Systèmes d'exploitation

Thème architectures matérielles et systèmes d'exploitation

Première NSI, Lycée du Parc

Table des matières

Crédits

Ce cours est inspiré du chapitre 24 du manuel NSI de la collection Tortue chez Ellipse, auteurs : Ballabonski, Conchon, Filliatre, N'Guyen. J'ai également consulté le cours de Guillaume Connan, le livre Splendeurs et servitudes des Systèmes d'exploitation Histoire, fonctionnement, enjeux de Laurent Bloch, le livre La ligne de commande par l'exemple de Vincent Fourmond et le livre Parlez-vous Shell ? de Thomas Hugel.

1 Chronologie des systèmes d'exploitation

- 1950 premiers calculateurs (ENIAC, EDSAC) branchements manuels pas de système d'exploitation
- 1956 GM-NAA I/O premier système d'exploitation sur IBM 704
- 1967 MULTICS développé aux Bell Labs premier système d'exploitation à temps partagé
- 1970 UNIX développé aux Bell Labs premier système d'exploitatio multitâches et multiutilisateurs, écrit en langage C
- 1978 BSD variante d'UNIX dévéloppée à l'université de Berkeley, ancêtre de MacOS
- 1980 les ordinateurs personnels d'IBM utilisent le système d'exploitation MS-DOS de Microsoft
- 1990 Microsoft développé Windows un système d'exploitation avec interface graphique inspiré du système disponible sur les machines Apple
- 1991 système d'exploitation libre Linux, noyau développé par Linus Torvalds, outils logiciels du projet GNU, distribué sous licence libre
- 2001 système d'exploitation MacOS sur les machines Apple, dérivé de BSD

2007 Apple développe un version de MacOS pour téléphones portables 2008 Google développe le système d'exploitation Android pour téléphones portables, basé sur le noyau libre Linux avec ajout de logiciels propriétaires

2 Définition d'un système d'exploitation

Un *système d'exploitation* est un logiciel, ou ensemble de programmes, qui sert d'interface entre les programmes exécuté par l'utilisateur et les ressources matérielles d'un ordinateur.

Il est à la fois :

- une *machine virtuelle* qui présente une interface simplifiée d'accès aux ressources (processeur, mémoire, périphériques d'entrée/sortie, réseau ...) pour les autres programmes et pour l'utilisateur
- un *chef d'orchestre* et un *administrateur* :
  - c'est le premier programme exécuté au démarrage de l'ordinateur
  - il gère l'accès concurrent aux ressources par les différents programmes (ordonnancement de l'utilisation du processeur par les programmes en cours d'exécution ou processus, sécurisation de la mémoire) ou utilisateurs (droits d'accès du système de fichiers)
- Sitographie :
  - Page Wikipedia
  - Présentation en video

3 Interface utilisateur d'un système d'exploitation : le shell

Une interface entre l'utilisateur et le système d'exploitation s'appelle un shell ou *interpréteur de commandes.

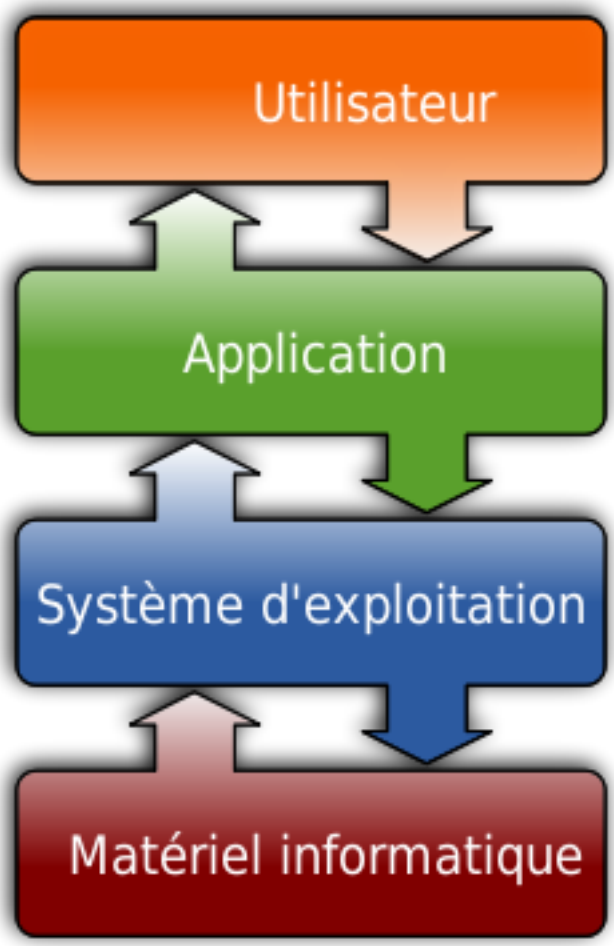

Figure 1: Illustration du rôle d'un système d'exploitation

Le rôle d'un *shell* est de prendre une entrée de l'utilisateur, de la traduire en instructions compréhensibles par le système d'exploitation et de renvoyer la réponse du système à l'utilisateur.

Il existe deux grandes catégories de *shell* :

- les *interfaces textuelles* comme bash, le plus commun sur les systèmes de la famille UNIX.
- les *interfaces graphiques* qu'on retrouve dans les systèmes d'exploitation grand public tels que Windows

Nous utiliserons bash dans ce TP et le suivant. La syntaxe d'une commande bash est simple : le nom de la commande peut être suivi d'options facultatives introduites par un tiret et d'arguments : `nom_commande -option1 -option2 argument1 argument2`.

