

Interprêteur de commandes (1)



Pour interagir avec l'utilisateur, un système informatique doit disposer au minimum d'un **interprêteur de commandes** (shell) couplé à une **interface** rudimentaire.²

Le *shell* attend les ordres que l'utilisateur transmet par le biais de l'interface, décode et décompose ces ordres en actions élémentaires, et finalement réalise ces actions en utilisant les services des couches plus profonde du système d'exploitations.

Le *shell* est donc un traducteur de type interprêteur, et le **langage de commande** qu'il prend en compte est un langage de programmation interprêté.

interface (terminal)	interface (fichier)		interface (console)		
shell (p.ex. cs	shell (p.ex. csh)		shell (p.ex. bash)		
couches basses du sys	stème				

Parmis les shells Unix les plus utilisés, citons: Bourne [Again] shell (sh et bash), C shell (csh), Korn shell (ksh), Z shell (zsh), et celui présent par défaut sur les comptes du cours, l'Enhanced C shell (tcsh).

^{2.} Contrairement à l'architecture figée de Dos ou Windows, les systèmes de la famille Unix définissent en tant que composants externes l'interface utilisateur et l'interprêteur de commandes. Ne faisant pas directement partie du système, ils peuvent être changés si nécessaire.



Interprêteur de commandes (2)



Pour faciliter le travail de l'utilisateur, les interprêteurs de commandes offrent la possibilité de **définir des variables d'environnement**, de **renommer les commandes**, d'en définir de nouvelles, par «paramètrage» et enchaînement des commandes de base, etc. La plupart offrent également des **facilités d'édition** comme le rappel des commandes précédentes (historique des commandes), la complétion, la correction (suggestion de correction) en cas de commande invalides, et bien d'autres fonctionnalités.

Tous les systèmes d'exploitation permettent par ailleurs, en plus de l'interaction «directe» (au moyens de terminaux ou de consoles dans le cas d'*Unix*), le «*traitement par lots*» (*batch*).

Ce mode de traitement non-interactif est obtenu en regroupant les comandes dans un fichier alors appelé *script*.



Mémoire virtuelle (1)



La mémoire centrale a toujours été une ressource critique: initialement très coûteuse et peu performante (torres magnétiques), elle était de très faible capacité.

Avec l'apparition des mémoires électroniques, la situation s'est améliorée, mais comme parallèlement la taille des programmes considérablement augmenté, la mémoire demeure, même de nos jours, **une ressource critique**, et donc à économiser.

Pour pallier le manque de mémoire centrale, l'idée est venue d'utiliser des **mémoires secondaires** [de type disque dur], **plus lentes**, mais de beaucoup **plus grandes capacités**.

Pour cela, un concept à la fois simple et élégant fut établit:

la **mémoire virtuelle**.³

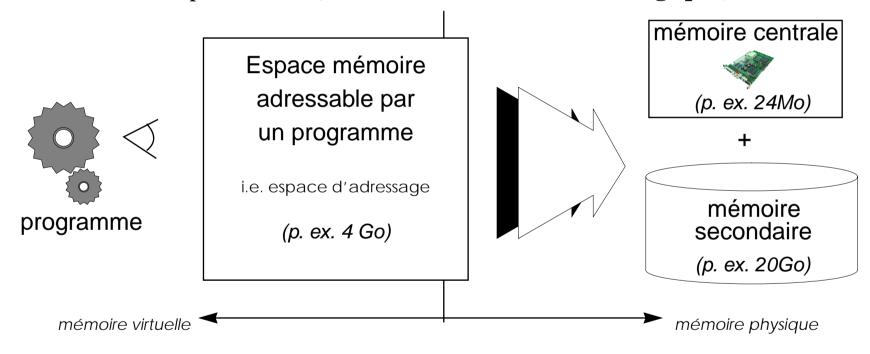
^{3.} Le concept de mémoire virtuelle est présenté dans le cadre des systèmes d'exploitation, car historiquement c'était le S.E. qui en était chargé. Pour des raison de performances, une grande partie du travail est maintenant réalisée par microprogrammation, directement dans les processeurs.



Mémoire virtuelle (2)



La mémoire virtuelle repose sur une **décorellation** entre la **mémoire physique** (centrale ou secondaire), présente sur la machine, et **l'espace mémoire mis à disposition des programmes** par le système d'exploitation (la **mémoire virtuelle**, ou **logique**).



C'est le système d'exploitation qui prend en charge la **mise en correspondance** entre les adresses utilisées dans les programmes (*adresses virtuelles*) et les adresses correspondant effectivement à de la mémoire physique (*adresses réelles*).

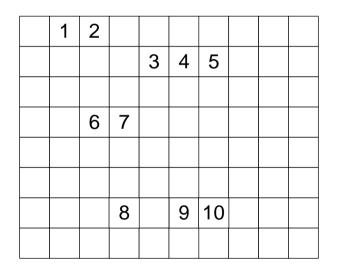


Mémoire virtuelle (3)

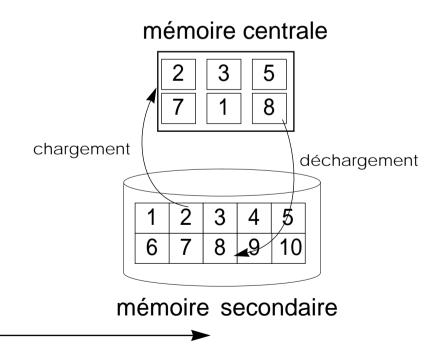


Une technique fréquemment utilisée pour réaliser cette mise en correspondance est la **pagination**.

