ARCHITECTURE MATERIELLE

NSI | 02

SYSTEMES D' EXPLOITATION

[Marion SZPIEG]

Savoir ce qu'est un système d'exploitation, ce qu'est le standard POSIX, et ce qu'est un système d'exploitation libre ou non.

Savoir utiliser/naviguer dans l'arborescence de fichier sous Linux

Savoir les commandes de base dans un terminal sous Linux

Savoir comment fonctionne un processus lorsqu'une commande est lancée sous Linux

Maîtriser le système de droit et permission des fichiers sous Linux

1. Qu’est-ce qu’un système d’exploitation ?

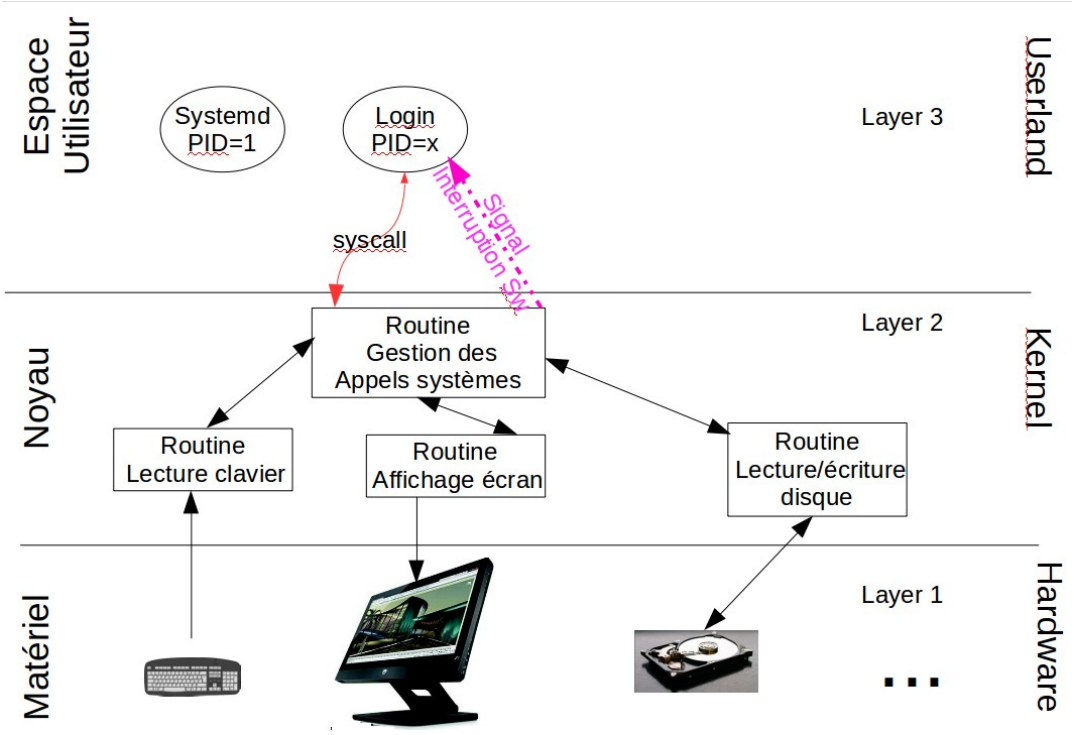
Vidéo sur l’histoire des systèmes d’exploitation

1.1. Schéma général

Un système d’exploitation (*Operating System*), est chargé d’assurer la liaison entre les ressources matérielles, l’utilisateur et les applications Attention parler de processus !! (traitement de texte, jeu vidéo, …).

Ainsi lorsqu’un programme désire accéder à une ressource matérielle, il ne lui est pas nécessaire d’envoyer des informations spécifiques au périphérique, il lui suffit d’envoyer les informations au système d’exploitation, qui se charge de les transmettre au périphérique concerné via son pilote(driver). Par exemple si vous avez un fichier texte et que vous voulez l’enregistrer sur le disque dur, c’est le système d’exploitation qui s’en occupe. En l’absence de pilotes il faudrait que chaque programme reconnaisse et prenne en compte la communication avec chaque type de périphérique … en langage machine !