Par exemple, la commande ci-dessous demande d'utiliser la commande `ls` pour afficher les informations détaillées (option `_l`) sur le fichier `exemple`:

```
junier@fredportable:~/bac$ ls -l exemple
-rw-rw-r-- 1 nobody mail 0 août  16 12:05 exemple
```

Chaque commande est un programme enregistré dans un fichier, par exemple `/usr/bin/ls` pour la commande `ls`:

```
junier@fredportable:~/bac$ which ls
/usr/bin/ls
junier@fredportable:~/bac$ ls -l /usr/bin/ls
-rwxr-xr-x 1 root root 142144 sept.  5  2019 /usr/bin/ls
```

Enfin chaque commande possède trois flux d'entrée / sortie :

- *l'entrée standard* qui est par défaut le clavier (les options et arguments sont traités puis transmis par bash sur l'entrée standard de la commande)
- *la sortie standard* qui est par défaut l'écran
- *la sortie d'erreur* qui est par défaut l'écran également

Il existe des opérateurs `>` et `<` pour rediriger les flux d'entrée / sortie.

Considérons un exemple . La commande `ls` sans argument affiche le contenu du répertoire courant sur la sortie standard et la commande `cat` renvoie son entrée standard sur sa sortie standard :

```
junier@fredportable:~/bac$ ls
entree
junier@fredportable:~/bac$ cat --help
utilisation : cat [OPTION] [FICHIER]...
Concaténer les FICHIERs vers la sortie standard.
.............
junier@fredportable:~/bac$ cat entree
contenu de entree
junier@fredportable:~/bac$ cat entree > sortie
junier@fredportable:~/bac$ ls
entree  sortie
junier@fredportable:~/bac$ cat sortie
contenu de entree
```

Nous fournirons un memento des principales commandes de bash communes à tous la plupart des *shell* des systèmes de la famille UNIX](https://fr.wikipedia.org/wiki/Bourne-Again_shell).

Un memento en ligne : https://juliend.github.io/linux-cheatsheet/

4 Attributs du système d'exploitation : le mode noyau

Les programmes ne peuvent pas accéder directement aux ressources matérielles (mémoire, processeur, périphériques d'entrée/sortie) sinon il y aurait des conflits : par exemple deux programmes pourraient écrire dans la même zone mémoire.

Le système d'exploitation possède un mode d'exécution privilégiée : le *mode noyau*, qui lui donne un accès unique et total aux ressources. Les autres programmes s'exécutent en *mode normal*. Pour accéder aux ressources, ils font appel au système d'exploitation à l'aide de primitives qu'on nomme *appels systèmes*. Pour les systèmes d'exploitation dérivés d'UNIX, ces appels systèmes sont normalisés par le standard POSIX pour Portable Operating System Interface.

Le **noyau** désigne l'ensemble des programmes au coeur du système d'exploitation, qui s'exécutent en particulier au démarrage de l'ordinateur. Par exemple, le noyau du système d'exploitation libre Linux est téléchargeable sur https://www.kernel.org/.

5 Attributs du système d'exploitation : le contrôle de l'activité des programmes

Un programme ne peut être exécuté que si le système d'exploitation l'autorise. L'exécution d'un programme peut être interrompue par le système d'exploitation, pour libérer une ressource ou en cas d'erreur, on parle d'interruption système.

Un programme en cours d'exécution s'appelle un *processus* et le système d'exploitation arbitre le partage de temps de calcul du processeur par les processus concurrents. Il jour un rôle *d'ordonnanceur* pour éviter par exemple qu'un processus s'accapare le processeur.

Le système d'exploitation maintient une base de temps unique pour tous les processus et fournit des services de gestion du temps : estampillage, chronologie, attente, réveil.

La commande de *shell* `ps` permet d'afficher la liste des processus : l'option `U` permet de sélectionner l'utilisateur qui a lancé les processus. S'il s'agit d'un processus qu'on a lancé ou si on est superutilisateur, peut alors "tuer" le processus avec la commande `kill` suivie du numéro du processus.

```
junier@fredportable:~$ ps U
PID TTY      STAT   TIME COMMAND
13845 pts/5    S+     0:00 python3
13900 pts/4    R+     0:00 ps U junier
junier@fredportable:~$ kill 13845
```

6 Attributs du système d'exploitation : le contrôle d'accès aux ressources

Le système d'exploitation est le seul à pouvoir contrôler l'accès aux ressources : processeur, mémoire, périphériques d'entrée/sortie.

La gestion et la sécurisation de la mémoire est une tâche importante, avec un système de *mémoire virtuelle* qui donne à chaque programme l'illusion d'accéder à toute la mémoire.

Des algorithmes permettent d'éviter les situations d'interblocage ou étreinte fatale.

Un exemple concret d'interblocage peut se produire lorsque deux processus essayent d'acquérir deux ressources dans un ordre différent. Par exemple avec deux ressources (M1 et M2), deux processus (P1 et P2) et la séquence d'opération suivante :

```
P1 acquiert M1.
P2 acquiert M2.
P1 attend pour acquérir M2 (qui est détenu par P2).
P2 attend pour acquérir M1 (qui est détenu par P1).
```

Dans cette situation, les deux processus légers sont définitivement bloqués.

La commande *shell* `free` permet d'afficher l'utilisation de la mémoire, l'option `-m` convertit les mesures en megaoctets (Mo). Dans l'exemple ci-dessous, la mémoire vive disponible est de 3856 Mo, 618 Mo sont utilisés pour les buffers/cache et 254 sont libres soit un total de 3856 - 618 - 254 = 2983 Mo utilisés. La seconde ligne concerne un espace du disque dur, nommé swap, utilisé par le système pour étendre la mémoire vive en mémoire virtuelle.

```
junier@fredportable:~/bac$ free -m
               total        utilisé      libre     partagé tamp/cache   disponible
Mem:            3856         2983         254          302         618         334
Partition d'échange:          1303         1288           14
```

7 Attributs du système d'exploitation : le contrôle du système de fichiers

7.1 Système de fichiers

Sur les supports de mémoire persistants (disques durs, clefs USB...), les informations sont regroupées par le système d'exploitation dans des *fichiers* qui sont organisés à travers un système de fichiers.