La mémoire virtuelle utilisable par un programme (aussi grande que le permet la taille du codage des adresses) est **segmentée** en zones de tailles identiques, les **pages**. A tout instant, **seul un nombre limité** de ces pages est effectivement (i.e. physiquement) présent dans la mémoire centrale, le reste [des pages allouées, i.e. utilisées] étant conservé en mémoire secondaire, et **chargé** en mémoire centrale en cas de besoin.



Mémoire virtuelle





Mémoire virtuelle (4)



La mémoire centrale ne pouvant contenir qu'un nombre limité de pages, il faut donc parfois effectuer des **remplacements** (swapping).⁴

Quelle stratégie employer pour réaliser ces remplacements ?

Dans le cas idéal, il faudrait remplacer les pages qui ne seront plus utilisées, du moins celle qui le seront le plus tard possible... Malheureusement, il n'est en général pas possible pour le système de prévoir le déroulement des processus.

Une stratégie largement répandue est celle du LRU (Least Recently Used), et consiste à remplacer la page la moins récemment utilisée. ⁵

Cette heuristique est basée sur l'observation suivante: ⁶

«les références mémoires d'un processus restent groupées, et évoluent lentement dans l'espace d'adressage»

^{4.} On parle de défaut de page lorsque survient un référencement à une page non présente en mémoire centrale.

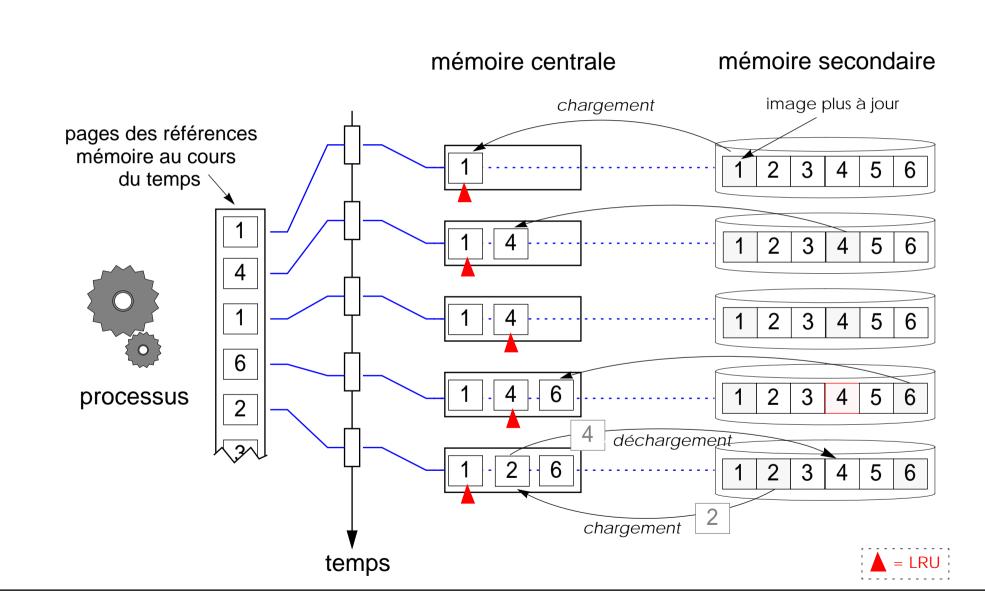
^{5.} Parmis les autres algorithmes courants, citons FIFO, qui consiste à éliminer de la mémoire la plus ancienne page, LFU (*Least Frequently Used*) la page la moins fréquemment référencée, et ... RANDOM, qui choisit la page de manière aléatoire (algorithme sous-optimal de référence)!

^{6.} Egalement appelé «principe de localisation des références mémoire». Cette observation entre dans le cadre de la *règle des 80-20* (aussi désignée sous le nom de *règle des 90-10*); nous aurons l'occasion de revoir cette règle en détails dans la suite du cours, mais l'on peut d'ores et déjà donner son interprétation dans le contexte de l'utilisation mémoire: 80% (respectivement 90%) des références mémoires d'un programme sont faite dans les mêmes 20% (respect. 10%) de l'espace d'adressage total.



Mémoire virtuelle (5)







Mémoire virtuelle (6)



Naturellement, l'utilisation d'un mécanisme de mémoire virtuelle a un **coût**:

• En espace mémoire:

la gestion des pages implique l'utilisation de tables de descripteurs (informations sur la localisation des pages, validité de l'image disque, droits d'accès en fonction des processus, pile du LRU, ...), ce qui consomme de l'espace mémoire, et les pages ainsi gérées sont sujettes à la fragmentation interne (les pages ne sont souvent que partiellement utilisées, mais la totalité de l'espace mémoire correspondant est réservé).

