption.
- L'avantage, c'est qu'une erreur commise par un programme en espace utilisateur n'entraîne pas d'écrans bleus. C'est donc un gage de sûreté et de fiabilité.

Noyau d'un OS : Implémentation

- Les programmes en espace utilisateur ne peuvent pas écrire ou lire dans la mémoire des autres programmes ou communiquer avec un périphérique, il doivent déléguer cette tâche à un programme système via une interruption.
- L'avantage, c'est qu'une erreur commise par un programme en espace utilisateur n'entraîne pas d'écrans bleus. C'est donc un gage de sûreté et de fiabilité.

Noyau d'un OS : Implémentation

- Les anneaux mémoires sont gérés par un circuit qui gère tout ce qui a rapport avec la gestion de la mémoire :
 - la **Memory Management Unit**, abrégée en MMU.
- Lorsqu'un programme effectue une instruction ou demande l'exécution d'une fonctionnalité du CPU interdite par son niveau de privilège, l'unité de gestion mémoire (la MMU, donc) va déclencher une interruption d'un type un peu particulier :
 - une **exception matérielle**.
- Généralement, le programme est arrêté sauvagement et un message d'erreur est affiché.
- Vous savez maintenant d'où viennent vos messages d'erreurs sous votre OS préféré.

Noyau d'un OS : Principaux types de noyaux

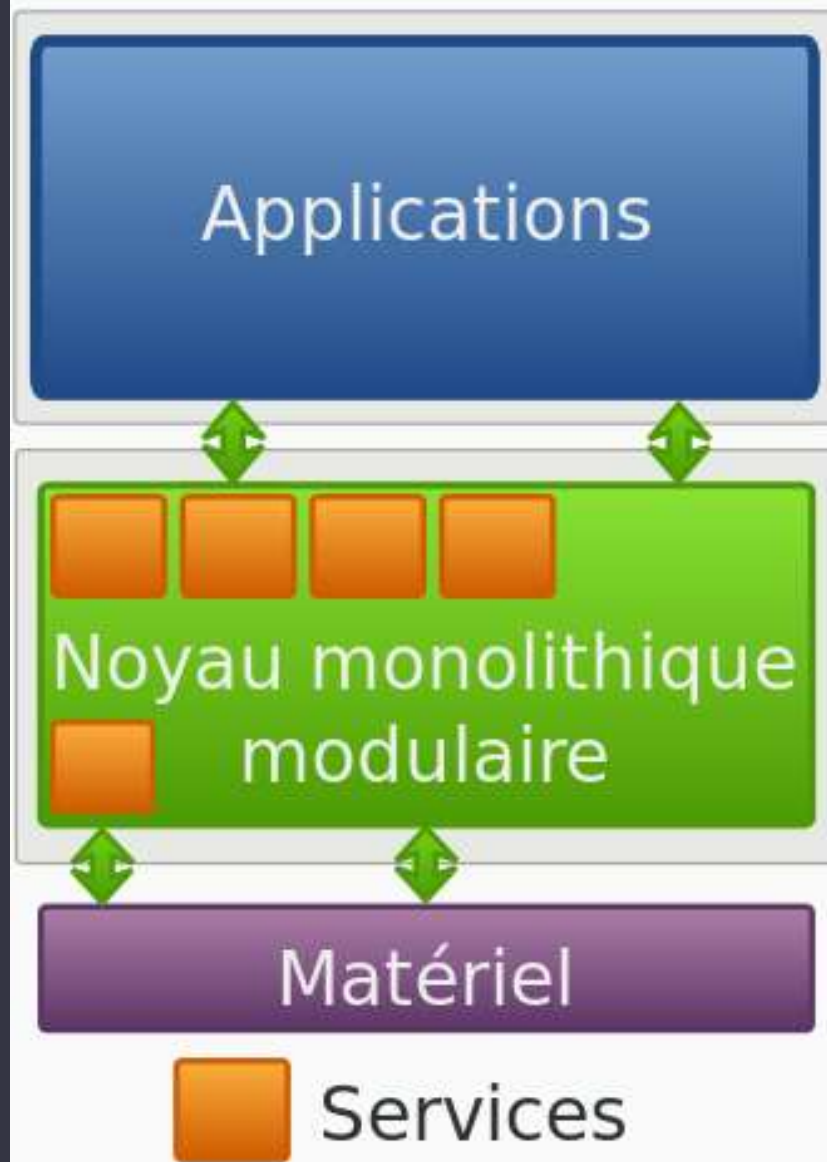
- Exécuter un appel système est très lent.
- Dès lors, exécuter des instructions juste pour changer de niveau de privilège et demander l'exécution d'une routine, devrait de préférence être évité.
- On se retrouve donc avec deux contraintes : **les performances et la sécurité.**

Noyau d'un OS : Principaux types de noyaux

- **Noyaux mégalithiques**
 - Avec les **noyaux mégalithiques**, tout l'OS est dans l'espace noyau.
- Dans le cas du **noyau monolithique**, un maximum de programmes systèmes est placé dans l'espace noyau.
- Avec ce genre d'organisation, très peu d'appels systèmes sont effectués, ce qui est un avantage en terme de performances.
- Mais cela est aussi un désavantage en terme de sûreté.

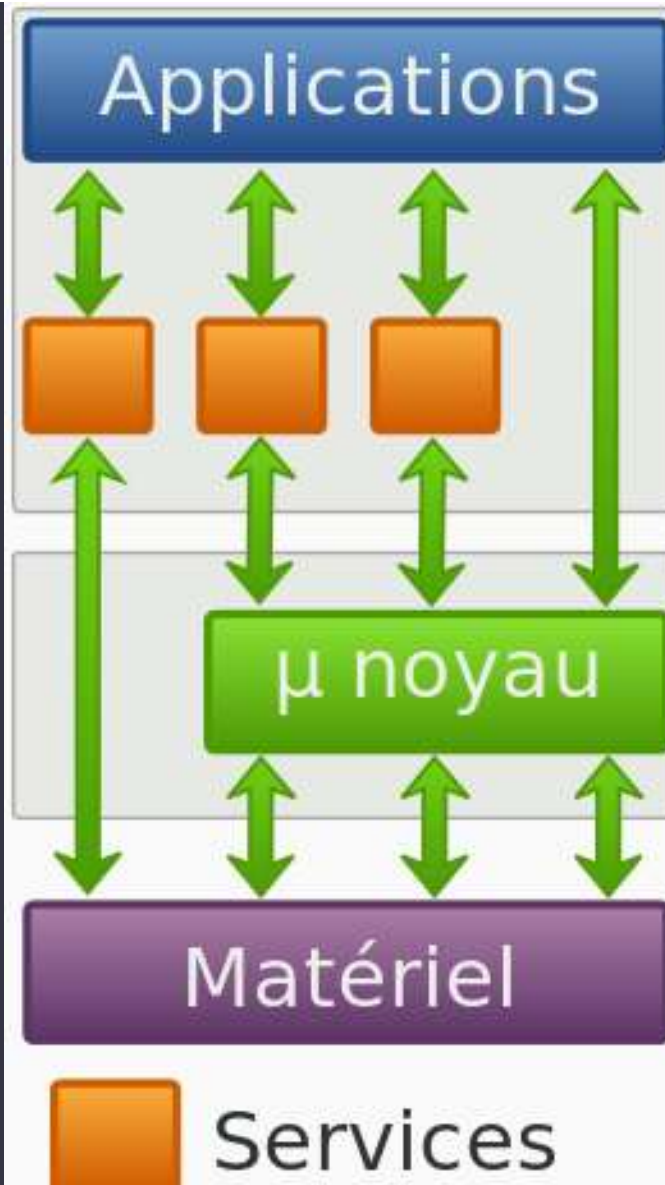
Noyau d'un OS : Principaux types de noyaux

- Un **noyau modulaire** est un noyau monolithique qui est divisé en plusieurs parties bien distinctes nommées les modules.
- Par exemple, chaque pilote de périphérique sera stocké dans un module séparé du reste du noyau.
- Avec ce genre d'organisation, on peut ne charger que ce dont on a besoin au lancement de l'ordinateur (par exemple, cela permet de ne pas charger le pilote d'un périphérique qui n'est pas branché sur l'ordinateur).
- Cela permet aussi de rajouter plus facilement des modules dans le noyau sans avoir à refaire celui-ci depuis le début.



