éduSCOL
Informer et accompagner les professionnels de l'éducation

Liberté • Égalité • Fraternité
RÉPUBLIQUE FRANÇAISE

MINISTÈRE DE L'ÉDUCATION NATIONALE ET DE LA JEUNESSE

VOIE GÉNÉRALE

2DE | 1RE | TLE

Numérique et Sciences informatiques

ENSEIGNEMENT
SPÉCIALITÉ

SYSTÈMES DE TYPE UNIX : LE POINT DE VUE UTILISATEUR

- Histoire de l'informatique
- Représentation des données
- Traitement des données
- Interactions entre l'homme et la machine sur le Web
- Architectures matérielles et systèmes d'exploitation
- Langages et programmation
- Algorithmique

Dans ces notes, nous nous intéressons uniquement à la représentation *logique* des données (fichiers et processus) pour un utilisateur d'un système de type UNIX. Pour des aspects plus bas niveau du système d'exploitation, voir l'autre polycopié sur le système.
Dans tous les exemples en ligne de commande, le caractère > désigne l'invite de commande du shell, et le caractère # un début de ligne de commentaire. Les exemples qui suivent ont été produits sous `bash`[1], mais les commandes sont suffisament peu spécifiques pour que n'importe quel shell classique fasse l'affaire.

1. Fichiers

1.1. La hiérarchie d'un Système de Gestion de Fichiers (SGF)

Intéressons-nous en premier lieu à une abstraction intellectuelle permettant de présenter l'organisation logique des données dans un système de type UNIX. Dans un tel système, les données sont rangées dans des *fichiers (réguliers)*, eux-mêmes organisés de façon hiérarchique, grâce à des *répertoires*.
Cette hiérarchie peut être représentée par un arbre au sens informatique du terme, dont les nœuds internes seraient les répertoires et les feuilles les fichiers réguliers (et les répertoires vides), et le vocabulaire utilisé est le vocabulaire classique quand on étudie les arbres : le répertoire qui se trouve en haut de cette hiérarchie s'appelle le *répertoire racine* (ou *racine*) et l'unique répertoire contenant un autre répertoire (il existe sauf pour la racine) s'appelle le *répertoire parent* (ou *répertoire père* ou *père*). Par convention, le répertoire père de la racine est la racine en elle-même.

Sur un système de type UNIX, le caractère `/` a deux significations possibles : il désigne la racine du SGF, et il permet de séparer les noms de deux répertoires lorsqu'on décrit un chemin dans l'arbre représentant le SGF ; le caractère `~` désigne le répertoire de login de l'utilisateur, c'est-à-dire le répertoire où se trouve l'utilisateur au moment où il se connecte à son compte.
Les utilisateurs et les processus ont souvent besoin de désigner des fichiers. Considérons par exemple le fichier de configuration du shell `bash` de l'utilisateur `moi` : il a pour nom `.bashrc` et se trouve dans le répertoire de login de l'utilisateur `moi`, qu'on suppose être le sous-répertoire `moi` du répertoire `home`, lui-même sous-répertoire de la racine.
Il y a deux façons de procéder pour désigner un fichier :

- *chemin absolu* : on décrit comment arriver jusqu'au fichier en partant d'un point fixe du sysème, généralement la racine, exemple : `/home/moi/.bashrc`, ou le répertoire de login de l'utilisateur, exemple : `~/.bashrc`.

1. Plus précisément : `Ubuntu 18.04.2 / GNU bash, version 4.4.20(1)`.

- *chemin relatif* : on décrit comment arriver jusqu'au fichier en partant du répertoire courant (c'est-à-dire le répertoire dans lequel on se trouve), exemple : `.bashrc` si le répertoire courant est `/home/moi`.