Dans les systèmes d'exploitation qui respectent le standard POSIX, on distingue plusieurs catégories de fichiers :

- *fichiers réguliers* ou *fichiers textes* qui contiennent des suites de caractères et qui sont lisibles par des humains
- *fichiers binaires* qui sont des suites d'octets non lisibles par des humains
- *fichiers exécutables* qui sont des programmes, ils peuvent être des fichiers textes binaires.
- *répertoires* qui sont des listes de fichiers : ils servent de conteneur à fichiers.

Les *répertoires* pouvant contenir d'autres fichiers, un système de fichiers POSIX possède une structure hiérarchique qui est une *arborescence* avec un *répertoire racine* symbolisé par un /.

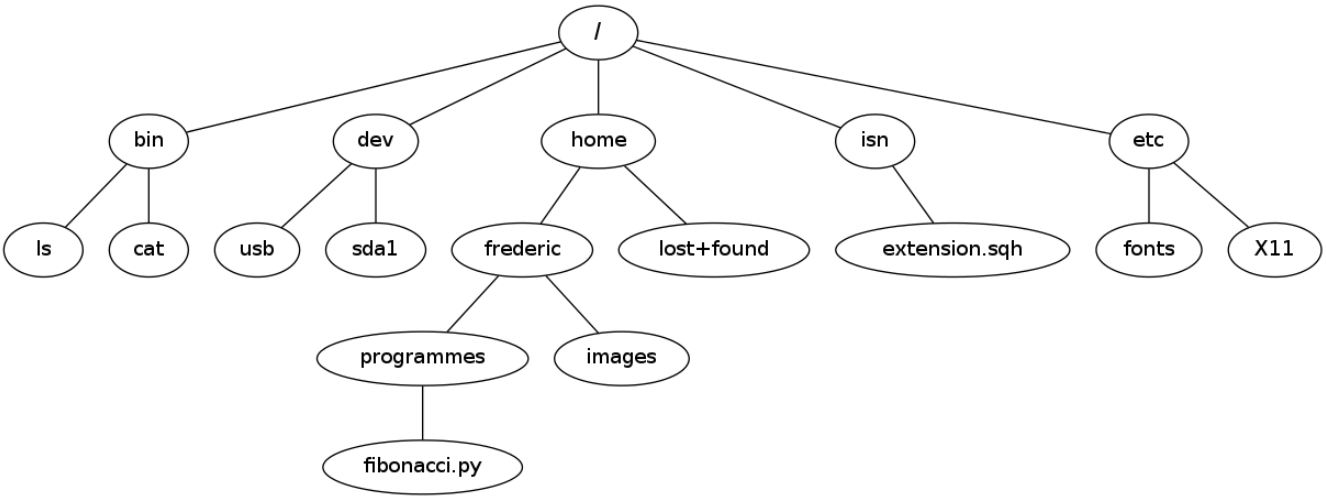

Figure 2: Un exemple d'arborescence

Les fichiers sont repérés par leur *chemin* :

- *chemin absolu* depuis la racine : par exemple si un fichier `exemple.txt` se trouve dans le répertoire `sandbox`, contenu dans le répertoire `maurice`, lui-même contenu dans le répertoire `home`, lui-même dans le répertoire racine, son *chemin absolu* est `/home/maurice/sandbox/exemple.txt`
- ou *chemin relatif* qui est relatif au *répertoire courant* ou *répertoire de travail* où l'utilisateur se trouve. Par exemple, si le répertoire courant est `maurice`, le *chemin relatif* du fichier précédent est `sandbox/exemple.txt`.

7.2 Gestion des droits et permissions

Le système d'exploitation gère les *droits et permissions* sur les fichiers (dont les répertoires) pour les différents *utilisateurs*.

7.2.1 Utilisateurs et permissions

Pour un fichier, on distingue :

- trois types d'utilisateurs :
  - le *propriétaire* (ou *owner*) noté `u`
  - le *groupe principal* (ou *group*) noté `g`
  - un *autre utilisateur* (ou *other*) noté `o`
- trois types de permissions :
  - *lecture* (caractère *r* si attribué ou - sinon)
  - *écriture* (caractère *w* si attribué ou - sinon)
  - *exécution* (caractère *x* si attribué ou - sinon)

