Linux

D.Moreaux

Table des matières

I Qu'est-ce qu'un OS

Depuis les premières machines programmables, différents problèmes se sont posés qui ont mené à la création d'un programme appelé OS (Operating System). Même si un bon nombre des techniques rencontrées dans le passé peuvent sembler désuètes, elles conditionnent encore de nos jours certains aspects de nos ordinateurs et donc de l'OS qui tourne dessus.

I.1 Les machines mécaniques

Bien avant les ordinateurs électriques, on trouve des mécaniques dont le comportement préfigure les ordinateurs.

- Au 17ème siècle apparaissent les orgues de barbarie et pianos mécaniques. Les premiers utilisaient un cylindre sur lequel étaient adapté des picots qui actionnaient les valves. Des mécanismes à disque ont ensuite vu le jour. Afin d'augmenter la durée possible des morceaux, ils ont évolué en 1892 pour utiliser des "livres de notes", série de cartes perforées attachées les unes aux autres et puis, les rouleaux de papier.
- En 1642, Blaise Pascal crée la Pascaline, une machine à additionner. Le mécanisme utilisé pour les reports est toujours utilisé de nos jours dans les compteurs kilométriques des voitures et dans les compteurs eau/gaz/électricité. A noter que certaines machines de nos jours traitent encore les nombres sous forme de série de chiffres et non de leur valeur binaire (on parle de codage BCD)
- Entre 1768 et 1774, Pierre Jaquet-Droz fabrique trois automates : le dessinateur, la pianiste et l'écrivain. Ces automates utilisent des arbres à came pour gérer les séquences de mouvements. L'écrivain est parfois considéré comme le premier système "programmable". En effet, le texté écrit était composé en adaptant des blocs de hauteurs différentes, chaque auteur correspondant à une lettre. Ces automates fonctionnent toujours et peuvent être vu au musée de Neuchâtel en Suisse.
- En 1728, Falcon crée un métier à tisser qui utilise des cartes perforées pour "programmer" les motifs. En 1801, Joseph Marie Jacquard améliore le travail de Falcon lance l'utilisation en masse de son métier à tisser
- En 1822, Charles Babbage invente sa machine différentielle[1], chargée de calculer des tables de nombres. La fabrication de cette machine fut abandonnée en cours. Il conçoit donc une nouvelle version, capable de sauver des résultats intermédiaires et d'effectuer des traitements en série et des opérations conditionnelles. Cette machine ne fut construite qu'en 1989-1991, en deux exemplaires. L'un se trouve au musée des sciences de Londres et le second au musée de l'histoire de l'informatique, en Californie.
- En 1890, Herman Hollerith propose une machine lisant des cartes perforées et comptabilisant les données en utilisant le mécanisme de la Pascaline pour l'établissement du

1 Cette machine se basait sur le principe des différences finies. Si on désire calculer les valeurs d'un polynôme, il est possible à partir des premiers termes de résoudre l'opération à de simples additions. Pour cela, on calcule les différences x(1)i entre les termes x_i et x_{i+1}, les différences de second ordre $x(2)_i$ entre $x(1)_i$ et $x(1)_{i+1}$ et ainsi de suite jusqu'à ce que les x(n) soient tous égaux. Pour calculer les termes suivants, on part des x(n) pour obtenir les x(n-1), qui permettent de remonter à x(n-2), ... jusqu'aux x, le tout se faisant à l'aide d'additions.

recensement aux USA. Il crée une entreprise appelée Tabulating Machine Company, devenue par la suite International Tabulating Machine et enfin International Business Machine (IBM).

Toutes ces mécaniques ont amené des concepts de base utilisés dans l'informatique.

I.2 Les machines à relais

L'arrivée de l'électronique a rendu la création d'ordinateurs plus aisée. On a vu ainsi des ordinateurs fonctionnant à base de relais (et puis de tubes électriques) apparaître. Les relais étaient câblés afin de leur faire effectuer les diverses opérations.

Les opérations de base en logique binaire peuvent facilement être effectuées à l'aide de relais, ainsi que les fonctions de mémorisation.

Changer la tâche de ces ordinateurs était long et très difficile. Il fallait en effet changer de nombreux câblages dans la machine. Les cartes (et puis les bandes) perforées étaient utilisées pour fournir les données. On peut parler de logique câblée (cette notion existe toujours de nos jours au niveau de l'électronique).

I.2.a Les tubes

Les tubes sous vide ont remplacé les relais avec de nombreux avantages : moins de bruit, consommation électrique plus faible, plus rapides, ...

Les tubes comportaient entre autre la diode (deux électrodes : une anode entourant une cathode chauffée) qui ne laissait passer le courant que dans un sens (les électrons circulaient de la cathode vers l'anode[2]), la triode (trois électrodes, une grille de contrôle entre anode et cathode permettant de bloquer ou de laisser passer le courant), le décatron (un point "chargé" se déplace entre 10 électrodes permettant de compter jusqu'à 10 (ou une valeur inférieure à 10) mais aussi de visualiser une valeur de 1 à 10), le Nixie (tube comportant plusieurs électrodes en formes de chiffres, l'électrode sous tension ionisant le gaz et faisant apparaître le chiffre lumineux correspondant), ...

Le passage aux tubes était également le passage de l'électromécanique à l'électronique. Plus tard, les transistors ont remplacé les triodes mais ils ont un principe de fonctionnement similaire[3]. Les puces électroniques ne font rien d'autre que d'intégrer sur une faible surface des centaines de transistors interconnectés.

Les tubes demandaient à être alimentés avec de fortes tensions et chauffaient assez fort. Cette chaleur attirait les insectes (bugs) qui, s'ils touchaient les contacts provoquaient un court-circuit. C'est de là que vient le terme de bug pour une erreur informatique.

I.2.b Avantages et désavantages de la logique cablée

Si à l'époque il ne s'agissait pas d'un choix, de nos jours, la logique câblée est toujours utilisée dans certaines situations. En effet, lorsqu'une fonction logique est réalisée en logique câblée, le calcul est fait de manière quasi immédiate, quelle que soit le nombre d'entrées concernées. Pour effectuer une

2 A noter que les termes d'anode et de cathodes sont toujours utilisés dans les diodes actuelles

3 Tout au moins, dans le cadre de l'électronique numérique qui concerne les ordinateurs et où l'on travaille en "tout ou rien". Pour les amplificateurs, les méthodes de design sont assez différentes.

addition de deux bits, les données ne doivent traverser que maximum 3 portes logiques, contre des centaines (voire des milliers) dans le cas de la programmation. De plus, la plus grande partie d'un microprocesseur doit être réalisée en logique câblée.

Le plus gros défaut de la logique câblée est qu'il est compliqué et long (si pas impossible) de changer la tâche qui doit être effectuée. On a donc plus un calculateur qu'un vrai ordinateur.

On trouve maintenant des composants qui forment une solution à mi-chemin entre la programmation et la logique câblée : les FPGA. Ces composants intègrent un grand nombre de réseaux logiques NON-ET-OU connectés à des cellules mémoire de 1 bit. Le choix des fonctions réalisées par les réseaux logiques, des données qui sont mémorisées et de celles qui sont transférées immédiatement et les interconnexions entre ces différentes cellules sont programmées dans la FPGA lorsqu'elle est mise sous tension. Elles sont donc idéales pour tester un design avant de créer une puce "en dur" ou pour des tâches où la vitesse de la logique câblée est demandée (sans les problèmes liés au recâblage).

I.3 Redécouverte de la programmation

Alors que Jacquard et Babbage avaient tous les deux amené la notion de programme, les machines qui avaient suivi étaient construites pour une tâche donnée. En 1941, Zuse conçoit une machine où le programme peut aisément être modifié... Des boutons rotatifs à 10 positions pour encoder des nombres et des panneaux de câbles pour encoder le programme.

Les réglages permettaient de commander les différents systèmes directement. On pouvait par exemple donner l'ordre à l'unité mathématique de faire une addition et aux mémoires de sauver le résultat dans un registre précis. Chaque instruction était directement liée au matériel, on parle parfois de microcode. Ce concept de microcode est toujours utilisé de nos jours à l'intérieur des processeurs (afin de commander les registres internes ("publics" ou réservés en interne), l'ULA[4] et les fils de contrôle externes au processeur). Les instructions de microcode des processeurs actuels possèdent un nombre de bits particulièrement élevé vu le nombre d'éléments à commander.

I.4 Architectures Von Neumann et Harvard

Von Neumann propose en 1945 d'utiliser la mémoire pour stocker le programme et de ne pas se contenter de l'utiliser pour les données. Le programme est alors chargé à l'aide de cartes ou de bandes perforées. Lors du démarrage de la machine, il lit le programme dans la mémoire et puis commence à la première instruction.

Ce modèle Von Neumann est toujours employé de nos jour dans la plupart des ordinateurs.

A côté du modèle Von Neumann, un second modèle existe, le modèle Harvard (titré du Mark I livré à l'université de Harvard et qui utilisait des stockages séparés pour le programme et les données). Dans le modèle Harvard, les mémoires où sont stockées le programme et les données sont séparées. Cela permet d'utiliser des nombre de bits différents pour les instructions de programme et pour les données. Ce modèle Harvard est encore utilisé dans des composants programmables comme les

4 Unité Logique et Arithmétique, un circuit capable d'effectuer les opérations de bases (addition, soustraction, multiplication, division, ET/OU/NON/OU-exclusif, décalages, ...)

micro-contrôleurs qui embarquent sur un seul composant processeur, mémoire vive pour les données, mémoire morte pour le programme et périphériques d'entrée/sortie.

Le modèle Von Neumann a l'avantage d'être plus flexible dans la mesure où une même mémoire peut, selon les besoins, être utilisée pour le programme ou pour les données. Le modèle Harvard par contre permet une plus grande rapidité (deux bus différents), une meilleure sécurité (des données incorrectes ne peuvent être exécutées comme un programme) et permet des mots de tailles différentes pour les données et le programme.

I.5 Le BIOS

Un des principaux défauts de ces programmes était que pour tout programme, il fallait reproduire les mêmes routines propres à la machine (principalement pour les entrées sorties). Dans le cas des programmes sur bande perforée par exemple, cela était effectué en collant des morceaux "tout prêts" qui contenaient les routines en question.

Une solution à ce problème est d'équiper la machine d'une mémoire morte contenant les routines en questions. Il suffit alors d'utiliser les fonctions prédéfinies plutôt que de les réencoder à chaque fois. On parle de Basic Input Output System ou de BIOS.

Ce BIOS possède différentes tâches dont celle d'initialiser la machine lorsqu'on la démarre, les lectures de cartes, de clavier, les sorties sur imprimante ou perforatrice (et plus tard, sur l'écran), ...

Une autre fonctionnalité souvent embarquée dans la ROM du BIOS est un moniteur langage machine de base (programme permettant d'entrer et d'exécuter le code) ou, plus tard, un interpréteur BASIC[5]. Qu'il s'agisse du moniteur langage machine ou de l'interpréteur BASIC, il s'agit d'un programme séparé du BIOS (et qui se sert d'ailleurs du BIOS). Ces programmes permettant l'interaction avec l'utilisateur est la première forme de "shell" que l'on trouve.

Pour les systèmes prévus pour travailler avec des disquettes, on ajoutait souvent un second ensemble de fonctions standard pour des accès au disquette de "haut niveau". C'est ainsi que les Apple II chargeaient en mémoire le code prévu pour les accès disques, à partir d'une disquette correctement initialisée (on parle maintenant de formatage), ou les Spectrum équipés d'interface Disciple ou Plus D chargeaient dans la mémoire de l'interface la partie avancée de la gestion de disquettes. Le BIOS ne contenait que les fonctions de bas niveau (lecture et écriture d'un secteur, formatage d'une piste et chargement du système disque à partir d'un emplacement particulier de la disquette).

I.6 L'Operating System

Là où les frontières entre BIOS, système de gestion de disquettes et le shell étaient souvent floues, les trois parties étant souvent mélangées, on voit une meilleure séparation arriver avec la complexification des ordinateurs. Avec l'apparition d'OS/360 (pour mainframe) et de CP/M pour d'autres systèmes, on se trouve devant la division encore présente de nos jours entre BIOS, OS et shell.

5 A noter que certaines machines ont fourni le Basic dans le seul but de faire classer l'appareil comme outil et non comme loisir. Les premières PlayStation proposaient un interpréteur Basic fourni avec la console afin de bénéficier de taxes moins élevées)

Le CP/M est un cas un peu particulier. En effet, il était prévu pour tourner sous différents matériels (dans le cas de l'APPLE II et de certains autres systèmes, cela passait par l'ajout d'une carte équipée d'un processeur Z80). Dès lors, au démarrage de CP/M, un BIOS spécifique au matériel était chargé en mémoire (au lieu d'être situé en ROM comme c'est généralement le cas) sur lequel s'appuiait le BDOS (Basic Disk Operating System) qui charge le CCP (Console Command Processor). Le BIOS du CP/M était donc un cas un peu particulier en ce sens qu'il ne se situait pas en mémoire morte. Cette façon de faire était nécessaire pour uniformiser l'interface du BIOS vu la diversité des machine supportant le CP/M.

A côté de cela, d'autres OS vont exister en un grand nombre de versions différentes, chacune adaptée à un matériel propre. Cela implique que la partie chargée de gérer les entrées-sorties doivent être changées selon le matériel.

Les premiers OS ne s'occupaient que de la gestion des fichiers, de la mémoire et de tâches simples comme le chargement et l'exécution d'un programme. Par la suite, des tâches telles que la gestion des processus (et donc le fonctionnement en multitâches), les mécanismes de communication entre processus et les mécanismes réseau se sont ajoutés.

I.7 Les grandes lignes de quelques OS

Les ordinateurs 8 bits disposaient d'un OS appelé CP/M. Lorsque IBM a créé la plateforme PC sur base d'un processeur à 16 bits, il a cherché un OS pour son hardware.

Microsoft, qui jusque là vendait principalement un interpréteur basic, a alors acheté un OS appelé QDOS (Quick and Dirty Operating System) qui était une preuve de concept de l'adaptation de CP/M sur 16 bits et l'a revendu à IBM comme étant le MS-DOS 1.0.

Par la suite, d'autres versions de MS-DOS sont sorties, avec quelques problèmes comme l'utilisation de la technologie de la firme Stacker dans le MS-DOS 6.0 sous le nom de "Doublespace" qui a résulté en un procès au terme duquel le MS-DOS 6.22 est sorti avec un outil équivalent qui n'entrait pas en conflit avec les brevets de Stacker.

De son côté, la firme Apple a utilisé une technologie de Xerox pour créer son premier OS pour Macintosh. Tant la souris que l'interface graphique reprennaient la technologie de Xerox. Lorsque Microsoft a copié cette interface pour son MS Windows, Apple a tenté d'intenter un procès à Microsoft mais a été débouté lorsque ce dernier a pointer s'être inspiré du travail de Xerox comme l'avait fait Apple.

L'environnement MS-Windows a ensuite évolué en masquant le MS-DOS (version 7.0) pour Windows 95 et 98.

D'un autre côté, une colaboration avait commencé avec IBM pour réaliser un OS conjointement. IBM devait fournir un noyau fiable, multitâches et multi-utilisateurs et Microsoft la compatibilité avec les applications Windows. Cela a mené à la création de OS/2 mais la compatibilité windows n'a jamais été complète, Microsoft gardant certaines API et fonctionnalités pour sa version de Windows. Au final, la collaboration s'est arrêtée, chaque côté gardant accès au travail commun.

Microsoft a alors débauché des informaticiens de chez Digital pour les faire travailler sur le noyau en question ce qui a mené aux premiers MS Windows NT. L'évolution de NT fur Windows 2000 puis Windows XP, Vista, 7.0, ...