• En temps de traitement: consommé par la gestion, le chargement et le déchargement des pages.







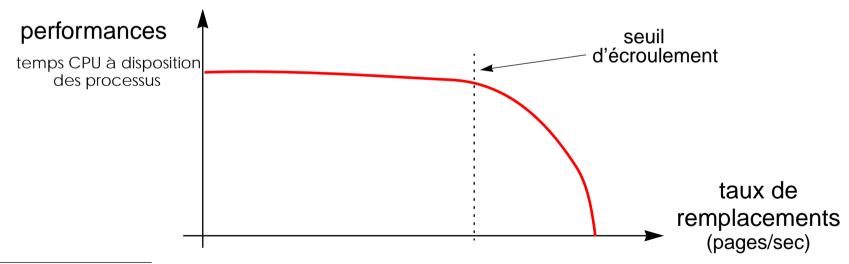
Mémoire virtuelle (7)



La **fréquence des remplacements de pages** est directement liée au rapport entre le nombre de pages pouvant être logée en mémoire et le nombre totale de pages allouables par un programme.

Lorsqu'un programme provoque des remplacements trop fréquents, on dit qu'il s'écroule.

Dans ce cas, il peut arriver que le système monopolise l'essentiel du temps pour les transferts de pages entre disque et mémoire, et ne permette plus aux programmes de progresser dans leur exécution, ce qu'il faut à tout prix éviter.



^{7.} La plupart des systèmes d'explopitation permettent de verrouiller des pages en mémoire, afin d'interdire leur remplacement. Cette possibilité permet de limiter le risque d'écroulement, typiquement dans le cas de processus temps-réel, en assurant la présence permanente dans la mémoire des pages «critiques» (i.e. périodiquement accédées)



Processus (1)



Les périphériques réalisent souvent des opérations lentes à l'échelle du processeur.

Considérons par exemple un programme qui effectue une requête de lecture de données sur un CD-ROM.



⇒ La première opération à faire, au niveau du système, sera de positionner la tête de lecture du CD-ROM sur le premier bloc à lire.

Admettons qu'un tel positionnement se fasse en moyenne en 50 [ms], et que le système soit doté d'un processeur de type RISC, cadensé à 500 [MHz].

Le temps de positionnement de la tête de lecture représente l'exécution de:

$$(50 \times 10^{-3})[s] \times (500 \times 10^{6})[op/s] = 25 \times 10^{6}[op]$$

soit 25 millions d'opérations

qui pourraient être exécutées en attendant la fin de l'opération d'entrée-sortie.

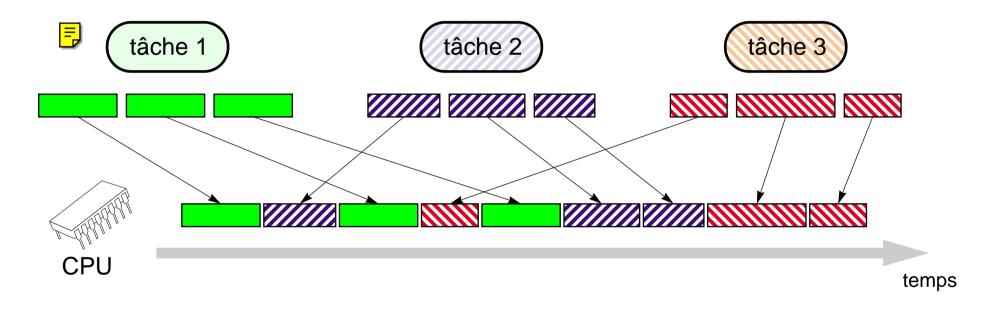


Processus (2)



L'idée est donc tout naturellement venue d'utiliser ce temps d'oisiveté au profit d'autres tâches se déroulant **en parallèle** des opérations d'entrée-sortie.

Cette idée a rapidement été étendue à une forme plus générale de parallélisme basé sur l'entrelacement de l'exécution des tâches (parallélisme logique).



On parle alors de systèmes à temps partagé [time sharing].



Processus (3)



L'un des rôles importants d'un système d'exploitation est de permettre la mise en œuvre effective du parallélisme logique.

Unix est ainsi un système qui permet de lancer similanément plusieurs programmes sur une même machine (mono ou multi-processeur):

On parle dans ce cas de système [ou environnement] multi-tâches.

Dans la suite, nous appelerons:

«processus» (process) ou «tâche» (task)

tout programme exécuté dans un environnement

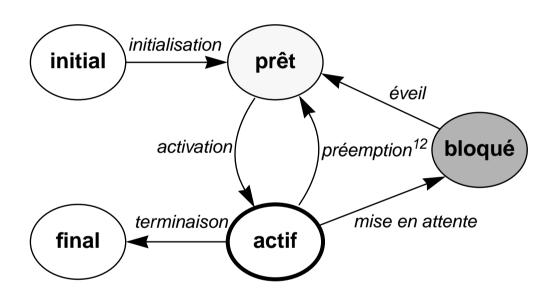
multi-tâches.