Attention : le principal élément du système d’exploitation qui facilite la communication entre les programmes et le matériel s’appelle le noyau. Il contient des programmes exécutables appelés « routines ».



1.2. Les différentes fonctionnalités d’un système d’exploitation

Voici une liste non exhaustive des différents rôles du système d’exploitation :

- **Gestion du processeur** : le système d’exploitation est chargé de gérer le partage du temps de calcul du processeur entre les différents programmes grâce à un algorithme d’ordonnancement.
- **Gestion de la mémoire vive** : le système d’exploitation est chargé de gérer l’espace mémoire alloué à chaque processus. En cas d’insuffisance de mémoire physique dans la RAM, le système d’exploitation peut utiliser une zone mémoire sur le disque dur, appelée «mémoire virtuelle» sous Windows ou « partition de SWAP » sous Linux. La mémoire virtuelle permet de faire fonctionner des applications nécessitant plus de mémoire qu’il n’y en a dans la RAM. En contrepartie cette mémoire est beaucoup plus lente.
- **Gestion des entrées/sorties** : le système d’exploitation permet d’unifier et de contrôler l’accès des programmes aux ressources matérielles par l’intermédiaire des pilotes (appelés également gestionnaires de périphériques ou gestionnaires d’entrée/sortie).

- **Gestion de l'exécution des applications** : le système d'exploitation est chargé de la bonne exécution des applications en leur affectant les ressources nécessaires à leur bon fonctionnement. Il se permet à ce titre de «tuer» une application ne répondant plus correctement.
- **Gestion des droits** : le système d'exploitation est chargé de la sécurité liée à l'exécution des programmes en garantissant que les ressources ne sont utilisées que par les programmes et utilisateurs possédant les droits adéquats.
- **Gestion des fichiers** : le système d'exploitation gère la lecture et l'écriture dans le système de fichiers et les droits d'accès aux fichiers par les utilisateurs et les applications.
- **Gestion des informations** : le système d'exploitation fournit un certain nombre d'indicateurs permettant de diagnostiquer le bon fonctionnement de la machine (exemple taux de charge du CPU, quantité de mémoire RAM utilisée).

1.3. UNIX et le standard POSIX

Vidéo sur l'histoire d'UNIX / Lien vers l'histoire de MULTICS (en anglais)

En 1964, le MIT, les Laboratoires Bell et General Electrics (alors fabricant de transistors) lancent un projet fou qu'ils appellent MAC (Mathematic And Computer). A cette époque, les systèmes d'exploitation n'existaient pas. Le projet MAC a pour ambition de construire un système multi-utilisateur et multi-tache qui faciliterait l'utilisation du matériel informatique par les utilisateurs (composés essentiellement de scientifiques à cette époque). Ce système a été baptisé MULTICS (Multiplexed Information and Computing Service). General Electrics devait s'occuper de construire la machine physique, et les Laboratoires Bell devaient s'occuper du code.

Le projet trop ambitieux tombe à l'eau dès la fin des année 60. Deux ingénieurs des laboratoires Bell labs (Ken Thompson et Dennis Ritchie) qui ont travaillé sur ce projet se lancent en solitaire (avec quelques autres collègues) sur leur propre projet bien moins ambitieux : créer un système conversationnel plus simple pour un seul utilisateur qu'ils baptisent UNICS (Uniplexed Information and Computing Service). Ils commencent par écrire ce système en langage assembleur. Ils progressent tellement vite qu'ils réussissent à créer un système multi-utilisateur, codé dans un second temps dans un langage qu'ils ont inventé (le langage B) qu'ils ont tellement transformé qu'il l'ont rebaptisé le langage C. Ce système final multi-utilisateur est rebaptisé à son tour Unics, puis UNIX.

