Le système d'exploitation Unix

Historique

1969 Ken Thomson, Bell Labs, première version mono-utilisateur, interactif, écrit en assembleur sur PDP-7 puis PDP-9.

1971 Version multi-utilisateur sur PDP-11

1971-1973 Développement du langage C par Dennis Ritchie, à la fois langage de programmation système avec des structures de contrôle de type ALGOL plus proches du langage machine pour l'efficacité.

1973 Première version d'UNIX en C pour assurer sa portabilité.

1975 UNIX version 6 issue des laboratoires Bell.

1978 UNIX version 7, première version réellement commercialisée.
- effort de portabilité : compilateur C portable, découpage du noyau pour isoler les parties dépendantes de la machine.
- nouveaux utilitaires (SCCS, …)

Développements AT&T

1982 UNIX system III
- outils de comptabilité, outils d'atelier logiciel (PWB)

1983 UNIX system V
- éditeur d'écran (vi)
- mécanisme pour la communication interne (files de messages, sémaphores, mémoire partagée.
- nouveau mode de commercialisation

Actuellement La version SystemVR4 de UNIX est commercialisée par USL(Unix System Laboratory)

Historique

Développements université de Berkeley

1981 UNIX BSD 4.1
- version VAX, avec gestion de mémoire virtuelle

1983 UNIX BSD 4.2
- gestion des périphériques VAX
- outils de communication (réseaux)

actuellement Version 4.4BSD

Normalisation

1991 Création de la norme POSIX normalisant les différentes version d'Unix BSD et System V 4.x

1993 Intégration à POSIX de la notion de "*processus léger (thread)*" avec la technologie des "*micro noyaux*"

Bien qu'il existe de nombreux types d'Unix, tous sont basés soit sur BSD, soit sur System V ou même les deux à la fois.

AIX	IBM
HP/UX	HP
Solaris	SUN Microsystems
SunOS	SUN Microsystems
Ultrix	DEC
Linux	gratuit sur l'internet

Le système Unix utilisé dans ce cours est celui de Sun Microsystems appelée Solaris 2.5 sur trex et zadig et OSF1 sur pauli, dirac et fermi.

Unix vs autres OS

Unix se distingue des autres OS de trois manières :

1 Le système est écrit dans un langage de haut niveau

2 Le système est distribué sous format source

3 Le système fournit des primitives puissantes normalement délivrées avec des OS qui tournent sur des matériels beaucoup plus puissants

Principales caractéristiques

Système d'exploitation

- Interactif
- Multi-utilisateur, multi-tâches => assure une bonne répartition des ressources (mémoire, processeurs, espace disque, imprimantes, utilitaires)
- Système de fichiers hiérarchisé
- Système hiérarchisé de processus (génétique des processus) avec une notion d'héritage.
- Compatibilité totale des E/S sur périphériques, fichiers, entre processus. Tout ressource physique ou logique du système (*device*) : terminal, disque physique ou logique, mémoire physique, est vue par l'intermédiaire d'un fichier.
- Exécution de processus avec contrôle synchrone (attente de fin) ou asynchrone (poursuite en parallèle)
- Haut degré de portabilité. Système "ouvert" supporté par de nombreuses machines
- Langages de commandes (*shells*), véritable langage de programmation avec structures de contrôle évoluées
- Le **noyau** (*kernel*) gére uniquement les **processus**, les **ressources**, les **fichiers**
- L'interface avec le noyau est assuré par un ensemble de gestionnaires de périphériques (*device drivers*)
- L'interface entre le noyau et, les programmes utilisateurs est assuré par un ensemble d'appels systèmes;

Système de développement

- Outils d'édition de textes (vi, ed)
- Outils de débogage (adb, dbx)
- Compilateurs
- Nombreux utilitaires très puissants

Généralités

Machine hôte et terminal

- La machine hôte supporte plus d'un utilisateur en même temps
- Son OS partage les ressources de l'hôte équitablement entre tous les utilisateurs
- On travaille sous Unix à travers un terminal : écran-clavier, terminal X.
- Le terminal est **connecté** à l'hôte.

Relation hôte-terminal

- Lorsqu'on appuie sur une touche du clavier, un signal est envoyé à l'hôte qui l'interprête.
- L'hôte répond par un signal d'affichage du caractère tapé indiquant au terminal de réaliser effectivement l'affichage.
- L'hôte fait un **écho** du caractère sur l'écran.

Types de connexion

- **Directe** => plusieurs terminaux, 1 hôte
- Par un **serveur** de terminaux => plusieurs terminaux, plusieurs hôtes. Ce mode de connexion nécessite une commande supplémentaire à destination du serveur (**connect** *hôte*) pour choisir l'hôte.
- Le serveur réalise ensuite la connexion.
- Avantage : un même terminal peut utiliser plusieurs machines.

Généralités

Console et terminal

- L'ensemble écran-clavier-souris d'un ordinateur est équivalent à un terminal.
- Ce terminal est appelé **console**.
- On le nomme ainsi car il n'a pas à être connecté. La console est intégrée à l'ordinateur.

Station de travail

- C'est un ordinateur Unix utilisé par une seule personne à la fois, bien qu'il puisse supporter plusieurs utilisateurs.
- Une station de travail ne possède qu'un terminal, la console.
- Par rapport à une station de travail, l'utilisation d'un terminal offre trois avantages :
  - moins cher
  - plus de facilités (connexion à un hôte plus puissant, plus de mémoire, plus de services).
  - maintenance faite par un technicien.
- L'utilisateur d'une station de travail doit prendre en charge l'installation et la maintenance du système Unix. Il doit réaliser les sauvagardes lui-même.

Connexion et réseau

- Les ordinateurs sont connectés entre eux pour partager les ressources (imprimantes, fichiers, courrier, utilisation à distance, …)
- L'écran-clavier peut jouer le rôle d'un terminal pour un autre ordinateur.

Réseaux

- **L**ocal **A**rea **N**etwork (**LAN**) : ordinateurs connectés à travers un cable
- **W**ide **A**rea **N**etwork (**WAN**) : LAN connectés entre eux à travers des lignes à haute vitesse (*BACKBONE*), permettant ainsi l'échange de courrier sur un vaste domaine par exemple.

Généralités

Relation client-serveur

- L'utilité d'un réseau est avant tout le **partage des ressources** (par exemple, le partage d'espace mémoire sur réseau implique que l'on peut stocker ses propres fichiers sur un ordinateur distant).
- Tout programme qui offre une ressource est appelé **serveur**.
- Tout programme qui utilise une ressource est appelé **client**.
- Exemple : un programme qui fournit un accès aux fichiers sur le réseau est un serveur de fichiers. Un programme qui coordonne les différentes imprimantes est un serveur d'impressions. On trouve aussi des serveurs de courrier, de news, …

Connexion réseau à grande échelle

- Beaucoup de réseaux (WAN) sont connectés à de grands réseaux régionnaux ou nationaux. Les ordinateurs qui assurent le lien avec le monde extérieur sont appelés passerelles (*GATEWAYS*). Les passerelles s'occupent du routage vers le réseau approprié.
- Exemple : Internet. On peut utiliser un ordinateur très lointain. La commande *talk* permet de connecter 2 ordinateurs et de converser (echo sur les deux machines).
- Les ordinateurs peuvent être hétérogènes.

Terminaux graphiques

- Ils permettent d'afficher aussi bien des caractères que des images, des formes géométriques, …
- Ils sont utilisés à travers une interface graphique
- Si l'interface graphique est basée sur X Window, on les appele des **terminaux X**.
- Le lien entre un terminal graphique et l'hôte est un lien à grande vitesse car la communication d'images nécéssite le transfert de grandes quantités de données. La vitesse des lignes téléphoniques se révèle insuffisante à cet usage.

X Window

- historique
  - W fut l'interface graphique développé pour le système d'exploitation V à l'université de Stanford
  - le MIT se servi de W comme base au système de fenêtrage X (projet ATHENA)
  - le système X est depuis 1987 maintenu par le X consortium
  - la version actuelle est X11R5.
- idée
  fournir des services standards aux programmes qui affichent des données graphiques
- exemple
  un programme s'exécute sur une machine et affiche ses résultats sur un terminal attaché à une autre machine. X est responsable de la gestion de cet affichage.

Système X, serveur X et client X

- Le **système X Window** a été conçu pour faciliter la création et l'utilisation de programmes graphiques (qui offrent une interface utilisateur graphique)
- X offre une serveur d'affichage (E/S), le **serveur X** qui gère les problèmes d'interfacage avec une interface utilisateur graphique.
- Un **client X** est un programme qui tourne sous X, utilisant les ressources du serveur X pour gérer les E/S
- Le système X comporte une cinquantaine de programmes utilitaires tournant sous X (xclock, xcalc, xterm, xedit, …)

Le gestionnaire de fenêtres (window manager)

- L'interface graphique réel n'est pas produit par X mais par un gestionnaire de fenêtres. On n'intéragit pas directement avec X.
- Les plus connus sont :
  - **mwm** (motif produit par OSF) proche de presentation manager sous OS/2
  - **olwm** (Open Look développé par AT&T et SUN)
- Le serveur X est celui de la machine sur laquelle on travaille, ou bien de la machine sur laquelle le terminal X "boote".
  Le gestionnaire de fenêtre (window manager) est un client X.

Utilisation de X pour exécuter des programmes à distance

- L'affichage (input/output) est séparé du traitement
- L'affichage (input/output) est géré par le serveur X
- Le serveur X peut gérer n'importe quel client X quelle que soit la machine sur laquelle le client est exécuté

Le client xterm

- Emulation d'un terminal VT100 (par défaut)

Démarrer un programme sur un système distant

- **>xterm&** (Se mettre dans une fenêtre xterm)
- **>xhost trex** (pour indiquer à notre serveur X, exécuté sur le terminal X, que la machine distante est autorisée à accéder notre machine)
- **>telnet trex** (ou **rlogin trex**, pour se logger sur la machine distante)
- **>setenv DISPLAY artichaut:0.0** (positionne la variable d'environnement pour indiquer à trex d'utiliser le serveur X d'artichaut)
- **>mon_prog&**

Identification de l'utilisateur : Login

Identification

L'utilisateur doit être :

- connu du système => **compte utilisateur**
- reconnu du système => **opération d'iedentification** (*login*)
- désidentifié à la fin de ses travaux => *logout*

Compte utilisateur

Il se compose :

- d'un identifiant (*User name + UID*)
- d'un groupe de rattachement (*GID*)
- d'un répertoire d'accueil (*Home directory*)
- d'un environnement de travail (fichier .*login*)

Sous Unix

```
login : nom utilisateur
password : mot de passe
```

2 lignes s'affichent :

- **Date** du dernier login, **nom** du périphérique utilisé => on peut savoir si quelqu'un a utilisé notre login
- Affichage du fichier **/etc/motd** (message of the day) géré par l'administrateur
- Affichage de l'invite (**$** pour le shell, **%** pour le C-shell)

remarques :

- Pour se déconnecter : **logout**
- Pour sortir du shell (**^D** ≡ fin de fichier) et donc se déloger
- **A chaque connection, les fichiers .login et .cshrc** sont éxécutés.