Processus (4)



De plus, on définit, pour tout processus, les états suivants: (cycle de vie)



- Les processus *prêts* sont les processus potentiellement exécutables;
- Les processus *actifs* sont ceux en cours d'exécution (par le processeur)
- Les processus *bloqués* sont ceux en attente de synchronisation (fin d'une opération d'entrée-sortie, ou plus généralement en attente d'un autre processus).

Sur un système monoprocesseur, il ne peut (évidemment) y avoir à tout instant qu'un et un seul processus actif.

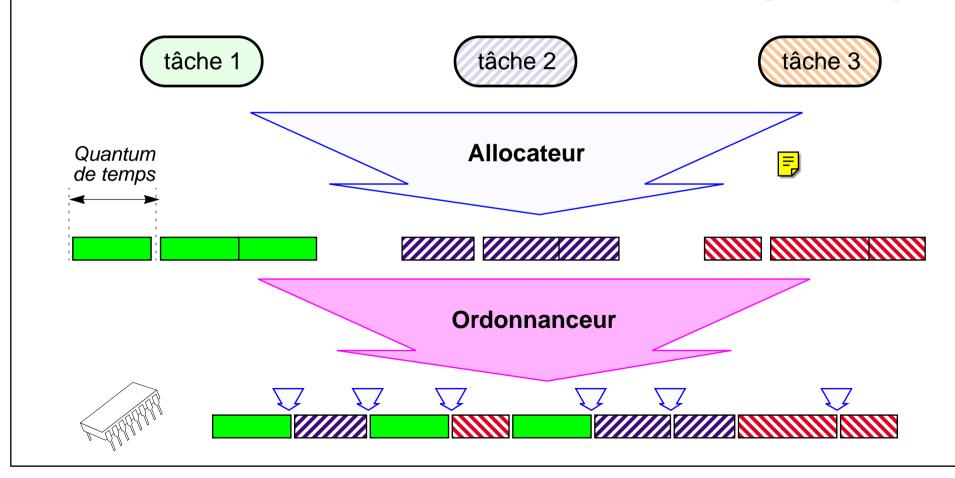
^{8.} La préemption correspond à l'interruption (temporaire) d'un processus, à la fin du quantum de temps qui lui était alloué.



Processus (5)



L'ordonnanceur (scheduler) est responsable du choix du processus à activer, parmis l'ensemble des processus prêts. L'allocateur (dispatcher) définit le temps maximum pendant lequel un processus pourra rester actif (le quantum de temps alloué au processus); il fixe les échéances des préemptions (i.e. des mise en veille temporaire des processus).



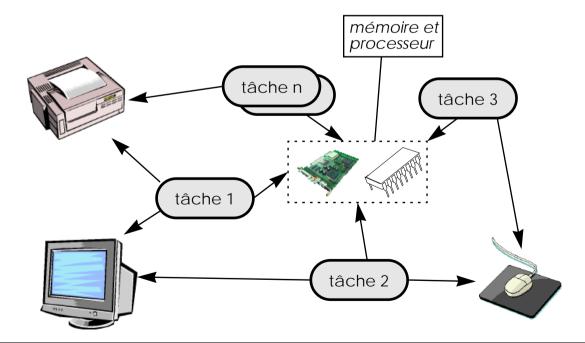


Processus (6)



La mise en œuvre d'un système multi-tâches implique que les processus puissent:

- s'exécuter comme s'ils étaient seuls sur la machine (monopole)
 - ⇒ concurrence d'accès aux ressources
 - coopérer avec d'autre processus s'exécutant simultanément
 - **⇒** communication
 - **⇒** échange de données
 - ⇒ nécessité de synchronisation.



Les systèmes multi-tâches sont ainsi des exemples de

systèmes concurrents,

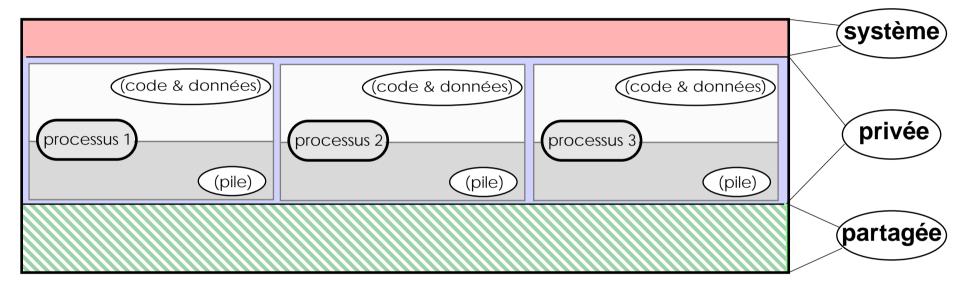
les tâches étant en compétition les unes par rapport aux autres pour l'obtention des ressources physiques



Concurrence d'utilisation mémoire



Pour régler les problèmes de concurrence d'utilisation mémoire, tout en permettant la coopération des processus, l'espace d'adressage est décomposée en zones: (par exemple)



- La zone «système» contient le noyau du système d'exploitation et les services de base.
- Une **zone privée** propre à chaque processus, dans laquelle le code et les données sont logées; il est assez aisé pour le gestionnaire de mémoire virtuelle de dupliquer cet espace pour chaque processus.
- Finalement, on trouve généralement un **espace commun** à tous les processus (accessible en écriture), permettant une communication appelée **communication par mémoire partagée**, ou *par boîtes aux lettres*)



Concurrence d'accès aux périphériques (1)



Une possibilité pour résoudre les concurrences d'accès est la **réservation** des périphériques par les processus devant en faire usage: le périphérique est alloué au processus, jusqu'à la terminaison de l'opération.