Les qualificatifs *absolu* et *relatif* sont à prendre dans le sens suivant : un chemin absolu désigne de façon non ambigu un fichier, quel que soit le répertoire courant, tandis qu'un chemin relatif le désigne relativement au répertoire courant (le répertoire courant d'un processus est une variable associée à ce processus). Dans la suite, on désignera par *nom* d'un fichier tout chemin absolu ou relatif permettant d'identifier ce fichier.
Un répertoire est un fichier *spécial* : le système n'utilise pas les mêmes algorithmes que pour les fichiers réguliers pour le créer et le manipuler (par exemple un répertoire contient à tout moment un lien vers lui-même noté `.` et un lien vers son père noté `..` ; dans le cas de la racine, `..` désigne la racine elle-même). Il existe d'autres types de fichiers spéciaux que nous ne verrons pas ici.
Quelques commandes importantes pour se déplacer dans le SGF[2] :

```
> pwd  #pour connaître le répertoire courant [print working directory]
/
> ls  #sans argument: liste le contenu du répertoire courant [list]
bin    dev    home   lost+found mnt    proc    run    sys    usr
boot   etc    lib    media      opt    root    sbin   tmp    var
> ls /home/moi  #liste le contenu du répertoire fourni en argument
Bureau   Documents   Francais   Informatique   Mathematiques
> cd /home/moi  #modification du répertoire courant [change directory]
> pwd
/home/moi
> mkdir Anglais  #création d'un nouveau répertoire [make directory]
> ls
Anglais Bureau   Documents   Francais   Informatique   Mathematiques
> cd ../momo
> pwd
/home/momo
```

Pour créer ou modifier un fichier, on utilise souvent un logiciel ou une commande, qui font eux-mêmes appel au système, à travers un *appel système*, pour toutes les opérations critiques (c'est-à-dire les actions susceptibles de corrompre l'intégrité du système). Le système vérifie que l'opération demandée est acceptable avant de l'exécuter. Par exemple tout répertoire doit contenir un lien vers lui-même et un lien vers son père : à la création du répertoire, le système crée ces liens ; quand on manipule le répertoire, le système s'assure que ces liens persistent. Le système vérifie également que l'utilisateur a les bons droits sur un fichier pour y accéder.

1.2. Les droits

Sous `UNIX`, chaque fichier (et répertoire) est la propriété d'un utilisateur particulier ; par défaut de l'utilisateur qui a créé le fichier. Les utilisateurs sont réunis en groupes (un utilisateur pouvant faire partie de plusieurs groupes, pour chaque fichier est spécifié le *groupe propriétaire*, c'est-à-dire en tant que membre de quel groupe le propriétaire détient le fichier). Les droits sur les fichiers sont alors définis en fonctions de la "position" du demandeur par rapport au propriétaire du fichier : propriétaire, faisant partie du groupe propriétaire, autre utilisateur.
Seul le super-utilisateur `root` peut modifier le propriétaire d'un fichier, en revanche chacun peut modifier le groupe propriétaire d'un de ses propres fichiers, à condition d'appartenir au nouveau groupe propriétaire.
Il est particulièrement intéressant de pouvoir appartenir à plusieurs groupes car cela permet de définir des ensembles d'actions qui sont autorisées pour certains et pas pour d'autres : on peut ainsi restreindre l'utilisation du lecteur DVD à certains utilisateurs et la possibilité de lire des clefs usb à d'autres, sans que ces deux groupes d'utilisateurs n'aient de liens logiques entre eux. On peut aussi créer un groupe restreint d'utilisateurs qui auraient accès à certains fichiers.
Il y a trois types de droits (lecture, écriture, exécution, respectivement identifiés par les lettres `r`, `w` et `x` et les valeurs 4, 2 et 1) dont la signification est résumée dans le tableau ci-dessous :

2. La commande `man` suivi du nom d'une autre commande permet d'obtenir la documentation de cette autre commande, et en particulier ses options.

			fichier régulier	répertoire
lecture	r	4	regarder le contenu	lister le contenu
écriture	w	2	modifier le contenu	ajouter ou supprimer un élément
exécution	x	1	exécuter	passer à travers

On peut ainsi noter que le droit de supprimer un fichier n'est pas lié à un droit sur le fichier, mais au droit d'écriture sur le répertoire dans lequel se trouve ce fichier (ceci est tout à fait naturel si on comprend ce qu'est le contenu d'un répertoire—voir la section 1.4 ci-dessous).