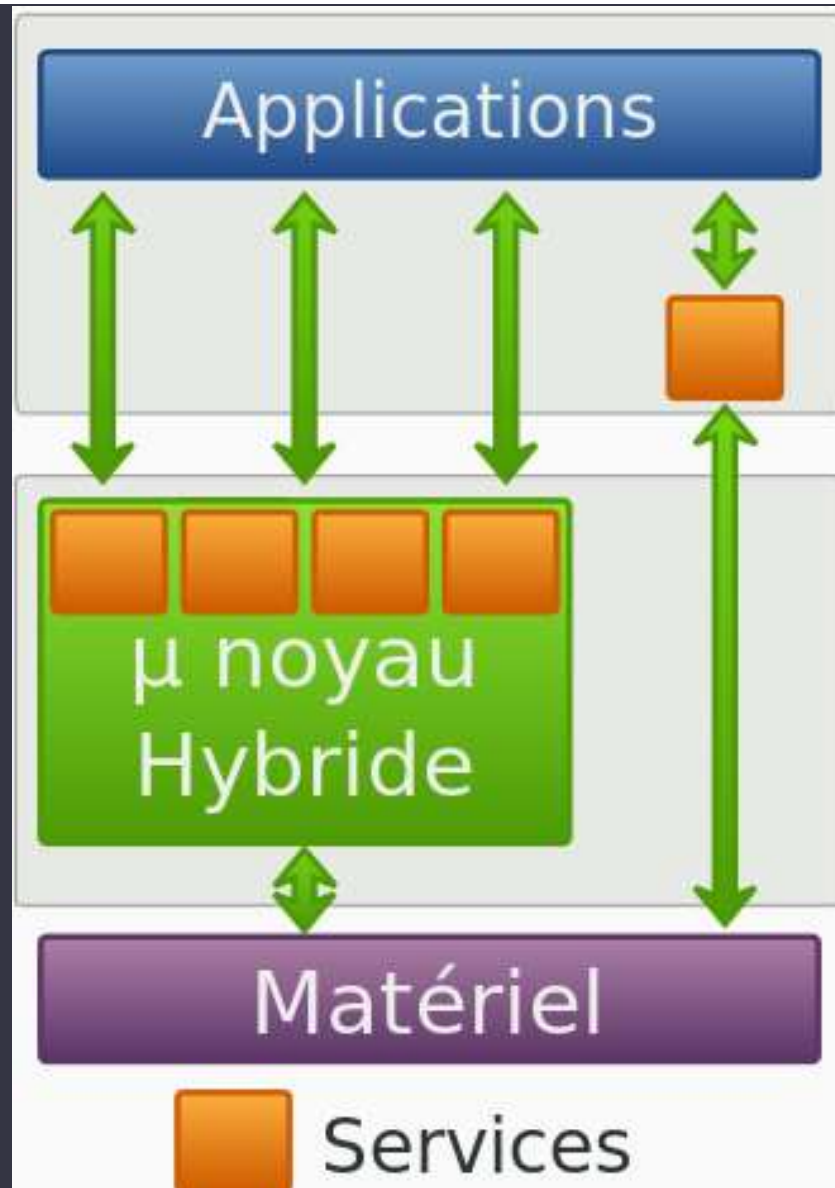
Noyau d'un OS : Principaux types de noyaux

- **Micro noyau**
- Pour gagner en sûreté de fonctionnement, certains créateurs de systèmes d'exploitation ont décidé de ne laisser dans le noyau que les programmes qui ont absolument besoin d'un niveau de privilège élevé.
- Ces **micro-noyaux** sont souvent très légers :
 - peu de programmes systèmes sont présents dans le noyaux, les autres étant évacués dans l'espace utilisateur.
- L'avantage, c'est qu'un *bug* a plus de chances de se retrouver dans l'espace utilisateur.
- Mais cela implique de nombreux appels systèmes entre les programmes systèmes en espace utilisateur et ceux en espace noyau, ce qui réduit les performances.



Noyau d'un OS : Principaux types de noyaux

- **Noyau hybride**
- Dans les **noyaux hybrides**, on garde la même philosophie que pour les micro-noyaux, en étant un peu plus souple :
 - on évacue un maximum de programmes systèmes dans l'espace utilisateur.
- Néanmoins, certains programmes systèmes, très demandeurs en appels systèmes sont placés en espace noyau.
- Cela évite de plomber les performances en générant trop d'appels systèmes.





Différence entre Bios et UEFI

BIOS vs UEFI

#UnhappyGhost

Bootting Old Way



Bootting New Way



For more posts visit:
unhappyghost.com

[Fb.com/geeksch00l](https://fb.com/geeksch00l)

Noyau Windows

- Démarrage de windows en mode UEFI, processus utilisés :
- **Bootmgr.exe or Bootmgr.efi** : Boot Configuration Data – contient les informations de démarrage de Windows.
 - Ce dernier remplace boot.ini
- **Windows Boot Loader (Winload.exe or Winload.efi)**
- **efibootmgr** : Modifier le chargeur de démarrage UEFI sur Linux

Noyau Windows

- Ci-dessous, la différence entre un démarrage UEFI et BIOS.
- En effet, le démarrage du PC et de l'OS est différent.
- Du moins au départ lors du chargement du BIOS et pour donner la main à Windows
- Ainsi en UEFI, le boot lance le firmware qui donne la main au EFI Boot Loader.
- La partition EFI stocke les firmwares des OS. Ainsi, les PC UEFI ne fonctionnent qu'avec des disques GPT.
- Ensuite le démarrage de Windows reste identique.

Noyau Windows

- En mode MBR :
 - **Ntldr** : Charge l'OS
- **Boot.ini** : Construit le menu de sélection
- **Bootsect.dos** : Chargé par Ntldr en vue d'un autre OS
- **Ntdetect.com** : Recherche le matériel disponible
- **Ntbootdd.sys** : Pour amorçage à partir d'un disque dont le contrôleur n'a pas le bios activé
- **Ntoskrnl.exe** : Noyau NT (System32).
- **System** : Paramètres de configuration (System32\configuration).
- **Hal.dll** : Couche HAL. Rend Ntoskrnl indépendant de la plate-forme sur laquelle il va fonctionner.

Noyau Windows

- Processus et séquence de démarrage d'un pc (POST, BIOS, OS)
- Plusieurs séquences et phases s'enchaînent dans **le processus de démarrage du PC** pour donner la main au système d'exploitation (Windows, Linux, ...).
- Cela notamment grâce à un chargeur de démarrage (boot loader) est un programme informatique qui charge le système d'exploitation principal ou l'environnement d'exécution de l'ordinateur après l'achèvement des auto-tests.

Noyau Windows

- Démarrage électrique du pc → exécution du code de démarrage dans la ROM (Read-only Memory).
- ROM + CPU → situé sur la carte mère
- Phase POST (Power-On Self Test)
- Ce test vérifie tous les matériels connectés, y compris la mémoire RAM et les périphériques de stockage secondaires pour être sûr que tout fonctionne correctement.

Noyau Windows

- Durant la phase de POST, chaque périphérique inclus dans la liste de démarrage charge son propre BIOS unique
- C'est dans cette phase du démarrage que l'on peut rencontrer des erreurs :
 - Smart hard disk error
 - BIOS ROM checksum error
 - System CMOS checksum bad
 - CMOS Checksum Mismatch
 - Des beep au démarrage du PC indiquant un matériel défectueux

Noyau Windows

- Une fois le matériel du PC initialisé et vérifié, le PC peut charger le système d'exploitation.
- Pour cela, le processus de démarrage **recherche la liste des périphériques de démarrage dans les réglages du BIOS**.
- Pour cela, le BIOS joue les périphériques dans **un ordre de démarrage (Boot Priority)** définis par l'utilisateur.
 - **Disk I/O error**
 - **Reboot and select proper Boot Device** or insert Boot Media in Selected Boot Device
 - **No bootable device, hit any key**
 - **non-system disk**
 - **Operating system not found**

Noyau Windows

- Une fois la fonctionnalité matérielle confirmée et que le système d'entrée/sortie est chargé, le processus de démarrage commence à charger le système d'exploitation à partir du périphérique de démarrage.
- C'est le **chargeur de démarrage (*BootLoader*) de l'OS** qui se charge de cela.
- Sur Windows, il s'agit de Windows Boot Loader, sur Linux, c'est Grub.