Se pose alors le problème de savoir à quel moment cette opération est terminée ?

Une première possibilité, et dans bien des cas la seule, est d'utiliser le principe du **drapeau** (*flag* ou *status*):

en début d'opération, une variable (drapeau) est positionnée à une certaine valeur; lorsque l'opération se termine, le périphérique modifie la valeur de ce drapeau.

Parallélement, le processus teste à intervalles réguliers la valeur du drapeau, jusqu'à ce que le périphérique signale la fin de l'opération.

On appelle ce mode de travail la *scrutation* (*polling*), et la mise en attente du processus sur une boucle de test est appellée: *attente active*.



Concurrence d'accès aux périphériques (2)



Afin de réduire le gaspillage de temps induit par les scrutations infructueuses, une autre solution à été envisagée, au niveau du matériel:

Au lieu de laisser le contrôle à la charge du processus invoquant l'opération, on va donner la possibilité au périphérique de signaler de manière active

la fin de l'opération (en fait tout événement digne d'être signalé), en interrompant le programme

en cours de traitement (*rupture de séquence*) au profit d'une routine spéciale (routine de service), destinée à effectuer le minimum d'opérations nécessaires à la prise en compte de l'événement.

Lorsque cette routine se termine, l'exécution du programme initial reprend.

L'occurrence d'un tel signal émis par le périphérique est appellée interruption.

La différence fondamentale entre la syncronisation par scruptation et celle par interruption réside au niveau du **passage de contrôle**:

- dans le premier cas, le programme garde le contrôle, et le «prête» explicitement au périphérique par le biais de la boucle d'attente;
 - dans le second cas, le contrôle *est pris* par le périphérique, de manière implicite vis à vis du programme.



Concurrence d'accès au terminal



Le fait que plusieurs programmes soient exécutés «simultanément» a une incidence sur les **interactions de ces programmes avec l'utilisateur**, en particulier pour ceux qui utilisent la console ou le **terminal** (i.e. les programmes qui n'ont pas d'interface de dialogue privée – ce sera le cas de la plupart de ceux écrits dans le cadre de ce cours).

Cependant, si le programme dispose de sa propre fenêtre d'interaction (c'est le cas de la plupart des applications – Netscape, Emacs – utilisées dans le cadre de ce cours), l'interaction dans le terminal se réduit à la commande de lancement du programme.

Dans ce cas, l'utilisateur peut lancer ce programme en **mode** «détaché» (on parle également de lancement en «arrière-plan»), par exemple en faisant suivre la commande du signe « &»⁹.

Le processus sera alors exécuté par le système, **indépendamment du terminal** qui aura servi à le lancer.

^{9.} CTRL+Z permet de suspendre le processus actif relié au clavier (il devient bloqué), tandis que CTRL+C l'interrompt définitivement. «bg», acronyme de *background*, permet de passer le processus en arrière-plan (son exécution reprendra, mais détachée du shell qui l'a lancé) tandis que «fg», acronyme de *foreground*, permet de ré-attacher (repasser au premier plan) le dernier processus détaché.



Processus, commandes pratiques



Exemples pratiques:

- Pour obtenir la liste des processus d'un utilisateur (user): \$>ps -fu <user> | more
- Pour obtenir de manière actualisée la liste des principaux processus, triés selon leur consomation CPU: \$>top

```
• Pour détruire un processus:
```

```
$>kill <PID>: on demande au processus son auto-destruction
$>kill -9 <PID>: destruction du processus commandée au niveau système
$>xkill [-frame]: permet de choisir à la souris l'application graphique à tuer.
```

```
[liasun20<del>/</del>178N<u>:</u>04pm /users/fseydoux]/bin/ps -fu fseydoux | more
    UID PID PPID C
fseydoux 1265 1070 0
                        Nov 06 pts/2
                                        0:02 audiocontrol -Wp 4 3
fseudoux 1112 1010 0
                        Nov 06 pts/2
                                        0:00 /usr/bin/tcsh -c kno
fseudoux
          902 29261 0
                                        0:00 /bin/ksh /usr/dt/cor
fseudoux
               1086 0
                         Nov 06 pts/2
                                        0:01 kscd -caption kscd
fseudoux
                                        0:00 /usr/bin/tcsh -c kvt
fseudoux 1073 1010 0
                                        0:00 /usr/bin/tcsh -c gkr
fseydoux 1406 1112 0
                                        0:02 knotes -knotes_resto
fseydoux 1483 1482
                                        0:00 (dns helper)
fseudoux 1009 956
                                        0:00 /bin/ksh /usr/dt/cor
fseudoux 1248 1231 0
                        Nov 06 pts/4
                                        0:01 tcsh
fseydoux 3060 1434 0 15:58:56 pts/5
                                        0:00 snapshot
fseydoux 918
                                        0:00 /usr/openwin/bin/spe
fseydoux 1405
              1093 0
                        Nov 06 pts/2
                                        1:21 kvt -restore kvtrc.2
fseudoux 2870 1248
                    0 14:38:36 pts/4
                                        0:23 ssh -l prog liasun20
        1054 1010 0
                                        0:00 /usr/bin/tcsh -c xte
fseudoux
fseydoux 1042 1010 0
                                        0:02 kfm
                        Nov 06 pts/2
fseydoux 1010 1009 0
                        Nov 06 pts/2
                                        0:22 kwm
fseudoux 954 1 0
                        Nov 06 ?
                                        0:02 /usr/dt/bin/dsdm
fseydoux 1469 1404 0
                                        0:00 libotop-server
```