On modifie les droits d'un fichier (ou d'un répertoire) avec la commande `chmod` avec deux arguments : le premier correspond aux nouveaux droits et le deuxième au nom de fichier.
Seuls le propriétaire d'un fichier et le super-utlisateur peuvent modifier les droits d'accès à un fichier. Le changement de droits se fait de deux façons différentes :

- *changement absolu* : il s'agit de spécifier de nouveaux droits, indépendamment des droits existants ; on calcule pour chaque type d'utilisateur un chiffre correspondant à la somme des droits à lui attribuer et on écrit côte à côte les droits en spécifiant d'abord ceux du propriétaire du fichier, puis ceux du groupe propriétaire et enfin ceux des autres utilisateurs. Par exemple pour un répertoire `Mathematiques`, si on veut que tout le monde ait les droits en exécution, que seuls le propriétaire et le groupe propriétaire aient le droit en lecture et que seul le propriétaire ait les droits en écriture, il suffira d'utiliser la ligne de commande
  ```
  chmod 751 Mathematiques
  ```
  Ainsi les utilisateurs appartenant au même groupe que le propriétaire pourront lister (`ls`) le contenu du répertoire, tandis que les autres ne pourront consulter un fichier du répertoire que s'ils connaissent son nom (et ont le droit de lecture sur le fichier, bien entendu) ;
- *changement relatif* : il s'agit de spécifier les droits à ajouter ou à supprimer relativement aux droits existant, en précisant :
  - le(s) utilisateur(s) concerné(s) : propriétaire du fichier (u), groupe propriétaire sans le propriétaire (g), autres utilisateurs (o), tous les utilisateurs (a) ; en l'absence de spécification, c'est l'ensemble des utilisateurs qui est concerné,
  - s'il s'agit d'un ajout (+) ou d'un retrait (−),
  - droits concernés (`r`, `w` ou `x`).

  Par exemple, pour enlever au fichier `~/Mathematiques/dm1` les droits en lecture pour le groupe propriétaire et les autres utilisateurs on utilisera la ligne de commande
  ```
  chmod go-r ~/Mathematiques/dm1
  ```

L'option `-l` de la commande `ls` permet de connaître (entre autres) les droits sur un fichier : `ls -l`.

1.3. D'autres opérations sur le SGF

Quelques autres commandes de base :

- `less` : pour voir le contenu d'un fichier ;
- `mv` : pour déplacer un fichier (ou modifier son nom, ce qui est semblable) ;
- `rm` : pour supprimer un fichier régulier ;
- `rmdir` : pour supprimer un répertoire vide (c'est-à-dire contenant uniquement une référence à lui-même et à son père).

1.4. I-nœuds

Les chaînes de caractères ne sont pas des objets pratiques à manipuler (on peut penser notament à tous les problèmes de stockages et de recherche). C'est pourquoi le système identifie les fichiers non par un nom (ce qui est pratique pour l'être humain, mais pas pour la machine), mais par un nombre, appelé *numéro d'i-nœud*. L'i-nœud (*inode* en anglais) est une entité qui regroupe un certain nombre de données nécessaires à l'accès à un fichier (par exemple le propriétaire, les droits, l'emplacement où se trouve les données du fichier). Le nom du fichier est en fait complètement indépendant de l'ensemble des données qu'il contient. Le lien entre l'i-nœud du système et le nom attribué par l'utilisateur se situe dans le contenu du répertoire dans lequel "se trouve" le fichier : le contenu d'un répertoire est une liste de paires (nom, numéro d'i-nœud) ; la suppression d'un fichier est en fait la suppression d'une paire dans cette liste, c'est donc la modification du contenu d'un répertoire : cela explique que le droit associé à cette action soit un droit d'écriture sur le répertoire.