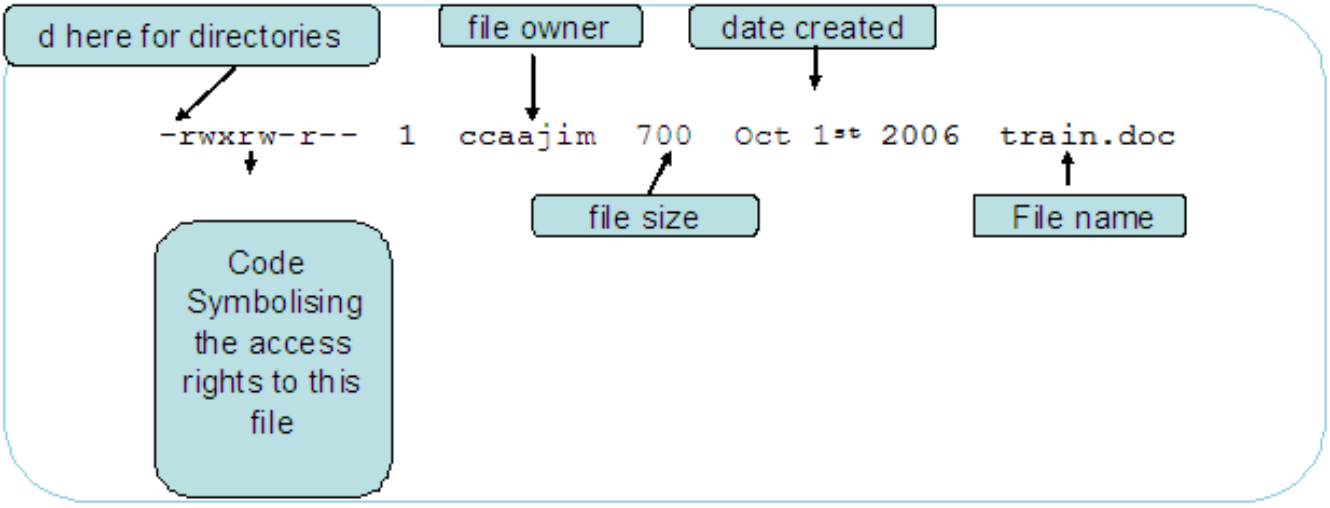

Figure 3: droits unix

Source : Wikimedia Commons, Jimbotyson / CC BY-SA (https://creativecommons.org/licenses/by-sa/3.0)

Pour les répertoires, la signification des permissions est précisée dans ce tableau. Attention, si on a le droit d'écriture sur un répertoire on peut supprimer tous les fichiers qu'il contient même ceux dont on n'est pas propriétaire !

	Type de permission	pour un répertoire
r	lecture	lister le contenu
w	écriture	ajouter, supprimer, renommer des fichiers
x	exécution	entrer dedans

En général le nom d'utilisateur figure dans le prompt de l'interpréteur de commandes / shell, mais la commande `id` permet de l'afficher avec la liste des groupes auxquels on appartient :

```
junier@fredportable:~$ id
uid=1000(junier) gid=1000(junier) groupes=1000(junier),4(adm),8(mail),24(cdrom),27(sudo),30(dip),
junier@fredportable:~$ sudo apt install pandoc
[sudo] Mot de passe de junier :
```

Certaines commandes, comme l'installation de logiciels avec un gestionnaire de paquets comme `apt` ne sont exécutables qu'avec les droits du *superutilisateur* `root`. Si on appartient au groupe `sudo` (groupe des administrateurs), on peut élever ses droits et agir comme `root` en préfixant la commande de `sudo`. et en saisissant son mot de passe utilisateur. Certains systèmes permettent de devenir directement `root` avec la commande `su` en saisissant le mot de passe de `root`. *L'utilisateur `root` a tous les droits sur le système de fichiers.*

7.2.2 Exemples de lectures de permissions

On peut afficher les droits d'un fichier avec la commande `ls -l`, en considérant les 10 premiers caractères :

- le premier précise le type de fichier : `d` pour un répertoire et tiret `-` pour un autre fichier
- les 9 caractères suivants dénotent les attributs pour les trois types de permission avec de gauche à droite le *propriétaire*, le *groupe*, les *autres*.
- Exemple 1 :

```
junier@fredportable:~/sandbox$ ls -l histoire-systeme.txt
-rw-rw-r-- 1 junier junier 1136 août  15 13:56 histoire-systeme.txt
```

```
junier@fredportable:~/sandbox$ ls -l /bin/python3.8
-rwxr-xr-x 1 root root 5453504 juil. 16 16:00 /bin/python3.8
```

Dans cet exemple le fichier binaire `/bin/python3.8` a pour propriétaire `root`, pour groupe `root` et pour droits :

	Lecture	Écriture	Exécution
Propriétaire	oui	oui	oui
Groupe	oui	non	oui
Autres	oui	non	oui

- Exemple 3 : pour afficher les informations détaillées d'un répertoires il faut rajouter l'option `-d` :

```
junier@fredportable:/var$ ls -l -d mail
drwxrwsr-x 2 root mail 4096 avril 23 09:32 mail
```

Dans cet exemple le dossier `/var/mail` a pour propriétaire `root`, pour groupe `mail` et pour droits :

	Lecture	Écriture	Exécution
Propriétaire	oui	oui	oui
Groupe	oui	oui	oui
Autres	oui	non	oui

On peut noter que pour le groupe, le droit d'exécution est positionné à `s`, qui correspond au sgid : tout fichier créé appartient au groupe propriétaire du répertoire.

Attention, si tout membre du groupe `mail` peut écrire dans le répertoire `var`, il peut aussi supprimer un fichier créé par un autre utilisateur même s'il s'agit du superutilisateur `root` :

```
junier@fredportable:/var/mail$ ls -l
total 0
-rw-r--r-- 1 root mail 0 août  16 10:27 mail_important
junier@fredportable:/var/mail$ echo 'Bonjour' > mon_message
junier@fredportable:/var/mail$ ls -l
total 4
-rw-r--r-- 1 root   mail 0 août  16 10:27 mail_important
-rw-rw-r-- 1 junier mail 8 août  16 10:28 mon_message
junier@fredportable:/var/mail$ rm mail_important
rm : supprimer 'mail_important' qui est protégé en écriture et est du type « fichier vide » ? y
junier@fredportable:/var/mail$ ls -l
total 4
-rw-rw-r-- 1 junier mail 8 août  16 10:28 mon_message
```

7.2.3 Modification des droits / permissions

- Seul le propriétaire d'un fichier (ou répertoire) ou l'utilisateur `root` peuvent modifier les permissions d'un fichier (ou répertoire) avec la commande `chmod`.