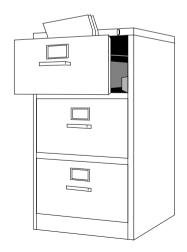
Fichiers



La capacité des mémoires auxiliaires – i.e. les mémoires secondaires (disques durs) et les mémoires de masse (CDs, bandes) – étant très grande, on peut y loger un grand nombre de données. A cette fin, les systèmes d'exploitation utilisent le concept de «fichier»:

Un *fichier* est un **regroupement logique de données** présente sur une mémoire secondaire, avec un début, une fin, une taille d'enregistrement de base, un ordre de lecture et un certain nombres d'attributs.

C'est donc une collection ordonnée de données, représentant une entité pour l'utilisateur.

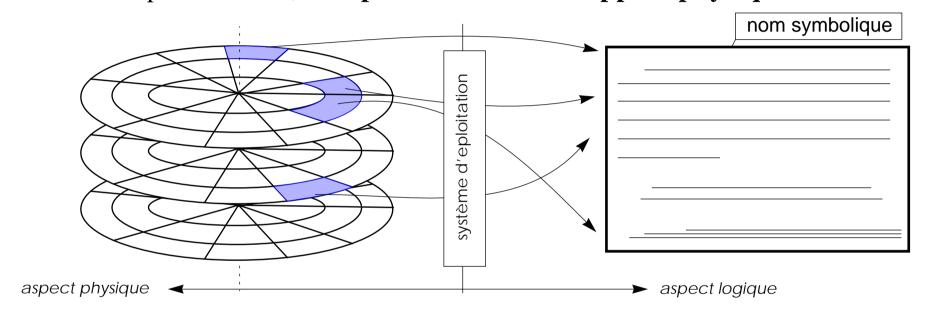




Organisation de la mémoire secondaire



L'un des rôles du système d'exploitation est de permettre la mise en œuvre du concept de fichier, **indépendamment du support physique utilisé**.



Le système d'exploitation doit donc permettre la **gestion** (création, destruction, modification et lecture) des fichiers, en offrant en particulier à l'utilisateur la possibilité de les désigner par des **noms symboliques**, sans avoir à se préocuper de leur organisation physique dans la mémoire auxiliaire..

Dans le cas de systèmes multi-utilisateurs, il faut de plus assurer la **confidentialité** des données, en protégant le contenu des fichiers du regard des autres utilisateurs.



Méthodes d'accès aux données



L'accès aux données stockées dans un fichier est contraint par la mémoire auxiliaire utilisée.

On distingue généralement deux grand types de fichiers, ¹⁰ suivant la nature des accès qu'ils permettent:

• les fichiers à accès séquentiel: suite d'enregistrements accessibles les uns après les autres. Pour lire une portion du fichier, il faut le parcourir depuis le début, jusqu'à atteindre la portion en question. (-> les bandes)

• les fichiers à acces directe: (aussi appelé fichiers à accès aléatoire) chaque enregistrement est accessible directement. Pour lire une portion du fichier, il suffit de connaître le déplacement (no de l'enregistrement concerné x la taille des enregistrements).

^{10.} Remarquons qu'il existe beaucoup d'autre types de fichiers (par exemple les fichiers relatifs, indexés, indexés à clefs multiples, ISAM, VSAM, etc). Toutefois, ces types «exotiques», apanage des «gros» systèmes (VMS, IBM) des années 80 ont tendance à disparaîtres avec eux.



Système de fichiers



Pour assurer la gestion des fichiers, un système d'exploitation utilise un [voire plusieurs] système[s] de fichiers (file system).

C'est le système de fichiers qui détermine les structures internes à employer pour organiser les fichiers.