Déplacer un fichier d'un répertoire à un autre revient à modifier les listes associées aux deux répertoires : supprimer une paire dans le répertoire origine et en ajouter une dans le répertoire destination. Il y a donc très peu de données manipulées, contrairement à ce qui se passe lors d'une copie d'un fichier. On peut le voir de façon empirique :

```
> ls -lh image.iso
-rw-rw-r-- 1 moi mon_groupe 764M mai   17  2018 image.iso
> time cp image.iso copie.iso  #time mesure divers temps associés à un processus
real  0m0,757s
user  0m0,000s
sys 0m0,432s
> time mv image.iso /tmp/bouge.iso
real  0m0,002s
user  0m0,002s
sys 0m0,000s
```

En fait sur un système de type UNIX, un fichier peut avoir plusieurs noms (voir la commande `ln`), voire aucun : on peut en faire l'expérience en lançant la lecture d'un fichier vidéo avec un logiciel *ad hoc* et en supprimant ce fichier en ligne de commande : la lecture de la vidéo continue, ce qui signifie que le fichier existe encore, et pourtant il n'a plus de nom. Une fois le logiciel de visionnage fermé, on n'a plus accès au fichier par le SGF : les données existent quelque part sur le disque et en mémoire tant que les emplacements où elles se trouvent n'ont pas été utilisés pour y mettre de nouvelles données, mais l'utilisateur n'y a plus accès (au moins de façon simple ; il existe des programmes qui analysent les données d'un disque pour tenter de récupérer des fichiers effacés[3]).
Les numéros d'i-nœud sont obtenus grâce à l'option `-i` de la commande `ls`. Le numéro d'i-nœud de la racine est toujours 2.

2. Processus

Un processus est l'objet dynamique associé à un programme : le programme est une suite d'instructions à exécuter associées à des données, le processus est une instance en mémoire de ce programme en train de s'exécuter. Il peut bien sûr y avoir plusieurs processus associés au même programme, par exemple on peut lancer simultanément plusieurs interpréteurs `python`. Les systèmes de type UNIX sont multi-tâches : on a l'impression que plusieurs processus peuvent s'exécuter en même temps (plus que le nombre de processeurs en cas de système multi-processeurs). Voir plus de détails dans l'autre polycopié de système sur ce point.
Chaque processus est identifié par un entier, son *pid* (*process identifier*). Cela permet de communiquer avec ce processus pour interrompre son cycle normal. Par exemple quand l'utilisateur se déconnecte de son compte, un signal est envoyé à chacune de ses applications pour qu'elle se ferme. L'envoi d'un signal peut se faire en ligne de commande grâce à la commande `kill` suivie du pid.
On peut voir la liste des processus qui tournent sur le système avec la commande `ps` :

```
> xclock &  #le & sert à lancer le processus en arrière-plan et récupérer la main
> ps  #liste les processus descendants du terminal
  PID TTY          TIME CMD
10733 pts/5    00:00:00 xclock
10740 pts/5    00:00:00 ps
29871 pts/5    00:00:00 bash
```

Le cycle de vie d'un processus est compliqué. En particulier, quand il utilise le processeur, un processus peut être en mode noyau ou en mode utilisateur (voir par exemple les résultats de la commande `time` dans la section 1.4) : le mode noyau correspond aux moments où le processus délègue au noyau du système son action quand celle-ci est critique : par exemple quand on demande l'ouverture d'un fichier en lecture, le système doit vérifier les droits, c'est donc à lui de faire cette ouverture ; quand le processus fait des actions non critiques (par exemple une multiplication), il reste en mode utilisateur. Un processus en mode noyau ne peut

3. En fait en raison d'un effet d'hystérésis, les nouvelles écritures sont légèrement décalées sur le disque à chaque réécriture et on peut toujours récupérer des vieilles données—mais ce n'est plus du domaine de l'informaticien ; cela explique en tout cas que les organismes manipulant des données sensibles demandent à leurs employés de protéger leurs disques par cryptage, voire détruisent ces disques en les réduisant en poussière quand ils ne sont plus utilisés.