Noyau Windows

- La recherche du périphérique de démarrage diffère sur les PC en BIOS hérité, d'un PC en BIOS UEFI :
 - en BIOS hérité, le BIOS charge le **Master boot Record (MBR)** de la partition active
 - Sur un PC UEFI, le BIOS charge le firmware UEFI de la partition EFI /ESP
- C'est lors de cette phase que l'on peut rencontrer des problèmes et erreurs de démarrage de Windows Par exemple :
 - Des erreurs BCD lorsque la configuration du démarrage de Windows est corrompue ou erronée
 - Windows qui boucle sur la réparation automatique lorsque le système est corrompu
 - Résoudre l'erreur INACCESSIBLE_BOOT_DEVICE au démarrage de Windows 10
 - BSOD UNMOUNTABLE_BOOT_VOLUME au démarrage de Windows 10

Noyau Windows

- Le transfert de protocole
 - Une fois que les étapes précédentes sont terminées et que le système d'exploitation est chargé en toute sécurité en RAM, le processus de démarrage renonce à la commande du système d'exploitation.
 - Le système d'exploitation procède ensuite à l'exécution de toutes les routines de démarrage préconfigurées pour définir la configuration utilisateur ou l'exécution des applications.
 - À la fin du transfert de contrôle, l'ordinateur est prêt à être utilisé.

Noyau Windows

- Attardons nous sur le boot loader et son fonctionnement
- Le PC démarre avec la phase du BIOS qui se décompose en différentes sous phases :
 - **POST (Power On Self Test)** et vérification des composants du PC. Si un composant est défectueux le PC beep
 - **Charge la configuration du BIOS** et l'ordre de démarrage (Boot Priority)
- Puis le BIOS initialise le chargeur de démarrage de l'OS suivant l'ordre de priorité. L'emplacement et la recherche diffère selon si le BIOS est Legacy (MBR) ou le PC est en UEFI
 - Ensuite le chargeur de démarrage effectue certaine opération pour charger le noyau du système d'exploitation dans la mémoire RAM
 - Puis suivent les différentes phases d'initialisation de l'OS avec chargement de pilotes, services puis environnement de bureau et application

Noyau Windows

- Un chargeur de démarrage comporte donc souvent une partie configuration que l'utilisateur ou l'OS peut modifier.
- Celle stocke en général l'emplacement du noyau de l'OS ou d'un firmware.
- On peut aussi adjoindre certaines options et paramètres spécifiques du démarrage de l'OS.

Noyau Windows

- Qu'est ce que la partition de démarrage
- La zone MBR est un secteur de démarrage spécial situé au début d'un lecteur d'une taille de 512 octets.
- Sa taille est de 512 octets et se place dans le secteur 0 du disque. Le boot loader se place dans un secteur spécifique dit **secteur d'amorçage (boot sector)**.
- Le BIOS cherche et lit son contenu à chaque démarrage.

Noyau Windows

- Qu'est ce que le MBR :
 - Table de partition
 - Le boot Loader
 - La signature du disque
 - Nombre magique
 - situé dans les deux derniers octets du MBR (511-512), cette section doit contenir la valeur hexagonale AA55, qui classe officiellement cela comme un MBR valide.
 - Si il n'y a pas ça, alors on ne peut pas démarrer

Noyau Windows

BIOS HÉRITE



UEFI



Noyau Windows

- Les types de boot sous windows
- **EFI (*Extensible Firmware Interface*)** : Nouveau système présent sur les ordinateurs, il s'agit d'un logiciel intermédiaire entre le micrologiciel (firmware) et le système d'exploitation (OS) d'un ordinateur.
- EFI permet l'amorçage du système d'exploitation sur des disques GPT.
- **GPT (*GUID Partition Table*)** : Nouveau standard de table de partition, et permet l'amorçage des ordinateurs EFI. Cette nouvelle norme permet aussi de gérer des partitions pouvant aller jusqu'à 9,4 Zo

Noyau Windows

- **MBR (*Master Boot Record*)** : Le MBR est la zone de disque contenant les informations d'amorçage de disque dur ainsi que la table des partitions. Par opposition, au partitionnement GPT et ordinateur EFI. on peut aussi parler de disque MBR.
- **CSM (*Compatibility Support Module*)** : permet de rendre le démarrage MBR possible sur un ordinateur UEFI. Plus d'informations : Activer ou Désactiver l'option CSM dans le BIOS
- **Partition de disque primaire et étendue.** Ce sont des types de partitions de disque. Se reporter à cet article : Les partitions de disque sur Windows et Linux : primaire, étendue, GPT, MBR

Noyau Windows

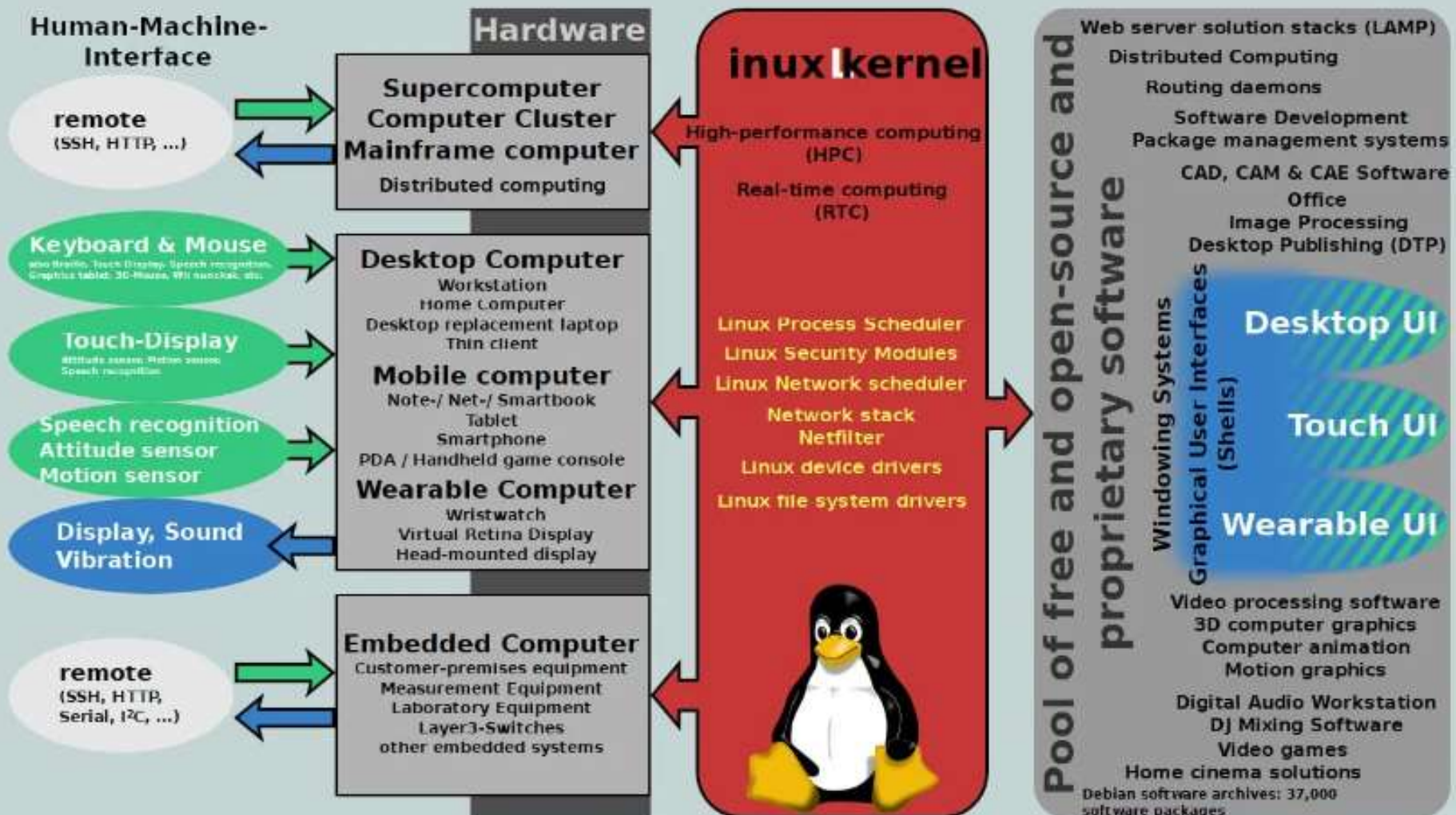
Noyau Windows

Noyaux Linux

- Qu'est-ce que le noyau Linux ?
- **La noyau Linux est le cœur principal du système d'exploitation.**
C'est en grande partie lui qui permet le fonctionnement du hardware et l'interaction entre l'utilisateur et le PC.
- Ainsi, le noyau gère la communication entre la partie logicielle et matériel du PC.
- On pense au PC mais le noyau Linux est aussi déployé sur une grande variété de systèmes informatiques :
 - tels que les appareils embarqués
 - les appareils mobiles (y compris son utilisation dans le système d'exploitation Android),
 - les ordinateurs personnels
 - les serveurs
 - les ordinateurs centraux et les supercalculateurs.