Ritchie et Thompson recevront le prix Turing en 1983 pour ces travaux.

En 1988, l'Institute of Electronical and Electronics Engineer (IEEE) définit la famille de norme POSIX afin de standardiser des interfaces de programmation et des logiciels destinés à fonctionner sur le système d'exploitation UNIX (et ses variantes).

1.4. Systèmes d'exploitation libres et propriétaires

Il existe 2 types de systèmes d'exploitation : les systèmes libres et les systèmes propriétaires.

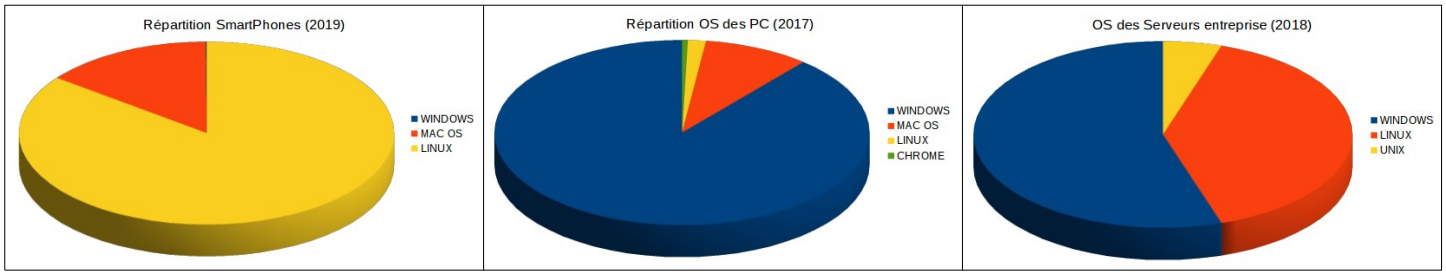
Un **système d'exploitation libre** est un système d'exploitation composé uniquement de logiciels et programmes libres. Un logiciel libre est un programme qui accorde 4 libertés essentielles à l'utilisateur : **liberté d'exécuter** le programme pour n'importe quel usage, **liberté de le modifier** (donc on peut télécharger le code source), **liberté de redistribuer** des copies du programme et **liberté de distribuer** des versions modifiées.

Un **système d'exploitation propriétaire** est un système d'exploitation qui, au contraire, est composé au moins en partie de logiciels et programmes non libres.

Voici les principaux :

SE propriétaires	SE libres
Windows Unix MacOS	Linux et ses dérivés : • debian • fedora, redhat • androïd

Standard POSIX



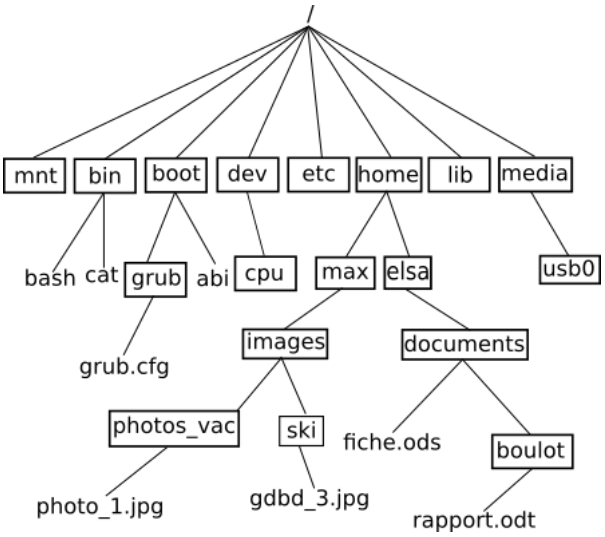
2. Linux

Naissance de Linux

2.1. Arborescence du système de fichier(1ère feuille d'exercices)

Les répertoires fondamentaux :