Le fichier /etc/passwd

- Les carctéristiques d'un utilisateur sont contenues dans le fichier */etc/passwd*

- Structure du fichier */etc/passwd*

Nom	Nom usuel de l'utilisateur
Mot de passe	(Codé)
Identificateur	Numéro individuel (UID)
Groupe d'appartenance	Numéro du groupe (GID)
Zone d'information	(Nom et Prénom)
Répertoire d'accueil	En référence absolue
Interpréteur de commandes	Bourne shell, C shell, Korn shell, …

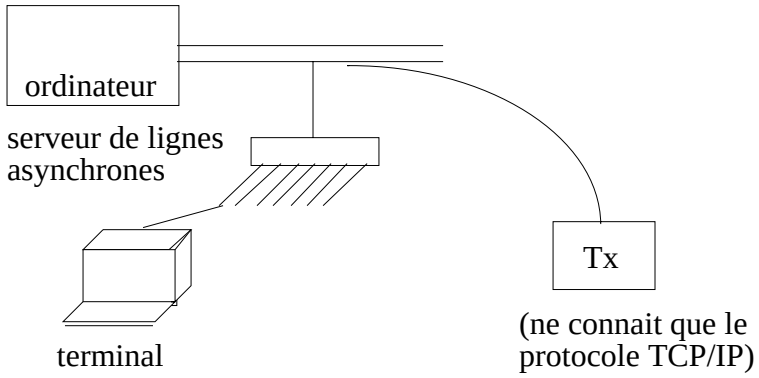
Nom		Passwd		UID		GID		Commentaire		Home		Shell
root	:	*	:	0	:	1	:	SuperUser	:	/	:	/bin/sh
jojo	:	*	:	103	:	13	:	Lucien Jojo	:	/u/jojo	:	/bin/csh
mad	:	*	:	66	:	21	:	Max Mad	:	/u/mad	:	/u/csh

Connexion à distance

La connexion nécessite une étape supplémentaire.

```
telnet> c tulipe
login : nom utilisateur
passwd : mot de passe
%
```

On se connecte ainsi à toutes les machines qui supportent le protocole du serveur de terminaux.



remarque **: telnet ne dépend pas du système d'exploitation. C'est un service du protocole TCP/IP. Si on veut connecter 2 stations de systèmes d'exploitation différents, on utilise telnet**.

rlogin est une commande UNIX. On utilise donc rlogin si les deux stations sont UNIX.

Login déjà établi

Connexion à partir d'une session

- A partir d'une session, il est possible de se connecter sur une autre machine par les commandes **rlogin** et **telnet** sur un réseau supportant le protocole TCP/IP.

rlogin Entre machines UNIX.
telnet Pour des machines possédant des systèmes d'exploitation différents.

Prendre l'identité d'un autre utilisateur

- Il est possible de prendre l'identité d'un autre utilisateur par les commandes **su** et **login**.

su On choisit son nouvel environnement de travail, puis, après abandon du shell crée, on retourne dans son environnement initial.

login Il est impossible de revenir dans son environnement initial.C'est comme si on faisait d'abord un logout chez soi.

- *su* avec l'option - permet de prendre l'identité d'un autre utilisateur ainsi que son environnement de travail exactement comme si cet autre utilisateur avait réalisé la connexion.

```
%su - max
%
```

- *su* sans option permet de prendre l'identité d'un autre utilisateur en conservant son propre environnement de travail.

```
%su max
%
```

Exercices

- Modifier votre mot de passe

```
% passwd
old password : ancien mot de passe
new password : nouveau mot de passe
re-enter new password : nouveau mot de passe
```

- A partir de votre session, ouvrir une session sur **trex** et **zadig**.
  (ouvrir un xterm : **%xterm&**)

- Fermer les sessions (**^D, exit**).

- Prendre l'identité d'un autre utilisateur en récupérant son environnement de travail, puis revenir à son shell initial.

- Prendre l'identité d'un autre utilisateur en restant dans son propre environnement, puis revenir à son shell initial.

- Mêmes manipulations sans nécessité de retour dans son environnement propre.

```
%env          -- on visualise l 'environnement de
                  -- travail
%setenv x 10 -- on ajoute une variable initialisée
à             -- 10
%su toto      -- on prend l'identité de toto
%env          -- on remarque la présence de x=10
%exit         -- retour à sa propre identité
%su - toto    -- on prend l'identité de toto
%env          -- la variable x est absente=>on est
                  -- dans l'environnement de toto
```

remarque : *.cshrc* permet essentiellement de positionner les alias et l' invite (*prompt*)
.login contient 4 variables de base : USER, PATH, HOME, DISPLAY

Le terminal

- En Unix, **tty** signifie terminal, abbréviation de télétype qui furent les premiers terminaux, périphériques éléctromécaniques qui réalisaient leurs sorties sur papier.

- Les différents types de terminaux sont décrits dans le fichier */etc/termcap*.Cette base de données permet aux programmes d'être compatibles avec une grande variété de terminaux.

- La variable d'environnement TERM permet à Unix de connaitre le type de terminal utilisé.

```
%echo $TERM  -- pour connaitre la valeur de TERM
```

- L'interface entre le système UNIX et tout périphérique est effectué par des programmes appelés **pilotes** (*drivers*).

- Le terminal, **tty**, est donc géré par un pilote de tty.

- L'interface est paramêtrable (commande **stty**)

- Les paramêtres de stty sont (ils sont visualisés par **%stty -a**) :

  erase destruction du dernier caractère
  kill destruction de la ligne
  intr interruption
  eof fin de fichier

Le manuel en ligne

- Collection de fichiers sur disque : un pour chaque commande ou sujet.

- La commande man suivie du nom de la commande affichera la documentation voulue.

```
%man su
%man man
%man su man cp
```

- La documentation est affichée à l'écran par la commande more qui permet une visualisation page à page (**espace** pour changer de page, **q** pour sortir).

- Le manuel est organisé en huit **sections** :

1	COMMANDES
2	APPELS SYSTEME
3	FONCTIONS BIBLIOTHEQUE
4	FICHIERS SPECIAUX
5	FORMATS DE FICHIER
6	JEUX
7	INFORMATIONS DIVERSES
8	COMMANDES DE MAINTENANCE

- Format d'une page du manuel

Name	: nom et but de la commande
Synopsis	: syntaxe
Description	: description complète
Files	: liste des fichiers utilisés par cette commande
See Also	: où trouver l'information associée
Diagnostics	: erreurs possibles et warnings
Bugs	: erreurs

Utilisation pratique du manuel en ligne

- Chaque section et sous-section contient une page d'introduction (bève description)

```
%man intro   (ou man 1 intro)
%man 1c intro
%man 6 intro
```

- La recherche d'une commande ou d'un sujet est effectuée à partir de la première section jusqu'à la première description trouvée.

```
%man kill     description trouvée en
              section 1 si elle existe
              sinon 1ère description
              trouvée
%man 2 kill   description trouvée en
              section 2 (si elle existe)
%man 3f kill  description contenue dans la
              section 3 sous-section f
%man 1 kill 2 kill 3f kill
```

- Références internes aux sections du manuel

```
SEE ALSO
    ls(1), chmod(2), …
```

Les n° 1 et 2 indiquent les n° de sections où trouver l'information

- Recherche rapide dans une page. L'affichage s'éffectuant sous **more**, il suffit de taper **h** (help) pour afficher un résumé de toutes les commandes disponibles.

```
%man csh
…
…
/SEE ALSO    les pages défilent jusqu'à
             SEE ALSO
```

Utilisation pratique du manuel en ligne

- Autres commandes utiles

```
whatis   =>  fournit une brève description
             de la commande
apropos  =>  localise des commandes à
             partir des mots clés
```

- **Exercices**

  1- Lancer la commande **`%man man`** pour étudier la structure générale du manuel et les options de la commande **`man`**

  2- Afficher le détail de l'option **`-x`** de la commande **`gcc`**

  3- Rechercher la description de la fonction **`sleep`**

  4- Etudier les commandes **`echo`** et **`who`**

  5- Explorer la description de la commande **`login`**, son environnement, les fichiers associés

- **Remarque**

  Lorsqu'on utilise X Window, il existe un client X nommé xman qui fournit une version graphique de la commande **`man`**.

Syntaxe générale des commandes Unix

• Ligne de commande

Plusieurs commandes peuvent apparaître sur une même ligne, à condition de les séparer par un **;**

```
%date;cd;ls -l toto
```

• Syntaxe générale d'une commande

nom_commande options paramètres

ou plus précisément,

nom_commande [-option [arg][,args]*]* [args]*

nom de commande, liste facultative d'options avec ou sans arguments, liste d'arguments

[1]

• Les options

Elles servent à préciser la nature du travail demandé

```
%ls -l -F toto
%ls -F -l toto
%ls -lF toto
%ls -Fl toto
```

• Les arguments

Ils peuvent être de différentes natures : par défaut, noms de fichiers, de commandes, mots clés,…

```
%ls  -a  -l
%ls  -ldg  /temp /bin
%cc  -o  cat  -O  cat.c
%man -k manual
```

• Remarques

- On peut concaténer les options sans arguments.
- Pour certaines commandes, l'ordre des arguments et des options n'a aucune importance.
- Le tiret est supprimé lorsque les options sont obligatoires.

[1] Les commandes prises en exemple sont celles du système OSF1

Comprendre la syntaxe d'une commande du manuel

ls [-aAbcCdfFgilLmnopqrRstuxl] [filename …]

On comprend que :

- La commande a 24 options
- Les options sont toutes facultatives car elles sont entre crochets
- Il y a un argument qui est facultatif (entre crochets)
- L'argument est suivi de …; cela signifie que plusieurs arguments peuvent être employés
- La commande peut n'avoir aucun argument (à cause des crochets)
- Le nom des arguments est choisi par l'utilisateur
- Le nom des options est fixé (minuscules et majuscules portent une signification différente).

Commandes de communication

- **users**

Affiche les utilisateurs loggés sur la machine

- **who**

Donne plus d'informations sur les utilisateurs logés sur la machine :

- UID
- Nom du terminal
- Date et heure du login
- Machine à partir de laquelle le login a été réalisé.

- **w**

Permet de savoir ce que fait quelqu'un sur une machine donnée.

- **finger**

Affiche des informations relatives à un utilisateur sur une machine (éventuellement distante du moment qu'elle est connectée à la votre).

- **ping**

Permet de savoir si une machine est connectée.

- **uname**

Permet deconnaitre le nom du système d'exploitation. Avec l'option **-a**, on peut connaitre aussi le nom de la machine sur laquelle on est loggé.