Noyaux Linux

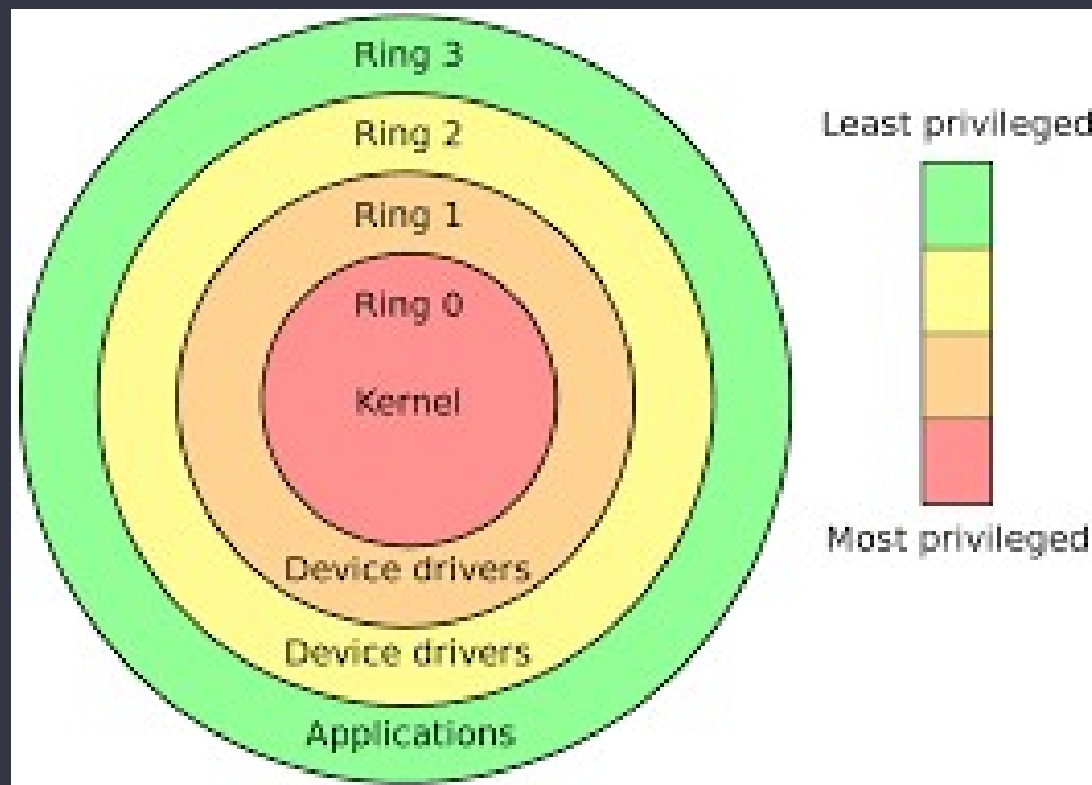
- Les principaux rôles du noyau linux :
 - **Gestion de la mémoire:** gardez une trace de la quantité de mémoire utilisée pour stocker quoi et où
 - Gestion des processus: déterminez quels processus peuvent utiliser l'unité centrale (CPU), quand et pendant combien de temps
 - **Pilotes de périphériques:** agissent en tant que médiateur / interprète entre le matériel et les processus
 - **Appels système et sécurité:** recevez les demandes de service des processus



Noyaux Linux

- Le noyau Linux fonctionne avec différences espaces et couches protégées.
- On nomme ces dernières anneau ou Ring et vont de 0 à 3 :
- **Kernel internal** : est l'emplacement où le code du noyau est stocké et s'exécute sous.
 - **Ring 0 (mode noyau)** : L'anneau 0 (espace noyau) est l'anneau le plus privilégié et a accès à toutes les instructions de la machine
 - **Ring 1 et 2** peut être utilisé par les hyperviseurs ou pilotes de machine virtuelle,
- **Kernel user space (Ring 3)** : qui est un ensemble d'emplacements où s'exécutent les processus utilisateur normaux (c'est-à-dire tout autre que le noyau).
- Le rôle du noyau est de gérer les applications qui s'exécutent dans cet espace

Noyaux Linux



Noyaux Linux

- Les pilotes de périphériques et les extensions de noyau s'exécutent dans **l'espace noyau** (anneau 0 dans de nombreuses architectures de processeur), avec un accès complet au matériel, bien que certaines exceptions s'exécutent dans l'espace utilisateur.
- Le matériel est représenté dans la hiérarchie des fichiers.
- Les applications utilisateur interagissent avec les pilotes de périphériques via des entrées dans **les répertoires /dev ou /sys**.
- Les informations sur les processus sont également mappées au système de fichiers via **le répertoire /proc**.
- La reconnaissance du noyau Linux se fait à travers **le daemon udev**.

Noyaux Linux

- Enfin **Linux est un noyau monolithique avec une conception modulaire.**
- On peut charger ou décharger des modules avec **les commandes modprobe et rmmod.**
- Cela permet de charger un pilote interne au noyau ou des fonctionnalités spécifiques.
- On peut alors configurer les modules à charger au démarrage du PC ou en charger/décharger à n'importe quel moment.

Noyaux Linux

- Un noyau monolithique, bien que plus rapide qu'un micro-noyau, présente l'inconvénient du manque de modularité et d'extensibilité.
- Un module de noyau est **un fichier d'objet qui contient du code** qui peut **étendre la fonctionnalité du noyau**.
- Lorsqu'un module de noyau n'est plus nécessaire, il peut être déchargé.
- La plupart des pilotes de l'appareil sont utilisés sous forme de modules de noyau.

Noyaux Linux

- Les fichiers du noyau Linux se trouvent dans le **répertoire /boot**.
On trouve trois principaux fichiers :
 - **initrd-XXX.img** : Initrd est l'abréviation de "Initial ramdisk. Initrd est généralement utilisé pour démarrer temporairement le matériel dans l'état où le noyau réel vmlinuz peut prendre le relais et continuer à démarrer
 - **System.map-XXX** : c'est la est une table de symboles de noyau d'un noyau spécifique. C'est le lien avec le système. Carte de votre noyau en cours d'exécution
 - **vmlinuz-XXXX** : Vmlinuz est un noyau amorçable et compressé. «VM» représente «Virtual Memory».

Noyaux Linux

Activités Terminal ▼



```
mak@mak-virtual-machine:~$ ls -lh /boot/
total 70M
-rw-r--r-- 1 root root 243K avril 12 17:02 config-5.8.0-50-generic
drwx----- 3 root root 4,0K janv.  1 1970 efi
drwxr-xr-x 4 root root 4,0K avril 23 10:23 grub
lrwxrwxrwx 1 root root  27 avril 23 08:59 initrd.img -> initrd.img-5.8.0-50-generic
-rw-r--r-- 1 root root  51M avril 23 10:22 initrd.img-5.8.0-50-generic
lrwxrwxrwx 1 root root  27 avril 23 10:23 initrd.img.old -> initrd.img-5.8.0-50-generic
-rw-r--r-- 1 root root 179K août  18 2020 memtest86+.bin
-rw-r--r-- 1 root root 181K août  18 2020 memtest86+.elf
-rw-r--r-- 1 root root 181K août  18 2020 memtest86+_multiboot.bin
-rw----- 1 root root 5,4M avril 12 17:02 System.map-5.8.0-50-generic
lrwxrwxrwx 1 root root  24 avril 23 08:59 vmlinuz -> vmlinuz-5.8.0-50-generic
-rw----- 1 root root  14M avril 12 18:48 vmlinuz-5.8.0-50-generic
lrwxrwxrwx 1 root root  24 avril 23 10:23 vmlinuz.old -> vmlinuz-5.8.0-50-generic
mak@mak-virtual-machine:~$
```

Noyaux Linux

Noyaux Linux

- **Le Secure Boot (démarrage sécurisé) est un mécanisme de sécurité** qui vise à interdire l'exécution de code inconnu.
- Il protège des **rootkits** et plus particulièrement des **bootkits**.
- Pour démarrer en UEFI, Linux propose deux firmwares
- **EFI/XXXX/shimx64.efi** – C'est le composant SHIM qui agit comme UEFI Boot Loader.
 - Microsoft le signe et SHIM intègre une autre clé CA spécifique à la distribution.
 - Cette dernière est utilisée pour signer à son tour d'autres programmes (par exemple Linux, GRUB, fwupdate).