A côté de cela, un OS multi-taches et multi-utilisateurs avait été créé d'abord pour PDP mais ensuite pour les autres plateformes. Il s'agissait de UNIX. UNIX s'est par la suite séparé en deux branches (System V et BSD) qui restent présentent malgré l'apparition d'un standard commun, le standard POSIX qui reprend les points forts de ces deux branches. Dans chacune des branches, de nombreux OS ont vu le jour (mais ne pouvant généralement pas utiliser le mot UNIX, on trouve ainsi des noms tels que Dynix, Xenix, Ultrix, Irix, ...). Parmis ces OS, on peut trouver Linux (à la base des systèmes Android, principalement d'inspiration System V) et les FreeBSD, NetBSD et OpenBSD (dont s'est inspiré MacOS/X).

Au final, les grandes inspirations des OS Modernes pour micro-ordinateurs sont le CP/M, OS/2 et Unix.

II Role d'un OS

Un OS contient une série de fonction qui vont de la gestion du matériel au support des applications qui tournent dessus.

La frontière entre l'OS et le shell est parfois difficile à faire (des entreprises telles que Microsoft ont d'ailleurs tendance à rendre cette limite floue volontairement, afin de lier à l'OS des fonctionnalités qui ne devraient pas s'y trouver comme ce fut le cas avec le navigateur web). En général, on peut dire que si une fonctionnalité est accessible (directement) par l'utilisateur, elle ne fait pas partie de l'OS. Les tâches effectuées par l'OS sont en effet en arrière plan, le plus discrètes possible. L'OS est sensé ne pas se voir, consommer le moins de ressources possibles, ...

II.1 Gestion de la mémoire

Une des tâches principales de l'OS est la gestion de la mémoire. Il doit en effet gérer la mémoire physique (celle qui est présente au niveau hardware), mais également la mémoire virtuelle (la mémoire visible par les applications, après application du mécanisme de pagination), être capable d'allouer des intervalles d'adresses mémoires consécutives mais aussi de s'assurer que la mémoire physique est utilisée de la manière la plus efficace possible, en évitant des sur-allocations.

On peut également placer dans la gestion de la mémoire la problématique du Swap : lorsque l'on ne dispose pas d'assez de mémoire physique, une partie de la mémoire peut être écrite sur le disque dur pour libérer de la mémoire physique et sera récupérée quand elle sera nécessaire.

II.2 Gestion des interruptions et des exceptions

Lorsqu'un événement se produit sur un périphérique, il peut déclencher une interruption qui sera traitée par l'OS afin de gérer l'événement : données lues du disque dur, buffer vide pour la carte son, ...

De la même manière, certains événements peuvent se produire directement au niveau du processeur. On parlera généralement d'exceptions même si le mécanisme de gestion est similaire. Il peut s'agir d'une tentative d'accès à une zone de mémoire non mappée au niveau de la pagination, d'une violation de privilèges, d'une division par 0, ...

Finalement, le mécanisme d'interruption est également souvent utilisé pour la communication entre l'espace USER et l'espace KERNEL (noyau).

II.3 Gestion des périphériques

Clavier, écran, disque dur et autres sont gérés au niveau de l'OS. Pour gérer ces périphériques, il est généralement nécessaire d'avoir accès au mécanisme d'interruption mais également à une série d'instructions assembleur réservées au mode Kernel (et donc à l'OS) et à des structures de mémoire gérées pour le reste par l'OS.

II.4 Gestion des tâches

L'OS se charge également de créer des tâches, environnements dans lesquels des programmes différents peuvent s'exécuter. On fait généralement la distinction entre deux principaux styles de

tâches : les processus, parfaitement isolés les uns des autres, et les threads, qui partagent le même espace mémoire. Cette gestion de tâche implique également la présence de code chargé de faciliter la communication entre ces tâches.

II.5 System Calls

Lorsqu'une application a besoin d'utiliser un service fourni par l'OS, elle a recours à un mécanisme qui peut être appelé System Call, Syscall, Supervisor Call, Trap, Diag, ...

Ce mécanisme permet, généralement au travers d'une interruption, d'appeler des fonctions mises à disposition par l'OS. On y trouve par exemple les fonctions d'accès aux fichiers, les fonctions qui permettent de créer une nouvelle tâche, de terminer la tâche en cours, ...

Il s'agit normalement de la seule interface entre le mode utilisateur et l'Operating System.

II.6 Mécanisme de boot

Lors du démarrage de la machine, le BIOS sait comment charger un morceau de code (généralement de petite taille) et le démarrer. Ce morceau de code, qui fait partie de l'OS, est généralement chargé de lire le reste de l'OS du support (disque dur ou autre), de mettre en place une configuration de base du processeur (par exemple une configuration "plate" de la mémoire au niveau des différents mécanismes de pagination et de segmentation, en attendant que la pagination définitive soit en place) et parfois d'accéder à certaines ressources du BIOS avant que ce dernier ne puisse plus être utilisé (taille mémoire par exemple).

Dans le cas des processeurs intel 32 bits, ce code sera également chargé de faire passer le processeur en mode 32 bits. En effet, au démarrage, le processeur travaille en mode 16 bits et le code de boot devra donc être partiellement écrit en mode 16bits.

Pour finir, le code de boot sera également chargé de placer la pile langage machine à un endroit correct. En effet, une grande partie des OS est programmée en langage de haut niveau qui utilise la pile pour le passage de paramètres. Il faut donc s'assurer que la pile ait une taille maximale suffisante.

III Gestion de la mémoire

La plupart des sous-systèmes de l'OS a besoin de réserver des zones de mémoire. Certains sous-systèmes pour y allouer des structures dans lesquelles ils sauveront leurs données, d'autres pour y allouer des tampons (buffers),...

Cette gestion de la mémoire dépend fortement de la plateforme (et surtout du processeur). Les mécanismes disponibles, la taille des pages mémoires, la largeur des données traitées, ... sont différentes d'un processeur à l'autre.

III.1 L'allocation de la mémoire

L'attribution de la mémoire présente deux problèmes distincts : d'une part ne pas attribuer la même zone de mémoire physique à deux tâches différentes et d'autre part, une optimisation de l'allocation de la mémoire.

III.1.a Mémoire physique

La mémoire physique sera découpée en "pages". Ces pages pourront être allouées à différents sous-systèmes et libérées quand elles ne seront plus nécessaires. Comme une page peut être utilisée par plusieurs parties du système (zone de mémoire partagée), il est nécessaire de connaitre le nombre de réservations effectuées sur la page concernée. Lorsque ce nombre tombe à 0, cela signifie que la page est de nouveau libre.

Pour gérer ses pages, il suffirait donc d'une table contenant autant d'entrées que de pages et reprenant le nombre de réservations de la page. Mais un tel dispositif serait très lent car il faudrait parcourir toute la table à la recherche d'une page libre. On utilisera donc deux listes chaînées, l'une reprenant les pages vides et l'autre les pages utilisées. Les structures chaînées seront soit référencées par un tableau, soit directement situées dans un tableau, afin de pouvoir atteindre rapidement la structure liée à une page donnée.

Lors de l'initialisation de ce sous-système, il faut marquer les pages correspondant à l'OS et à ses structures de base (table d'interruption, pile, ...) comme "réservées" afin d'éviter qu'elles ne soient allouée pour sauver des données. Il faut également créer les listes de pages libres et occupées.

III.1.b Optimisation de l'allocation

Le système découpe la mémoire en pages. Souvent, la taille de ces pages sera égale à celle des pages gérées par la pagination (dans le cas des x86, les pages font 4Ko). Cela veut dire que la plus petite zone de mémoire qui puisse être allouée est de une page, même si le but est de sauver une structure de quelques octets.

Deux problèmes différents peuvent se présenter : d'une part, allouer une page complète pour une structure de petite taille provoquerait un gros gaspillage de mémoire, d'autre part, si l'on désire allouer une structure de plusieurs pages, elles doivent être mappées dans des pages virtuelles qui se suivent.

Pour éviter le gaspillage de mémoire, on va sauver plusieurs structures dans la même page. Afin de faciliter les traitements et d'éviter des pertes de temps pour trouver des zones de taille suffisante, on

créera plusieurs groupes de pages, un groupe par taille de structures. Ces groupes s'appellent des slabs.

Ainsi, la plupart des structures utilisées dans le noyau auront un slab associé, auquel seront ajoutées des pages lorsque le besoin se fera sentir. Par contre, pour les structures de taille variable ou peu utilisées, on utilisera des slabs de taille fixe, en choisissant le plus petit pouvant accueillir la structure demandée.

Ce mécanisme de slabs utilise également une série de structures de contrôle : la description d'un slab (pointeur vers la liste des zones libres, nombre de structures libres, slab suivant pour le même type de structures) mais aussi la description d'un groupe de slab (nombre de pages par slab, nombre de structures par slab, nombre de slab réservés, ...) Ces structures sont également gérées par le mécanisme de slab.

Pour les allocations de plusieurs pages, on utilisera une liste des intervalles de mémoire virtuelle disponible. Ces intervalles sont maintenus par ordre d'adresse croissantes. Lors des allocations, un intervalle pourra être coupé en deux morceaux, l'un occupé et l'autre libre. Lors de la libération d'un intervalle, il pourra être fusionné avec des intervalles libres qui le précéderaient ou le suivraient.

Ces deux mécanismes sont liés. En effet, l'allocation d'un slab passera par l'allocation d'un intervalle de mémoire virtuelle et lorsqu'un intervalle sera divisé en deux, il faudra réserver une structure à cet effet dans un slab. Pour éviter un blocage, il sera nécessaire de garder un certain nombre d'entrées libres.

III.2

III.3 Mémoire paginée (mémoire virtuelle)

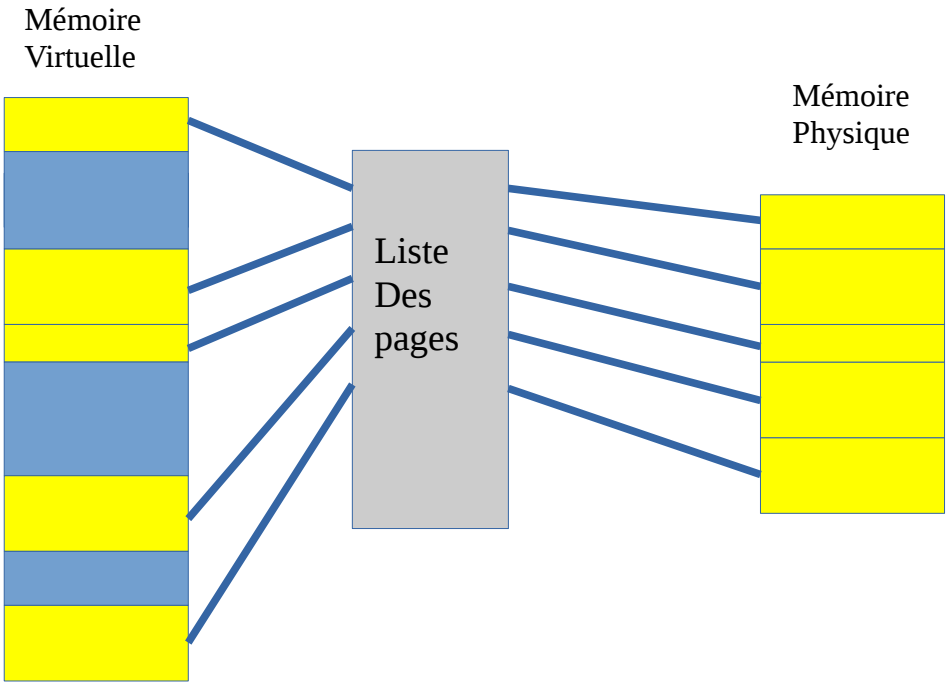
Là où les registres du processeur permettent un accès à la mémoire sur 32 bits, pendant longtemps la mémoire réelle disponible était inférieure aux 4Go. Une partie de l'espace adressable était donc inutilisé.

Il est donc nécessaire de définir les relations entre les zones adressables utilisées et la mémoire physique. Ce rôle est géré par le MMU qui peut se trouver à l'intérieur du processeur ou être un composant externe.

Le principe est d'utiliser une table qui associera les zones de la mémoire adressable (aussi appelée mémoire Virtuelle) à des zones de la mémoire physique.

Dans le cas d'un OS multitâches, on peut utiliser des tables différentes pour chaque tâche, ce qui permettra de garder les mêmes zones d'adresses d'un programme à l'autre (code, données, pile, ...)

Dans le cas des processeurs 80x86 un deuxième mécanisme est prévu, il s'agit de la segmentation. La conversion est faite à l'aide d'une table qui associe un numéro de segment à une zone mémoire réelle. Dans un segment, les adresses sont définies à partir de l'adresse 0. L'adresse obtenue à l'aide de la segmentation sera ensuite transformée grâce au mécanisme de pagination classique. En pratique, ce mécanisme est mis "hors des pieds" et ignoré par l'OS qui se contente de la pagination.



Le mécanisme de mémoire paginée permet donc d'attribuer certains morceaux de la mémoire physique à des zones de la mémoire virtuelle. A noter que certaines parties de la mémoire physique peuvent ne pas être assignées ou l'être plusieurs fois. On parle de "mapper" une page de mémoire physique dans une zone de mémoire virtuelle.

Lorsque l'on tente d'accéder à une zone mémoire qui n'existe pas, le processeur déclenche une exception. Dans le cas d'un accès à une zone mémoire non définie, on parle de "Page Fault". Cette exception peut également apparaître si l'on tente d'écrire dans une zone définie en lecture seule.

Parfois, les pages de mémoire physique ne seront réservées et mappées que lorsque l'on rencontrera un Page Fault. On parle de réservation "à la demande" (même si les zones mémoire accessibles ont été définies au préalable).

Les tables qui servent à gérer la pagination sont généralement définies par des adresses en mémoire physique. Pour pouvoir les manipuler, il faut qu'elles soient mappées en mémoire virtuelle. Cela demande quelques précautions au niveau du code de gestion de la pagination qui soit toujours s'assurer avoir des entrées libres dans ces tables présentes en mémoire mappée.

IV Gestion des Interruptions

Lorsqu'une interruption ou une exception survient sur le système, il est nécessaire de sauver les différents registres susceptibles d'être utilisés lors du traitement de cette dernière afin de pouvoir retourner au code interrompu de manière transparente.

Comme le code principal derrière cette gestion est généralement programmé dans un langage de haut niveau tel que le C, les registres utilisés dépendent du compilateur et des options de compilation utilisées. Dès lors, il est plus prudent de sauver tous les registres.

Pour des raisons de facilité, cette sauvegarde se fait sur la pile.

Mais cela a également un autre avantage : la quasi totalité de l'état du programme interrompu se trouve sur la pile : fonctions appelées, paramètres passés à ces dernières, variables locales et l'état au moment de l'interruption. Les seules variables qui ne se trouvent pas sur la pile sont les variables statiques.

On peut donc passer aisément d'une tâche à une autre en changeant de pile langage machine. Les tâches en question partageront le même espace mémoire et on parlera de Threads.

Les interruptions peuvent être déclenchées pour diverses raisons : timer arrivé à 0 (qui permet de générer des interruptions à intervalle régulier), données disponibles venant d'un périphérique ou buffer d'un périphérique de sortie vide.

De même, les exceptions sont déclenchées par le processeurs lors d'événements tels qu'une division par 0, une tentative d'accéder à une zone mémoire inaccessible (pas de mémoire mappée) ou dont l'accès est interdit ou encore d'exécuter une instruction illégale ou restreinte.

Pour finir, il est souvent également possible de déclencher une interruption directement par programme. Ce type d'interruption sera généralement utilisé par le mode utilisateur pour appeller un code situé en espace kernel. On parlera de System Call (Syscall), de Supervisor Call (svc), de trap, diag, ...

Dans le cas d'un OS multitâches, on profitera des interruptions pour effectuer éventuellement un changement de tâche. Si les tâches sont dans des espaces d'adressage différents (configuration de la pagination différentes), il sera nécessaire de rétablir la pagination de la tâche vers laquelle on compte retourner en fin de routine d'interruption.