. : répertoire dans lequel on est, appelé « répertoire courant »
.. : répertoire père du répertoire dans lequel on est
/ : racine de l'arborescence de fichier
bin : contient des exécutables couramment utilisés
dev : contient les fichiers permettant d'accéder aux périphériques
etc : contient les fichiers de configuration des différents programmes
home : répertoire qui contient tous les répertoires des utilisateurs
lib : contient les bibliothèques de fonctions que les programmes peuvent utiliser
mnt et media : permet de monter les systèmes de fichiers des périphériques amovibles (clé USB par exemple)
root : contient les fichiers de l'administrateur de l'OS



Pour désigner le chemin d'accès à un répertoire ou à un fichier, il y a 2 façons possibles :

- par le chemin absolu : c'est le chemin d'accès au fichier ou au répertoire depuis la racine
- par le chemin relatif : c'est le chemin d'accès au fichier ou au répertoire depuis le répertoire où l'on se trouve à l'instant t.

Exemples :

1. Quel est le chemin absolu pour accéder au fichier fiche.ods ?

2. Quel est le chemin relatif pour accéder au fiche.ods depuis le répertoire elsa ?

3. Quel est le chemin relatif pour accéder au fiche.ods depuis le répertoire max ?

4. Quel est le chemin relatif pour accéder au fiche.ods depuis le répertoire images?

Remarques :

- le chemin absolu du répertoire « Home » de l'utilisateur Dupont du SE peut s'écrire de 3 façons différentes : /home/dupont ou ~ ou $HOME.
- ainsi, un chemin absolu commencera toujours pas « / » ou « ~ » ou « $HOME ».
- dans un chemin relatif, on part automatiquement du répertoire courant. Ainsi, pour les deux derniers exemples, on peut mettre ./ devant, cela ne change rien. Le caractère « . » indique le répertoire courant.

2.2. Quelques commandes de base (1ère et 2e feuille d'exercices)

Sous Linux, on peut utiliser les système d'exploitation grâce à l'interface graphique (utilisation de la souris obligatoire) OU par la console (ou terminal) qui contient un processus appelé « Interpréteur de commandes» ou « Shell ». Nous allons voir différentes commandes à connaître et maîtriser pour pouvoir dialoguer avec le shell.

Une commande est toujours de la même forme :

commande	option(s) (non obligatoire)	opérandes (quand il y en a)

Ces 3 éléments sont toujours séparés par un **espace**. Sous Linux il faut généralement les taper en minuscules.

Voici les commandes qu'on utilisera dans la 1ère feuille d'exercices (par ordre d'apparition) :

uname	affiche les caractéristiques du système d'exploitation
man	permet d'obtenir la documentation d'une commande. Par exemple, "man ls" vous donnera la documentation de la commande "ls"
ls	liste le contenu du répertoire courant si on ajoute -s : permet de voir plus en détail le contenu du répertoire courant si on rajoute -a : permet de voir également les fichiers cachés
tree	permet de voir l'arborescence des fichiers et répertoires en format texte
pwd	permet de connaître le répertoire courant (Path Working Directory)
mkdir	permet de créer un répertoire dans le répertoire courant. La commande est de la forme "mkdir nom_du_repertoire" =>éviter les accents et les espaces dans les noms de répertoire ou de fichier !
cd	permet de changer le répertoire courant. Il suffit d'indiquer le chemin (relatif ou absolu) qui permet d'atteindre le nouveau répertoire
cp	permet de copier un fichier. La commande est de la forme "cp /repertoire_source/nom_fichier_a_copier /repertoire_destination/nom_fichier" Remarque 1 : le nom du fichier "destination" n'est pas obligatoirement le même que le nom du fichier "source". Dans ce cas, il faut préciser le nouveau nom à la fin du chemin du répertoire destinataire (on peut avoir "cp fic.txt info/fiche.txt") Remarque 2 : on peut copier un fichier dans le répertoire courant en changeant de nom (exemple : "cp fic.txt fiche.txt")
rm	permet de supprimer un fichier ou un répertoire. La commande est de la forme "rm nom_du_repertoire_ou_nom_du_fichier" si on ajoute -r après rm, cela permet de supprimer tout un répertoire non vide (y compris ses sous répertoires)
history	Permet de voir l'historique de toutes les commandes tapées précédemment