Exercices

- Etudier les options **-h** et **-s** de la commande **w,** ainsi qu'une combinaison des deux.
- Explorer les commandes **ping**, **finger**, **users** …

Les fichiers Unix

- En général, un fichier est une collection de données stockée sur un disque ou sur une bande magnétique

- Un fichier Unix revêt une définition plus large : il s'agit en plus de toute source de données qui peut être lue ou toute cible sur laquelle des données peuvent être écrites (clavier, écran, imprimante, …)

- Un fichier Unix est une chaine de caractères non structurée. Il n'existe aucune notion d'organisation de fichiers (indexé, séquentiel indexé, direct,…).

- A tout fichier physique est associé un bloc d'informations (**i-node**) contenant :
  - type du fichier (ordinaire, spécial, catalogue)
  - adresse des blocs utilisés sur le disque
  - nom du propriétaire et du groupe d'appartenance
  - droits d'accès
  - nombre de liens physiques sur le fichier
  - dates de création, de dernier accès, de dernière modification.
  - taille

- L'**i-node** ne contient pas de nom pour le fichier. La désignation se fait au moyen de **catalogues** (répertoires, "directories") qui sont des fichiers de couples **(nom, numéro de i-node)**.
  Le **nom** identifie le fichier relativement au catalogue.
  Le **numero d'i-node** est l'index par lequel le système identifie le fichier.

- **Remarques**

Rien n'empêche de référencer un même fichier sous des noms différents dans des catalogues différents.

Un **catalogue** est lui aussi un fichier; il est donc désigné par un nom dans un autre fichier. Ce système suppose l'existence d'un fichier sans nom dont le système connait la localisation. Ce fichier particulier est la racine absolue du système de fichiers.

Le système de fichiers a donc une structure arborescente. Les noeuds sont des catalogues, les feuilles des fichiers non catalogues ou catalogue vide.

Structure d'un Système de Gestion de Fichiers

Bloc d'initialisation
Super bloc
table des i-node
blocs de données

• Bloc d'initialisation

Il peut être utilisé au chargement du système (BOOTSTRAP)

• Super bloc

Il contient les informations générales du S.G.F. :

- Nombre de noeuds alloués
- Nombre de noeuds libres
- Liste d'un certain nombre de noeuds libres
- Nombre de blocs de données libres
- Liste de quelques blocs libres

• Table des i-nodes

Un i-node contient des adresses, dont quelques adresses directes de blocs de données, 1 adresse indirecte simple, et éventuellement une double et une triple.

• Blocs de données

Ce sont des blocs logiques non fragmentables de taille 512,1024, 2048, 4096 octets.

Commandes associées au Système de Gestion de Fichiers

- df

Permet de visualiser les statistiques sur la quantité d'espace disque libre

```
df [options] [nom]
```

```
%df
```

les informations affichées sont :

partition
capacité en Kbytes
Kbytes utilisés
Kbytes disponibles
% de la partition occupé
"file system" monté

- **du**

```
du [options] [répertoires]
```

Indique le nombre de blocs de 512K contenus dans les fichiers d'un catalogue et récursivement de tous les catalogues fils

```
%du /users
```

Types de fichiers

• Fichiers de données

Ce sont des tableaux à une dimension de caractères. Ils contiennent des programmes sources ou binaires, des données (fichiers ASCII).
Etant le résultat de programmes (éditeurs pour les fichiers sources, compilateurs pour les fichiers objets,…), leur structure est imposée pour chaque type d'utilisation par un codage particulier.

Exemple : les fichiers texte source sont des ensembles de lignes (c. à d. des suites de caractères terminées par le caractère <*line feed*>.

• Fichiers spéciaux (périphériques)

Un périphérique est vu comme un fichier par l'utilisateur (en particulier pour sa désignation). Toutefois, étant associé à des dispositifs d'E/S, les opérations d'E/S sont traitées différemment par le système.

Exemples : les terminaux sont des fichiers spéciaux en mode caractère; les E/S sont donc réalisées en mode caractère.
les disques sont des fichiers spéciaux en mode bloc; les E/S sont donc réalisées par bloc (en général 512 ou 1024 caractères).

• Catalogues (répertoires, "directories")

C'est un fichier qui contient des noms de fichiers, liste de couples (index d'identification système, nom d'identification relatif au catalogue).
L'information contenue dans un catalogue n'est manipulée que par le système (par l'intermédiaire de directives utilisateur).

• Fichiers "FIFO" (tubes nommés)

Ce sont des files d'attente de communication entre processus (boite à lettres). Elles sont créees par une directive particulière (mknod).

Fichiers texte, fichiers binaires

- les fichiers **textes** sont des fichiers de données qui contiennent uniquement des caractères ASCII (jeu de 128 codes représentant les caractères majuscules, minuscules, les caractères numériques, de ponctuation, spéciaux et les caractères de contrôle.

- Les fichiers textes peuvent être édités à l'aide d'un éditeur de textes sur écran, par exemple.Les codes ASCII sont alors traduits en caractères normaux, lisibles.

- Les fichiers **binaires** sont des fichiers de données non textuelles. Leur contenu ne prend sens que s'il est interprété par un programme. Par exemple, un fichier binaire obtenu par compilation et édition de liens est exécutable par la machine.

Différences techniques

- Les données sont représentées par des chaines de bits divisées en bytes (séquence de 8 bits)

- un fichier texte est une suite de séquences de 8 bits, chaque byte représentant un élément du code ASCII (exemple :01000000 pour le caractère 'A').

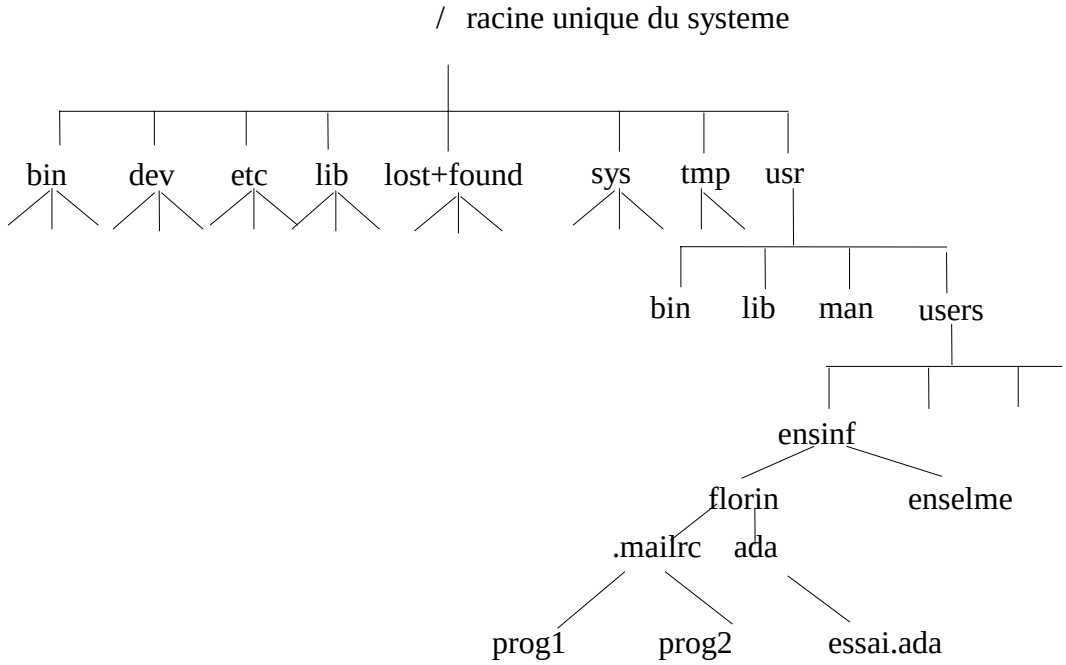
- en fait, l'information nécéssaire pour représenter un caractère tient sur 7 bits. Le plus à droite a toujours 0 pour valeur.

- le codage des fichiers binaires dépend des programmes qui les ont créés, les 8 bits sont donc significatifs.

    - le transfert d'un fichier binaire est donc une opération différente du transfert d'un fichier texte. Il est nécéssaire que les 8 bits soient correctement transférés.

Hiérarchie des catalogues d'un système UNIX

- **Exemple de hiérarchie**



- **Catalogues principaux**

lost+found	fichiers orphelins
sys	fichiers de configuration
usr	fichiers utilisateurs
doc	fichiers de documentation
man	manuel de référence
tmp	fichiers temporaires
pub	fichiers de données publiques
etc	utilitaires d'administration
dev	fichiers spéciaux (périphériques)
bin	commandes système
lib	bibliothèques

Désignation des fichiers

Nom de base

- Chaine de caractères limitée à 14 caractères.
- On se limite aux caractères alphanumériques et aux caractères "_" et ".".
- Les caractères "/" et **espace** ne sont pas acceptés.
- UNIX fait une différence entre les caractères **minuscules** et **majuscules**.

Génération des noms de base

***** remplace une chaine de caractères de 0 ou plus.
? remplace un unique caractere.
[] délimite un ensemble ou un intervalle de caractères optionnels.

Exercice : soient les noms de base :
proj.c
proj.o
projet.c
mon_texte.dvi
tpB.moi.ll
tpbV1
tpb.lui.ll
tp1.ada
tp2.ada

quels sont ceux que les expressions suivantes génèrent?

p*
p*.c
proj.*
*.dvi
tp[B b]*
tp?.*
oj.[o c]
tp[0-9]*

catalogue courant	**nom relatif**	**nom absolu**
racine	bin	/bin
/bin	ls	/bin/ls
/users/ensinf/enselme	proj.c	/users/ensinf/florin/proj.c

Chemin d'accès

Chemin d'accès absolu

Le système de fichiers a une structure arborescente dont la racine a pour référence /. Le chemin de la racine au fichier identifie donc sans ambiguité ce fichier.
Il s'écrit symboliquement en la liste successive des catalogues traversés à partir de la racine séparés par des "/".

/	catalogue racine
/usr/bin	catalogue des commandes système
/users/florin/prog	fichier utilisateur

Chemin d'accès relatif

A tout instant, chaque utilisateur est positionné sur un **catalogue de travail** qu'il peut changer librement.
Toute désignation de fichier peut dès lors être faite relativement à ce catalogue courant.
Un **nom relatif** est un chemin d'accès défini à partir du catalogue courant. Il ne commence donc pas par "/".
Un **nom absolu** est donc obtenu par concaténation du nom absolu du répertoire courant et du nom relatif du fichier.

Conventions de désignation

Chaque utilisateur possède un **catalogue privilégié** ("home directory"). Ce catalogue devient automatiquement le catalogue de travail à l'ouverture d'une session (login).
Le point "**.**" désigne le catalogue courant.
Deux points "**..**" désignent le père du catalogue courant.
Exemple : si le catalogue courant est **/local/bin** alors **../..** désigne la racine.

Disques logiques

- Tout support physique, organisable en partitions (disque logique) (région d'un disque, disk-pack, cd-rom, …) peut contenir un ou plusieurs sous-arbres de l'arborescence totale.