Noyaux Linux

- **EFI/XXXX/grubx64.efi** – c'est le Bootloader GRUB qui charge ensuite le noyau Linux (Kernel).

```
root@mak-virtual-machine: /  
root@mak-virtual-machine:/# ls -lh /boot/efi/EFI/ubuntu/  
total 4,1M  
-rwx----- 1 root root 108 juil. 30 20:40 BOOTX64.CSV  
-rwx----- 1 root root 126 juil. 30 20:40 grub.cfg  
-rwx----- 1 root root 1,6M juil. 30 20:40 grubx64.efi  
-rwx----- 1 root root 1,3M juil. 30 20:40 mmx64.efi  
-rwx----- 1 root root 1,3M juil. 30 20:40 shimx64.efi  
root@mak-virtual-machine:/#
```

Noyaux Linux

- Les firmware uefi de debian :

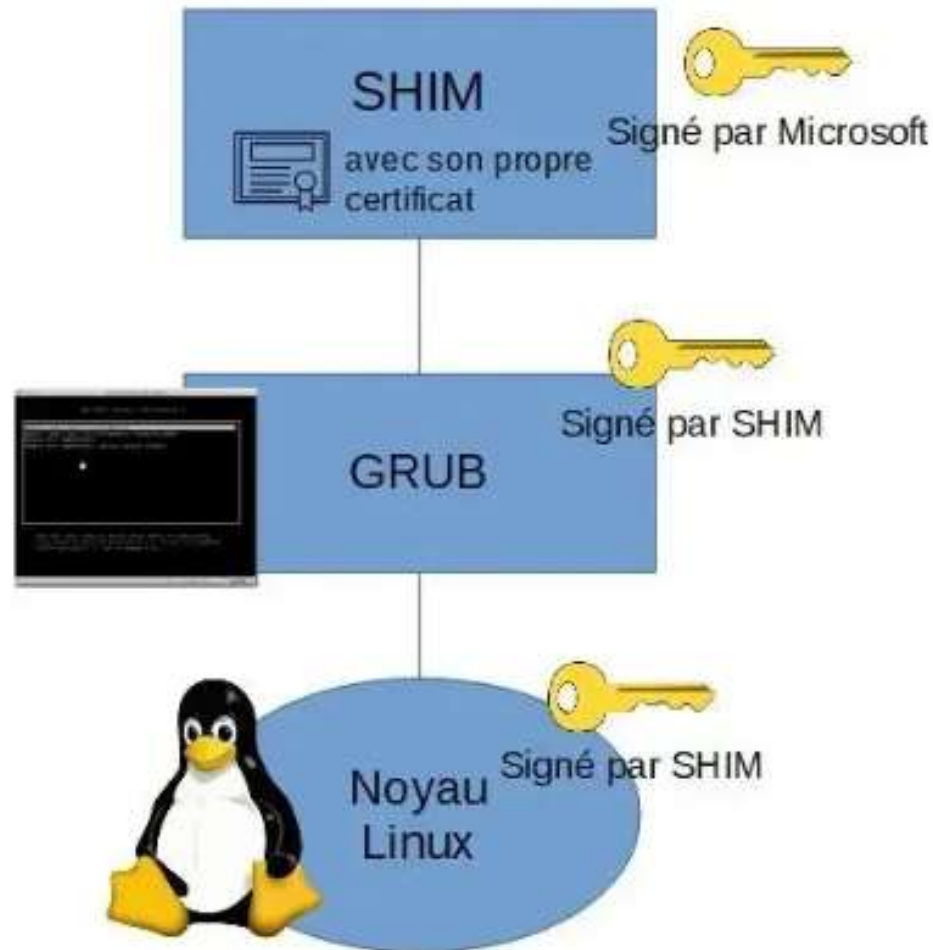
| Architecture | Chemin |
|--------------|--------------------------|
| amd64 | \EFI\debian\grubx64.efi |
| i386 | \EFI\debian\grubia32.efi |
| arm64 | \EFI\debian\grubaa64.efi |
| armhf | \EFI\debian\grubarm.efi |

Noyaux Linux

- Le firmware SHIM est signé avec le CA Microsoft mais embarque à son tour son propre CA.
- Ce certificat sert à son tour à signer tous les sous-composants Linux : le boot loader Grub et le kernel Linux.

Noyaux Linux

Schéma démarrage PC Linux UEFI



Noyaux Linux

- Différente phase de chargement du bios UEFI Linux
- **SEC Phase** : C'est la phase d'initialisation où rien n'est exécuté. Il s'agit de vérifier le matériel.
- **PEI Phase** : elle prépare la plate-forme pour l'initialisation du système principal dans la phase DXE.
- **DXE Phase** : Cette phase exécute des pilotes basée sur les ressources découvertes et décrites dans la phase PEI.
- **BDS** : On localise le l'OS Boot Manager depuis les périphériques de démarrage (USB, Disque locaux ou réseau). La partition EFI est trouvée et le firmware shim est chargé, si sa signature numérique est valide.

Noyaux Linux

- Une fois le BIOS chargé, on s'attaque au Kernel
- Le chargeur de démarrage grub2 démarre un noyau Linux et vérifie sa signature numérique.
 - *Si le noyau n'est pas signé, grub2 appellera **ExitBootServices** avant de démarrer le noyau non signé)*
 - Sinon le noyau Linux se charge

Noyaux Linux

- **Initialisation du kernel Linux et systemd ou init**
- Le noyau Linux s'initialise et monte la partition système /.
- Cela peut se faire au préalable par le disque Ram initramfs.
- Lorsque celle-ci est marquée comme propre (clean), le boot continue.
- Sinon vous pouvez avoir une erreur qui indique d'effectuer une analyse et réparation sfck.
- Dans ces là, Linux demande le mot de passe root.

Noyaux Linux

- Pour initialiser notre OS, nous avons besoin de GRUB
- Un chargeur de démarrage est le premier programme logiciel qui s'exécute au démarrage d'un ordinateur.
- Il est responsable **du chargement du le logiciel du noyau du système d'exploitation** (tel que Hurd ou Linux).
- Le noyau, à son tour, initialise le reste du système
- d'exploitation.

Noyaux Linux

- En 1995 GRUB 1 est publié, mais c'est en 2002 que GRUB 2 fait son apparition.
- Il est alors devenu rapidement **le chargeur de démarrage** pour la plupart des distributions Linux.
- A cette époque, il remplace alors **LILO (Linux LOader)**

Noyaux Linux

- **Le Bootloader** se doit de permettre de charger le système d'exploitation Linux.
- Lors du démarrage du PC, le BIOS lit les informations du Bootloader :
 - soit depuis le secteur de boot pour un disque MBR
 - soit depuis la partition EFI pour un PC en UEFI

Noyaux Linux

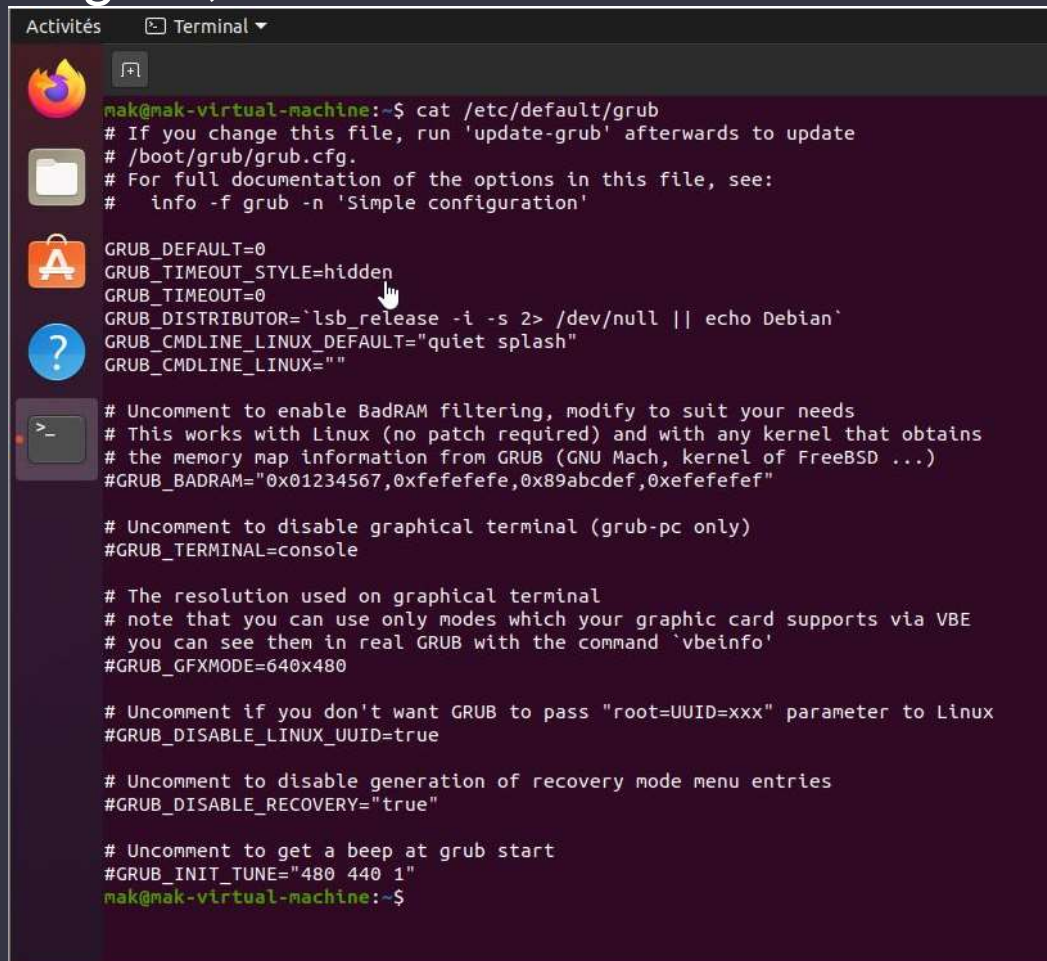
- Enfin GRUB prend en charge plusieurs noyaux Linux et permet à l'utilisateur de choisir entre eux au démarrage à l'aide d'un menu.
- Il permet aussi de charger d'autres OS comme Windows dans le cas d'un **dual-boot ou multiboot**.
- Cela se fait grâce au mécanisme de **chargement en chaîne (chain loading)** qui permet de charger un autre loader en l'occurrence **Windows Boot Manager**.