V Gestion des tâches

V.1 Threads ou Processus ?

Lorsque plusieurs tâches tournent sur la même machine, on peut prévoir deux niveaux de séparation des tâches :

Les threads, qui sont des tâches séparées mais partagent le même espace mémoire. Les différentes threads ne sont pas protégées les unes des autres et ont accès à la même mémoire.

Les processus qui sont des tâches isolées les unes des autres. Chaque processus disposera de son propre espace mémoire.

Les deux notions peuvent être mélangées, chaque processus peut ainsi être composé de plusieurs threads.

Le principal avantage des threads est qu'il est aisé de partager des données entre les différentes threads vu qu'elles partagent les mêmes pages de mémoire.

Le principal avantage des processus est l'isolation entre les processus qui permet de les protéger l'un de l'autre, protection utile dans le cas d'un environnement multi-utilisateurs.

Au niveau de l'OS en lui-même, ce sont généralement des threads qui seront utilisées car l'OS n'a pas besoin d'être protégé de lui-même, ce qui permet d'éviter les lourdeurs (changements de pagination) rencontrées lors des changements de tâches.

V.2 Le Scheduler

Lorsqu'il est décidé de changer de tâche, une fonction appelée scheduler sera chargée de déterminer la tâche suivante. Le rôle de cette fonction est crucial tant au niveau des possibilités disponibles qu'au niveau des performances du système.

Le scheduler peut gérer des niveaux de priorité différentes entre les différentes tâches : les tâches disposant d'une priorité supérieure seront choisie en priorité, disposeront de plus de temps d'exécution, ...

De la même manière, le scheduler peut prévoir un niveau appelé "real-time" (temps réel). Les tâches de ce niveau s'exécuteront dès qu'elle le désireront. Elles peuvent donc potentiellement bloquer toutes les autres tâches du système. Le scheduler ne pourra exécuter les autres tâches que quand les tâches real-time seront terminées ou en attente d'un événement.

Lié au scheduler se trouve le mécanisme des "wait queues" (aussi appelées wait-q). Il s'agit de files dans lesquelles sont placées les tâches qui doivent attendre un événement avant de pouvoir continuer. Une file dans une wait-q ne sera pas prise en compte par le scheduler. Le système disposera de plusieurs wait-q selon le rôle : attente de données d'un périphérique, d'être débloqué par une autre tâche, ...

Un mécanisme souvent utilisé pour le scheduler utilise deux listes de tâches : les tâches actives et les tâches inactives. Lorsqu'il est nécessaire de changer de tâche, si la tâche en cours a terminé son temps d'exécution (lié à sa priorité), il est transféré dans la file inactive sinon, il est remis (en tête

de liste pour sa priorité) dans la file active. La tâche suivante sera ensuite choisie dans la file active. Si la file active est vide, les files actives et inactives sont permutées. Dans ce système, une tâche temps réel sera toujours remise dans la file active.

Comme le processeur doit toujours être occupé, une tâche spéciale, de priorité inférieure à toutes les autres, est créée. Cette tâche ne fait rien et n'est utilisée que dans le cas où aucune autre tâche ne peut s'exécuter.

VI Les sémaphores

VI.1 Définition

Le sémaphore est un mécanisme qui accepte deux opérations P() et V()[6]

- P(SEM) qui sert à tester le sémaphore. Si la valeur du sémaphore est supérieure à 0, elle est diminuée de un. Sinon, le process sera mis en attente jusqu'à ce que la valeur aie été augmentée

- V(SEM) qui sert à incrémenter le sémaphore. Cette opération peut toujours être effectuée.

Ces deux opérations sont dites "atomiques" : pendant qu'elles s'exécutent, aucune autre opération ne peut être faite sur les sémaphores concernés.

La valeur d'un sémaphore sera donc toujours supérieure ou égale à 0 (un P() qui amènerait le sémaphore à une valeur inférieure à 0 sera bloquée).

P() est parfois appelée "Down" ou "Take" et V() est parfois appelée "Up" ou "Release" dans la documentation. Lorsque l'on crée le sémaphore, on lui donne une valeur initiale. Par la suite, le sémaphore n'est plus sensé être manipulé que par les fonctions P() et V().

Les sémaphores permettent de gérer les principaux problèmes de synchronisation de process : accès à une zone critique (un seul process peut accéder à une ressource à la fois), synchronisation (problème du rendez-vous, tous les processus doivent arriver au point de rendez-vous avant d'aller plus loin), problème des producteurs/consommateurs (un ou des process produit des ressources, un ou plusieurs autres les consomment), problème des lecteurs/rédacteurs (un ou plusieurs process peuvent accéder en lecture, un peut accéder en écriture et empêche les accès en lecture).

VI.2 Zone critique (MutEx)

Le problème de la zone critique peut être décrit comme suit : on désire qu'un et un seul process puisse accéder à la fois à une ressource.

Cette ressource peut être une variable partagée, afin d'éviter que entre la lecture et la réécriture de cette variable par un process, cette variable n'aie été modifiée par un autre process. Cela peut également être un ensemble de valeurs sur lesquelles plusieurs opérations doivent être faites avant de réutiliser la zone de mémoire, ...

Le terme de MutEx fait référence à l'aspect exclusif de l'accès : Mutually Exclusive (l'un ou l'autre mais pas les deux). Ce terme est souvent utilisé pour un objet qui n'est capable que de gérer cet accès précis d'un sémaphore.

Pour gérer un MutEx avec un sémaphore, on initialise le sémaphore à 1 (le nombre de process qui peuvent le prendre simultanément). Lorsqu'un process désire entrer dans la zone critique, il prend le sémaphore (P()). En prenant le sémaphore, la valeur de ce dernier est amenée à 0, ce qui empêchera qu'un second process puisse prendre le sémaphore pour accéder à la zone critique. Une fois que le process quitte la zone critique, il libère le sémaphore (V()) ce qui le ramènera à 1 et permettra donc à un autre process de le prendre.

6 Proberen (tester, essayer) et Verhogen (incrémenter).

VI.3 Producteur-Consommateur

Un ensemble de process peuvent "produire" des blocs de données que d'autres process peuvent "consommer". On peut par exemple imaginer une situation style "travail à la chaîne" où une série de process s'enchaînent pour décomposer le travail. Un premier process peut lire les données du fichier, les découper en petits morceaux qu'un (ou des) process traitera avant de les transmettre à un dernier process qui assemblera le fichier final.

On peut exploiter l'aspect bloquant des sémaphores afin de ne pas devoir tester en boucle si de nouvelles données sont disponibles. On initialise le sémaphore à 0, chaque fois qu'une donnée sera produite, on l'augmentera (V()) et, avant d'en utiliser une, on le testera (P()). La valeur du sémaphore sera égale au nombre de données disponibles et s'il n'y en a aucune, le P() bloquera jusqu'à ce qu'une donnée exploitable soit déposée.

Une variante est le cas où la place dans la file est limitée, on initialise le sémaphore au nombre d'entrées libres, avant d'utiliser un emplacement de la file, on prendra le sémaphore (P()) et quand on libérera un emplacement, on l'incrémentera (V()). On peut imaginer que le sémaphore contient le nombre de cellules libres et que l'on commence par en créer un certain nombre (lors de l'initialisation) et, par la suite, elles seront consommées (on en rempli une) ou produite (on en libère une).

Souvent, on utilisera les deux facettes en même temps, avec un sémaphore indiquant le nombre de données présentes dans la file et un second indiquant le nombre d'emplacements libres.

VI.4 Le problème du rendez-vous

Plusieurs processus doivent attendre d'avoir tous atteint le point de rendez-vous avant de pouvoir aller plus loin. Si on désire synchroniser N processus, une solution est d'utiliser N sémaphores initialisés à 0. Lorsqu'un processus arrive au point de rendez-vous, il libérera N-1 fois le sémaphore qui lui correspond et puis attendra sur chacun des N-1 autres sémaphores.

Une autre solution utilise deux sémaphores et une variable :

```
Initialisation : count = 0, S1=1, S2=0
Rendez-vous :
  P(S1)  // mutex
    count++
    if (count==N)  V(S2) ;
  V(S1)  // fin mutex
  P(S2)
  V(S2)
```

Le mutex protège l'accès à la variable commune. Chaque process qui arrive au point de rendez-vous augmentera le compteur de 1 et se mettra en attente sur S2. Lorsque le dernier arrivera au point de rendez-vous, count sera égal à N et S2 sera libéré. Un des process sera donc débloqué et libérera le suivant par l'instruction V(S2) qui suit le P(S2), et ainsi de suite jusqu'à ce qu'ils aient tous été libérés.

VI.5 Problème des lecteurs/rédacteurs

Les process doivent accéder à une ressource partagée. D'une part en lecture, d'autre part en écriture.

On peut avoir un nombre quelconque de process qui accèdent à la ressource en lecture en même temps. Par contre, si un process y accède en écriture, il doit être le seul à y accéder.

```
Initialisation : count=0, S1=1, S2=1
Rédacteur :
  P(S1)
    utilisation
  V(S1)
Lecteur :
  P(S2)
    count++
    if (count==1) P(S1)
  V(S2)
    lecture
  P(S2)
    count--
    if (count==0) V(S1)
  V(S2)
```

S1 sert de "mutex" à l'utilisation lecture/écriture. Il n'est changé qu'une fois au niveau lecture (lorsque le premier lecteur accède à la ressource ou le dernier cesse de lire)) mais à chaque rédacteur (pour s'assurer qu'il n'y aie qu'un seul rédacteur).

VI.6 Problème du Deadlock

Lorsque l'on tente de prendre un sémaphore (P()), le process peut être mis en attente qu'un autre process libère le sémaphore en question.

Si on a deux process A et B et deux sémaphores S1 et S2, la situation suivante peut arriver :

A	B	S1=1, S2=1
P(S1)		A prend S1
	P(S2)	B prend S2
P(S2)	P(S1)	A essaye de prendre S2 mais est bloqué, B essaye de prendre S1 mais est bloqué
V(S2)	V(S1)	On n'arrive pas à ces lignes qui permettraient de débloquer les process
V(S1)	V(S2)	

Les deux process A et B se bloquent mutuellement et définitivement. On parle de deadlock (verrou mortel)

Si par contre les process A et B prennent les sémaphores dans le même ordre (d'abord S1 puis S2), la situation ne peut pas se produire. En effet, une fois que l'un des process aura pris S1, l'autre ne pourra pas le prendre et donc, à plus forte raison, prendre S2.

En général, si on dispose d'un ensemble de n sémaphores S1, S2, ... , Sn, on peut éviter les situations de deadlock si tous les process prennent les sémaphore dans l'ordre numérique.

Si un process Pa est bloqué sur un sémaphore Si, ce sémaphore est pris par un autre process Pb qui ne doit plus prendre que des sémaphores dont le numéro est supérieur à i.

Si ce process Pb est lui même bloqué sur un sémaphore Sj, ce sera pas un process Pc qui ne doit plus prendre que des sémaphores de numéro supérieur à j (et donc, pas Pa), ...

Comme le nombre de sémaphore est limité à n, on arrivera a un moment à un process qui ne pourra pas être bloqué (il devrait être bloqué par un sémaphore dont le numéro serait supérieur à n). Ce process pourra donc libérer les sémaphores qu'il avait pris et ainsi de suite jusqu'à ce que Pa soit débloqué.

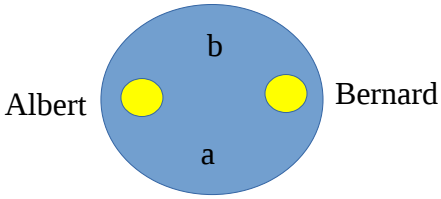
Dans le cas où on a un problème de producteur/consommateur, il faut s'assurer qu'un process en attente sur un sémaphore pour "consommer" ne bloque pas de sémaphores mutex qui seraient nécessaires pour que le producteur puisse libérer le sémaphore de producteur/consommateur. Le P() d'un producteur-consommateur doit donc se trouver hors de toute zone critique. De même, le producteur ne peut libérer le sémaphore qu'une fois la donnée effectivement disponible et donc, après les zones critiques.

VI.7 Le problème du dîner des philosophes

Autour d'une table, on dispose *n* philosophes. Comme ces derniers parlent beaucoup, on décide de faire des économies au niveau des couverts et de ne mettre qu'un couvert entre deux philosophes. Chaque philosophe pour manger devra prendre le couvert qui est à sa gauche et celui qui est à sa droite[7].

Comment faire pour être sur que tous les philosophes puissent manger (un philosophe qui a fini son assiette ne prend plus aucun couvert).

Si on considère deux philosophes, Albert et Bernard, on a deux couverts que nous appellerons a et b. Si albert prend le couvert à sa droite (a) et Bernard le couvert à sa droite (b), ils se retrouvent tous deux avec un seul couvert en main et ne savent manger ni l'un, ni l'autre, attendant chacun que l'autre dépose le couvert qu'il a pris.



Par contre, si Albert commence par prendre le couvert à sa droite mais Bernard le couvert à sa gauche, le plus rapide prendra le couvert a et l'autre attendra. Si le plus rapide est Albert (A), il pourra prendre le couvert b tout à son aise et une fois qu'il les déposera, Bernard pourra manger.

De même, si B prend le couvert a en premier, il pourra prendre le couvert b (A attend le couvert a) et manger.

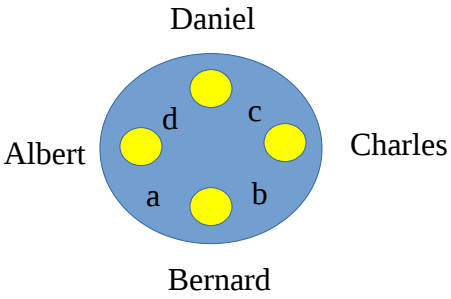
Si celui qui a pris les couverts les dépose avant d'avoir fini, on revient à la situation de départ. S'il a fini, les deux couverts sont disponibles pour l'autre.

7 On peut donc imaginer qu'ils sont chinois et mangent avec des baguettes puisque les couverts sont "interchangeables".

Si on prend quatre philosophes (Albert, Bernard, Charles et Daniel), avec les couverts a,b,c et d, le même raisonnement sera possible.

A et B tenteront de prendre d'abord le couvert a puis respectivement le couvert d ou b. C et D tenteront de prendre d'abord le couvert c puis respectivement le couvert b ou d.

Une fois que deux des philosophes auront leur premier couvert, deux cas de figure peuvent se présenter. Soit ils sont diamétralement opposés (A et C ou B et D) et leur deuxième couvert sera libre (A prend les couverts a et d, C prend les couverts c et b). Soit ils sont l'un à côté de l'autre et ce



sera au plus rapide pour prendre le second couvert (A et D voudront le couvert d, B et C voudront le couvert b)

Dans le second cas, lorsque celui qui aura pris le second couvert le déposera, l'autre pourra le prendre, manger et déposer les deux couverts. On est donc sur qu'ils pourront tous les quatre manger.

Le même raisonnement pourra être fait pour n'importe quel nombre pair de philosophes. Si le nombre est impair, le raisonnement fonctionnera aussi (on aura un philosophe qui ne sera pas en compétition avec un autre pour prendre son second couvert)

VII UNIX et Linux

VII.1 Les origines de UNIX

Au début, les sources de UNIX étaient disponibles, ce qui a permis pas mal de contributions et d'améliorations... Jusqu'à ce que AT&T décide de commercialiser le système.

UNIX était principalement programmé en langage C, un langage disponible sur la majorité des architectures.

Deux principales variantes sont apparues : System V (utilisé entre autre sur les systèmes de chez Sun) et BSD. Chacune de ces variantes avait des caractéristiques propres et les incompatibilités ont commencé à être de plus en plus nombreuses.

Afin de ramener la compatibilité entre les système de type UNIX, une norme intégrant les éléments les plus intéressants de chaque côté a été créée : la norme POSIX. De nos jours, on peut trouver les IPC System V ou le système NIS[8] tant sur les variantes BSD que sur les variantes System V et les nouveaux programmes n'ont plus qu'une seule norme (POSIX) à respecter pour pouvoir être de nouveau compatibles entre les différents systèmes UNIX.