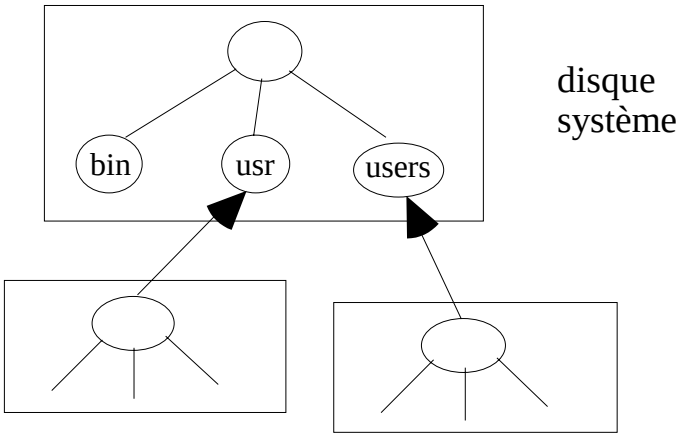
- L'espace physique global sur disque est divisé en disques logiques (partitions), référençées dans le catalogue **/dev** et possédant leur propre racine.

- Les systèmes de fichiers possèdent leur propre racine et sont crées sur des partitions distinctes

- L'un des disques logiques, le **disque système** joue un rôle privilégié. Il est **toujours accessible**. Il contient le noyau, les fichiers systèmes.

- Les autres systèmes de fichiers sont montés (attachés) selon les besoins par une commande système particulière (**mount**) en un point particulier de la hiérarchie (un nom de chemin d'un catalogue)

Exemple



Commandes relatives aux catalogues

Référence au catalogue courant : pwd

- Fournit la référence absolue du noeud catalogue courant

```
%pwd
/users/ensinf/robert
%
```

Changement de catalogue de travail : cd

- Permet de se déplacer dans l'arborescence des catalogues.
- Utilisée sans paramêtre, le catalogue privé ("home directory") devient catalogue de travail.
- Utilisé avec un paramêtre (chemin absolu ou relatif de catalogue), le catalogue désigné devient catalogue de travail.

```
%cd
%pwd
/users/ensinf/enselme
%cd ../florin
%pwd
/users/ensinf/florin
%cd ..
%pwd
/users/ensinf
%
```

Contenu d'un catalogue : ls

- Permet de lister le contenu et les caractéristiques d'un ou plusieurs catalogues.

```
%ls
f1
f2
souscat
%
```

Commandes relatives aux catalogues

Commande ls (suite)

options :

-l principales caractéristiques en format long

-a liste aussi les fichiers cachés (commençant par ".") comme .login, .mailrc

-R liste les catalogues de manière récursive

-d liste les caractéristiques du catalogue plutôt que celle des fichiers qu'il contient

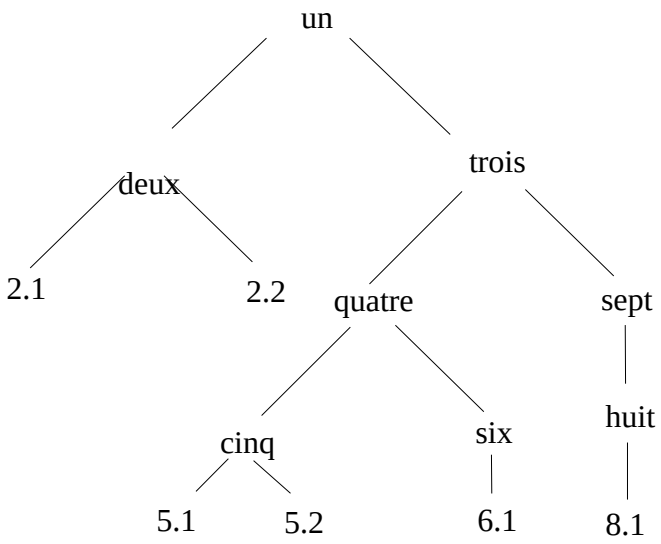
Création et destruction de catalogues : mkdir, rmdir

Un catalogue est crée par rapport à un catalogue père existant.
Un catalogue ne peut être détruit que s'il est vide.

%mkdir src /users/ensinf/dupont src/editor
%rmdir src/editor src

Exercices

- Créer l 'arborescence



Commandes relatives aux fichiers ordinaires

Destruction d'un fichier: rm [-eifrR] noms_de_fichiers

Efface un lien sur un fichier. Si ce lien est unique, les données sont également détruites.

options

-i	interactif
-r et **-R**	si l'argument est un catalogue, effaçage récursif de tous les fichiers et catalogues qu'il contient.

remarques :

```
%rm *        efface tous les fichiers
%rm -r *     efface tous les fichiers et
             catalogues
```

Affichage du contenu : cat noms_de_fichiers

Permet l'affichage de fichiers sur la sortie standard

```
%cat f1 f2
ceci est le contenu du fichier f1
ceci est le contenu du fichier f2
```

Commandes relatives aux fichiers ordinaires

Copie de fichiers :

cp [-fhiprR] *fichiers_source fichier_dest*

Effectue une copie physique d'un fichier dans un autre. Un nouvel i-node et une nouvelle entrée dans un catalogue sont crées.

```
%cp f1 f2          f1 est recopié dans f2=>ancien
                       f2 écrasé (à condition que f2
                       possède une autorisation
                           d'écriture)

%cp -i f1 f2       signale interactivement la
                       possibilité d'écrasement de
f2

%cp f1 f2 f3 dir   copie de f1, f2, f3 dans le
                       catalogue dir (dir doit
                           préalablement exister)


%cp -p dir1/* dir     copie tous les fichiers de
                   dir1 dans dir en préservant leurs
                   modes d'accès et autres
                   informations attachées à chacun
                   des fichiers copiés

%cp -r f1 d1 f2 d2 dir    copie récursive des
                          fichiers et catalogues
                              dans le catalogue dir
```

Commandes relatives aux fichiers ordinaires

Renommer des fichiers : mv [-if] *noms_de_fichiers*

Permet de renommer les fichiers ou de les déplacer dans l'arborescence.

```
%ls
     f1
     f3
     %mv f1 f2
     %ls
     f2

     %mv f2 f3 dir
     %ls
     dir
     %ls dir
     f2
     f3
```

Exercices

- Editer les fichiers **2.1, 2.2, 5.1, 5.2, 6.1, 8.1**

- Se placer sous le catalogue **un**

- Déplacer la directory **sept** sous la directory **six**

```
%mv -i ~/un/trois/sept/huit
~/un/trois/quatre/six/huit
```

ou

```
%set x=~/un/trois
%mv -i $x/sept/huit $x/quatre/six/huit
```

remarque : option **-i** pour vérifier qu'on n'écrase pas un fichier existant. on ne peut pas faire de déplacement inter-volumes. **mv** ne provoque pas la création d'un nouvel i-node.

- Déplacer **8.1** et **6.1** dans **trois**

```
%set x=~/un/trois/quatre/six
```

etc…

- Créer **5.1** dans **trois**

- Déplacer **5.1** de **trois** vers **cinq** (comme **5.1** existe dans **cinq**, il est écrasé => option -i)

- Copier le catalogue **trois** dans **six (impossible)**

- Copier le catalogue **deux** dans **cinq**

%cp -r ~/un/deux ~/un/trois/quatre/cinq

Liens

- La relation entre un **nom** de fichier ordinaire ou spécial et un **fichier physique** (son **i-node**) s'appelle un **lien**.

- Plusieurs noms peuvent correspondre à un même fichier physique.

- Créer un lien, c'est créer un nom dans un catalogue. Son descriptif (**i-node**) est celui du fichier déjà existant.

- On crée ainsi des **synonymes** d'un même objet sans création d' i-node.

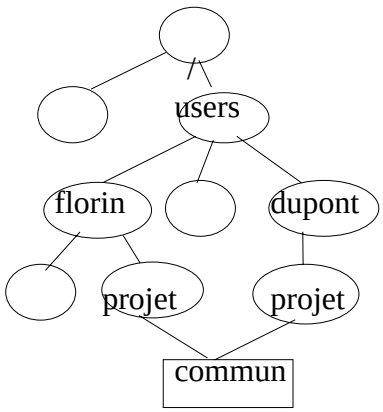
```
%ln f /usr/stagiaire/jmf/f-bis
%ls -il f /usr/stagiaire/jmf/f-bis
3081-rw-r--r-- 2 moi appli 95 Oct 10 17h43 f
3081-rw-r--r-- 2 moi appli 95 Oct 10 17h47
    /usr/stagiaire/jmf/f-bis
```

Restrictions

- Il **ne** peut **pas** y avoir plusieurs **liens** sur un **catalogue**.

- On **ne** peut **pas** établir de **liens** à travers des frontières de **disques logiques**.

Exemple



Liens symboliques

Pour créer un lien vers un répertoire ou vers un fichier dans un système de fichiers différent (donc sur une partition différente), on ne peut pas créer de lien physique puisque les fichiers n'ont pas le même i-node.
On réalise donc un lien symbolique. Plutôt que de contenir la valeur de l'i-node, le lien contient le nom du chemin du fichier origine.
Lorqu'on utilise un lien symbolique, Unix utilise le nom du chemin pour trouver et accéder le fichier origine.

- Création d'un nouvel i-node
  `%ln -s D C`
  -- On crée l 'objet C à partir duquel on pourra voir D et ses
  -- descendants.

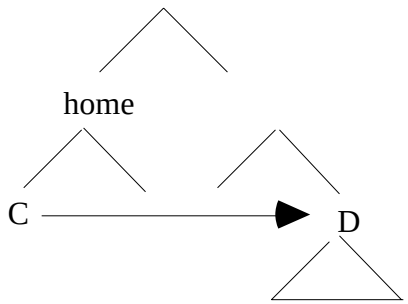
- C ne peut pas posséder de sous arbres.

- Si D est détruit, C ne voit plus rien.

- Si on détruit C, D reste en place.

Intérêt

Si on n'a pas assez de place dans une partition pour ajouter un sous arbre, alors on crée le sous arbre dans une autre partition (appelons le D) puis on crée un lien symbolique (C) vers le noeud auquel on désirait ajouter D.
On traverse ainsi les volumes.



Exercices

- Créer un lien entre un fichier existant et un nouveau fichier.

```
%ln titi toto
```

- Vérifier qu'il possède le même numéro d'i-node.

```
%ls -il titi toto
```

- Créer le fichier D dans la "home directory" et créer un lien symbolique entre B/C et D. Vérifier les caractéristiques du lien symbolique.

```
%touch D
%mkdir B
%cd B
% ln -s ../D C   -- on cree le nom C sous le
                 -- répertoire courant
%pwd
/B           -- nom du répertoire courant
%ls
C            -- C appartient à B
%cat C       -- le contenu de D s'affiche

%mkdir D;cd D;mkdir E;cd E;touch F
%ln -s D C
%cd C
%pwd
D
```

remarque : la destruction d'un lien ne provoque que la destruction d'une référence au niveau de l'i-node.