Parmis les très nombreux systèmes, citons:

- FAT, VFat, HPFS, NTFS (Dos & Windows),
- ISO9600, UDF, UFS, Joliet, RockRidge (CD),
- Ext, Ext2, Ext3, (*Linux*)
- SystemV, VxFS, Spiralog (Solaris, VMS)
- **NFS** (Network File System)



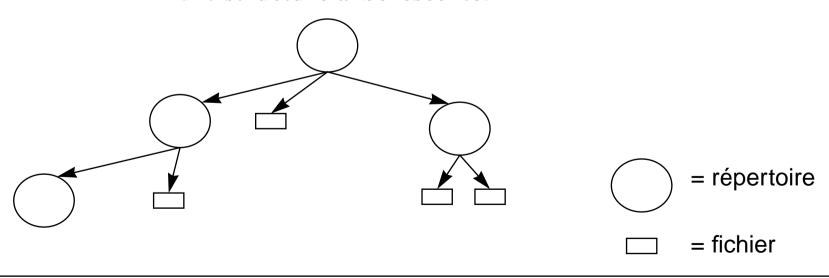
Structuration d'un système de fichiers (1)



La capacité des mémoires auxiliaires étant très grande, on peut y loger un **grand nombre de fichiers**. Par exemple, un compte d'utilisateur peut facilement contenir plusieurs milliers (voire dizaines de milliers) de fichiers, un système de fichiers complet pouvant contenir quant à lui plusieurs millions de fichiers.

Il est par conséquent indispensable de fournir un moyen pour organiser ces fichiers. A cette fin, pratiquement tous les systèmes d'exploitation utilisent le concept de **répertoire** (aussi appelé catalogue ou directory).

Un répertoire est une collection (généralement non ordonnée) de fichiers ou de répertoires (alors appelés sous-répertoires). Ils permettent d'organiser l'ensemble des fichiers dans une **structure arborescente**:



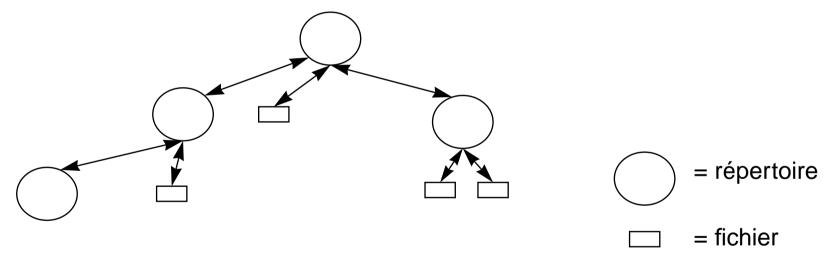


Structuration d'un système de fichiers (2)



De nombreux systèmes (p.ex. Windows) utilise en fait une **forêt**, c'est à dire qu'il maintiennent plusieurs structures arborescentes, en associant les racines de ces structures à des noms de lecteurs. Ce n'est pas le cas de Unix, qui ne possède qu'une arborescence.

Une arborescence de fichiers peut être parcourue dans les deux sens. Cette possibilité de double parcours est assurée par le système d'exploitation, qui conserve, pour chaque répertoire, une référence vers le répertoire parent.

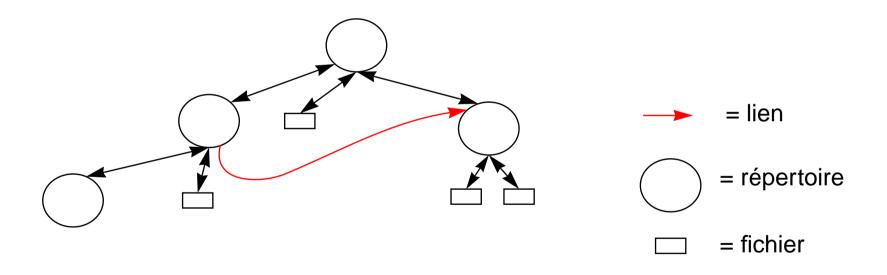




Structuration d'un système de fichiers (3)



En plus de la notion de répertoire, la plupart des systèmes permettent également de définir des **liens symboliques** vers des fichiers ou des répertoires (soft/hard links avec Unix, raccourcis (~) avec Windows), qui permettent d'une part de définir des **alias**, et d'autre part d'assouplir la structure d'arbre.





Nommage des fichiers



On appelle «**chemin**» la succession des répertoires conduisant à un fichier donné. Par extension, le chemin désigne la succession des répertoires et le nom du fichier lui-même.

Pour désigner un fichier, il est possible de procéder de deux manières:

à l'aide d'un chemin absolu: on prend comme convention un parcours de l'arbre partant de la racine (dans le cas d'une forêt, le nom du lecteur (i.e. la désignation de la racine de l'arbre) doit être tout d'abord spécifié. (analogie: Av. des Morgines 18, CH-1213 Petit-Lancy)

- à l'aide d'un chemin relatif: c'est la succession des répertoires à traverser, à partir d'un répertoire donné de l'arborescence (autre que la racine) (analogie: tout droit, puis à gauche au carrefour, 2e immeuble, 3e allée)



Système de fichiers UNIX



Sous unix, le nom d'un fichier est composé d'au plus 255 caractères. Les fichiers sont généralement (mais il n'y a aucune obligation) muni d'une extension, délimitée par un point («.»). L'extension sert à indiquer la nature du fichier, i.e. l'application à laquelle il est associé. Cependant, cela n'est absolument pas une nécessité sous Unix: les fichiers peuvent avoir 0, 1 ou plusieurs «extension».