La syntaxe est la suivante, on a noté entre crochets les options facultatives :

```
chmod [ugo][+-=][rwx] fichier
```

Voici un exemple commenté :

- Lorsqu'on crée un fichier, par défaut il a les droits de lecture et d'écriture pour l'utilisateur et son groupe principal et de lecture pour les autres.

```
junier@fredportable:~/bac$ touch fichier
junier@fredportable:~/bac$ ls -l
total 0
-rw-rw-r-- 1 junier junier 0 août  16 11:05 fichier
junier@fredportable:~/bac$ echo "echo exécution" > fichier
junier@fredportable:~/bac$ cat fichier
echo exécution
```

- On ajoute le droite d'exécution à `fichier` pour son propriétaire et on vérifie que la commande qu'il contient est bien exécutée lorsqu'on écrit `.\fichier` :

```
junier@fredportable:~/bac$ chmod u+x fichier
junier@fredportable:~/bac$ ./fichier
exécution
junier@fredportable:~/bac$ ls -l
total 4
-rwxrw-r-- 1 junier junier 16 août  16 11:06 fichier
```

- On enlève le droit d'écriture au propriétaire et on vérifie qu'on ne peut plus écrire dans 'fichier

```
junier@fredportable:~/bac$ chmod u-w fichier
junier@fredportable:~/bac$ echo "echo ajout" > fichier
bash: fichier: Permission non accordée
junier@fredportable:~/bac$ ls -l
total 4
-r-xrw-r-- 1 junier junier 16 août  16 11:06 fichier
```

- On ajoute le droit d'écriture au propriétaire et aux autres :

```
junier@fredportable:~/bac$ chmod uo+w fichier
junier@fredportable:~/bac$ ls -l
total 4
-rwxrw-rw- 1 junier junier 16 août  16 11:06 fichier
```

- On enlève le droit de lecture au propriétaire, à son groupe et aux autres :

```
junier@fredportable:~/bac$ chmod ugo-r fichier
junier@fredportable:~/bac$ ls -l
total 4
--wx-w--w- 1 junier junier 16 août  16 11:06 fichier
```

- Pour remettre ce droit de lecture à tous, on peut utiliser le raccourci `a` au lieu de `ugo` :

```
junier@fredportable:~/bac$ chmod a+r fichier
junier@fredportable:~/bac$ ls -l
```

```
total 4
-rwxrw-rw- 1 junier junier 16 août  16 11:06 fichier
```

– On positionne tous les droits à `rwx` sur `fichier` pour tous les utilisateurs. On aurait pu écrire `chmod a=rwx fichier`.

```
junier@fredportable:~/bac$ chmod ugo=rwx fichier
junier@fredportable:~/bac$ ls -l
total 8
-rwxrwxrwx 1 junier junier   16 août  16 11:06 fichier
```

– On crée un répertoire nommé `repertoire` avec deux fichiers. On peut noter qu'un répertoire est toujours créé par défaut avec le droit d'exécution pour tous, car celui-ci permet (ou non) de traverser le répertoire.

```
junier@fredportable:~/bac$ mkdir repertoire
junier@fredportable:~/bac$ ls -l -d repertoire/
drwxrwxr-x 2 junier junier 4096 août  16 11:10 repertoire/
junier@fredportable:~/bac$ cd repertoire/
junier@fredportable:~/bac/repertoire$ touch fichier2
junier@fredportable:~/bac/repertoire$ touch fichier3
junier@fredportable:~/bac/repertoire$ ls -l
total 0
-rw-rw-r-- 1 junier junier 0 août  16 11:10 fichier2
-rw-rw-r-- 1 junier junier 0 août  16 11:12 fichier3
```

– On modifie récursivement les droits sur le répertoire `repertoire` et tous ses fichiers avec l'option `-R` dans la commande `chmod -R ugo=rx repertoire`. On peut vérifier qu'on ne peut plus supprimer les fichiers contenus dans `repertoire`.

```
junier@fredportable:~/bac$ chmod -R ugo=rx repertoire
junier@fredportable:~/bac$ ls -l repertoire
total 0
-r-xr-xr-x 1 junier junier 0 août  16 11:10 fichier2
-r-xr-xr-x 1 junier junier 0 août  16 11:12 fichier3
junier@fredportable:~/bac$ ls -l -d repertoire/
dr-xr-xr-x 2 junier junier 4096 août  16 11:12 repertoire/
junier@fredportable:~/bac$ cd repertoire/
junier@fredportable:~/bac/repertoire$ rm fichier2
rm : supprimer 'fichier2' qui est protégé en écriture et est du type « fichier vide »
rm: impossible de supprimer 'fichier2': Permission non accordée
```

- Le superutilisateur `root` peut changer le propriétaire d'un fichier avec la commande `chown` ou son groupe principal avec la commande `chgrp` :

```
junier@fredportable:~/bac$ ls -l fichier
-rw-rw-r-- 1 nobody mail 0 août  16 12:05 fichier
junier@fredportable:~/bac$ sudo chown nobody fichier
junier@fredportable:~/bac$ ls -l
-rw-rw-r-- 1 nobody junier    0 août  16 12:05 fichier
junier@fredportable:~/bac$ sudo chgrp mail  fichier
```

```
junier@fredportable:~/bac$ ls -l
-rw-rw-r-- 1 nobody mail      0 août  16 12:05 fichier
```

8 Licence logicielle et système d'exploitation

Une licence de logiciel est un contrat qui octroie un droit d'usage non exclusif et par lequel le titulaire des droits d'auteur du logiciel définit les conditions dans lesquelles ce programme peut être utilisé, diffusé ou modifié par l'usager.