Voici les commandes qu'on utilisera dans la 2e feuille d'exercices(par ordre d'apparition) :

echo	affiche un texte ou le contenu d'une variable à l'écran
cat	permet d'afficher dans la console le contenu d'un fichier au format texte. La commande est de la forme "cat nom_du_fichier_a_afficher"
find	cherche un fichier ou un répertoire dans le système de fichiers
chmod	permet de modifier les droits d'un fichier ou d'un répertoire (voir partie 2.4. du cours)
mv	permet de déplacer un fichier dans un autre répertoire. Par exemple un "mv toto.txt titi" déplacera le fichier "toto.txt" dans le répertoire "titi" (attention on parle bien de déplacement pas de copie). si en plus le nom du chemin cible se termine par un nom de fichier, le fichier déplacé est en plus renommé

Voici quelques commandes en plus

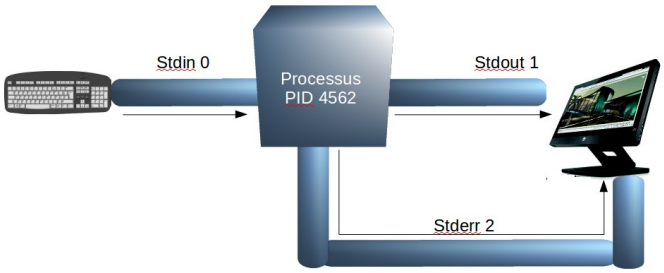
touch	permet de créer un fichier vide. La commande est de la forme "touch nom_du_fichier_a_creer"
who	permet de savoir quels sont les autres utilisateurs connectés au même serveur
whoami	permet de savoir quel est notre nom d'utilisateur avec lequel on est connecté sur le serveur

Remarque : quand on ouvre une console, on se trouve par défaut dans notre propre répertoire home.

2.3. Flux entrée/sortie, redirection (2e feuille d'exercices)

2.3.1. Qu'est-ce que c'est ?

Lorsqu'on exécute une commande dans une console Linux, un processus se lance dans le « userland » du système d'exploitation. Par défaut, ce processus possède 3 canaux (flux) qui le relient aux périphériques d'entrée-sortie :

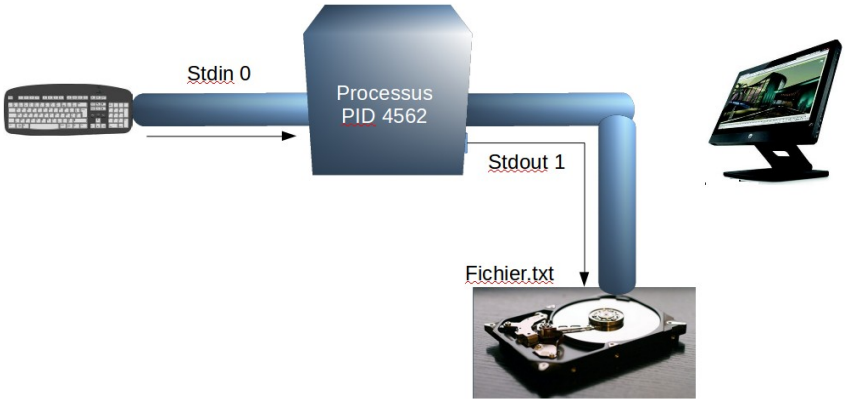


- Un canal d'entrée standard (stdin - n°0) : qui permet au processus de récupérer généralement ce qui a été entré au clavier.
- Un canal de sortie standard (stdout - n°1) : qui est par défaut connecté à l'écran et qui permet au processus d'écrire des messages
- Un canal de sortie d'erreur (stderr - n°2) : qui sert au processus à indiquer un message d'erreur en l'écrivant en général sur l'écran