Noyaux Linux

- Fonctionnalité supporté par GRUB :
 - Conformité avec la spécification **Multiboot**
 - Les fonctions de base sont faciles à utiliser pour un utilisateur final. Bonne fonctionnalité pour les experts / concepteurs OS.
 - **Compatibilité** pour démarrer FreeBSD, NetBSD, OpenBSD et GNU / Linux. Les systèmes d'exploitation propriétaires tels que la plupart des versions actuelles de Windows sont pris en charge via une fonction de **chargement en chaîne (chain loader)** .
 - Fournit **un mode rescue avec un shell** limité pour pouvoir booter lorsque les fichiers de configuration sont manquants, corrompus
 - Peut se charger par le réseau

Noyaux Linux

- Pour modifier grub, il faut éditer le fichier /etc/default/grub



The image shows a terminal window titled 'Terminal' with a dark background. The prompt is 'mak@mak-virtual-machine:~\$'. The user has run the command 'cat /etc/default/grub', and the output is displayed. The output includes several comments and configuration lines for GRUB. A mouse cursor is pointing at the line 'GRUB_DISTRIBUTOR=\`lsb_release -i -s 2> /dev/null || echo Debian\''. The terminal window has a sidebar on the left with icons for 'Activités', 'Terminal', and other applications.

```
mak@mak-virtual-machine:~$ cat /etc/default/grub
# If you change this file, run 'update-grub' afterwards to update
# /boot/grub/grub.cfg.
# For full documentation of the options in this file, see:
#   info -f grub -n 'Simple configuration'

GRUB_DEFAULT=0
GRUB_TIMEOUT_STYLE=hidden
GRUB_TIMEOUT=0
GRUB_DISTRIBUTOR=\`lsb_release -i -s 2> /dev/null || echo Debian`
GRUB_CMDLINE_LINUX_DEFAULT="quiet splash"
GRUB_CMDLINE_LINUX=""

# Uncomment to enable BadRAM filtering, modify to suit your needs
# This works with Linux (no patch required) and with any kernel that obtains
# the memory map information from GRUB (GNU Mach, kernel of FreeBSD ...)
#GRUB_BADRAM="0x01234567,0xfefefefefefefefefefef,0x89abcdef,0xfefefefef"

# Uncomment to disable graphical terminal (grub-pc only)
#GRUB_TERMINAL=console

# The resolution used on graphical terminal
# note that you can use only modes which your graphic card supports via VBE
# you can see them in real GRUB with the command `vbeinfo'
#GRUB_GFXMODE=640x480

# Uncomment if you don't want GRUB to pass "root=UUID=xxx" parameter to Linux
#GRUB_DISABLE_LINUX_UUID=true

# Uncomment to disable generation of recovery mode menu entries
#GRUB_DISABLE_RECOVERY="true"

# Uncomment to get a beep at grub start
#GRUB_INIT_TUNE="480 440 1"
mak@mak-virtual-machine:~$
```

Noyaux Linux

```
[    1.088346] piix4_smbus 0000:00:07.3: SMBus Host Controller not enabled!  
[    1.641899] sd 32:0:0:0: [sda] Assuming drive cache: write through  
/dev/sda5: clean, 208072/2588672 files, 2419048/10353920 blocks
```

—

- Puis **charge systemd ou init** (selon la distribution et le système privilégié).
- Enfin ce dernier charge les scripts de démarrage dans un ordre prédéfini, selon la configuration et les dépendances.

Noyaux Linux

- Linux termine de booter pour lancer les daemons et services un à un.

```
[ OK ] Finished Detect the available GPUs and deal with any system changes.
[ OK ] Started Login Service.
systemd-logind.service
[ OK ] Started Avahi mDNS/DNS-SD Stack.
avahi-daemon.service
[ OK ] Started Make remote CUPS printers available locally.
cups-browsed.service
[ OK ] Started WPA supplicant.
wpa_supplicant.service
[ OK ] Started Thermal Daemon Service.
thermald.service
[ OK ] Started Switcheroo Control Proxy service.
Starting Save/Restore Sound Card State...
switcheroo-control.service
[ OK ] Finished Save/Restore Sound Card State.
alsa-restore.service
[ OK ] Started Network Manager.
[ OK ] Reached target Network.
[ OK ] Reached target Sound Card.
NetworkManager.service
Starting Network Manager Wait Online...
Starting OpenVPN service...
Starting Permit User Sessions...
[ OK ] Started Unattended Upgrades Shutdown.
unattended-upgrades.service
[ OK ] Finished OpenVPN service.
openvpn.service
[ OK ] Finished Permit User Sessions.
systemd-user-sessions.service
```

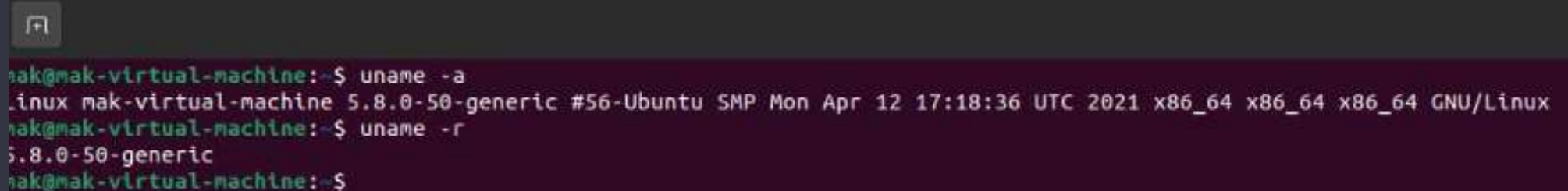
Noyaux Linux

- Enfin le démarrage se termine par le chargement du Display Manager (gdm, kdm, etc) ou simplement d'un terminal si aucune environnement graphique n'est installée.

```
Starting GNOME Display Manager...  
Starting Hold until boot process finishes up...  
[ OK ] Started Dispatcher daemon for systemd-networkd.  
networkd-dispatcher.service  
[ OK ] Started Authorization Manager.  
polkit.service  
Starting Modem Manager...
```

Noyaux Linux

- Comment connaître sa version du noyaux linux ?
- Uname -a → **La commande uname** est la plus simple car elle vous donne le nom de la machine, la version du noyau et l'architecture.
Pour obtenir toutes ces informations systèmes dont la version du noyau Linux

A terminal window with a dark background and light-colored text. The prompt is 'mak@mak-virtual-machine:~\$'. The first command is 'uname -a', and the output is 'Linux mak-virtual-machine 5.8.0-50-generic #56-Ubuntu SMP Mon Apr 12 17:18:36 UTC 2021 x86_64 x86_64 x86_64 GNU/Linux'. The second command is 'uname -r', and the output is '5.8.0-50-generic'.

```
mak@mak-virtual-machine:~$ uname -a
Linux mak-virtual-machine 5.8.0-50-generic #56-Ubuntu SMP Mon Apr 12 17:18:36 UTC 2021 x86_64 x86_64 x86_64 GNU/Linux
mak@mak-virtual-machine:~$ uname -r
5.8.0-50-generic
mak@mak-virtual-machine:~$
```