exemple: cours-1.ps.gz

Le délimiteur entre nom de répertoire et nom de fichier est la barre oblique 'slash' «/». Le slash est également le nom de la racine de l'arborescence Unix.

exemple: /users/toto/cours-1.ps.gz

Le catalogue parent d'un sous-catalogue est désigné par deux points («..»), tandis que le répertoire lui-même par un point («.»)

exemple: cd ../titi/



Fichiers et shell



Un certain nombre des attributions du shell sont relatives au système de fichiers. Le shell a ainsi la responsabilité de:

- permettre la navigation dans la structure des fichiers (pour cela, il définit la notion de répertoire courant toutes les commandes se font relativement à ce répertoire et une primitive de modification de ce répertoire (cd pour Change Directory))
- Définir le répertoire de base de l'utilisateur, et offrir un certain nombre d'alias pour s'y rendre rapidement (la variable d'environnement \$HOME contient le chemin absolu du répertoire de l'utilisateur courant, et on peut le désigner par tilde «~» dans les commandes)
- Permettre l'utilisation de jockers dans le nommages des fichiers et répertoires. Ainsi, le point d'interrogation «?» peut être utilisé comme substitut de 1 caractère, tandis que l'étoile sers à substiuer une chaîne de caractères (n positions).

exemples:

- ls cours-?.ps.gz
 - ls cours*



Attributs de fichiers



Les attributs de fichiers typiques sont:

- son nom, - le propriétaire (créateur)

- sa taille, - les droits d'accès des autres utilisateurs

- la date et heure de création - .

- modifiable, exécutable, caché,... - la date et l'auteur de la denière révision,

- système, possédant des alias,... - no de version (système VMS),...

Dans le cas du système Unix:

- On distingue les fichiers cachés au moyen d'une convention de nommage: ils sont préfixé par un «.»

- A chaque fichier est associé un utilisateur propriétaire (le créateur du fichier) et un groupe propriétaire (l'un des groupes auxquels appartient l'utilisateur)
 - Les droits d'accès définissent 3 attributs, et sont paramétrable pour 3 classes d'utilisateurs:

Attributs		
Visibilité (lecture)		
Modification (écriture, effacement)		
Exécution		

Classes d'utilisateurs	
Propriétaire (owner ou user)	
Groupe (group)	
Autres (others)	



Utilisateurs et groupes

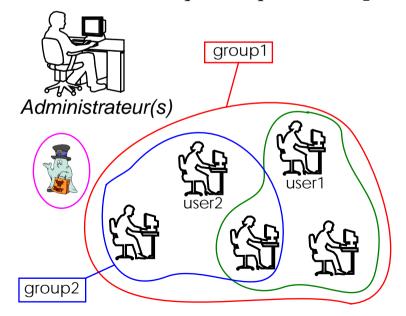


Dans le cas d'architectures multi-utilisateurs, il est primordial d'identifier les personnes désireuses de travailler avec le système, afin de pouvoir assurer la confidentialité de leur données, et parfois de leur facturer les ressources utilisées.

A cette fin, les systèmes d'exploitation utilisent un modèle à 2 entités:

- les utilisateurs, et
- les **groupes** d'utilisateurs

Ceci permet de définir une politique des droits (ressources utilisables et dans quelle quantité – *privilèges*) commune à un ensemble d'utilisateurs.



- Chaque utilisateur appartient à au moins 1 groupe
- Il existe des utilisateurs et des groupes d'utilisateurs particuliers:
 - les administrateurs (utilisateurs en charge du système, avec tous les privilèges)
 - les utilisateurs «fantômes», utilisés par des programmes particuliers (par exemple les programmes réalisant les sauvegardes, les démons, ...



Quotas



Dès lors que plusieurs personnes travaillent ensemble, et utilisent une ressource commune, il faut mettre en œuvre des mécanismes assurant qu'un utilisateur ne monopolise pas cette ressource au détriment des autres.

C'est également le cas avec les disques durs.

Le «quota disque» représente l'espace qu'un utilisateur peut utiliser sur le disque contenant son répertoire de base (home directory). C'est en fait l'espace mémoire maximum admis pour la somme des tailles des fichiers du sous-arbre dont la racine est le répertoire de base de l'utilisateur. 11

Lorsqu'un utilisateur «atteint son quota» 12, le système refusera toute tentative conduisant à une augmentation de l'espace utilisé. Par ailleurs, l'utilisateur ne pourra plus démarrer de sessions en tant qu'utilisateur local).

Pour vérifier votre utilisation d'espace disque, utiliser la commande: quota -v

^{11.} En effet, tous les élements hors de ce sous-arbre (lecteur de disquettes, /tmp, ...) ne sont pas pris en compte dans le calcul de l'espace occupé. Par contre, leur usage peut également parfois être restreints par le biais de quotas.

^{12.} En réalité, le quota disque est dépassable dans une petite proportion, pour un court laps de temps (typiquement, +10% pendant 7 jours au maximum)