être interrompu, ceci afin d'assurer que toutes les opérations visant à garantir l'intégrité des données du systèmes ont bien été appliquées.
Le tout premier processus créé, au moment où on allume son ordinateur, est le processus `boot` d'identifiant `0` : il a été codé en dur et devient ensuite l'ordonnanceur (*ie* il décide quel processus obtient le processeur, voir plus de détails dans l'autre poly systèmes). Ce processus n'est pas un processus UNIX à proprement parlé (pour des raisons de cycle de vie par exemple), c'est pourquoi on ne le voit pas avec la commande `ps`. A partir du processus suivant, tous les processus créés le sont de la même façon : un processus existant (le processus *père*) est clôné à sa propre demande et un nouveau pid est associé à cette copie (le processus *fils*) ; le fils reçoit une copie de toute la mémoire du processus père : a priori il exécute donc le même programme et commence dans l'état où se trouvait le père au moment de son clônage. C'est comme cela que fonctionne un shell : quand on lance une commande externe, le processus du shell se duplique et le nouveau processus ainsi créé écrase son propre code (qui au moment de la duplication est donc le code du shell) par le code de la commande (qui est contenu dans le fichier ELF de la commande, obtenu par compilation souvent au moment de l'installation du système ou d'un paquet). Le droit en exécution sur un fichier correspond donc au droit de mettre en mémoire le contenu du fichier pour l'exécuter.
Ainsi, les processus sont organisés sous forme de hiérarchie. Le seul processus créé par le processus 0 est le processus `init` de pid `1`.
Le pid du processus père est appelé le *ppid* :

```
> ps o ppid,pid,comm
  PID PPID     CMD
10733 29871    xclock
10740 29871    ps
29871 23615    bash
```

On peut voir la hiérarchie de processus grâce à la commande `pstree`.
Quand un processus a fini son travail (par exemple la commande `date` a affiché la date, ou l'utilisateur a fermé une fenêtre), il passe dans l'état *zombie* : il ne peut plus avoir accès au processeur, mais ses données sont conservées en mémoire tant que son père n'a pas constaté sa fin, et éventuellement récupéré son code de retour ; le processus disparaît alors de la table des processus. S'il avait lui-même créé d'autres processus, ceux-ci sont *orphelins* et récupérés par le processus de pid `1` qui devient leur nouveau père.
Les fichiers *core* qui sont parfois créés lors d'un arrêt intempestif d'un processus (par exemple un *segmentation fault*) sont en fait l'état de la mémoire d'un processus au moment de son interruption.

3. Processus et SGF

Les commandes écrites dans cette partie n'ont de sens que sous linux. En effet le pseudo-système de fichiers `/proc` n'existe pas sur les systèmes BSD (FreeBSD par exemple, mais aussi OS X).
Sous linux, le répertoire `/proc` reflète une partie des informations sur les processus du système. Il contient en particulier un sous-répertoire par processus, dont le nom est le pid du processus.
On peut par exemple voir le répertoire courant d'un processus e la manière suivante :

```
> pwd
/home/moi/tmp
> xclock &
[1] 4340
> ls -l /proc/4340/cwd
lrwxrwxrwx 1 moi mon_groupe 0 juil. 24 11:10 /proc/4340/cwd -> /home/moi/tmp
```

On peut également voir les ouvertures d'entrées/sorties :

```
> chromium-browser &
[2] 5462
> ls -l /proc/5462/fd | head -n 5  #pour n'obtenir que les 5 premières lignes
total 0
lrwx------ 1 moi mon_groupe 64 juil. 24 11:22 0 -> /dev/pts/6
```

```
lrwx------ 1 moi mon_groupe 64 juil. 24 11:22 1 -> /dev/pts/6
lrwx------ 1 moi mon_groupe 64 juil. 24 11:22 10 -> socket:[3179410]
lrwx------ 1 moi mon_groupe 64 juil. 24 11:22 100 -> /home/moi/.config/chromium/
    ↪ Default/Favicons
```