VII.2 Le projet GNU

En 1984, Richard Stallman a lancé le projet GNU qui visait à défendre le concept d'informatique libre (dans le sens liberté et pas gratuite). Ce concept est appuyé à l'aide d'une licence appelée la GPL (GNU Public Licence)

Le premier élément de ce projet était l'éditeur de fichiers EMACS mais il a rapidement été suivi de GCC, un compilateur C et d'autres éléments nécessaires à la réalisation d'un système d'exploitation. Tout ces outils sont distribués sous la GPL et on peut les retrouver à de nombreux endroits.

La GPL, aussi appelée "copyleft", est une licence qui autorise la distribution, la modification, la création d'œuvres dérivées à la condition que la licence GPL soit conservée sur les programmes dérivés et que les sources (listings) de ces derniers soient rendu disponibles à ceux à qui on distribue les nouvelles versions.

VII.3 Linux

En 1991, un étudiant finlandais, Linus Torvalds, décide de développer un petit système d'exploitation de type UNIX. Une fois ce noyau créé, il décide de le lâcher sur internet en le laissant ouvert aux modifications et ajoutes.

Même s'il ne se sent pas concerné par le projet GNU, il décide d'utiliser la GPL qui correspond bien à ce qu'il désire, plutôt que de passer son temps à créer lui-même un licence pour son OS.

Depuis, des développeurs du monde entier ont ajouté des nouvelles fonctionnalités, parfois juste pour le plaisir, parfois payés par des entreprises qui désirent l'une ou l'autre fonctionnalité.

8 Anciennement appelé Yellow Pages, ce système permet le partage des identifiants entre plusieurs machines connectées en réseau

Un système d'exploitation en lui même ne sert à rien sans une série d'outils qui permettent de l'exploiter. Plusieurs personnes ont donc regroupé Linux (l'OS) avec des outils de différentes sources, parmi lesquels de nombreux outils du projet GNU (GCC, Bash, les outils de base UNIX, ...) mais également des projets non GNU (Xfree puis Xorg pour le système graphique, Gimp, Audacity, OpenOffice, ...)

Si Linux a été initialement développé pour les processeurs Intel 32 bits, on a depuis ajouté de nombreuses architectures telles que MIPS, Alpha, SPARC, ARM, ... De nombreuses ajoutes ont également été apportées pour suivre les normes POSIX.

VII.4 Les distributions Linux

Une distribution Linux est un ensemble formé de

- un noyau Linux
- des outils de base et des applications
- un installeur

Les premières distributions étaient stockées sur des ensembles de disquettes (SLS, MCC, Slackware,...), souvent sous la forme de plusieurs groupes de disquettes, chaque groupe concernant un aspect (réseau, Emacs, compilateur, X11 (le système graphique), ...)

Les distributions suivantes étaient réalisées sous forme d'un puis plusieurs CD et maintenant de DVD, la taille des distributions augmentant avec le nombre d'application présentes.

On trouve plusieurs familles de distributions, généralement articulées autour du programme qui sert à gérer les différents packages

- La famille Debian (Ubuntu, Mint, SteamOS, ...) basée sur des packages .deb
- La famille RedHat (Fedora, Centos, Suse, ...) basée sur des packages .rpm
- Gentoo (pas de packages mais des scripts qui téléchargent et compilent les sources des différents éléments)
- La famille Slackware (orientée sur des archives tgz (équivalent UNIX des fichiers .ZIP) contenant les différents programmes)
- ...

La plupart de ces distributions sont téléchargeables gratuitement, l'entreprise qui se trouve derrière proposant un support payant. Certaines sont par contre purement commerciales (RedHat, Suse, ...).

Toutes ces distributions proposent principalement les mêmes outils (venant du monde du logiciel libre) souvent accompagnés de leurs propres outils pour la configuration, la gestion des programmes installés, ...

VII.5 Philosophie UNIX

Les systèmes UNIX s'articulent autour de deux grands principes :

- Tout est fichier

Qu'il s'agisse des données, des programmes, des connexions réseaux, des périphériques (clavier, écran, disque dur, lecteur de CD-ROM/DVD, connexion série, carte son, ...), les fonctions d'accès sont celles liées aux fichiers.

On utilisera ainsi la même fonction pour écrire dans un fichier, envoyer des données à l'autre extrémité d'une connexion TCP/IP (réseau), écrire sur la console (écran en mode texte) ou envoyer des échantillons sonores vers le haut parleur.

- Des outils simples, qui font peu de choses mais le font bien

Les outils de base sous UNIX sont extrêmement spécialisés, restreint à une seule tâche, ce qui permet qu'il restent de petite taille et limite fortement les risques d'erreur de programmation.

Pour réaliser une tâche complexe, on combinera plusieurs de ces outils, le résultat de l'un servant de données au suivant.

VII.6 Un système multi-tâches et multi-utilisateurs

Depuis le début, les systèmes de type UNIX sont prévus pour permettre la connexion *simultanée* de plusieurs utilisateurs sur la même machine et le lancement de plusieurs programmes simultanément.

Cela demande dès lors des mécanismes de protection et de contrôle d'accès

- au niveau des fichiers, où il doit être possible de préciser quels utilisateurs ont accès à un fichier et quels accès
- au niveau des processus où un utilisateur ne doit pas être autorisé à toucher aux processus appartenant aux autres utilisateurs.
- un système permettant d'identifier un utilisateur (login) lorsqu'il se connecte sur la machine ou de passer d'un utilisateur à un autre.

Un utilisateur spécial peut outrepasser toutes ces restrictions. Cet utilisateur, *root*, est réservé à l'administration du système.

Les premiers ordinateurs sous UNIX utilisaient des télétypes (appareils composés d'un clavier et d'une imprimante) puis des terminaux (appareils composés d'un écran et d'un clavier, qui envoient sur un port série les codes des touches enfoncées et affichent à l'écran les caractères reçus par ce même port série).

Les systèmes UNIX actuels se tournent maintenant vers des terminaux virtuels reliés soit à des "consoles virtuelles" (sortes de terminaux gérés en mode texte à l'aide du clavier et de l'écran de la machine), à des connexions réseau (telnet ou ssh) ou à des programmes terminaux sur un écran graphique (xterm, rxvt, gnome-terminal, ...) mais ils restent généralement capable de gérer des terminaux sur le port série.

VII.7 Le Shell

L'entrée de commandes afin d'exécuter des programme se fait au travers d'un programme appelé *shell*. De nombreux shells existent, allant des plus simples en mode texte aux plus évolués en mode graphiques (le "menu démarrer" d'une interface graphique est en fait une forme de shell).

Si les interface graphiques ont l'avantage de pouvoir être facilement utilisées, les shells en ligne de commande (on parle de CLI : Command Line Interface) permettent généralement d'effectuer des tâches bien plus complexes.

VII.7.a Les différents shell

- Le shell Bourne, œuvre de Steve Bourne, est le plus ancien. Il a été créé pour le système UNIX développé au sein de la société AT&T. Anciennement défini par la commande sh, on le rencontre souvent aujourd'hui sous le nom de commande bsh (ou /usr/old/bin/sh).
- Le C-shell (csh), créé par Bill Joy pour les systèmes Unix BSD de l'université de Berkeley, a introduit des nouveautés comme le rappel de commandes ou les alias de commandes. Il doit son nom à la très grande parenté entre sa syntaxe et celle du langage C.
- Le Korn shell (ksh) est aussi une réalisation de la société AT&T. Il a été créé par David Korn. Il complète le shell Bourne avec de nouvelles fonctionnalités comprenant notamment les avantages du C-shell.
- Le shell POSIX (sh) est dérivé du shell Bourne et intègre de nombreuses fonctionnalités du Korn shell. Normalisé par l'IEEE et l'ISO, il est devenu le standard des interpréteurs et assure ainsi la portabilité des applications écrites en langage shell.
- Le shell restreint (rsh) est un shell Bourne auquel une série de restrictions ont été ajoutées. Il est prévu pour être utilisé par des programmes amenés à exécuter des scripts shell (comme Sendmail) tout en s'assurant que ces scripts tourneront dans un environnement confiné.
- Le shell bash (« Bourne Again Shell ») est un shell libre de droit et téléchargeable sur internet. Il a été écrit par Brian Fox et Chet Ramey. Il est compatible avec le shell POSIX. C'est le shell par défaut du système UNIX et Linux.
- Le shell tcsh est une version améliorée et libre du célèbre C-shell de l'université de Berkeley.
- Le shell zsh est un shell conçu principalement pour une utilisation interactive. On peut cependant l'utiliser pour exécuter des scripts. Il incorpore la plupart des caractéristiques des shells bash, ksh et tcsh en plus de caractéristiques qui lui sont propres.
- Le shell tclsh est lié au langage de programmation TCL et permet de mélanger commandes de type shell et instructions de haut niveau en TCL
- Le shell pdmenu est un shell qui propose une interface sous forme de menus en mode textes. Il permet de n'autoriser que quelques instructions prédéfinies à un utilisateurs sous une forme simple (ensemble de menus textes)
- Le shell ash est un shell dérivé du shell POSIX qui se veut rapide et compact ce qui fait qu'on le trouve souvent dans le monde des systèmes embarqués (il est d'ailleurs intégré dans

l'outil busybox) mais également sous FreeBSD, NetBSD et une de ses variantes (dash) sous Ubuntu.

- ...

Au niveau de ce cours, nous nous intéresserons principalement au shell bash présent en standard sous Linux mais une grande majorité des explications s'appliqueront également aux autres shells

VII.8 Arborescence des fichiers

Les fichiers sur un système UNIX sont organisés dans des *répertoires* (dossiers). Ces répertoires sont organisés en arbre et contiendront les différents fichiers du système[9].

Le répertoire de base (aussi appelé répertoire racine) est désigné par le caractère /. Ce caractère sert aussi à séparer les différents noms de répertoires pour créer un chemin complexe (*/usr/local/bin* pour désigner le répertoire *bin* contenu dans le répertoire *local*, lui même contenu dans le répertoire *usr* qui se trouve dans le répertoire racine par exemple).

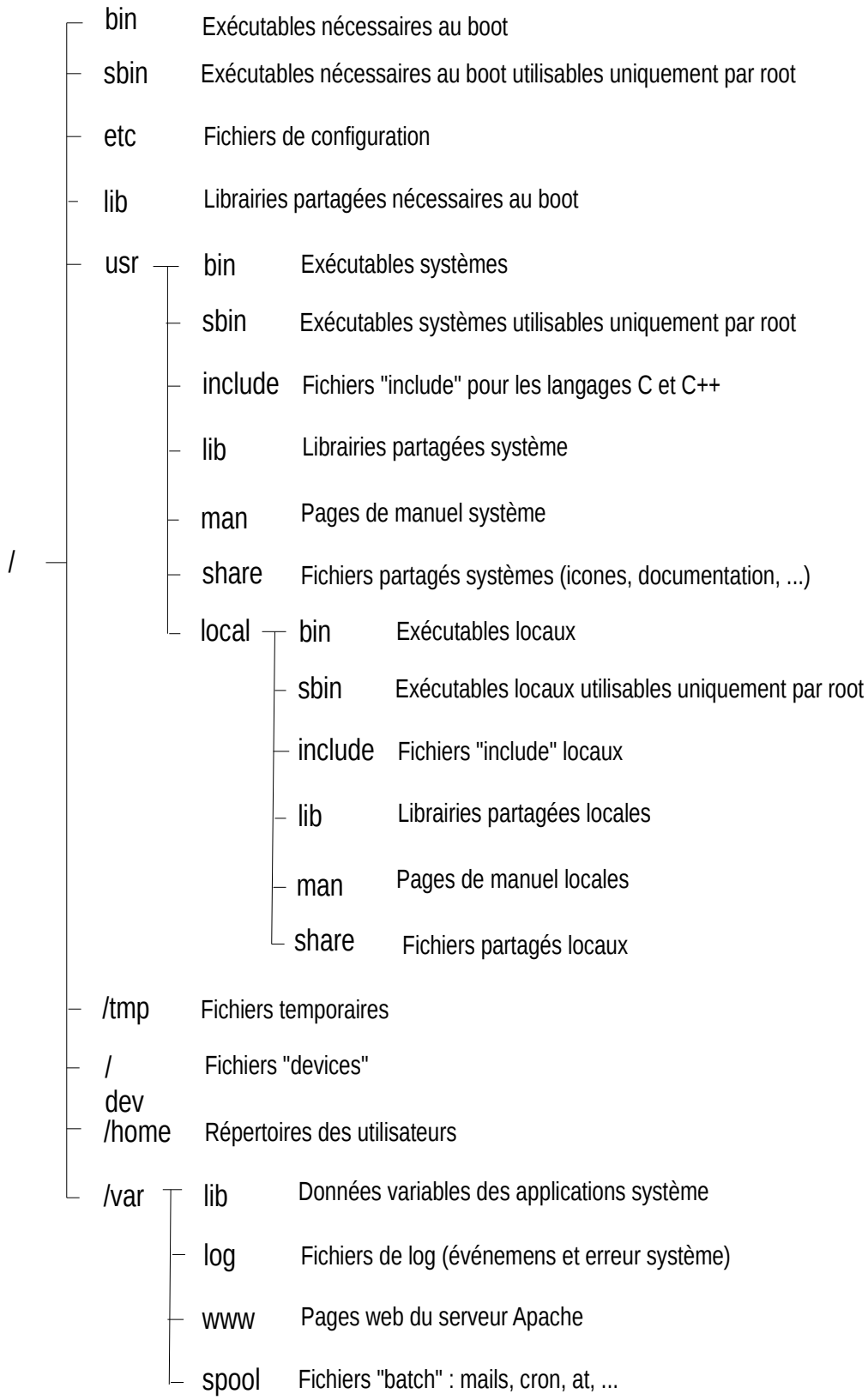
L'ensemble des répertoires et des fichiers qui se trouve dans ces derniers présents sur un périphérique (partition de disque dur, CD-ROM, partage réseau) est organisé en un "file system". Le terme "file system" désigne aussi bien cet ensemble d'éléments que la méthode utilisée pour structurer les informations (ext3 ou ext4 pour des données sur une partition Linux, iso9660 pour un CDROM, udf pour un DVD, fat pour une carte SD, ntfs pour une partition windows, ...)

L'arborescence de base UNIX a été fortement influencée par les premiers systèmes où l'espace disque était peu disponible. Une grande partie du système se trouvait au niveau d'un serveur central et les machines ne conservaient en local que le strict minimum.

Les fichiers fixes étaient également partagés entre les différentes machines et donc, étaient partagées en lecture seule dans */usr* alors que les parties qui pouvaient changer (variables) se trouvaient dans */var*.

Sous UNIX, contrairement à DOS/Windows, les systèmes de fichiers supplémentaires (CD-ROM, disquettes, partages réseaux et autres partitions) sont accrochés à des sous-répertoires. C'est ainsi que l'on trouve souvent un répertoire */mnt* prévu pour accrocher de façon temporaire une telle partition. Les UNIX modernes possèdent souvent également un */media* pour les supports extérieurs tels que CDROM, clé USB, carte SD, ...

9 A noter qu'un répertoire est un type de fichier particulier qui contient des noms de fichiers et des numéros d'i-nodes (entités décrivant le fichier)


/
bin
Exécutables nécessaires au boot
sbin
Exécutables nécessaires au boot utilisables uniquement par root
etc
Fichiers de configuration
lib
Librairies partagées nécessaires au boot
usr
bin
Exécutables systèmes
sbin
Exécutables systèmes utilisables uniquement par root
include
Fichiers "include" pour les langages C et C++
lib
Librairies partagées système
man
Pages de manuel système
share
Fichiers partagés systèmes (icones, documentation, ...)
local
bin
Exécutables locaux
sbin
Exécutables locaux utilisables uniquement par root
include
Fichiers "include" locaux
lib
Librairies partagées locales
man
Pages de manuel locales
share
Fichiers partagés locaux
/tmp
Fichiers temporaires
/dev
Fichiers "devices"
/home
Répertoires des utilisateurs
/var
lib
Données variables des applications système
log
Fichiers de log (événemens et erreur système)
www
Pages web du serveur Apache
spool
Fichiers "batch" : mails, cron, at, ...