- Créer un lien symbolique entre deux partitions

```
%df -- pour visualiser les différentes partitions
    -- choisir une partition cible et un sous
    -- arbre dans cette partition.
%ln -s /tmp/essai lien-symb
    -- création du lien symbolique sous un
    -- répertoire choisi
```

Protections

- A tout utilisateur est associé :
  - un identificateur (n° de compte)
  - un groupe particulier
  - droit d'accès

- Pour un fichier donné, on distingue :
  - le propriétaire
  - les utilisateurs du même groupe
  - tous les autres utilisateurs

- Pour un fichier et pour chaque classe précédente sont associés des droits d'accès :
  - **r** lecture
  - **w** écriture
  - **x** éxécution

remarque : le super-utilisateur (root) possède tous les droits.

- Lorsque le fichier est un catalogue :

  r signifie droit de lire les noms référencés dans le catalogue

  w signifie droit d'écrire dans le catalogue (droit de détruire un fichier du catalogue et aussi de créer des fichiers dans ce catalogue

  x signifie droit de "traversée"
  => définir le catalogue comme le catalogue courant
  => droit d'accéder à un fichier dont le nom absolu comporte le nom du catalogue

commande : **chmod [-fR] mode nom_de_fichier**

-R descente récursive dans la directory pour fixer le mode spécifié à chaque fichier.

mode symbolique [qui] opérateur droit [opérateur droit]

u	**+**	**r**
g	**-**	**w**
o	**=**	**x**

(par défaut) **a**

l'opérateur **=** permet de forcer un droit

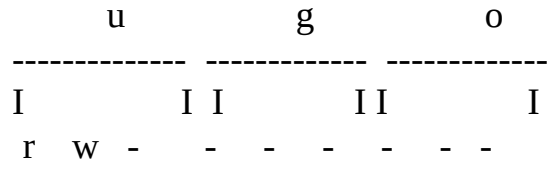
Protections

Exemple

```
%chmod o -w  f1
%chmod +x    f2
%chmod g -x+w  f3
```

Exercices

- Visionner les droits d'accès des fichiers du catalogue **cinq**



lecture et écriture de **5.1** par l'utilisateur

- Que signifie : **rwxr-xr--** lecture générale
  écriture par le propriétaire
  éxécution propriétaire + groupe

- Autoriser la lecture pour tous et l'éxécution et l'écriture pour le propriétaire et son groupe sur tous les fichiers de la directory **trois** et ses **sous directories**.

```
%cd ~/un/trois
%chmod -R  a +r .
%chmod -R ug +x+w .
```

- Essayer de lire un fichier dont l'accès est protégé en lecture.

- Copier un fichier d'un utilisateur à un autre.

remarque : **condition nécessaire et suffisante**
- les noeuds du chemin absolu sont en **x** au niveau du groupe ou du monde selon l'utilisateur copié.
- le fichier copié doit avoir les droits en lecture pour l'utilisateur qui copie

- Copier un fichier d'un utilisateur **durand** vers un utilisateur **dupont** à partir de ce dernier chez lui-même.

```
%cp ~durand/f ~  les protections ne sont pas copiées
%cp -p ~durand/f ~  les protections sont copiées
```

Type de fichier et droits

- Le type et les droits d'un fichier sont contenus dans une structure (*st_mode*)

Type de fichier

-	**régulier**	**d**	**répertoire**
b	**spécial bloc**	**c**	**spécial caractère**
l	**lien symbolique**	**p**	**tube nommé**

Bits particuliers

set-uid	**Lorsqu'il est positionné pour un fichier exécutable, le processus correspondant au programme possède les droits du propriétaire du programme et non ceux de l'utilisateur qui le lance**
set-gid	**Même rôle que le set-uid mais relativement au groupe**
sticky bit	**Indique au système que le programme doit être maintenu (collage) dans le swap même si aucun processus lui correspondant n'est actif.**

Droits d'accès

lecture par le propriétaire
écriture par le propriétaire
exécution par le propriétaire
lecture par les membres du groupe
écriture par les membres du groupe
exécution par les membres du groupe
lecture par les autres utilisateurs
écriture par les autres utilisateurs
exécution par les autres utilisateurs

Le langage de commandes

- C'est l'**interface externe** d'un système d'exploitation

Intérêt

- l'exécution des fonctions de l'interpréteur du langage
- le chargement, le paramêtrage et l'exécution des utilitaires et des programmes usagers

Trois approches

- approche langage de commande
- approche graphique
- approche de programmation

Approche langage de commandes

- On travaille directement sous l'interpréteur de commandes, qui doit être un véritable langage évolué.
- On peut lancer des scripts (programmes de commandes)
- On accède directement à un grand nombre de fonctions internes du système.

Approche graphique

Le langage de commandes est défini sous forme graphique (boutons, icones, menus, …)
Il complète un interface graphique (X Window) pour offrir des fonctions conviviales d'interface constructeur.

Approche de programmation

Les tâches lourdes d'exploitation sont directement définies par programmes (C).

Les interpréteurs de commandes

- Les interpréteurs de commandes ou shells sont nombreux. Le premier utilisé fut le Bourne shell (**sh**) du nom de son réalisateur.

La famille Bourne shell

- Le Korn shell (**ksh**) est compatible ascendant avec **sh** et offre de nouvelles caractéristiques (aliasing, historique,…)

- Le **bash** (Bourne again shell, 1989) est un produit freeware (Free Software Foundation, projet GNU)

- Le **zsh** (1990) pour les programmeurs Unix avancés

La famille C-shell

- Le C-shell (**csh**) est le plus populaire des interpréteurs de commandes. Il est très proche du langage C.

- Caractéristiques :
  - historiques de commandes
  - alias
  - gestion des processus pour suivre les travaux

- **tcsh** est une amélioration de csh.

- Caractéristiques :
  - compatibilité ascendante avec csh
  - amélioration de l'historique avec la gestion des curseurs
  - variables d'environnement plus nombreuses

Activation de shell

Activation par défaut

- Au démarrage d'une session, on se trouve sous un interpréteur de commandes (avec fermi, c'est le **tcsh**).
- Le fichier **/etc/passwd** contient l'interpréteur par défaut
- L'interpréteur courant est défini par le contenu de la variable d'environnement **SHELL**.

```
%echo $SHELL
```

Activation en session

- Au cours d'une session, donc sous un interpréteur de commandes, on peut créer un nouveau processus et activer un autre interpréteur de commandes
- Le processus fils du processus précédent exécute le nouvel interpréteur.
- On termine le processus fils et on retourne sous l'ancien interpréteur par : **crtl/d, exit, logout, …**

Initialisation d'une session

- A chaque début de session, avec un interpréteur de commande quelconque, on exécute une liste de commandes contenue dans un fichier.

`.login` pour **`csh`**
`.profile` pour **`sh`**

Initialisation de processus

- Pour tout nouvel interpréteur de commande, on exécute une liste de commandes pour repositionner le contexte du processus correctement

`.cshrc` pour **`csh`**

Processus et fichiers

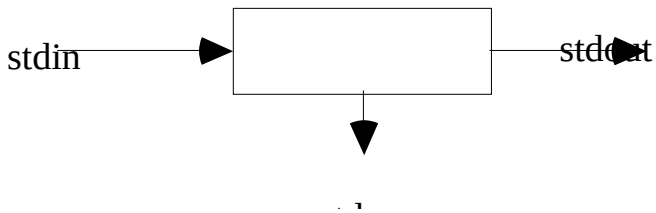
- Chaque programme doit être capable d'accepter des entrées à partir de n'importe quelle *source* et d'effectuer ses sorties vers n'importe quelle *cible*.

- **Idée** : l'input et l'output n'ont pas à être précisé par le programmeur

- A l'exécution du programme, c'est le **shell** qui connectera les *source* et *cible* appropriées aux E/S.

- D'où l'idée d'une source générale pour les entrées : *standard input* (**stdin**)
  et d'une cible générale pour écrire les résultats : s*tandard output* (**stdout**)
  à cela s'ajoute la sortie standard pour les messages d'erreur : **stderr**.

- Tout processus Unix fonctionne avec trois fichiers au minimum



- Par défaut, ces trois fichiers sont assignés au terminal de l'utilisateur.

- Leur descripteur (unité logique) sont 0, 1, 2

- Pour sauvegarder les résultats d'un programme, il faut signifier au **shell** de diriger le *standard output* vers un fichier particulier.

Redirection du standard output

Opérateur de redirection : >

- Pour sauvegarder le fichier **/etc/passwd** trié selon l'UID dans le fichier **mot_de_passe** :

```
%sort -t: /etc/passwd>mot_de_passe
```

remarque :

- si le fichier n'existe pas, il est crée
- si le fichier existe, il écrase son contenu

```
%ls -l >mot_de_passe
```

Opérateur de concaténation : >>

- Pour concaténer des données à un fichier existant :

```
%sort -t: /etc/passwd>>mot_de_passe
```

remarque :

- si le fichier n'existe pas, il est crée

Protection

- L'utilisation de > au lieu de >> peut provoquer des accidents

- Pour se protéger de tels accidents, on positionne une variable du C-shell **noclobber** :

```
%set noclobber
```

- Le shell ne remplacera pas un fichier existant si l'on utilise >, ni ne créera un fichier avec >>. Le shell se conformera exclusivement à l'intention du programmeur.

- Malgré le positionnement de la variable noclobber, il est possible de forcer la protection : **>!**, **>>!**

- Pour supprimer la protection :

```
%unset noclobber
```

Redirection du standard input

Opérateur de redirection : <

- La lecture des données est réalisée à partir d'un fichier quelconque

```
%who > toto
%cat toto
%sort <toto
%wc <toto>tata
%cat tata
```

Schéma d'exécution des commandes

Commandes

- Dans la plupart des cas, une commande est un utilitaire compilé et relié dans un fichier binaire qui doit être chargé.

- L'interpréteur utilise pour cela les mécanismes Unix :

```
fork
exec
wait
exit
```

- Une commande est donc le nom d'un fichier binaire exécutable appartenant à l'un des répertoires connus (variable PATH).

Commandes internes

- Dans le cas des commandes internes, l'interpréteur exécute par lui même les commandes.

- Il peut le faire immédiatement parce qu'il s'agit d'une commande de type structure de contrôle du langage ou d'un utilitaire associé.

```
%set     -- donne l'environnement en variables
             -- locales utilisateur
```

Liste de quelques commandes internes

```
setenv  affectation variable globale
foreach boucle pour
goto    branchement inconditionnel
history liste des commandes récentes
alias   création d'alias de commande
repeat  répétition de commandes
switch  aiguillage
while   boucle tant que
```

Création et terminaison des processus Unix

- Les opérations sur les processus sont basées sur 4 primitives qui ont un analogue dans le C-shell : **fork**, **exec**, **exit**, **wait**.