2.3.2. Redirection

Si on ne veut pas que l'entrée (ou la sortie) standard soit connectée sur le périphérique par défaut, on peut la rediriger :

- le signe « < » sert à rediriger l'entrée standard et permet de lire le contenu d'un fichier plutôt que les données provenant du clavier
- le signe « > » sert à rediriger la sortie standard (ou erreur en rajoutant un « 2 ») dans un fichier (texte par exemple). Si le fichier existe déjà, il est écrasé, et s'il n'existe pas, il est créé.
- le signe « >> » sert à rediriger la sortie standard (ou erreur en rajoutant un « 2 ») en mettant les nouvelles données à la fin du fichier cible si celui-ci existe déjà et contient des données. Si le fichier cible n'existe pas il est créé automatiquement.



2.4. Droit et permission des fichiers (2e feuille d'exercices)

Un utilisateur peut interagir avec un fichier de 3 façons différentes :

- en le lisant (r de read)
- en modifiant son contenu (w de write)
- en exécutant le code qui est contenu à l'intérieur (x de exécute)

Pour un fichier, on peut donner ou non les droits rwx à 4 types d'utilisateurs différents :

- au propriétaire du fichier (u de user). Par défaut, celui qui crée un fichier est le propriétaire.
- au groupe attaché au fichier (g de groupe). Par défaut, c'est le groupe principal auquel appartient le créateur du fichier.
- tous les autres utilisateurs sauf les 2 précédents (o de other)
- tout le monde (a de all)

Pour visualiser les droits d'accès d'un fichier, dans la console taper la commande ls -l. La liste de tous les fichiers (répertoires etc.) apparaît, avec en premier le détails des droits d'accès.

=> on peut juste faire ls -l un_fichier ou ls -l un_chemin

Exemple : après avoir tapé la commande « ls -l » on obtient , la ligne suivante :

-rwxr-xr-- 1 polo equipe1 06 mai 15 12:05 monfichier.txt

- 1er caractère : le premier caractère sert à donner la nature de l'entité : - pour un fichier ; d pour un répertoire ...
- 3 caractères suivants : 3 types de droits donnés ou non au propriétaire du fichier (r ou – puis w ou – puis x ou –) : ici le propriétaire a tous les droits.
- 3 caractères suivants : 3 types de droits donnés ou non au groupe attaché au fichier (même chose) : ici le groupe a le droit de lecture et celui d'exécution
- 3 caractères suivants : 3 types de droits donnés aux « autres » (même chose) : ici ils ont seulement le droit de lecture.
- puis un compteur qui sort du cadre du cours (compteur de lien dur pour les curieux)
- puis le nom propriétaire du document : ici polo
- puis le nom groupe rattaché au document : ici equipe1
- puis la date et l'heure de la dernière modification du document
- puis le nom du document

En pratique : comment changer les droits d'un fichier ?

- Se placer dans le répertoire contenant le fichier.
- Utiliser la commande chmod avec les options suivantes:
  1. l'entité pour laquelle on veut changer les droits : u, g, o ou a
  2. + le droit supplémentaire qu'on veut rajouter (r, w ou x)

  OU
  3. - le droit qu'on veut retirer (r, w ou x)
  4. le nom du fichier dont vous voulez changer les droits

Voici la syntaxe de cette commande :

```
chmod [u g o a] [+ -] [r w x] nom_du_fichier
```

Remarque : seul le propriétaire peut changer les droits de son fichier (et l'utilisateur root , appelé aussi l'administrateur du système d'exploitation)