VII.9 Types de fichiers

Un des grands principes de UNIX, c'est que toutes ressources matérielles (imprimante, unité de disque, …) et autres (fichiers de données, répertoire) sont assimilées à des fichiers.

Plusieurs types de fichiers sont différenciés :

- fichiers normaux (classiques, fichiers de données) : contiennent des textes en clair (fichiers de données) ou un programme exécutable. S'ils contiennent un programme exécutable, on parle également de fichiers binaires.
- Répertoires (ou catalogues, directories) : permettent d'organiser l'espace du disque dur. Les répertoires peuvent contenir des sous-répertoires, … qui peuvent eux-mêmes contenir des sous-répertoires et des fichiers. Ces répertoires associent des noms de fichiers à des numéros d'i-nodes. Ils sont représentés par la lettre **d**
- Fichiers devices de type blocs ou caractères correspondent et se substituent aux différents dispositifs d'E/S. Une lecture ou écriture vers ces fichiers provoque en réalité un acheminement de données entre la mémoire et l'unité d'E/S. C'est ce qui explique pourquoi un process utilisateur peut envoyer ou recevoir indifféremment des données vers ou en provenance de fichiers ou de périphériques. Ces fichiers spéciaux sont généralement repris dans le répertoire */dev* et sont représentés par les lettres **c** ou **b**.
- Lien « symbolique » : objet pointant sur un autre. Ceci permet de simuler l'existence de plusieurs copies d'un fichier alors que physiquement, il n'y en qu'une. Les liens symboliques peuvent remplir les mêmes usages que les « raccourcis » de Windows. Ils sont représentés par la lettre *l*
- Les sockets : représentent des points d'accès réseau de type UNIX (par opposition aux types IPV4 et IPV6 pour les connexions de type "internet" version 4 et 6 par exemple) utilisables uniquement en local (pas entre plusieurs ordinateurs) et représentés par la lettre **s**
- Pipes nommés : ces fichiers sont prévus pour êtres ouverts par deux programmes : l'un en lecture et l'autre en écriture. Tout ce que le second écrira sera lu dans le même ordre par le premier. Ils sont représentés par la lettre **p**

VII.10 Les i-nodes

Contrairement à Windows qui stocke ensemble le nom d'un fichier et ses autres caractéristiques (taille, date de création, droits d'accès, ...), UNIX lie les noms de fichiers à des numéros d'i-nodes et conserve les i-nodes dans une table à part.

Ces i-nodes contiennent toutes les informations concernant le fichier : taille, date de création, de dernière modification, droits d'accès et blocs utilisés sur le disque dur.

Cela permet d'associer plusieurs noms de fichiers au même numéro d'i-node et donc de disposer d'aliases sur un fichier. Chacun de ces noms est totalement équivalent aux autres et le fichier ne sera effacé que lorsqu'il ne restera plus aucun nom qui lui est associé. On parle de liens physiques (par

opposition aux liens symboliques qui sont des fichiers à part entière, avec leur propre numéro d'i-node).

Afin d'éviter des problèmes (récursion infinie), seul l'operating system peut créer des liens physique sur des répertoires. C'est ce qui est fait pour les noms spéciaux "." qui est un lien physique vers le répertoire où il se trouve et ".." qui est un lien physique vers le répertoire parent.

VII.11 Les noms de fichiers

Le nom d'un fichier se compose de caractères alphabétiques, numériques et de certains caractères spéciaux. (le nombre de caractères utilisé varie d'un système à l'autre).

Vous avez donc la possibilité de créer des noms significatifs, mais il vaut mieux que les noms ne soient pas trop longs non plus. (voire jusque 128 à 255 caractères).

Aussi, vous pouvez utiliser des lettres minuscules et majuscules dans les noms de vos répertoires et de vos fichiers, mais souvenez-vous que UNIX fait la différence entre les lettres minuscules et majuscules. Ainsi, les noms PROJET et Projet sont 2 noms différents pour UNIX.

Voici quelques conseils qui vous pourront vous éviter des ennuis :

- N'utilisez pas de caractères accentués dans vos noms de fichiers et de répertoires; limitez-vous aux lettres, aux chiffres et, comme caractères spéciaux, au tiret (-, à éviter comme premier caractère) et au sous-tiret (_), les caractères spéciaux ayant parfois des fonctions particulières
- Évitez d'avoir les mêmes noms avec des caractères dans des casses différentes (i.e. évitez d'avoir à la fois Texte et texte dans le même répertoire)
- Les noms de fichiers commençant par un point (.) sont des fichiers masqués. Ils n'apparaissent que si on le demande explicitement. Cela permet de ne pas afficher les fichiers de configuration et semblables par défaut.
- nom.ext : par convention l'extension donne une indication sur le type de fichier (.txt ; .c ; .h ; .s ;…) mais fait partie du nom sous UNIX. Lorsqu'il s'agira d'identifier le type de données, UNIX utilise souvent le principe des "magic-names" (séquences d'octets à une position donnée spécifiques à un type de fichier, par exemple, les images GIF commencent toutes par "GIF87" ou "GIF89", les fichiers Java par la valeur entière 0xcafe (en hexadécimal), les fichiers ZIP par les deux lettres PK, ...)

VII.12 Chemins et répertoire actif

Pour identifier un fichier (ou un répertoire), on peut utiliser différentes notations qui permettront d'identifier le fichier et sa position dans l'arborescence des répertoires

VII.12.a Chemin absolu

Un chemin absolu commence par le caractère / et précise tous les répertoires à traverser pour arriver au fichier (ou répertoire) concerné en les séparant par des caractères /.

On trouvera par exemple */usr/include/stdio.h* qui désignera le fichier *stdio.h* présent dans le répertoire *include* lui même présent dans le répertoire *usr* présent dans le répertoire racine.

VII.12.b Répertoire actif

Chaque programme tournant sur la machine (y compris le shell) possède un répertoire actif (généralement le répertoire dans lequel il a été lancé).

Si on précise un nom de fichier directement, UNIX considérera qu'il se trouve dans le répertoire actif. Le nom de fichier *fich1* concernera donc le fichier *fich1* situé dans le répertoire actif.

Si on désire désigner le répertoire actif, on pourra utiliser le nom de fichier ".". De même, il est possible de désigner le répertoire parent (un niveau dans la direction du répertoire racine) en utilisant le nom de fichier ".."

VII.12.c Chemin relatif

On peut également préciser le nom d'un fichier en indiquant un chemin à partir du répertoire actif. On parle de chemin relatif.

Pour ce faire, on indiquera le chemin à suivre en commençant par un nom de répertoire (et non par un / comme pour le chemin absolu) et en séparant les différents noms par des caractères / (comme pour un chemin absolu).

Si on désire indiquer un fichier se trouvant dans le répertoire actuel, on pourra utiliser le fichier "." et donc préciser un fichier tel que *./fich1*.

De même, il est possible de remonter d'un (ou plusieurs) niveaux de répertoires à l'aide des fichiers ".." présents dans les différents répertoires (par exemple, *../../fich1* pour remonter de deux niveaux et prendre le fichier *fich1*).

VII.12.d Répertoire personnel

Le répertoire personnel de l'utilisateur (généralement */home/login* où *login* est l'identifiant de l'utilisateur) est identifié par le caractère ~. On pourra donc utiliser un nom de fichier tel que *~/fich1* pour désigner le fichier *fich1* se trouvant dans son répertoire personnel.

Ce répertoire personnel est appelé "home directory" ou "homedir".

Il est également possible d'accéder au homedir d'un autre utilisateur en utilisant la syntaxe *~login* où *login* est le nom de l'utilisateur concerné.

VIII Les commandes UNIX

Une fois que l'on se trouve dans le shell, on peut utiliser une série de commande pour effectuer diverses actions.

Ces commandes respectent la structure suivante :

`commande [-`*`option [paramètre option]`*`] `*`paramètre`*

La commande peut être suivie d'une ou plusieurs options. Ces options ont deux formes possibles : une forme courte composée d'un tiret suivi d'une lettre (-o) et une forme longue composée de deux tirets suivis d'un mot (--help).

Lorsque l'on a plusieurs options courtes, il est possible de les regrouper (-a -l -s -F => -alsF). Ce genre de regroupement n'est pas possible avec les options longues.

Certaines options acceptent un paramètre. Il doit être placé directement après l'option.

La commande peut ensuite être suivie d'un ou plusieurs paramètres. Ces derniers sont séparés par des espaces. Si on désire utiliser un espace dans un tel paramètre, on le placera entre guillemets ou on précédera l'espace par un caractère \

Dans ce qui suit, seules les options les plus importantes seront citées.

VIII.1 Fileglob

Partout où un ou des noms de fichiers peuvent être utilisés, on peut utiliser des caractères spéciaux qui permettront de citer plusieurs fichiers simultanément. On parle de fileglob.

- L'étoile (*) remplace une chaîne de caractères quelconque (même vide). ***.c** indique tous les fichiers se terminant par .c, **h*c** indique tous les fichiers qui commencent par **h** et finissent par **c** (aussi bien *hello.c* que *hc* ou *hic*). On peut utiliser plusieurs étoiles dans un nom de fichier
- Le point d'interrogation remplace un caractère unique. Ainsi, **hello?.c** représente **hello1.c**, **helloa.c** mais pas **hello.c** (il faut obligatoirement un caractère)
- Une série de caractères entre crochets carré représente un caractère dans la liste citée **[abc]def** représentera ainsi **adef**, **bdef** et **cdef**.
- On peut préciser des intervalles de caractères entre crochets carrés : **[a-zA-Z0-9]** représente ainsi une lettre minuscule, majuscule ou un chiffre.
- On peut préciser un ensemble de fichiers séparés par des virgules entre accolades **{abc,def}** représente **abc** et **def**

VIII.2 Pages de manuel

VIII.2.a man

La commande man permet d'afficher la page de manuel associée à une commande. Sa syntaxe est

man [section] commande

Si on précise la section, la commande sera recherchée dans la section correspondante des manuels (comparable à des tomes séparés d'une encyclopédie). Sinon la commande sera cherchée dans les différentes sections par ordre de section et la première commande trouvée sera affichée

La section 1 concerne principalement les commandes UNIX standard.
La section 2 concerne les appels systèmes
La section 3 concerne les fonctions du langage C (strcpy, printf, ...)
La section 4 concerne les fichiers spéciaux et les drivers
La section 5 concerne les fichiers de configuration et autres
La section 6 concerne les jeux installés
La section 7 est un fourre tout (packages de macros, conventions, ...)
La section 8 contient les commandes d'administration systèmes
La section 9 contient les informations sur les routines du système (non standard)

Les sections les plus intéressantes sont 1 et 8 pour les commandes du shell, 2 et 3 pour la programmation en C/C++ et 5 pour les fichiers de configuration.

VIII.3 Manipulation des répertoires

VIII.3.a ls

La commande ls (LiSt) permet d'afficher la liste des fichiers présents dans un répertoires. On peut lui fournir en paramètre un ou des fichiers à afficher (y compris des fileglob) auquel cas, seuls ces fichiers seront affichés. Si on fourni un (ou des) répertoire, tous les fichiers du répertoire concerné seront affichés.

Quelques options permettent de modifier l'affichage.

-a	Affiche tous les fichiers. Par défaut, les fichiers qui commencent par un point ne sont pas affichés.
-l	Affichage long. Cet affichage contiendra, en plus des noms des fichiers, leurs droits, le nombre de liens physiques sur le fichier, leur taille et la date de dernière modification
-F	Ajoute un caractère en fin des noms de fichiers pour aider à leur identification. Les fichiers exécutables auront une étoile ajoutée (*), les répertoire un /, les sockets UNIX un =, les pipes une barre verticale (|), les liens symboliques un arobase (@).
-s	Affiche le fichier pointé pour un lien symbolique. Si -F est présent, l'arobase ne sera pas affichée. Le fichier pointé apparaîtra sous la forme -> *fichier* après le nom.
-R	Affiche de façon récursive : affiche le contenu des sous-répertoires en plus de celui du répertoire listé.

VIII.3.b cd

La commande cd (Change Dir) permet de changer le répertoire actif. Utilisée seule, elle permet de revenir dans le répertoire home de l'utilisateur (ce qui est l'équivalent de cd ~). Sinon, elle permet d'aller dans le répertoire indiqué.

VIII.3.c pwd

La commande pwd (print working directory) affiche le répertoire courant. Elle ne prend aucun paramètre.

VIII.3.d mkdir

La commande mkdir (MaKe DIR) permet de créer un (ou des) répertoire passé en paramètre. Si le répertoire existe déjà, la commande affichera une erreur.

On peut utiliser l'option suivante :

-p	Crée les répertoires parents : si le répertoire parent du répertoire à créer n'existe pas, il sera créé et ainsi de suite en remontant jusqu'au répertoire racine.

VIII.3.e rmdir

La commande rmdir (ReMove DIR) permet d'effacer un répertoire vide. Si le répertoire n'est pas vide, il ne sera pas effacé. Le répertoire à effacer sera passé en paramètre.

VIII.3.f pushd/popd

Ces commandes doivent être utilisées conjointement.

La commande pushd empile le répertoire actif et se déplace (comme cd) dans le répertoire indiqué.

La commande popd dépile le dernier répertoire empilé et y retourne.

Ces commandes permettent ainsi de changer temporairement de répertoire et de revenir ensuite dans celui dans lequel on se trouvait.

VIII.4 Gestion des fichiers

VIII.4.a cp

La commande cp (CoPy) permet de copier des fichiers. On l'utilise de la forme

cp [options] source1 [source2 ...] destination

Par défaut, les sources ne peuvent pas être des répertoires (voir option -r). La destination peut être un répertoire (la copie sera faite dans le répertoire en question) ou un nom de fichier (dans ce cas, on ne peut utiliser qu'un seul fichier source, la copie aura ne nouveau nom).

On peut utiliser les options suivantes :

-r	Permet de copier une arborescence de répertoires. La source sera un répertoire
-f	Permet de forcer la copie (si le fichier destination existe déjà)

-i	Interactif : si le fichier destination existe, demande une confirmation
-p	Conserve les droits du fichier d'origine

VIII.4.b mv

La commande mv (MoVe) permet de déplacer ou renommer un fichier. La syntaxe est la même que pour la commande cp.

Si la destination est un répertoire, le fichier ou répertoire source y sera déplacé. Si le nom destination n'existe pas ou est un nom de fichier, la source sera renommée.

On peut utiliser les options suivantes :

-f	Force le renommage (si la destination existe)
-i	Interactif : si la destination existe, demande une confirmation

VIII.4.c ln

La commande ln (LiNk) permet de créer un lien physique ou symbolique de la source vers la destination. La syntaxe est semblable à celle des commandes cp et mv.

Il n'est pas possible de créer un lien physique d'un répertoire (il s'agit d'une protection permettant d'éviter des bouclages qui provoqueraient des appels récursifs sans fin)

Si la destination est un répertoire, le ou les liens y seront créés.

Les options suivantes peuvent être utilisées

-s	Crée un lien symbolique.

Un lien physique est une seconde entrée de répertoire qui pointe vers le même i-node (et donc, le même fichier). Un fichier peut ainsi avoir un nombre quelconque de noms qui le désignent (aliases) et il ne sera effacé que lorsque le dernier lien sera effacé (en langage C, la commande qui permet d'effacer un fichier ou plutôt, une entrée de répertoire, est la commande *unlink()*)

Un lien symbolique crée un nouveau fichier (de type lien symbolique) qui contient le nom du fichier (ou répertoire) pointé. Si le fichier pointé est effacé, le lien pointe vers un emplacement vide (on parle de *dangling link*). Si on recrée un fichier avec le nom du fichier pointé, le lien pointera vers ce nouveau fichier.