Fork

- Permet à un processus (le **père**) de créer un autre processus (le **fils**) par **copie** de lui-même.
- Elle est utilisée, en particulier, à l'exécution d'une commande (cf transparent suivant)

Exec

- Remplace le code (image binaire) d'un processus en exécution par un autre code (image binaire) et reprend l'exécution au début du nouveau code.
- Exemple : `%exec csh`

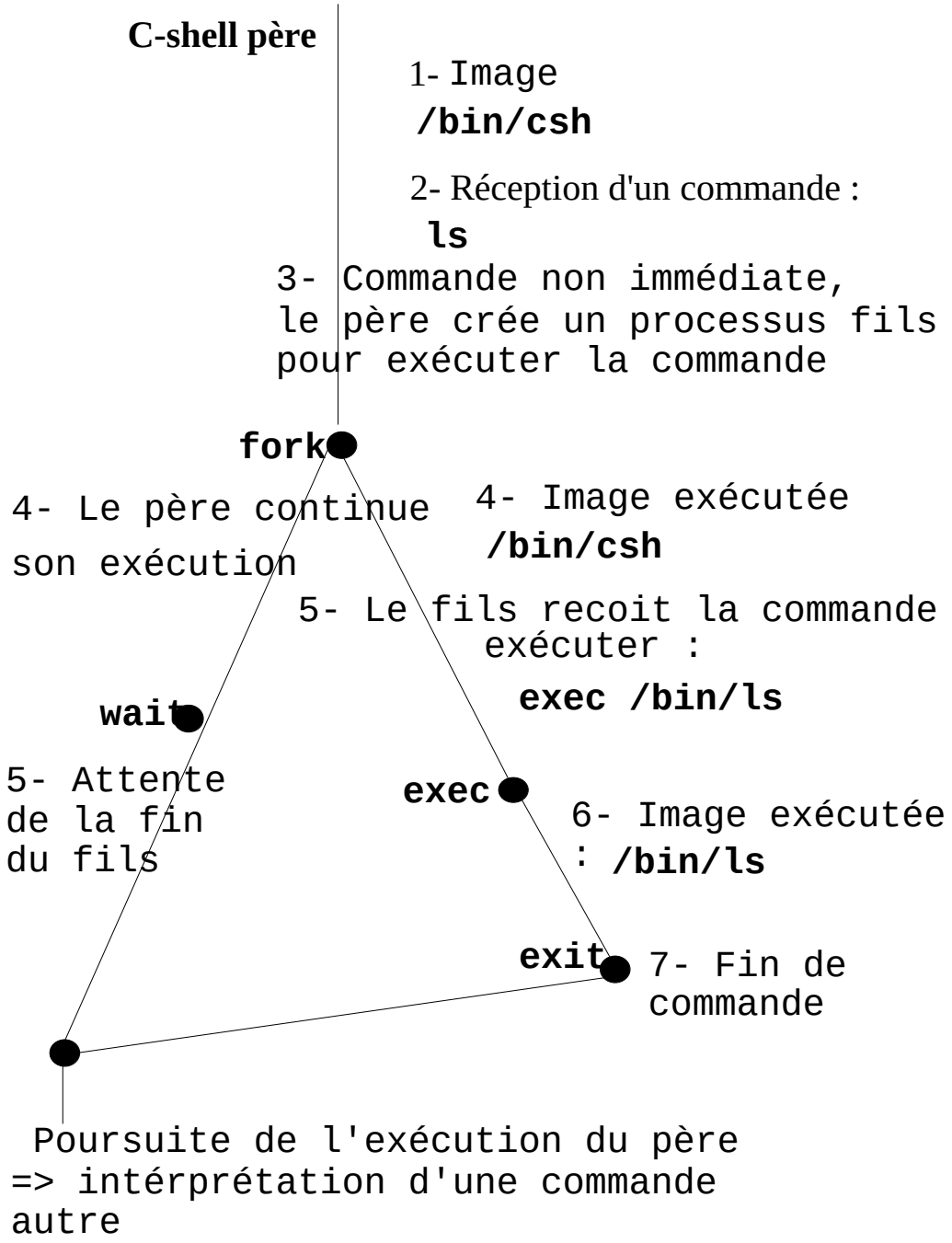
Exit

- Commande la **terminaison** d'un processus
- Un **code de retour** est associé à la terminaison du processus (**exit status**)
- La fin normale produit la valeur 0.
- Exemple : `%exit` (fin de session)

Wait

- Suspend l'exécution d'un processus et attend la terminaison de l'un de ses fils.
- Exemple : `wait` (dans un procedure)

Exécution standard d'une commande



Remarque :

- Le processus qui exécute la commande n'est pas le shell
- L'environnement du processus fils est nécessairement fixé par le père (fichier de commande .cshrc)

Exécution en parallèle d'une commande

- On peut exécuter une commande Unix sans que le père ne se bloque en attente de la fin de la commande.

Syntaxe

<nom_de_la_commande> **&**

exemple :

```
%xclock&
```

- Après le lancement de la commande, deux processus s'exécutent en parallèle

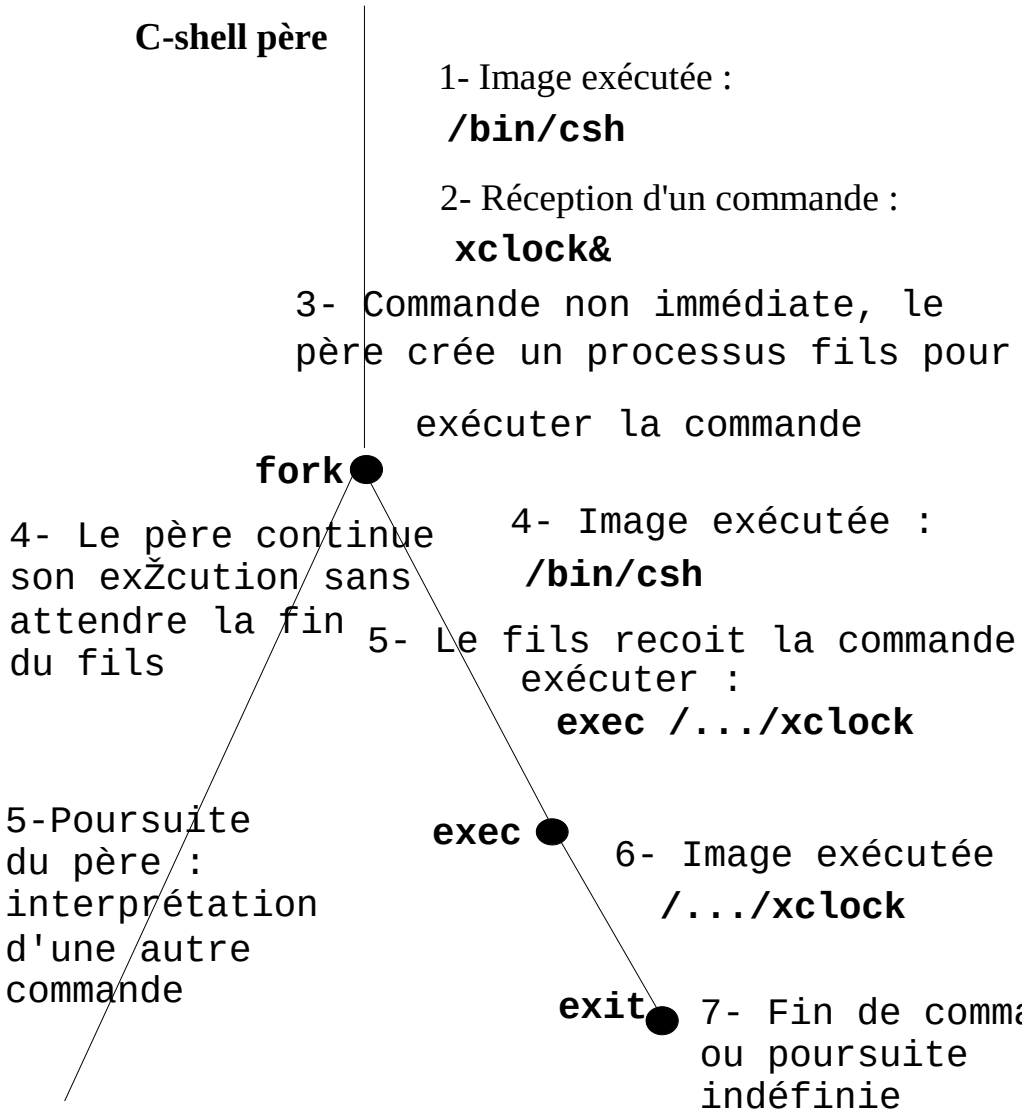
Processus d'avant plan (*foreground*)

- L'usager dialogue avec le processus d'avant plan au moyen de son terminal. Le processus qui tourne est le csh à partir duquel la commande d'arrière plan a été lançée.

Processus d'arrière plan (*Background)*

- Il exécute la nouvelle commande
- Il peut s'exécuter aussi longtemps que le processus père est actif (un même processus père peut avoir plusieurs fils)
- Il ne peut dialoguer avec l'usager sur le terminal standard réservé au **fg**

Schéma d'exécution en mode d'arrière plan



- Sous interface graphique, la gestion des processus d'avant plan et d'arrière plan est assurée par le multi fenêtrage.

Contrôle d'exécution des processus en mode avant plan, arrière plan

jobs

- Donne la liste de tous les processus dans l'environnement utilisateur avec un numéro d'identification entier.

crtl/z

- Suspend le processus d'avant plan

stop %numéro

- Suspend n'importe quel processus défini par son numéro

fg %numéro

- Ramène en avant plan le processus arrière plan identifié par son numéro

bg %numéro

- Place en arrière plan le processus arrière plan identifié par son numéro

Exercice

- Créer deux processus avant plan et arrière plan, éditer la liste des jobs, suspendre le processus d'avant plan, rappeler le processus d'arrière plan puis terminer les deux processus.

Commandes générales de manipulation de processus

ps

- Donne la liste des processus existants selon leur identification interne dans le système

- **Options**

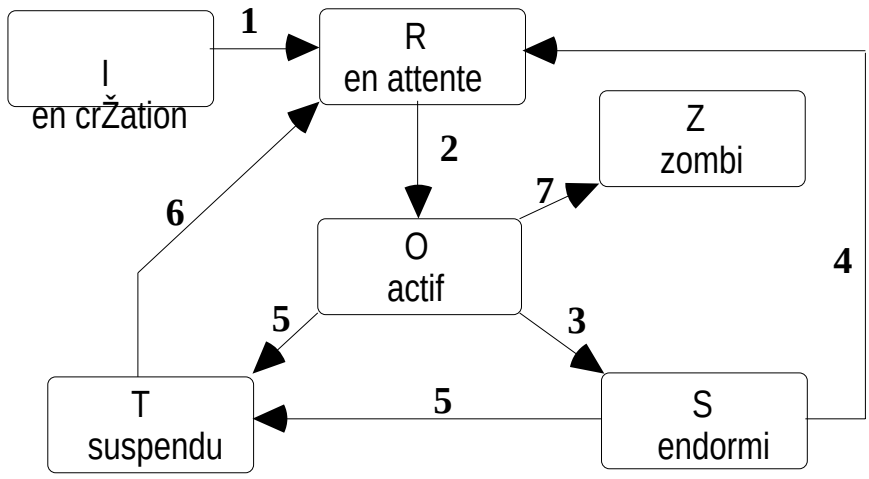
  -l informations détaillées
  -a informations les plus généralement demandées
  -x informations concernant les processus en exécution attachés à un même terminal

- **Informations fournies**

  PID identification
  PPID identification du père
  TT terminal de rattachement
  S état (status)
    O en exécution
    S endormi (en attente d'un évènement pour terminer)
    R en attente dans la queue d'exécution
    I en création (idle)
    Z zombi (le processus termine, les parents ne l'attendent pas)
    T suspendu
  C informations pour l'ordonnancement
  PRI priorité (nb élevé=basse priorité)
  NI valeur utilisée pour le calcul de la priorité
  ADDR adresse mémoire du processus
  SZ taille de l'image swappable du processus
  WCHAN adresse d'un événement pour lequel le processus est endormi
  STIME heure de départ du processus
  F Flag hexa associé au processus donnant des informations sur l'occupation mémoire, …

Les différents états d'un processus

- Le système d'exploitation ayant pour fonction de partager ses ressources, un processus ne peut pas être constamment actif.
- Un processus passe donc, au cours de son exécution, par différents états.