Noyaux Linux

| Commandes | Description |
|--|---|
| <code>-a</code> | Afficher toutes les informations |
| <code>-m</code> <code>--machine</code> | Afficher le le nom du matériel de la machine |
| <code>-o</code> <code>--operating-system</code> | Afficher le système d'exploitation (généralement GNU / Linux) |
| <code>-p</code> <code>--processor</code> | Afficher le type de processeur ou «inconnu» |
| <code>-r</code> <code>--kernel-release</code> | Afficher la version du noyau |
| <code>-s</code> <code>--kernel-name</code> | Affiche le nom du noyau |
| <code>-v</code> <code>--kernel-version</code> | Afficher la version du noyau (comprend généralement le système d'exploitation de base et l'heure à laquelle le noyau a été compilé) |

Noyaux Linux

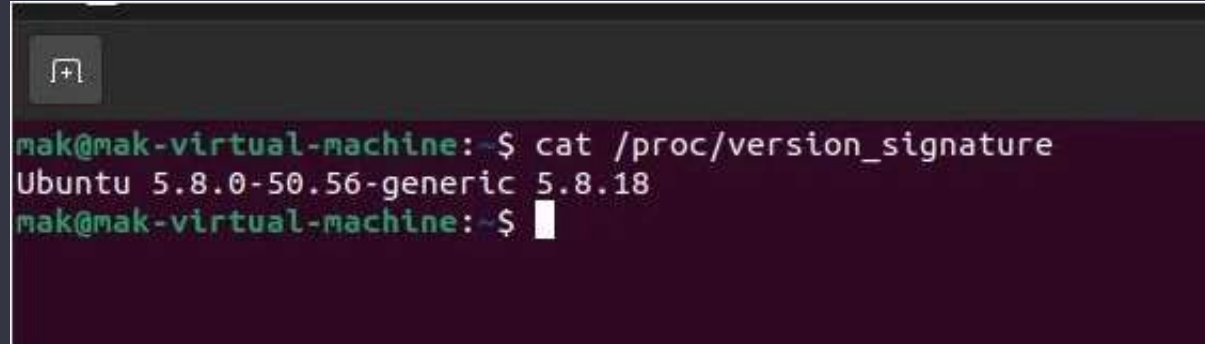
- Hostnamectl
- Cat /proc/version

```
mak@mak-virtual-machine:~$ hostnamectl
  Static hostname: mak-virtual-machine
            Icon name: computer-vm
            Chassis: vm
            Machine ID: 1868c8b1c7664956978a9d21e8054bb8
            Boot ID: 013cb643134a4cfaaed88f78ef0e8e43
    Virtualization: vmware
  Operating System: Ubuntu 20.10
            Kernel: Linux 5.8.0-50-generic
    Architecture: x86_64
mak@mak-virtual-machine:~$
```

```
mak@mak-virtual-machine:~$ cat /proc/version
Linux version 5.8.0-50-generic (bulldog@lgw01-and64-051) (gcc (Ubuntu 10.2.0-13ubuntu1) 10.2.0, GNU ld (GNU Binutils for Ubuntu) 2.35.1) #56-Ubuntu SMP Mon Apr 12
17:18:36 UTC 2021
mak@mak-virtual-machine:~$
```

Noyaux Linux

- Cat /proc/version_signature

A terminal window with a dark background and a title bar containing a window icon and a plus sign. The prompt is 'mak@mak-virtual-machine:~\$'. The command 'cat /proc/version_signature' has been entered and executed, resulting in the output 'Ubuntu 5.8.0-50.56-generic 5.8.18'. The prompt is now 'mak@mak-virtual-machine:~\$' with a cursor.

```
mak@mak-virtual-machine:~$ cat /proc/version_signature
Ubuntu 5.8.0-50.56-generic 5.8.18
mak@mak-virtual-machine:~$
```

Processus et threads



• Notion de **Processus**

- Concept central pour tout SE
 - Séquence d'actions produites par l'exécution d'une suite d'instructions
- 1 processus = **1 instance d'un programme en cours d'exécution**
 - Tâche (fr) / Task (en)
 - Environnement propre → contexte d'exécution
 - » Création, état, destruction dynamiques
 - Mémoire propre → espace d'adressage
 - » Code, données du programme...

Processus et threads



Différence entre Processus et Programme

- Un programme est une entité « statique »
 - Occupe un espace mémoire
 - » Code + données
 - Ne produit aucune action
- Un processus est une entité « dynamique »
 - Différents états : naît, vie... et meurt
 - Instance d'un programme
 - » Demande d'exécution d'un programme → création d'un processus
 - » Exécution possible de plusieurs instances d'un même programme simultanément = plusieurs processus d'un même programme

Processus et threads



• **Différence entre Processus et Programme**

- Terme « programme » est ambigu dans le langage courant
 - « mon programme ne compile pas » : fichier source
 - « je lance mon programme » : fichier exécutable
 - « mon programme a planté » : processus

Processus et threads



■ Notion de Processus

– Multiprogrammation

- **système à temps partagé** → l'UC passe d'un processus à l'autre « rapidement »
 - » Plusieurs fils d'exécution séquentielle s'exécutent à la suite
 - » *Exécute un programme utilisateur + lit les données d'un disque + affiche des résultats sur le terminal...*
 - » Illusion d'une exécution simultanée → **système multi-tâches**
- Virtualisation d'une Unité Centrale (UC)
 - » N UC virtuelles → 1 UC physique
 - » N compteurs ordinaux logiques → 1 compteur ordinal physique
 - » ≠ d'un « vrai parallélisme » au niveau matériel (multiprocesseurs)

Processus et threads



● **Notion de Processus**

– Multiprogrammation

- Exemple d'exécution de plusieurs tâches sur un mono-processeur
 - » L'utilisateur lance 2 programmes A ainsi que 2 autres programmes : A1, A2, B et C
 - » A1 et A2 sont donc deux processus (différents) et possèdent le même code source / binaire
- » Notes : l'ordre d'exécution des processus est décidé par l'**ordonnanceur** (scheduler) - c.f. cours n°5 (XXX)

Processus et threads



■ Notion de Processus

– Rappel : Multi-tâches \neq multi-utilisateurs

- Multi-tâches : 1 processeur est partagé entre plusieurs tâches
- Multi-utilisateurs : 1 machine est partagée entre plusieurs utilisateurs
- Système mono-tâche et mono-utilisateur
 - » Aucun partage du processeur
- Système multi-tâches et mono-utilisateur :
 - » Partage du processeur entre différentes tâches de l'utilisateur
- Système multi-tâches et multi-utilisateurs :
 - » Partage du processeur entre les différentes tâches de tous les utilisateurs

Processus et threads

● 2 modes d'exécution

- Processus utilisateurs : exécution en **mode utilisateur**
 - Les applications doivent toujours s'exécuter dans ce mode
 - Un utilisateur particulier : super-utilisateur ou superviseur (root)
 - Certaines instructions sont interdites ou limitées (même en root !)
 - » **Appel système** : demande un service au noyau
- Processus systèmes : exécution en **mode noyau (kernel)**
- Mode noyau \neq mode super-utilisateur (superviseur)



Processus et threads



● Notion de Processus

– Contexte ou vecteur d'état

- Regroupe les informations essentielles pour son exécution
- Définit à un point interruptible
 - » instant durant lequel le contexte contient des informations stables

– Principaux états d'un processus

- Bloqué (Blocked)
 - » Ressources nécessaires pour exécuter l'instruction suivante non disponibles
- Activable (Runnable or Waiting)
 - » Ressources nécessaires disponibles
 - » En attente de l'Unité Centrale (i.e. CPU)
- Actif (Run or Running)
 - » Ressources nécessaires disponibles
 - » S'exécute sur l'Unité Centrale (i.e. CPU)

Processus et threads

● Transitions d'état

– Activable → Actif

- Modification du vecteur d'état par un processus particulier
 - » **Ordonnanceur** (scheduler)
- S'accompagne d'une commutation de contexte

– Actif → Activable

- Interruption plus prioritaire que le niveau en cours
 - » Ordonnanceur (fin d'un quantum de temps par exemple)
- S'accompagne d'une commutation de contexte



Processus et threads



• Transitions d'état

– Actif → Bloqué

- Provoqué par le processus lui-même
 - » Blocage matériel
 - » Demande de ressource non disponible
 - » Non prévu par le programmeur
 - » Blocage logiciel (intrinsèque)
 - » Prévu par le programmeur (c.f. cours n°XXX)
- S'accompagne d'une commutation de contexte

– Bloqué → Activable

- Sur l'arrivée d'un événement
 - » Signalé par une **interruption**
- Commutation de contexte pas automatique

Processus et threads



• **Les interruptions**

- Arrivé d'un évènement = arrivé d'une interruption
 - « signal » sous UNIX (c.f. cours n° XX)
- Prise en compte d'un évènement = traitement de l'interruption
 - Gestionnaire d'interruption
 - Pas automatiquement traitée à son arrivée !
 - » Point interruptible, masquage, priorité...
- Rôle très important
 - Unique moyen pour le SE de reprendre la main (de manière forcée) sur le CPU

Processus et threads

• Gestion des processus

- Ensemble de « Bloc de contrôle de processus » (Process Control Block ou **PCB**)
 - Structure de données dépendant du SE
 - » Tables des processus
 - Information accessible uniquement par le noyau → appels systèmes
 - » Identité
 - » PID, PPID, droits
 - » Exécution
 - » Etat, contexte, priorité...
 - » Ressources
 - » Mémoires utilisée / utilisable, Temps, Fichier ouverts...