VIII.4.d rm

La commande rm (ReMove) permet d'effacer un fichier. Sauf dans le cas de l'option -r, le fichier effacé ne peut pas être un répertoire.

La syntaxe de la commande rm est la suivante :

```
rm [options] fichier [fichier...]
```

Si le fichier que l'on demande d'effacer n'existe pas, rm affichera un message d'erreur.
Rm accepte les options suivantes

-f	Force l'effacement. Permet d'éviter le message d'erreur si le fichier n'existe pas (ou de "contrer" l'option -i)
-i	Interactif : demande confirmation pour chaque fichier (certaines distributions Linux créent par défaut un alias qui ajoute automatiquement l'option -i)
-r	Récursif : si le fichier à effacer est un répertoire, efface le contenu de ce dernier et puis efface le répertoire.

VIII.4.e touch

La commande touch permet de changer l'heure de dernière modification du fichier passé en paramètre. Si le fichier n'existe pas, cette commande créera un fichier vide.

Cela peut être utilisé pour forcer une recompilation (make utilise les heures de dernières modification pour déterminer ce qu'il doit recompiler), pour forcer la sauvegarde d'un fichier par un logiciel de backup qui ne copierait que les fichier récemment modifiés, ...

VIII.4.f cat

La commande cat permet d'afficher le contenu d'un fichier texte. La totalité du fichier est affichée à l'écran (stdout).

Si plusieurs noms de fichiers sont fournis, ils seront affichés à la suite l'un de l'autre.

VIII.4.g more/less

Les commandes more et less permettent d'afficher un fichier page par page. La commande more est la commande standard UNIX mais de nombreux Linux utilisent less par défaut.

Le fichier à afficher est passé en paramètre à l'instruction more (ou less). Dans le cas de more, une fois la fin du fichier atteinte, la commande se termine. Dans le cas de less, il faudra arrêter soi même la commande (à l'aide de la touche Q)

Les touches suivantes peuvent être utilisées pendant l'affichage par more/less

[Enter]	Avancer d'une ligne
[Espace]	Avance d'une page
Q	Quitte le programme

VIII.4.h tail

La commande tail permet d'afficher la fin d'un fichier. Cette commande est intéressante quand seules les quelques dernières lignes sont intéressantes (fichiers de log par exemple).

Cette commande est exécutée comme suit

```
tail [options] fichier
```

-f	Follow : une fois les dernières lignes affichées, le programme continue à tourner et affichera les nouvelles lignes au fur et à mesure qu'elle apparaîtront dans le fichier.
-n 15	Permet de changer le nombre de lignes affichées. L'option doit être suivie du nombre de lignes que l'on veut afficher (ici 15).

VIII.4.i head

La commande head permet d'afficher le début d'un fichier. Sa syntaxe est semblable à celle de la commande tail.

On peut utiliser l'option suivante :

-n 15	Permet de changer le nombre de lignes affichées. L'option doit être suivie du nombre de lignes que l'on veut afficher (ici 15).

VIII.5 La gestion des droits

Le système Linux supporte plusieurs mécanismes de contrôle de droits sur les fichiers :

- permissions standard UNIX. L'utilisateur root (administrateur) peut accéder à un fichier/répertoire même si les permissions UNIX ne lui permettent pas.
- Les attributs des fichiers (immutable, append-only, ...). Ces attributs ne peuvent être changés que par root et s'appliquent à tous les utilisateurs (dont root).
- ACL : les permissions sont gérées par utilisateur/groupe, comme sous MS Windows[10]
- SELinux[11] (la gestion des droits se fait non seulement au niveau utilisateur mais également au niveau application, à l'aide de règles définies au niveau du système) pour les systèmes sécurisés.

Nous n'examinerons ici que les permissions de type UNIX.

VIII.5.a Comment fonctionnent les permissions de base ?

Chaque élément du système de fichier (fichier, répertoire, ...) possède trois attributs : un numéro d'utilisateur, un numéro de groupe et des drapeaux d'accès.

Lorsqu'un programme s'exécute, il possède un ensemble de 4 valeurs : *effective user*, *effective group*, *real user*, *real group*.

Lorsqu'un utilisateur lance un programme, les effective et real user/group correspondront aux identifiants de l'utilisateur.

10 Ce mécanisme demande bien plus de ressources pour vérifier les autorisations et donc doit être évité dans la mesure du possible.

11 Ce système a été mis en place par la NSA

- Lorsque le programme veut accéder à un fichier, on vérifie d'abord si l'*effective user* correspond au numéro d'utilisateur du fichier. Si c'est le cas, les permission de l'utilisateur seront utilisée.
- Sinon, on vérifie si le groupe du fichier est le *effective group* ou dans la liste des groupes de l'*effective user*. Si c'est le cas, les permissions du groupe seront utilisées.
- Si aucun des deux cas n'est rencontré, on utilise les permissions autres.

Les drapeaux de droits sont une valeur de 3 ou 4 chiffres en octal (base 8, des valeurs de 0 à 7). Les trois derniers chiffres sont dans l'ordre les droits de l'utilisateur, du groupe et des autres. Ainsi, une valeur de 2751 ou de 751 correspond à 7 pour l'utilisateur, 5 pour le groupe et 1 pour les autres.

Les droits utilisateurs/groupes/autres sont la somme de 4 pour les droits en lecture, 2 pour les droits en écriture et 1 pour les droits en exécution. Ainsi, la valeur 751 donne les droits en lecture, écriture, exécution pour l'utilisateur, lecture et exécution pour le groupe et juste exécution pour les autres.

valeur	Signification	Pour les fichiers	Pour les répertoires
4	Read (r)	Permet de lire le contenu du fichier	Permet d'afficher la liste des fichiers contenus dans le répertoire
2	Write (w)	Permet de modifier le contenu du fichier	Permet de créer un nouveau fichier, d'en effacer un ou d'en renommer un dans le répertoire
1	eXecute (x)	Indique que le fichier est exécutable (programme compilé, script bash, Perl, PHP, ...)	Permet de rentrer dans le répertoire et d'accéder aux fichiers qu'il contient

VIII.5.b SetUID/SetGID/Sticky

Le 4ème chiffre (au début) reprend quand il est présent le *SetUID* bit, le *SetGID* bit et le *Sticky* bit.

Les trois bits spéciaux ont des significations différentes selons qu'il sont utilisés sur un répertoire ou un fichier.

4	SetUIDbit	Lorsque le programme sera exécuté, le effective user contiendra l'utilisateur propriétaire du fichier au lieu de l'utilisateur qui l'a exécuté	
2	SetGIDbit	Lorsque le programme sera exécuté, le effective group contiendra le groupe du fichier au lieu du groupe principal de l'utilisateur qui l'a exécuté	Si on crée un fichier ou un répertoire dans ce répertoire, son groupe sera automatiquement celui du répertoire (à condition que l'utilisateur ait accès au groupe en question). Si on crée un répertoire, il aura automatiquement le SetGIDbit activé
1	Stickybit	Indique que le programme doit rester en mémoire (peu utilisé)	Seul le propriétaire d'un fichier contenu dans le répertoire pourra effacer le fichier en question (root peut toujours le faire)

Pour un répertoire, le SetGID bit indique qu'un fichier créé dans le répertoire aura le même groupe que le répertoire (si l'utilisateur a accès au groupe en question) et le Sticky-Bit indique que les fichiers contenus dans le répertoire ne peuvent être effacés que par l'utilisateur propriétaire ou par root (utilisé par exemple pour le répertoire /tmp pour éviter qu'un utilisateur n'efface les fichiers temporaires d'un autre utilisateur).

Lorsqu'un fichier possède le SetUID bit (ou le SetGID bit), si un utilisateur exécute ce fichier, l'effective user (ou group) du programme auront pour valeur le propriétaire du fichier (ou son groupe). Les real user et real group seront eux toujours égaux à l'utilisateur qui a lancé le programme.

VIII.5.c Umask

Lorsqu'un fichier ou répertoire est créé, il possédera automatiquement comme propriétaire l'utilisateur qui le crée (le effective user du programme qui le crée). Son groupe sera le groupe "actif" de l'utilisateur (par défaut, le groupe principal) sauf si le répertoire possède le SetGIDbit.

Pour les droits, cela dépendra de la situation :

Fichier données	rw-rw-rw-
Fichier exécutable (programme compilé)	rwxrwxrwx
Répertoire	rwxrwxrwx

L'utilisateur a une valeur appelée *umask* qui permet de modifier ces droits par défauts : les droits indiqués dans umask seront RETIRES aux droits par défaut.

Ainsi, si l'umask vaut 022 et que l'on crée un fichier, les droits qui seront placés seront rw-r--r-- (les droits en écriture (2) seront enlevés pour le groupe et pour les autres)

Le umask n'affecte que les nouveaux fichiers et répertoires créés.

VIII.5.d Quelques cas d'utilisation

Dans les UNIX récent, chaque utilisateur possède un groupe personnel qui est son groupe principal et est membre d'une série de groupes dépendant des droits qu'il possède. Par exemple, dans la VM, l'utilisateur est membre du groupe vboxsf qui lui donne le droit d'accéder aux dossiers partagés entre la VM et la machine hôte.

Répertoire de l'utilisateur

Le propriétaire et le groupe du répertoire personnel sont égaux à l'utilisateur et son groupe personnel. On peut donc donner des droits 770 (rwxrwx--- seul l'utilisateur peut y accéder) ou 775 (rwxrwxr-x tout le monde peut y accéder en lecture).

Ressource à accès limité

Un groupe est créé pour la ressource (comme vboxsf). Les utilisateurs qui ont le droit d'utiliser cette ressource sont ajoutés au groupe en question. Les fichiers sont associés à ce groupe et les droits sont donnés au groupe mais pas aux autres (640,660, 664, 750, 770 ou 775 selon le cas).

Projet commun

Un groupe est créé pour le projet et les utilisateurs associés au projet y sont ajouté. Les répertoires liés à ce projet (et leurs sous-répertoires) ainsi que les fichiers qui s'y trouvent sont associés à ce groupe.

Les permissions sur les répertoires sont de 2770 ou 2775 (selon qu'on ne désire pas un accès public ou qu'on désire laisser un accès en lecture). Le SetGID bit permet de s'assurer que les nouveaux fichiers créés dans le projet seront également associés au groupe en question.

Les fichiers auront des droits 660 ou 664, les programmes 770 ou 775.

Afin que les nouveaux fichiers créés aient les droits corrects, le umask doit valoir 002 (à noter que de nombreuses distributions Linux utilisent une valeur de 022 par défaut, définie dans le fichier .profile)

Répertoire tunnel

Si un utilisateur désire rendre un répertoire accessible à certains utilisateurs, il peut créer un répertoire tunnel. ce répertoire aura des permissions 771 ou 711 qui permettront à tous de rentrer à l'intérieur mais empêcheront de voir son contenu.

Dans ce répertoire, il créera un nouveau répertoire dont le nom est le *mot de passe d'accès* et les droits seront 777 ou 775.

Les utilisateurs qui connaîtront le nom de ce répertoire pourront faire cd ~user/tunnel/secret pour se rendre dans le répertoire en question.

VIII.6 Les commandes pour gérer les droits

Ces commandes ne peuvent être utilisées que par le propriétaire du fichier.

VIII.6.a chown

La commande chown (CHange OWNer) permet de changer le propriétaire d'un fichier. Seul root peut mettre un autre propriétaire que lui même ce qui rend cette commande inutile pour les utilisateurs normaux

La syntaxe est la suivante :

```
chown [options] propriétaire fichier(s)
```

```
Le propriétaire peut être l'identifiant (login) du nouveau propriétaire ou son
numéro d'utilisateur.
```

-R	Sur un répertoire, change les permission sur le répertoire et tout son contenu, de façon récursive.

VIII.6.b chgrp

La commande chgrp (CHange GrouP) permet de changer le groupe d'un fichier. Un utilisateur peut placer n'importe lequel des groupes auxquels il a accès, l'utilisateur root peut placer n'importe quel groupe. Il faut être le propriétaire d'un fichier (ou root) pour pouvoir changer son groupe

La syntaxe est la suivante :

```
chgrp [options] groupe fichier(s)
```

`Le groupe peut être le nom du groupe ou son numéro de groupe`

-R	Sur un répertoire, change les permission sur le répertoire et tout son contenu, de façon récursive.

VIII.6.c id

Cette commande affiche les informations sur l'utilisateur connecté (nom et numéro d'utilisateur, groupe principal et numéro de ce groupe, autres groupes de l'utilisateur et leurs numéros)

Elle ne prend aucun paramètres

VIII.6.d chmod

La commande chmod (CHange MOde) permet de changer les droits d'accès d'un fichier. Seul le propriétaire du fichier (ou root) peut changer les droits d'accès.

La syntaxe de cette commande est la suivante

```
chmod [options] permissions fichier(s)
```

-R	Applique le changement de permission au répertoire donné et à son contenu (récursivement)

Les permissions peuvent être fournies sous deux formes

Valeur numérique

On peut fournir une valeur numérique de 3 ou 4 chiffres correspondant aux droits que l'on veut appliquer

	User	Group	Other
SetUID/SetGID/Sticky	User	Group	Other

Modifications

La seconde forme consiste en trois groupes de caractères

Quels cibles (user/group/other)

Quelle modification (ajouter, retirer, changer)

Quels droits (read/write/execute)

La cible sera une ou plusieurs lettres

u	Droits pour le propriétaire (user)
g	Droits pour le groupe
o	Droits pour les autres (other)
a	Droits pour tous (All) = ugo

La modification sera un des caractères suivants :

+	Ajoute les droits indiqués
-	Retire les droits indiqués
=	Fixe les droits selon les droits indiqués

Les droits sont une ou plusieurs lettres r, w et x correspondant aux droits que l'on veut modifier/placer.

Par exemple, u+w ajoutera le droit en écriture pour le propriétaire, g-x enlèvera le droit d'exécution pour le groupe, go-wx retirera les droits en écriture et exécution pour le groupe et les autres, ugo=rx fixera les droits en lectures et exécution pour tout le monde (user, group et autres)

VIII.6.e umask

La commande umask permet de fixer la valeur (courante) de umask.

```
umask masque
```

```
Cette nouvelle valeur sera conservée jusqu'à ce qu'une autre commande umask soit utilisée ou que le shell soit quitté.
```

VIII.7 Les redirections

Lorsqu'un programme s'exécute, il "ouvre" automatiquement 3 fichiers spéciaux :

Numéro	Nom/langage C	C++	Utilité
0	stdin	cin	Lecture par le programme de caractères entrés sur la console
1	stdout	cout	Écriture par le programme de texte qui s'affichera sur la console
2	stderr	cerr	Écriture par le programme de messages d'erreurs qui s'affichera sur la console

En interne, les systèmes UNIX représentent les fichiers par des numéros appelés "file descriptors". Ce sont ces numéros qui sont utilisés en interne par les librairies C/C++ pour les accès à stdin,stdout ou stderr (cin, cout et cerr en C++).

Les redirections consistent à lier un de ces descripteurs de fichiers spéciaux[12] à des fichiers.

12 Il est en fait possible de lier ainsi les file descriptors de 0 à 9, les numéros 3 et suivants sont cependant plus rares

VIII.7.a Redirection en lecture

Il est possible d'indiquer qu'au lieu de lire des caractères sur la console, un programme doit les lire à partir d'un autre fichier. Cela se fait comme suit :

```
commande < fichier
commande 0< fichier
```

On préférera généralement la première forme, plus simple que de préciser le numéro du file descriptor concerné.

Cela permettra de transmettre au programme lancé le contenu du fichier indiqué comme si ce contenu avait été tapé au clavier.