1 Le processus a acquis les ressources nécessaires à son exécution

2 Le processus vient d'être élu par l'ordonnanceur

3 Le processus se met en attente d'un évènement

4 L'évènement s'est produit

5 Délivrance d'un signal (SIGSTOP, SIGTSTP)

6 Réveil du processus par un signal (SIGCONT)

7 Le processus se termine

Commandes générales de manipulation de processus

kill

- Permet de détruire un processus en lui envoyant un signal de n° n

- Exemple

```
%kill -9 1089  -- tue le processus 1089
               -- en toutescirconstances
```

nice

- Permet de définir une priorité au processus

- Exemple

```
%nice +20 appli  -- la priorité de appli
                 -- est augmentée de
la               -- valeur 20
```

- La priorité 10 est recommandée pour les programmes utilisateur dont le temps de traitement est important.

- La priorité d'un processus s'étage de la valeur -20 (le plus prioritaire) à la valeur +20 (le moins prioritaire).

- La priorité d'un processus est transmise à ses fils par la fonction `fork`

sleep, at "hour"

- `sleep` permet la suspension d'un processus pendant quelques secondes, `at "hour"` l'exécution à une heure absolue.

- Exemples

```
%sleep 5
%at "11am"
```

Enchainement parallèle des commandes : les tubes

- Nous avons vu comment enchainer séquentiellement les commandes

```
%ls /usr;cd
```

- Nous avons vu comment rediriger les entrées ou les sorties

```
%(ls;pwd)>toto
```

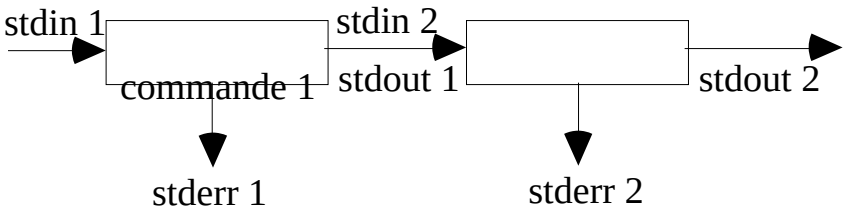
Les tubes

- Les commandes tapées sur une même ligne et séparées par une barre verticale ("*pipe*") sont exécutées en **parallèle**

- La sortie standard de la commande précédente est l'entrée standard de la commande suivante

- Exemple

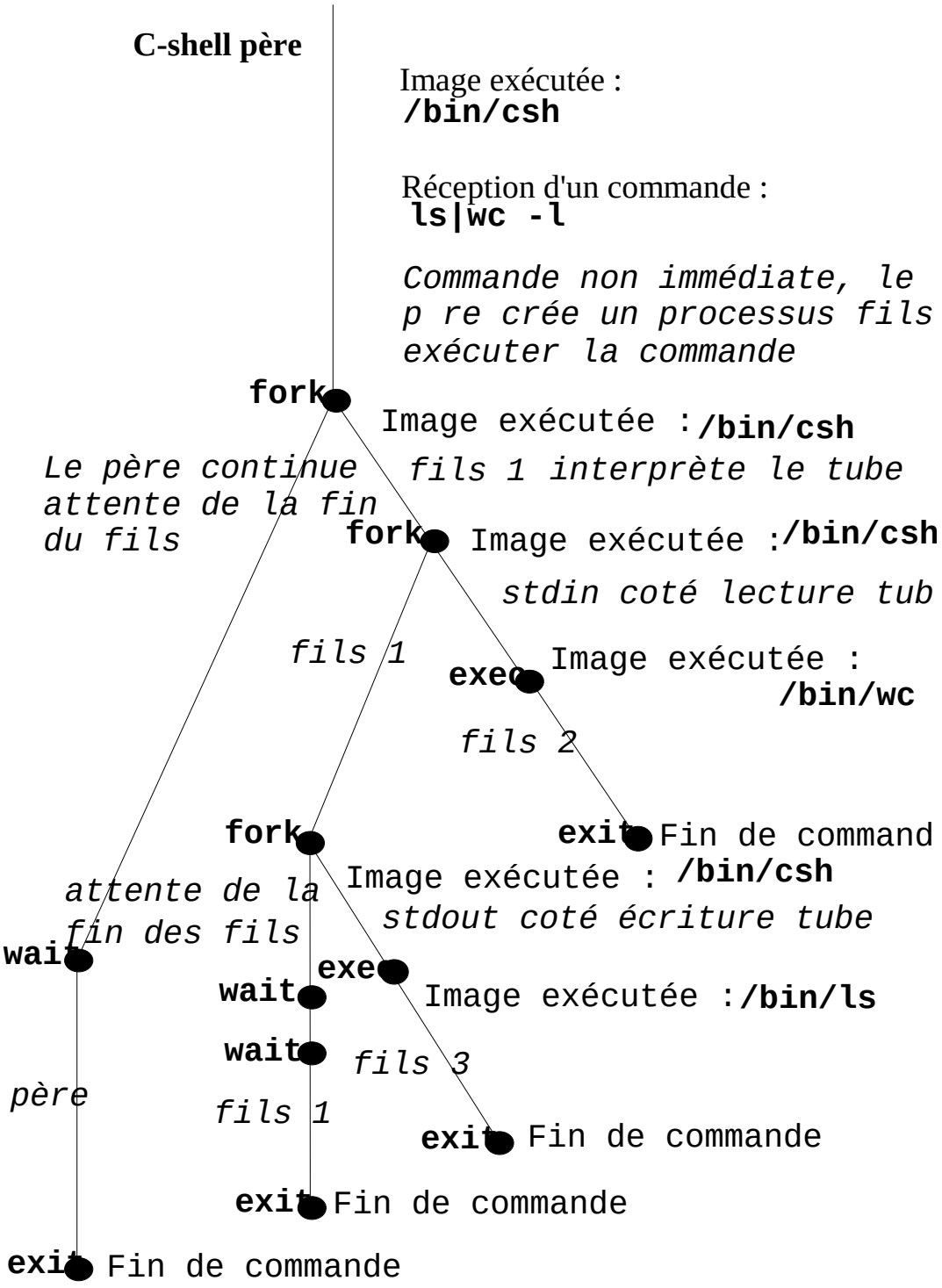
```
%who|sort    -- affiche la liste alphabétique
                -- des utilisateurs coonectés
```



Exercice

- Comptez en une seule ligne le nombre de vos fichiers (utiliser la commande `wc`)

Exécution avec tube



Dérivation en T

- Chaque ligne de commande , comprenant un ou plusieurs tubes, ne possède qu'une seule entrée standard et une seulesortie standard

- Si l'on veut observer ce qui se passe dans le tube, il faut employer un utilitaire.

tee nom_de_fichier

- Cet utilitaire permet de recopier l'entrée standard sur la sortie standard en écrivant aussi les informations sur un ou plusieurs fichiers

- Exemple

```
%ls|wc -c|tee toto tata
```

Exercice

- Ecrire une ligne de commande avec deux tubes qui liste dans un fichier le nombre de lignes d'un répertoire quelconque

Programme de commandes

- Les commandes peuvent être préparées dans un fichier formant un programme dont l'exécution est obtenue en évoquant simplement le nom du fichier.

- Ces programmes de commandes se nomment aussi :
  - procédure de commandes
  - macro commande
  - "script shell"

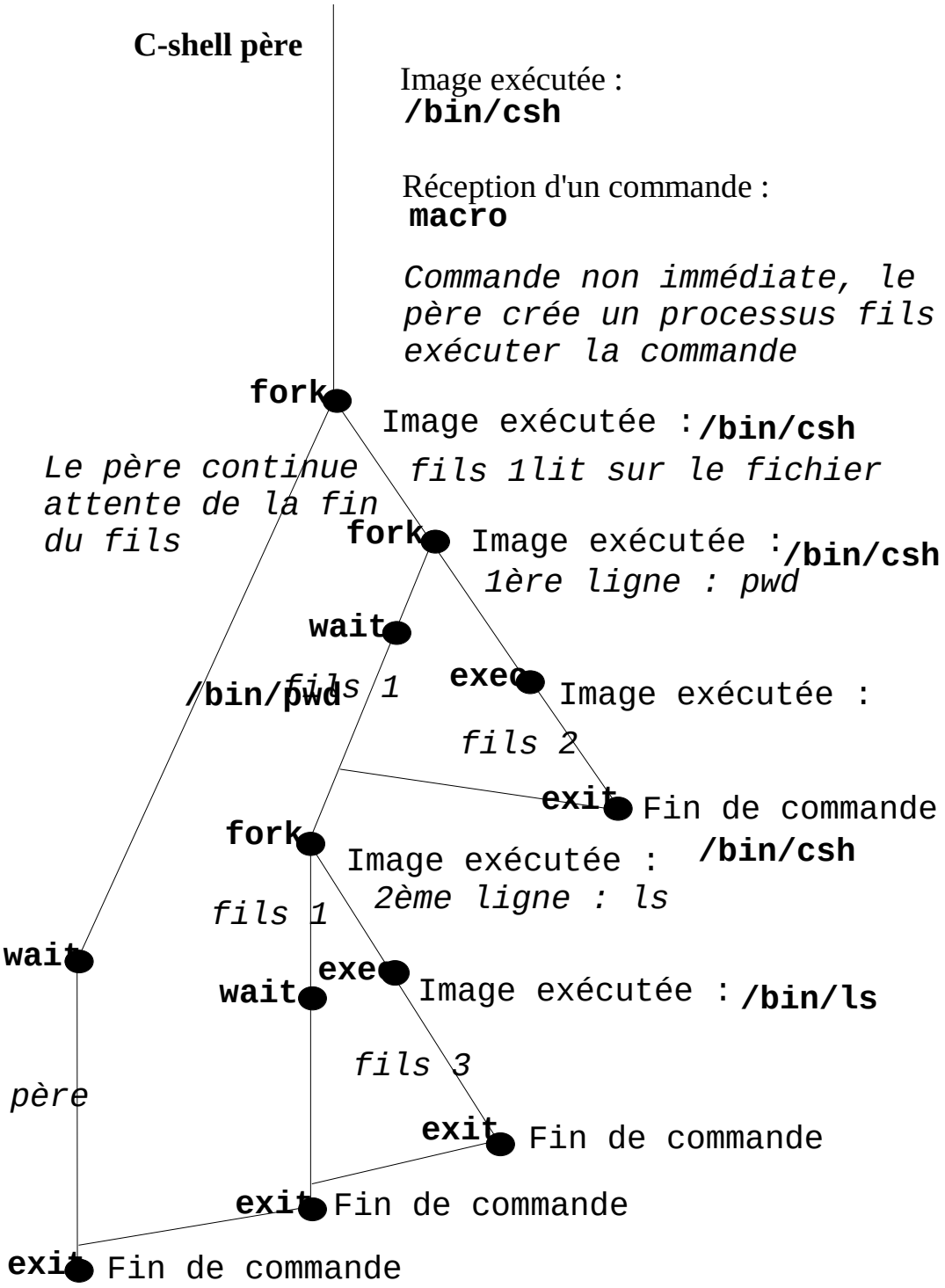
- Pour que le fichier soit réellement exécutable, il faut que la protection "execute" soit positionnée.