8.1 Contexte historique

De ses débuts jusqu'à l'aube des années 1980, le monde de l'informatique fut dominé par fabricants de `hardware` tels que IBM. Dans les années 50 et 60, les ordinateurs étaient de gros calculateurs et l'informatique concernait principalement des spécialistes (ingénieurs, chercheurs). Les logiciels ou `software` existaient mais étaient indissociables de la machine sur laquelle ils étaient installés par le fabricant. L'interopérabilité des programmes, la normalisation des formats de données n'étaient pas dans l'air du temps. D'ailleurs, le constructeur ne facturait pas le logiciel qu'il livrait avec leur machine et fournissait les codes sources aux utilisateurs afin qu'ils puissent adapter les ressources de l'ordinateur à leur usage ou rectifier certains bugs.

Dans les années 70, des projets de développement logiciels d'importance émergèrent comme le projet **UNIX** d'un système d'exploitation multiutilisateurs (plusieurs comptes utilisateurs mais un seul utilisateur à la fosi alors que le système MULTICS avait pour ambition de supporter plusieurs utilisateurs simultanés). Il fut d'abord développé par Ken Thompson et Denis Ritchie (créateur du lagage C) dans les laboratoires Bell. Cette entreprise livra le code source à l'université de Berkeley dont les étudiants améliorèrent le projet qui fut distribué sous le nom de **Berkeley Software Distribution** (BSD).

A cette époque, les échanges de code sources entre développeurs étaient naturels et ils furent accélérés par le déploiement du réseau Arpanet d'interconnexion des universités et centres de recherche, ancêtre de l'internet.

L'essor de la micro-informatique dans les années 80, le changement de profil des usagers (plus forcément des techniciens), firent du logiciel le facteur dominant du business informatique. Les sociétés d'édition de logiciels comme Microsoft supplantèrent les fabricants de hardware comme IBM. Mais la protection de leurs savoir-faire et de leur code ressource devint un facteur essentiel de leur stratégie de développement. L'époque des pionniers de l'informatique qui partageaient leurs codes sources et leurs idées était révolue.

En 1984, **Richard Stallman** démissionnait du MIT, crée la **Free Software Foundation** (FSF) et lançait le projet **GNU** de développement d'un système d'exploitation libre dont aucun élément ne serait soumis à une **licence propriétaire** (la première distribution Unix sans code propriétaire ne fut publiée qu'en 1989). Stallman créa la première **licence logicielle libre** : la General Public License (GPL). Le projet GNU aboutit en 1991 au système d'exploitation libre **GNU/Linux** après incorporation du dernier élément manquant, le noyau (ou kernel), apporté par Linus Torvalds.

Des références :

- http://www.gnu.org/gnu/gnu.html
- http://fr.wikipedia.org/wiki/Unix

Source : Wikimedia Commons, René Mérou et autres auteurs

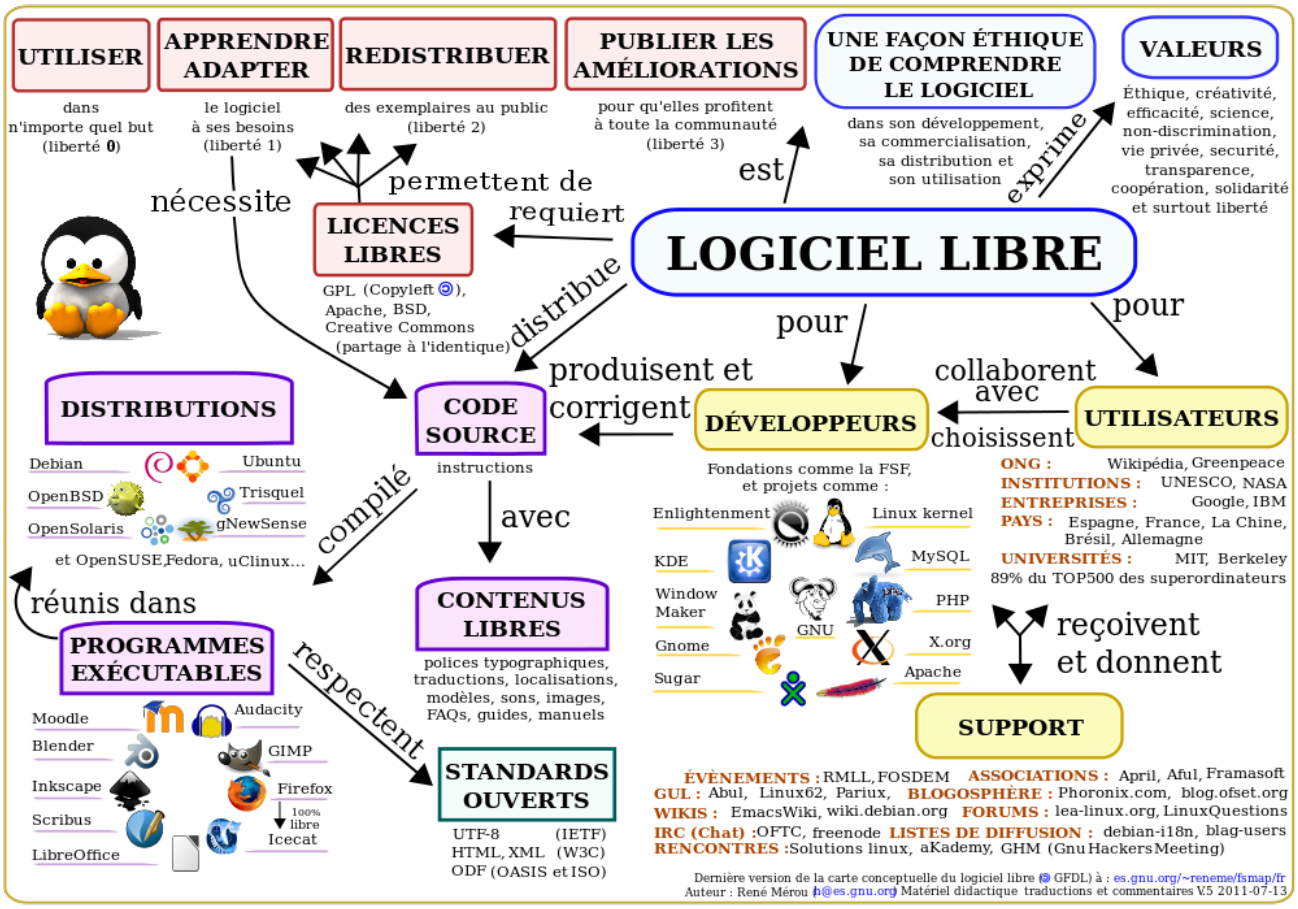

Figure 4: Carte du libre

8.2 Les différents types de licences logicielles

On distingue :