VIII.7.b Redirection en écriture

Il est également possible d'écrire les textes transmis par le programme dans un fichier au lieu de les afficher. La syntaxe est la suivante :

```
commande > fichier
commande 1> fichier
commande 2> fichier
```

Par défaut, si le descripteur de fichier n'est pas communiqué, ce sera le descripteur 1 (stdout) qui sera utilisé. Pour stdout, on utilisera de préférence la première forme.

Pour rediriger les messages d'erreurs (mais pas le texte normal), on utilisera la troisième forme.

Cela permet de sauver dans un fichier le résultat d'une commande (au lieu de l'afficher à l'écran) ou de sauver les messages d'erreur.

Si le fichier existe, il sera vidé avant de commencer la redirection

VIII.7.c Redirection en ajoute

Si on désire conserver l'ancien contenu d'un fichier et y AJOUTER le texte redirigé, on utilisera la syntaxe suivante :

```
commande >> fichier
commande 1>> fichier
commande 2>> fichier
```

Tout comme la redirection en écriture, le file descriptor 1 (stdout) est optionnel et on préférera la première forme.

Le fichier précisé ne sera pas vidé avant de commencer la redirection. Le texte redirigé sera ajouté en fin de fichier.

VIII.7.d Redirection chaînée

Il est possible de rediriger stderr vers stdout (qui peut lui-même avoir été redirigé vers un fichier). Cela se fait en utilisant la syntaxe suivante :

```
commande 2>&1
commande > fichier 2>&1
```

Dans ces deux cas, les messages affichés sur stderr seront transmis sur stdout. Dans le second cas, stdout a été redirigé vers un fichier et donc, on retrouvera dans ce fichier les messages imprimés par stdout ET par stderr

le &1 indique que la redirection est faite vers le file descriptor 1 (stdout).

VIII.7.e Pipe

Dans certains cas, on désire utiliser une commande sur le texte généré par une autre commande. On pourrait utiliser un fichier intermédiaire de la façon suivante :

```
commande1 > fichier_temporaire
commande2 < fichier_temporaire
rm  fichier_temporaire
```

Mais UNIX propose d'effectuer cette opération sans passer par un tel fichier temporaire, en utilisant les "pipes". Le pipe peut être vu comme un tuyau (pipe en anglais) qui relie le stdout d'un programme au stdin d'un second programme. Le pipe se représente par la barre verticale |

```
commande1 | commande2
```

De nombreuses commandes permettent d'agir sur un fichier mais, si aucun fichier n'est indiqué, utiliseront stdin à la place. Ces commande seront appelées filtres. La commande *more* (ou *less*) qui permet d'afficher un fichier "page par page" est un exemple de filtre. On peut donc l'utiliser quand la sortie d'une commande est trop longue, pour éviter qu'elle ne défile à l'écran sans qu'on ait le temps de la lire

```
ls -alsF /usr/bin | more
```

Permettra de lister les fichiers de /usr/bin en attendant que l'on pousse sur la barre d'espace à chaque page affichée.

VIII.8 Quelques filtres

La plupart de ces filtres peuvent être appelés avec un nom de fichier pour s'exécuter directement sur le fichier ou en faisant une redirection d'entrée (<) pour lire d'un fichier précis au lieu de lire à partir de stdout

VIII.8.a more/less/head/tail/cat

Ces commandes ont été décrites plus haut. Si on ne précise pas de nom de fichier, elles utiliseront stdin ce qui permet d'en faire des filtres pour afficher page par page, le début, la fin ou la totalité du fichier.

VIII.8.b grep

La commande *grep* permet de ne garder que les lignes contenant (ou ne contenant pas) un certain texte. Par exemple, pour ne garder que les lignes qui contiennent le mot "ERROR", on utilisera

```
commande | grep ERROR
```

Cette commande accepte une série d'options avant le "motif" à rechercher.

-i	Ne fait aucune différence entre majuscules et minuscules
-v	Ne retourne que les lignes qui ne CONTIENNENT PAS la chaîne cherchée
-c	Retourne le nombre de lignes qui seraient affichées au lieu des lignes en elles-même

L'expression de recherche est une "expression régulière" qui suit les règles suivantes :

. (point)	Un caractère quelconque
[abc]	Un des caractères a, b ou c
[a-z]	Un des caractères de a à z
[a-zA-Z_]	Un des caractères de a à z, de A à Z ou _
[^a-zA-Z_]	Un caractère qui n'est pas de a à z, de A à Z ou _
^	Début de ligne
$	Fin de ligne
X?	0 ou 1 caractère X (*)
X*	0 ou plus caractères X
X+	1 ou plus caractères X (*)
X{5}	Exactement 5 caractères X (*)
X{2,4}	Entre 2 et 4 caractères X (*)
abc|def	La chaîne abc ou la chaîne def (*)
a(bc|de)f	Les chaînes abcf ou adef (les parenthèses permettent de grouper des éléments ou d'en séparer) (*)

Attention, les formes marquées (*) demandent parfois l'option -E pour être reconnues. Certaines peuvent même ne simplement pas être reconnues.

On peut retirer la signification spéciale d'un symbole en le précédent d'un \ (par exemple \. pour un point)

VIII.8.c sort

La commande sort permet de trier le contenu de l'entrée standard ou les fichiers passés en argument (à noter qu'on utilise souvent sort < nontrie > trie).

Le tri s'effectue par défaut sur base des codes ASCII, ce qui veut dire que 24 se trouvera avant 3. De nombreuses options permettent de modifier la façon dont la chaîne est triée

-u	Ne garde que les lignes Uniques (retire les doublons)
-r	Ordre inversé (Reverse)
-n	Trie des valeurs numériques (terminées au premier caractère non numérique, 24 se retrouve après 3)

-f	Ignore les différences majuscules/minuscules
-R	Mélange les lignes (Random)
-t Séparateur	Indique quel caractère sert de séparateur de champs. Par exemple, -t, permet de traiter des fichier CSV. A utiliser avec -k
-k POS	Trie à partir de la position (commençant à 1) indiquée. Si -t n'est pas précisé, les champs sont séparés par des caractères blancs (espaces, tabulation,...) -t, -k2 permet ainsi de trier à partir de la 2eme colonne d'un fichier CSV

VIII.8.d cut

La commande cut permet de découper un morceau des lignes fournies en entrées. La manière dont sont faites les découpes dépend des options fournies

-c liste	Ne garde que les morceaux spécifiés, comptés en caractères depuis le début de la ligne
-f liste	Ne garde que les champs spécifiés
-d séparateur	Indique le caractère de séparation de champs utilisé (par défaut, TAB)

Les listes de caractères ou de champs sont composées comme suit :

- N-M : caractères/champs de N à M (le premier caractère est numéroté 1)
- N- : caractères/champs de N jusqu'à la fin
- -M : caractères/champs du début jusqu'à M
- N : juste le Nième caractère/champ

On peut préciser plusieurs intervalles séparés par des virgules.

Ainsi, la commande

```
cut -d: -f1,3-5,7
```

Permettra de récupérer les champs 1, 3,4,5, 7 d'un fichier où les champs sont séparés par le caractère :

VIII.8.e wc

La commande wc permet de compter le nombre de lignes, de mots et de caractères de stdin (ou du fichier passé en paramètre)

Les options suivantes peuvent être utilisées pour n'afficher qu'une de ces valeurs (n'utiliser qu'une option)

-c	Octets
-m	Caractères (attention, dans certains encodages, certains caractères peuvent utiliser plusieurs octets)
-w	Mots

-l	Lignes
-L	Taille de la ligne la plus longue

VIII.8.f tee

La commande tee permet de copier stdin sur à la fois stdout et dans un fichier.

```
commande | tee nontrie | sort >  trie
```

Dans l'exemple ci-dessus, le résultat de la commande sera copié dans le fichier nontrie mais sera également trié et sauvé dans le fichier trie.

Cette commande permet donc de sauver l'état intermédiaire d'une série de commandes chaînées par des |

VIII.8.g sed

La commande sed permet de faire des modifications du texte "au vol". Une utilisation courante est pour faire des "chercher-remplacer" :

```
commande | sed -e 's/chercher/remplacer/'
```

La chaîne "chercher" est une expression régulière (regexp, voir la commande grep) qui correspond à la chaîne à chercher et qui sera remplacée par la chaîne "remplacer".

Par exemple, pour remplacer les extensions .jpg et .jpeg en .png, on utilisera

```
sed -e 's/\.jpe?g/.png/'
```

VIII.9 Éditer un fichier

A l'origine sous UNIX, on utilisait des "TeleTYpes" (combinaison clavier et imprimante, on utilise toujours TTY pour désigner les consoles, même si elles sont dans une fenêtre d'un environnement graphique).

Pour éditer un fichier, on utilisait un éditeur "de ligne" appelé *ed*. Cet éditeur proposait des commandes qui permettaient d'afficher certaines lignes, d'insérer/effacer des lignes à une position précise, d'écrire le fichier sur le disque, ... En cas d'erreur, ed affichait simplement un ?, les affichages étant réduits au strict minimum (pour économiser papier et ruban d'imprimante).

Avec les terminaux textes, composés d'un écran et d'un clavier, un nouvel éditeur est apparu : vi (Visuel). Cet éditeur utilisait toujours des commandes pour déplacer le curseur, passer en mode d'édition, ... mais était nettement plus convivial. Il est également l'éditeur de fichier le plus présent sur les machines UNIX (ou un clone tel que vim).

D'autres éditeurs de fichiers sont apparus, plus conviviaux, tels que nano, joe, emacs, ...

Emacs est un éditeur à tout faire. Outre le fait de permettre d'éditer des fichiers, il peut servir à lire ses E-Mails, à afficher un calendrier, servir de calculatrice, de lecteur de news, d'interpréteur LISP, ... Par contre, son utilisation requiert l'usage de nombreuses séquences de touches telles que Controle+x Controle+s, Meta+x, ... qui sont assez fastidieuses à retenir

VIII.9.a joe

Pour éditer un fichier avec joe, on appelle la commande `joe nomfichier`. Si le fichier n'existe pas, un nouveau fichier sera créé lorsque l'on sauvera le contenu de l'éditeur.

La première commande à retenir est Control+K H qui affiche la page d'aide. Escape . (point) permet de passer à la page d'aide suivante et Escape , (virgule) de revenir à la page précédente. Pour fermer l'aide, il suffit de refaire Control+K H (Par la suite, Controle+X sera abrégé en ^X)

Pour éditer le contenu, il suffit d'utiliser les touches de curseur, de taper le texte que l'on veut insérer, d'utiliser Suppr et Backspace pour effacer les caractères de trop, ...

Pour quitter joe, on utilisera ^C (Control+C) pour quitter sans sauver ou ^KX (eXit) pour sauver et quitter. Si joe a été lancé sans nom de fichier, ^KX demandera le nom de fichier sous lequel on veut sauver le contenu de l'éditeur.

Si on désire sauver le fichier en cours d'édition, il suffit d'utiliser ^KD. Joe demandera le nom de fichier en proposant le dernier nom utilisé (il suffit de taper Enter si on ne désire pas changer le nom). Pour insérer le contenu d'un fichier à la position du curseur, il faudra utilser ^KR, joe demandera le nom du fichier à insérer.

Pour faire les couper/coller, il faut d'abord marquer le **B**loc**K** que l'on veut manipuler. On se place au début du bloc et on fait ^K**B**, à la fin et on fait ^K**K**. Il reste alors à se placer à la destination et faire ^KC pour Copier, ^KM pour déplacer (Move) ou ^KY pour effacer (Yank). ^KW permet de sauver le bloc dans un fichier (Write)

Joe possède plusieurs options de formatage qu'il peut être nécessaire d'activer ou de désactiver. Pour y accéder, il faut faire ^T. Les options apparaissent dans le bas de l'écran. Il suffit alors de taper sur la lettre correspondant à l'option que l'on désire changer. Les options sélectionnées apparaissent en haut de l'écran, avant le nom du fichier (I pour auto-indent, W pour Word Wrap, ...) Le plus souvent, on voudra désactiver le mode Word Wrap s'il est activé (retour à la ligne automatique) ou le mode Auto-Indent (ajout automatique d'espaces en début de ligne si la ligne précédente commençait par des espaces, particulièrement gênant en cas de copier/coller d'une fenêtre à une autre).

A noter que les options par défauts dépendent de l'extension du fichier créé. C'est ainsi que si on édite un fichier C, l'option W sera désactivée par défaut. Si on se rend compte qu'une ligne qui devient trop longue est scindée en deux, cela signifie que le Word Wrap est activé.

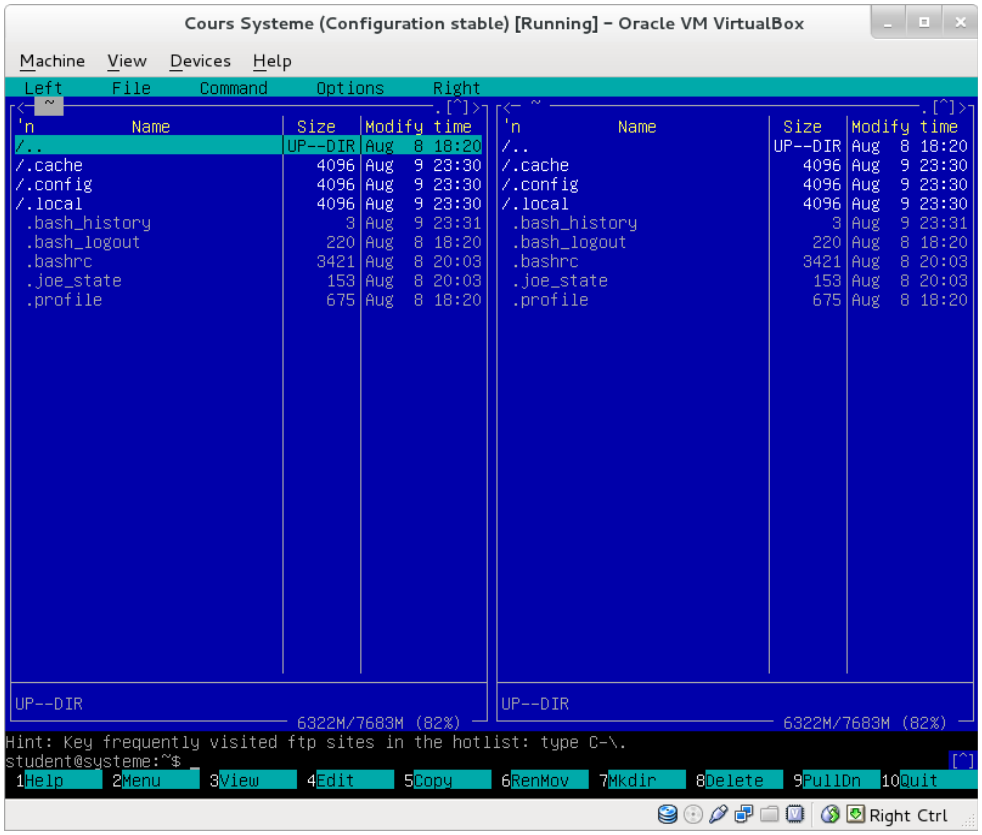
^KH	Page d'aide (ESC . et ESC , pour naviguer entre les pages)
^C	Quitter sans sauver
^KX	Sauver et quitter
^KD	Sauver le fichier sous ...
^KR	Insérer un fichier
^KB ... ^KK	Marque le début/la fin d'un bloc
^KC	Copie le bloc sélectionné à la position du curseur
^KM	Déplace le bloc sélectionné à la position du curseur
^KY	Efface le bloc sélectionné

^Y	Efface la ligne sur laquelle on se trouve
^KW	Sauve le bloc sélectionné dans un fichier
^KB^KK	Désélectionne le block (sélectionne un bloc vide)
^T	Affiche les options de formatage

VIII.10 Midnight Commander

Midnight Commander est un outil semblable à Norton Commander sous MS DOS. Cet outil facilite la navigation dans les répertoire et la manipulation des fichiers et répertoires en mode texte. Il est d'un usage nettement plus pratique que la plupart des interfaces graphiques de gestion de fichiers, d'autant plus qu'il laisse l'accès direct à la ligne de commande.