- L'interpréteur d'un fichier de commandes est le shell (pour nous csh)

- Pour forcer l'interprétation par C-shell en toute circonstances, il faut commencer le fichier par une ligne de commentaires

```
#Ceci est une procedure csh
```

Exécution d'une procédure de commande



Les variables

Variables locales et globales

- Les variables **locales** sont celles qui sont connues dans l'**environnement** dans lequel elles ont été définies.

- Les variables **globales** sont **exportées** dans l'environnement des fils.

Les identificateurs de variables

- Les noms de variables sont formées de caractères **alphanumériques** et du caractère **_**

Types de variables

- 4 types :
  le type **booléen** (ex : `noclobber` est de type booléen)
  le type **chaines de caractères**.
  le type numérique **entier**
  le type **tableau**

Quelques variables définies au niveau système

ignoreof	On ignore crtl/d
noclobber	L'écriture en redirection sur un fichier existant est impossible
nonomatch	la commande est effectuée même si les mécanismes de substitution donnent une chaine vide. Il n'y a pas de message nomatch sur le terminal.
argv	Liste des paramêtres
#argv	Le nombre de paramêtres
path	Liste des répertoires où chercher
prompt	L'invite
status	Le code de retour de la dernière commande
$	Le n° du processus shell en cours

L'affectation

Affectation d'une variable booléenne locale

- Affectation à vrai d'une variable booléenne locale ident

```
%set ident
```

Affectation d'une variable chaine locale

- Une chaine entre quotes est une chaine brute, chaque caractère est intepréte simplement comme un caractère

```
%set ident='Le loup $et le chien'
%echo $ident
%Le loup $et le chien
```

- Une chaine entre guillemets est une chaine où les caractères spéciaux sont intepretés. Le caractère $ permet de récupérer le contenu d'une variable.

```
%set a='Le loup'
%set b="$a et le chien"
%echo $b
%Le loup $et le chien
%echo "${a} et le chien"
%Le loup $et le chien
```

Affectation d'un numérique local

```
%set x=15
%@y=30
%@z=$x+$y+1
```

Affectation d'une variable exportable

```
%setenv ident valeur
```

Suppression d'un variable d'environnement

```
%unset
```

Edition de l'environnement local

```
%set   ou bien  %@
```

Ordre d'interprétation des commandes

4 étapes

1 Lecture d'une ligne complète

- La décomposition est effectuée au moyen des opérateurs de base : *espace tab ; carriage return*

2 Vérification syntaxique initiale

- Mise en place des redirections de fichiers
- Si la commande n'existe pas, la redirection est malgré tout interpété
- Exemple

```
%une_commande>toto
command not found
```

mais le fichier `toto` a été crée

3 Ordre d'application des substitutions

- Substitution de variables
- Substitution de commandes
- Expansion des noms de fichiers

4 Exécution de la commande

Substitution de commandes

- Au sein d'une chaine de caractères, il est possible de rendre une sous chaine interprétable comme une commande.
- Pour obtenir la substitution, il faut placer la sous chaine entre "*back quotes*"
- Exemple

```
%set liste_fich="`ls`"
%echo $liste_fich
```

Expansion des noms de fichiers

- Dans un nom de fichier :
  - ***** vaut pour n'importe quel chaine
  - **?** vaut pour n'importe quel caractère
  - **[-]** vaut pour un intervalle (ex **[0-5]**)

Exercices

- Création d'un prompt personnalisé (utiliser la variable d'environnement `prompt`).

- On pourra utiliser :
  - la variable `hostname` pour afficher le nom de la machine
  - la commande `who i am` pour afficher le nom de l'usager
  - la commande `date` pour afficher la date du jour
  - `\!` le n° de la commande

- On pourra aussi indiquer le répertoire courant dans l'invite

Les structures de contrôle

Branchement inconditionnel

- Définition d'une étiquette
```
etiq :
```

- Branchement
```
goto etiq
```

Branchement conditionnel

```
if (expression logique) commande_simple
```

```
if (expression logique) then
     liste_de_commande_1
else
     liste_de_commande_2
endif
```

```
if (expression logique 1) then
     liste_de_commande_1
else if (expression logique 2) then
     liste_de_commande_2
else if (expression logique 3) then
     liste_de_commande_3
endif
```

Itération bornée

```
foreach variable(liste de valeurs)
     liste_de_commande
end
```

Itération non bornée

```
while (expression booléenne)
     liste_de_commande
end
```

Les structures de contrôle

Aiguillages

```
switch (valeur)
    case valeur1 :
        liste_de_commande 1
        breaksw
    case valeur1 :
        liste_de_commande 1
        breaksw
    ...
    default :
        liste_de_commande 1
endsw
```

Opérations sur les fichiers

-r fichier	vrai si l'on peut lire
-w fichier	vrai si l'on peut écrire
-x fichier	vrai si l'on peut exécuter
-e fichier	vrai si le fichier existe
-d fichier	vrai si c'est un répertoire
-f fichier	vrai si c'est un fichier ordinaire
-z fichier	vrai si le fichier est de taille nulle
-o fichier	vrai si l'on est propriétaire du fichier

Opérateurs sur les variables

`>  <  >=  <=  !=  ==`

Exercice

- Ecrire une macro qui utilise un fichier et qui teste avant son utilisation s'il existe sinon envoie un message d'erreur

Les structures de contrôle

Les paramêtres d'appel

- Une commande est presque toujours associée à des options ou des paramêtres d'appel
- La procédure de commande doit pouvoir récupérer ces paramêtres pour orienter l'exécution
- Le tableau des paramêtres est dans la variable `argv`
- La variable `#argv` en donne le nombre

Exercice

- Ecrire une macro (à nombre variable de paramêtres) qui imprime son premier paramêtre et le nombre de ses paramêtres.

Récursivité

- Une procédure de commande peut s'appeler elle-même en évoquant son nom dans son code

Exercices

- Ecrire une macro qui édite la date en français (la commande `date` l'édite en anglais)
- Ecrire une commande à deux paramêtres dont le premier est un répertoire et le second est une commande à exécuter récursivement dans tous les répertoires sous le répertoire premier paramêtre.

Généralités

- Un processus est un programme en exécution (**entité dynamique**)

  => nécessité de **ressources** : cpu, mémoire, structures de données, routines, etc, …)

  => L'exécution s'effectue dans un contexte (**environnement** déterminé)

- Les processus s'exécutent en **parallèle** et échangent des **données** et des **signaux**

- Un processus est constitué :

du programme (**code exécutable**)
des **données** que le programme manipule
du **contexte** d'exécution (bloc de contrôle du processus)

- Le module du noyau appelé **ordonnanceur** (scheduler) est chargé d'allouer la **ressource** *processeur* aux différents processus prêts à exécuter une instruction.

–

Mode d'exécution

Mode utilisateur

- Moindre privilège
- Durée de vie limitée
- Utilise ses propres ressources sans accéder celles du noyau

Mode système

- Pour exécuter du code système. Ils sont **ininterruptibles**.
- Les processus système ne sont sous le contrôle d'aucun terminal et ont comme propriétaire le super utilisateur, *processus démons*
- Le passage en mode système s'effectue à partir d'**évènements**

Evènements internes

- Les **appels systèmes**
- Les **trappes** (pour les traitements exceptionnels tel que les violations de segment, les erreurs bus, …)

Evènements externes

- Ce sont les **interruptions** (asynchrone) qui peuvent se produire à tout moment de l'exécution du processus.

Exercice

- Voir quel est le premier processus exécuté à la connexion.

Caractéristiques

Création d'un processus

- La création de tout processus (excepté le processus d'identification) est le résultat de l'exécution, par le système, à la demande d'un autre processus, de la fonction de création (*fork*). Le processus ainsi crée hérite ainsi d'un certain nombre de caractéristiques de son père.

Bloc de contrôle

- Il contient l'ensemble des carctéristiques d'un processus
- Il est divisé en deux parties :

  - la première contient les informations nécessaires au système même lorsque le processus n'est pas actif (identité, calcul de priorité, …)

  - la seconde contient les informations nécessaires uniquement lorsque le processus est actif

Réentrance

- Les codes programmes produits par l'éditeur de liens sont par défaut réentrants.
- Cela signifie que :

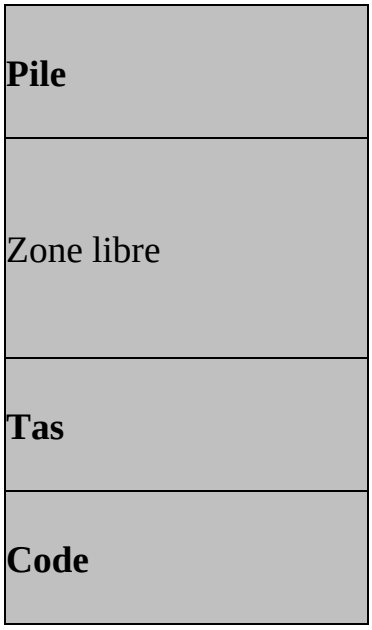
  - Un seul exemplaire du programme en M.C. pour plusieurs processus correspondant à l'exécution du même programme.

  - La zone programme du processus est protégée en écriture.

  - Les zones de données sont séparées (même si elles peuvent être partagées)

  - Chaque processus possède un espace d'adressage

Organisation mémoire



- La pile contient les objets manipulées dans le programme
- Le tas (*heap*) contient les objets crées dynamiquement
- Ces deux zones mémoire augmentent leur taille en allant l'une vers l'autre. En cas de dépassement, une image mémoire (*core*) du processus est enregistrée sur dique. Le message *core dumped* est affiché.