- les **licences propriétaires** :
  Un éditeur concède à titre non exclusif un droit d'usage sur un logiciel dont il conserve les droits de propriété intellectuelle. Un **CLUF** (Contrat Licence Utilisateur Final) délimite les conditions d'usage du logiciel : limitation du nombre de copies, de postes sur lequel le logiciel peut être installé, interdiction de modification et de redistribution . En général, le code source n'est pas accessible (on parle de logiciel fermé) ....
- les **licences libres** ou **open-source** :
  elles respectent les quatre libertés fondamentales du logiciel libre :

  1. **liberté d'exécuter le programme, pour tous les usages.**
  2. **liberté d'étudier le fonctionnement du programme, et de le modifier pour l'adapter à ses besoins**
  3. **liberté de redistribuer des copies.**
  4. **liberté de redistribuer aux autres des copies de versions modifiées**

  Les libertés 2 et 3 nécessitent l'accès au code source.
  Il ne faut pas confondre licence libre et domaine public : mettre un logiciel sous licence libre ne signifie pas abandonner ses droits d'auteurs (droit moral inaliénable) mais donner des permissions aux usagers qui vont au-delà de ce qu'autorise le droit d'auteur par défaut.

Attention à *ne pas confondre logiciels libres et gratuits* (double sens du mot free en Anglais):

- un **freeware** désigne usuellement un logiciel distribué gratuitement, indépendamment de sa licence d'utilisation. Le code source n'est pas forcément fourni ; dans ce cas ce n'est pas un logiciel libre.
- un **shareware** désigne un logiciel distribué gratuitement et librement pendant une durée ou un nombre d'utilisations qui sont fixées par l'auteur. Au delà de ce délai il faut payer des royalties et le code source n'est pas fourni donc ce n'est pas un logiciel libre.

Références :

- http://www.cndp.fr/savoirscdi/societe-de-linformation/cadre-reglementaire/le-coin-du-juriste/logiciel-libre-et-gratuiciel.html
- http://www.gnu.org/philosophy/categories.html et http://www.gnu.org/philosophy/philosophy.html

8.3 Licences de logiciels libres

On distingue deux types de licences libres :

1. Les **licences avec copyleft** :

   Le copyleft (gauche d'auteur , jeu de mots sur copyright), n'est pas l'abandon du copyright (droits patrimoniaux) mais au contraire s'appuie dessus pour rendre libre un programme (respect des 4 libertés fondamentales) en obligeant toutes les versions modifiées à être libres également et redistribuées sous la même licence. Ainsi le logiciel (ou l'oeuvre numérique) sera éternellement libre. La licence `GPL` (General Public License) créée par Richard Stallman en 1989 fut la première licence

libre avec copyleft. La licence `CeCill` créée par l'INRIA est une adaptation de la licence GPL au droit français. La licence `Creative Commons` CC BY-SA 3.0, paternité et partage à l'identique, est une licence libre avec copyleft pour les oeuvres numériques (image, video ...).

Il existe des formes plus ou moins fortes de copyleft. Ainsi si on inclut du code sous license GPL dans un autre, le programme hybride créé est contaminé et doit être distribué sous licence GPL. La license **LGPL** présente un copyleft plus faible : du code sous cette license cette license peut être inclus dans un code sous une autre license sans que le programme hybride soit nécessairement sous license LGPL. C'est utile pour développer des bibliothèques de fonctions.

2. Les **licences sans copyleft** :

   L'utilisateur peut redistribuer et modifier le logiciel, mais aussi ajouter des restrictions et distribuer une version modifiée sous licence propriétaire. On parle aussi de licence permissive . La première licence libre sans copyleft fut la licence `BSD` pour le distributions de logiciels créés par l'université de Berkeley. *La licence de Python est une licence libre sans copyleft.* Ce sont des licences compatibles avec la GPL mais la réciproque est fausse.

8.4 Exemple des systèmes d'exploitation

- **Systèmes d'exploitation propriétaires :** Windows et partiellement MacOS (une partie UNIX est sous licence BSD)
- **Systèmes d'exploitation libres :** de nombreuses distributions basées sur le noyau Linux, système dérivé d'UNIX : Debian, Ubuntu, Fedora, ArchLinux .....