Processus et threads



● Création de processus

- Un processus peut créer un autre processus, qui à leur tour, peuvent en créer d'autres
- Sous Windows
 - Pas de hiérarchie explicite, tous les processus sont « égaux »
- Sous UNIX
 - Parent, enfants, descendants → groupe
 - Identifié par un numéro (entier) unique
 - » Process ID (**PID**) & Parent PID (**PPID**)
 - Arbre de processus
 - » Démarrage : processus « init » (racine de l'arborescence) PID=1

Processus et threads



● **Lancement d'un nouveau programme**

- **Seul moyen de créer un processus : `fork()`**
 - Le processus appelant est dupliqué = code identique au père
 - Renvoie 0 si c'est le fils
 - Renvoie le pid du fils si c'est le père
- **Possibilité de remplacer l'espace mémoire du processus appelant par le code et les données d'une nouvelle application**
 - Appel système de la famille « exec »
 - » Succès : processus appelant entièrement remplacé
 - » Echec : retour au processus appelant

Processus et threads



• Fonctions de la famille « exec »

- Un seul véritable appel système : `execve ()`
- Les variantes sont implémentées à partir de `execve()`
 - `execl()`, `execle()`, `execlp()`, `execv()`, `execvp()`
 - » Suffixe « l » : liste d'arguments
 - » Suffixe « v » : tableau (à la manière de `argv[]`)
 - » Termine par « e » : transmet l'environnement (tableau `envp[]`)
 - » Termine par « p » : recherche de l'application à lancer à partir de la variable `PATH`, pour les autres il faut envoyer le chemin d'accès

Processus et threads

● Informations d'identité

- Récupérer le **PID** d'un processus
 - D'un terminal : « echo \$\$ »
 - Par les commande « ps » ou « pstree »
 - Appel système « pid_t getpid() »
- Récupérer le **PPID** d'un processus
 - Appel système « pid_t getppid() »



Processus et threads



● **Terminaison d'un processus**

- « **void exit (int status)** » : **retourne une valeur entière**
 - Contrôle la bonne exécution du processus lorsqu'il se termine
 - » 0 en cas de fin normale sinon indique un comportement anormal
 - ➔ « **return 0** » de la fonction main
 - » Des macro-constantes sont définies
 - » **EXIT_SUCCESS** & **EXIT_FAILURE**
 - Cette valeur doit être récupérée par le processus parent
 - Pas de libération des ressources tant que ce n'est pas fait...
 - » Etat particulier appelé « **Zombie** » sous UNIX

Processus et threads

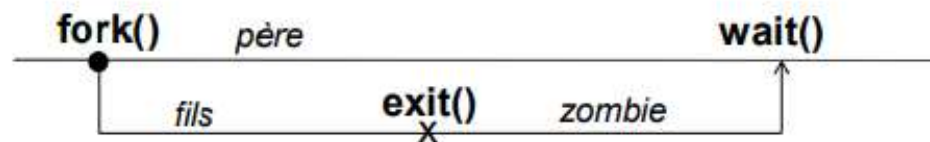
■ Attente de fin d'un processus

- « `pid_t wait(int* status)` »
 - Attendre la fin d'un processus fils
 - » Si le processus n'a aucun fils : retourne -1
 - » Si plusieurs : impossible d'en sélectionner un en particulier
 - » Attendre la fin de l'ensemble des fils → boucle de « `wait()` »
 - « `status` » permet de récupérer la valeur retournée par le fils
- « `pid_t waitpid(pid_t pid,int* stat_infos, int options)` »
 - Attendre la fin d'un processus fils en particulier

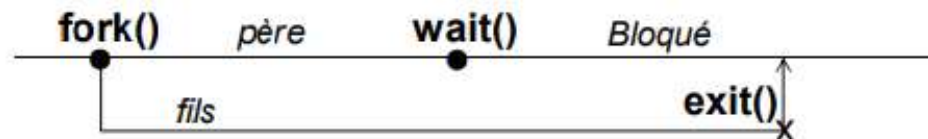
Processus et threads

■ Synthèse création / terminaison de processus

– Cas n°1



– Cas n°2



Processus et threads

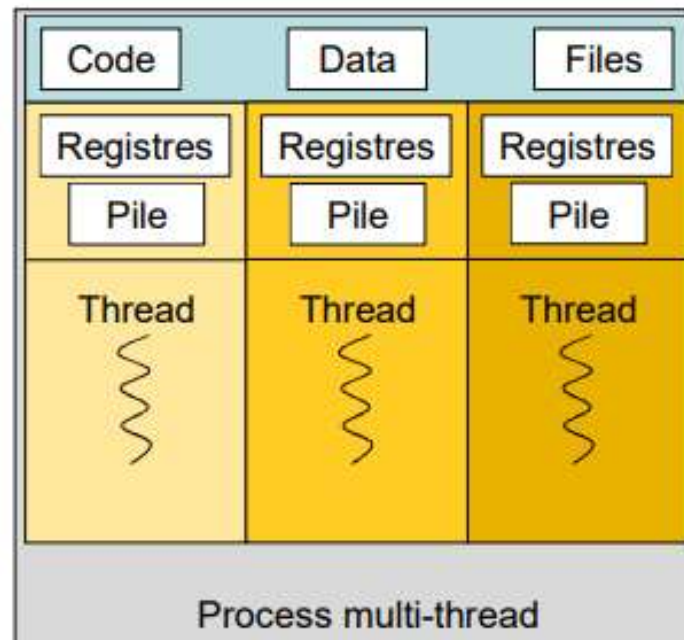
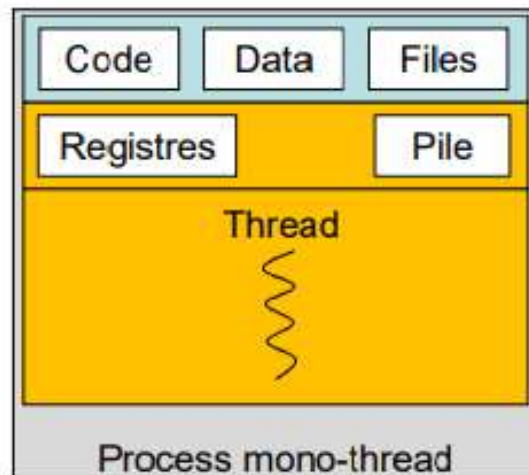


● Notion de Thread

- Création d'un processus est couteuse en temps
 - Contexte « lourd »
 - » PID, répertoire de travail, umask, descripteurs, propriétaires, registres, priorité d'ordonnancement, signaux, pile d'exécution...
- Aussi appelé « Processus léger »
- Espace mémoire partagé
 - Simplification du contexte d'exécution
 - Réduit à l'essentiel pour son exécution
 - » Registres, priorité d'ordonnancement, masque des signaux, pile d'exécution
 - Des données peuvent être partagées facilement
 - ➔ nécessite des synchronisations !

Processus et threads

● Processus multithread



Processus et threads



● **Création d'un Thread**

- `int pthread_create (pthread_t* idptr, pthread_attr_t attr, void*(*fonc)(void*), void*arg)`
 - `Idptr` = pointeur sur la zone où sera retournée l'identité du thread
 - `Attr` = attribut du thread : `pthread_attr_default` ou 0
 - `Fonc` = pointeur sur la fonction exécutée par le thread
 - `Arg` = pointeur sur les arguments passés au thread
- Contrairement à la création de processus, le code n'est pas dupliqué
 - On associe le thread à une fonction définie dans le programme

Processus et threads



• **Terminaison d'un Thread**

– `Int pthread_exit (void* status)`

- Status = pointeur sur le résultat retourné par le thread



• **Attente de la terminaison d'un thread**

– `Int pthread_join (pthread_t id, void* status)`

- Id = identité du thread attendu
- Status = pointeur sur le code de retour du thread attendu
 - » -1 si thread ne se termine pas correctement



Processus et threads



• Les processus et les thread peuvent...



- ... être indépendants
 - Cas idéal mais rarement rencontré !
- ... devoir se synchroniser
 - Sujet du cours n°2
- ... devoir communiquer de l'information
 - Sujet du cours n°3