Exercice 1

1. Lequel de ces systèmes d'exploitation est sous licence libre ?

Réponses

A) Android **B)** Linux **C)** Windows **D)** MacOS

2. Une et une seule de ces affirmations est **fausse**. Laquelle ?

Réponses

A) Un système d'exploitation libre est la plupart du temps gratuit

B) Je peux contribuer à un système d'exploitation libre

C) Il est interdit d'étudier un système d'exploitation propriétaire

D) Un système d'exploitation propriétaire est plus sécurisé

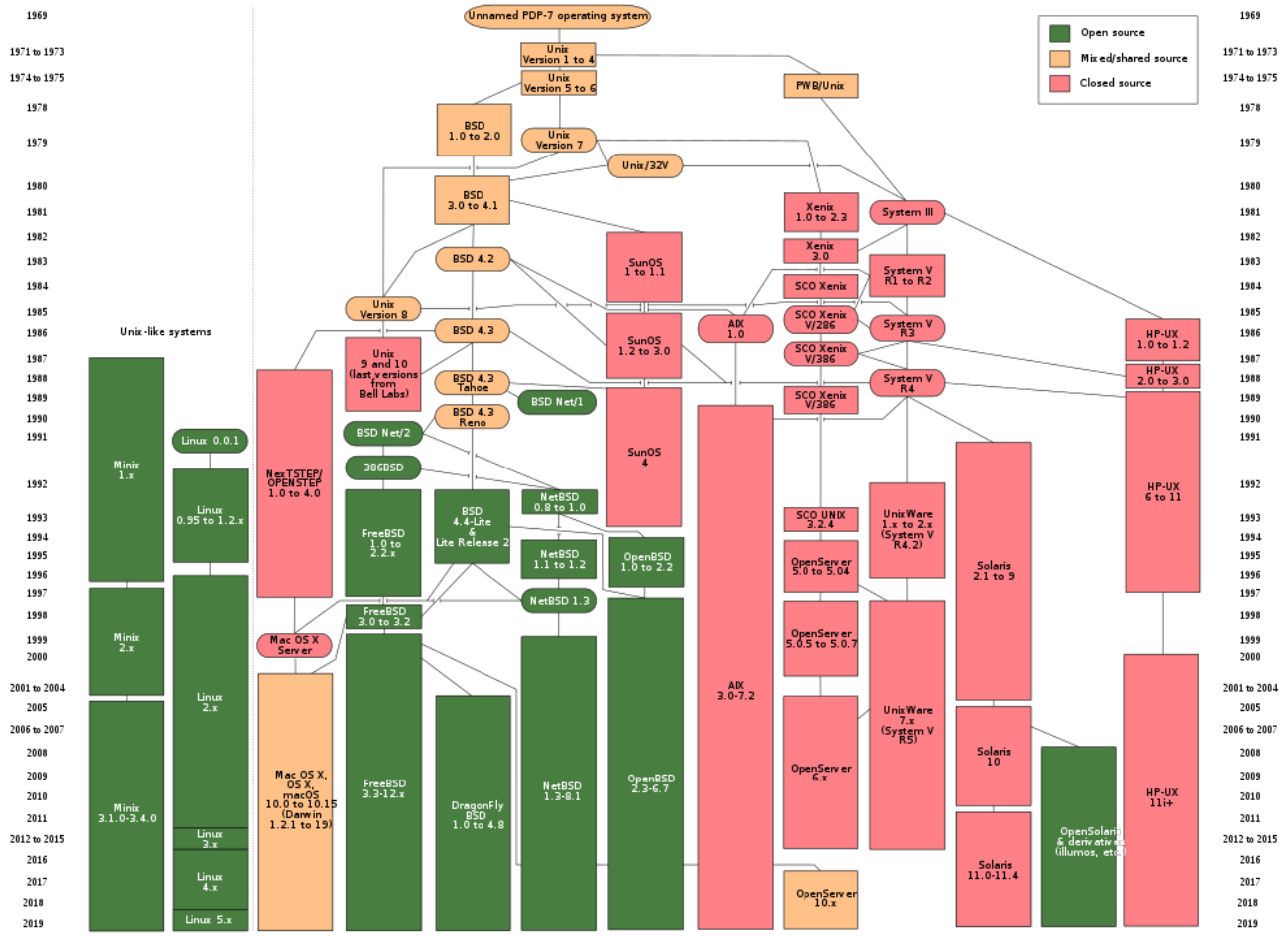

Figure 5: Arbre généalogique des systèmes UNIX

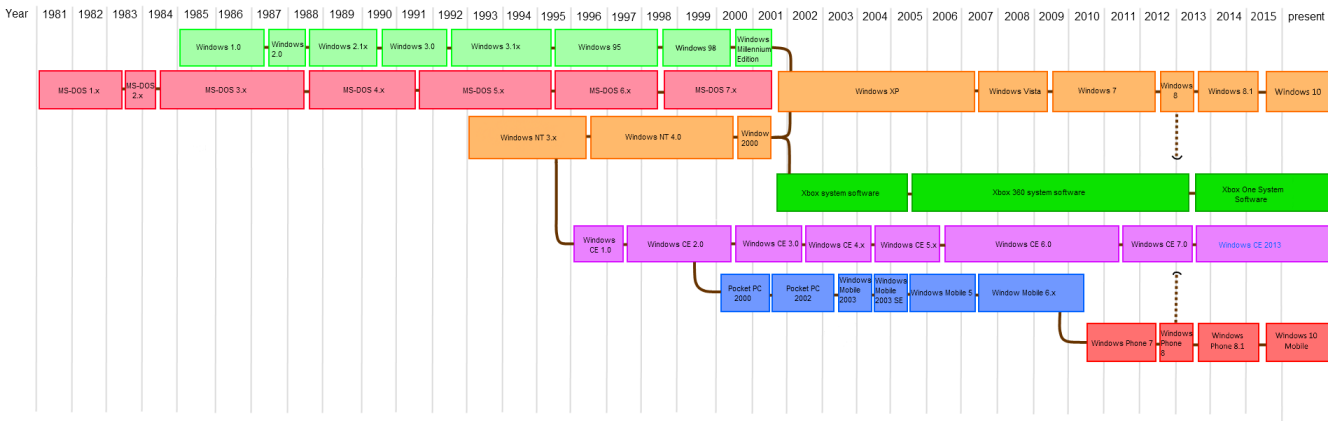

Figure 6: Arbre généalogique des systèmes Windows