On le lance simplement par la commande `mc`. Il se présente alors sous la forme reprise à la capture VIII.1.



Capture VIII.1: Midnight Commander

On dispose de deux panneaux. En haut de chaque panneau, on a le répertoire courant (ici ~, le répertoire de l'utilisateur). Le panneau actif pourra être identifié par le nom du répertoire courant et une des lignes dans le panneau apparaissant en vidéo inverse.

Pour changer le panneau actif, il suffit d'utiliser la touche TAB.

On peut déplacer le curseur du fichier (ou répertoire) dans le panneau actif par les flèches haut et bas.

Pour rentrer dans un sous-répertoire, il suffit d'amener le curseur sur son nom et d'utiliser la touche Enter. Le répertoire .. permet de remonter d'un niveau.

Pour copier/déplacer/effacer un fichier ou répertoire, il suffit d'amener le curseur sur son nom et d'utiliser F8 pour effacer ou F5 pour copier et F6 pour déplacer vers le répertoire sélectionné par l'autre panneau.

Pour renommer un fichier ou répertoire, on utilise F6 comme pour le déplacer mais on remplace la destination par le nouveau nom.

F3 permet de visionner le contenu d'un fichier et F4 permet de l'éditer. Par défaut, l'éditeur système est utilisé mais une option permet d'utiliser l'éditeur de Midnight Commander[13].

La touche F7 permet de créer un nouveau répertoire. Il suffit d'entrer le nom du nouveau répertoire et de valider.

Si on désire copier, déplacer ou effacer plusieurs fichiers, il suffit de les sélectionner à l'aide de la touche Insert.

F10 permet de quitter Midnight Commander et Control+O permet de masquer temporairement les panneaux (pour pouvoir voir l'écran situé derrière) et de les réafficher.

On peut taper directement des commandes. Tant que la ligne de commande n'est pas vide, la touche Enter cesse de permettre la navigation et permet de valider la commande tapée.

Si on presse Enter en ayant un fichier exécutable sélectionné, cela a pour effet de le lancer sans paramètres.

Midnight Commander permet également de rentrer dans une archive ZIP, TGZ, ... comme s'il s'agissait d'un simple répertoire.

La touche F2 appelle un menu qui dépend de l'item sélectionné. Une des fonctions les plus pratiques est quand une archive (tar.gz, tar.bz2, tgz,...) est sélectionnée et permet de décompresser directement l'archive.

Il y a bien d'autres fonctionnalités disponibles, client FTP, recherche de fichier, changement des permissions, ... mais cela dépasse de cadre de cette introduction.

VIII.11 Gestion des processus

Les systèmes UNIX sont des systèmes multitâches. Chaque programme tourne dans un environnement protégé appelé *process*. Un programme peut tourner sur plusieurs process qui communiquent entre eux à l'aide de mécanismes tels que les *IPC*. Chaque processeur de l'ordinateur va exécuter un process différent, diviser un programme en plusieurs process lui permettent donc d'exploiter plusieurs processeurs.

Chaque process possède son propre espace d'adressage (liaison entre les adresses utilisables par le programme et la mémoire physique de l'ordinateur). Cela donne à chaque process l'impression d'être seul sur l'ordinateur et permet de protéger les process entre eux (un process ne peut pas modifier la mémoire d'un autre process).

13 Cette option a été activée sur la VM qui vous est fournie.

Lorsque la machine démarre, elle lance une série de process en arrière plan que l'on appelle *daemons*.

Lorsque l'on tape une commande dans un shell, le shell va créer un nouveau process et exécutera la commande dans ce process. Il peut lancer ce process de deux façons : en avant plan (le programme garde le contrôle de la console) ou en arrière plan. Dans ce dernier cas, on récupère le contrôle de la console pour entrer d'autres commandes et le programme continue à tourner en arrière plan. Il peut toujours imprimer des messages sur stdout (cout) mais ne peut plus lire stdin (cin)

VIII.11.a Lancement d'un programme en arrière plan

Pour lancer une commande en arrière plan, on fait suivre la ligne de commande par le signe **&**.

```
commande paramètre &
```

Si on désire effectuer des redirections, on les place avant le signe &

```
commande > fichier &
```

Le shell répondra par un message de la forme

```
[1] 2857
```

Le nombre entre [] est le numéro de *job*. Le nombre qui suit est le numéro de process aussi appelé *PID*.

Quand il se passera quelque chose au niveau du process (par exemple, quand il s'arrêtera), le shell rappellera le numéro de job concerné.

Pour désigner un job, on utilise le signe % suivi nu numéro de job

```
kill %1
```

VIII.11.b jobs

Pour voir la liste des process lancés par un shell (ses jobs), on utilisera la commande jobs.

Cette commande affichera le numéro de job, l'état du job et la commande utilisée pour le lancer.

Elle accepte l'option suivante :

-l	Affiche le numéro de process à côté du numéro de job dans le listing

VIII.11.c fg/bg

La commande fg suivie d'un identificateur de job permet de faire passer un job au premier plan et s'il était en mode Stopped (arrêté), de le relancer. Le shell ne récupère pas la main et ce que l'on tapera sera directement envoyé au programme qui tourne dans le job concerné.

Le mode Stopped ne doit pas être confondu avec l'arrêt d'un process parce qu'il se termine normalement (fin du main()), parce qu'il a rencontré un appel exit(); ou parce qu'on lui a envoyé un signal pour l'arrêter (le terminer).

La commande bg suivie d'un identificateur de job permet de relancer un job à l'arrière plan et de le relancer s'il était en mode Stopped. Le shell récupère la main et on peut taper d'autres commandes.

La combinaison de touches **CTRL+Z** (^Z) permet de faire passer en mode Stopped le programme en arrière plan (et donc de récupérer la main sur le shell, par exemple pour relancer le process en arrière plan).

Quand un programme est au premier plan, **CTRL+C** permet de l'arrêter. Le programme est alors terminé et disparaît des jobs du shell.

```
      ./programme                   Le programme est en premier plan
      CTRL+Z                        On récupère le shell
      [2]+ Stopped ./programme
      bg %2                         On relance le programme en arrière plan
      jobs                          On demande d'afficher les jobs
      [2] Running ./programme
      fg %2                         On fait repasser le programme au premier
plan
      CTRL+C                        Arrêt complet du programme.
```

VIII.11.d ps

La commande ps permet d'afficher la liste des process de l'utilisateur. Différentes options permettront d'afficher les process de tous les utilisateurs ou ceux qui sont détachés de terminaux.

Il est souvent conseillé de soit utiliser | more, soit utiliser | grep xxx (pour trouver le process xxx) vu que la liste de tous les process est souvent longue.

Une forme souvent utilisée (parce qu'affichant une liste complète) est

```
ps aux | more
```

Cette forme affiche tous les process, de tous les utilisateurs, quel que soit leur statuts. Les options reprises ici n'ont pas de – contrairement aux autres commande, il s'agit d'un archaïsme des anciens UNIX de type BSD que l'on retrouve aussi dans quelques autres commandes.

La première colonne donne le nom du propriétaire du process, la seconde donne le numéro de process, la troisième, la charge machine, les trois suivantes concernent l'utilisation de la mémoire, la 7eme indique le terminal concerné, la 8eme le statut (Running, Sleeping, Zombie), suivent l'heure de lancement du process et le temps processeur qu'il a utilisé et, enfin, la ligne de commande utilisée pour lancer le process. Les colonnes les plus intéressantes sont la seconde (PID, numéro de process) et la dernière (qui reprend la commande lancée).

Si la commande tapée est trop longue que pour apparaître en fin de ligne, on peut permettre l'utilisation de lignes supplémentaires en ajoutant des "w" en fin de la ligne d'options (un par ligne supplémentaire autorisée à l'origine, maintenant, dès qu'il est présent deux fois, illimité). Ainsi, pour permettre l'affichage des commandes en utilisant autant de lignes supplémentaires que nécessaire, on utilisera

```
ps auxww | more
```

Les états de process sont les suivants :

S	Sleeping	Le process attend un événement qui lui permettra de nouveau de revendiquer l'usage du CPU (données lues d'un disque dur, sémaphore, données réseau, ...) ou est simplement en attente de CPU
R	Running	Le process est en train de s'exécuter. Le nombre de process Running est au maximum le nombre de processeurs (ou de cœurs) présents sur la machine
Z	Zombie	Le process s'est terminé mais son process père n'a pas encore lu sa valeur de fin (retournée par exemple par exit()). Le process est conservé tant que le process père existe et n'a pas lu la valeur en question
T	sTopped	Le process est arrêté (CTRL+Z par exemple)

La commande utilisée seule affichera uniquement les process du même utilisateur et lancés à partir du même terminal

La forme non-BSD qui permet d'afficher tous les process est

```
ps -eF
```

A noter que cette forme affiche quelques informations en moins que la forme BSD.

VIII.11.e kill

L'instruction kill permet d'envoyer un signal à un process. On peut soit utiliser le PID (obtenu par exemple à l'aide de l'instruction ps), soit utiliser un identifiant de job (%1, %2, ...)

Les process peuvent intercepter les signaux (sauf SIGKILL), par exemple pour effectuer une sauvegarde de secours avant de se terminer mais aussi pour toute autre tâche.

Les principaux signaux sont les suivants :

1	SIGHUP	A l'origine, les terminaux pouvaient être connectés au travers d'une ligne téléphonique. Lorsque la ligne était raccrochée, un signal HangUP était envoyé pour indiquer aux programmes qu'ils devaient s'arrêter. Un daemon n'étant pas accroché à un terminal, le signal SIGHUP n'a aucun sens pour eux et il est généralement réutilisé pour indiquer que l'on désire recharger les fichiers de configuration.
2	SIGINT	Indique qu'un programme est interrompu (**CTRL+C**) et doit se terminer
8	SIGFPE	Indique une erreur de calcul virgule flottante (division par 0, tangente de 90°, ...)
9	SIGKILL	Ce signal fait se terminer le programme **immédiatement**. Il n'est pas possible de l'intercepter. C'est l'option "de dernier recours" lorsqu'un programme refuse de s'arrêter.
11	SIGSEGV	Erreur d'accès à la mémoire. Cela arrive par exemple lorsqu'un pointeur est utilisé de façon incorrecte (non initialisé, pointeur NULL, ...)
14 (26)	SIGALRM	Il est possible de programmer une "alarme" dans un process. Quand elle se déclenche, le process reçoit ce signal. Cela permet par exemple d'amener le process à effectuer une tâche à intervalle régulier.
15	SIGTERM	Demande au programme de se terminer proprement. C'est le signal par défaut de l'instruction kill

18	SIGSTP	On a demander au process de se stopper (**CTRL+Z**). Le mode du process passe en Stopped mais le process ne se termine pas
30 31	SIGUSR1 SIGUSR2	Ces signaux sont disponibles pour le programmeur qui peut leur donner n'importe quelle signification.

Pour envoyer un signal autre que SIGTERM, on peut soit utiliser - suivi du numéro de signal (-9 pour SIGKILL), soit utiliser - suivi du nom du signal sans le SIG (-KILL pour SIGKILL)

```
kill -1 1428
kill -HUP 1428
```

Comme certains UNIX peuvent utiliser des numéros différents pour certains signaux, la seconde forme est souvent préférable. A noter cependant que SIGHUP, SIGKILL et SIGTERM auront toujours les valeurs 1, 9 et 15. Dans le cas de SIGTERM, on ne doit d'ailleurs pas préciser le signal.

Le signal SIGKILL ne doit être envoyé qu'en dernier recours. Toujours essayer de commencer par un SIGTERM (et laisser le temps au process de se terminer).

VIII.11.f killall

La commande killall permet d'envoyer un signal à tous les process dont le programme est celui fourni en paramètre. Par exemple, pour envoyer un signal SIGHUP à tous les process *bind* (le programme qui gère la traduction des noms de machines en adresses IP), on utilisera

```
killall -HUP bind
```

De nouveau, si on désire envoyer le signal TERM, on peut ignorer l'option qui précise le signal. Attention que si cette commande est utilisée sous root, elle affecte tous les process concernés de **tous** les utilisateurs.

VIII.11.g nohup

Si on précède une commande de nohup, cela permet de s'assurer que le shell n'enverra pas le signal SIGHUP¨au programme en question quand on terminera la session. A noter que pour que cela ait une utilité, il faut également que le programme soit lancé en arrière plan (&) et qu'il n'ait pas besoin de lire sur la console (stdin).

VIII.11.h uptime

La commande uptime permet d'afficher le temps depuis lequel la machine est bootée mais également la charge de la machine.

La charge est affichée sous forme de trois valeurs qui sont les moyenne sur 1 minute, 5 minutes et 15 minutes. Cette charge est le nombre moyen de process qui sont "Running". Si la machine est utilisée à 100 %, la charge est égale au nombre de processeurs (ou de cœurs). Si la machine est inactive, cette charge est de 0.

VIII.11.i time

La commande time permet d'évaluer le temps d'exécution nécessaire à un programme. D'une fois à l'autre, ce temps peu changer selon la charge machine, les temps d'accès aux disques, ...

La commande retourne trois valeurs : le temps "réel" (heure de fin-heure de début), le temps passé en mode utilisateur (dans le code du programme) et le temps passé dans le noyau (pour les appels système effectués par le programme).

Cette commande s'exécute comme suit :

```
time [options] commande
```

Les options possibles sont les suivantes :

-o fichier	Écrit les résultats dans un fichier au lieu de stdout
-p	Permet l'affichage standard POSIX. Sous linux, l'affichage par défaut est plus complet que l'affichage standard et l'ordre des informations est différent

VIII.12 Autres commandes

VIII.12.a date

La commande date permet d'afficher l'heure (sous root, elle permet également de la changer).

VIII.12.b echo

La commande echo affiche la chaîne qui suit à l'écran (stdout). Par défaut, cette commande envoie un caractère de retour de ligne après la chaîne.

```
echo Hello World
echo "Hello World"
```

On peut utiliser l'option suivante :

-n	N'envoie pas de caractère retour de chariot

VIII.12.c file

La commande file affiche le type de données contenu dans le fichier passé en paramètre. Il utilise les Magic Names pour déterminer ce type de fichier.

IX Bases de script Shell

Il est possible de créer des petits programmes en utilisant les commandes UNIX et quelques commandes propres au shell utilisé. Ces programmes s'appellent *shell scripts* et ont souvent des extensions .sh (bourne shell, bash,...) ou .csh (C-Shell).

Pour créer un tel programme, il faut

- mettre **#!/bin/bash** en première ligne (magic name indiquant qu'il s'agit d'un script bash)

- faire suivre par les commandes à exécuter (dans l'ordre)

- donner le droit d'exécution au fichier (*chmod ugo+x fichier.sh*)

IX.1 Variables et paramètres

Dans un script shell, il est possible d'utiliser des variables pour sauver des chaînes de caractères. Il suffit d'utiliser la syntaxe suivante:

```
A="un texte"
```

Lorsque l'on désirera utiliser le contenu de cette variable, il suffira d'utiliser une des formes *$var* ou *${var}*.

Le Shell prévoit une série de variables spéciales :

$1 $2 $3 ... $9	Paramètres passés à la commande (un "mot" = un paramètre)
${10} ${11} ...	Paramètres passés à la commande à partir du 10ème
$*	Liste des paramètres. "$*" => "$1 $2 $3 ..."
$@	Liste des paramètres. "$@" => "$1" "$2" "$3" ...
$#	Nombre de paramètres
$?	Statut de retour de la dernière commande exécutée (0= OK, autre = erreur)
$PWD	Répertoire courant
$HOME	Homedir de l'utilisateur en cours
$PATH	Chemin de recherche des commandes : liste de répertoires examinés lorsque l'on tape une commande, séparés par des ":"
$HISTSIZE	Nombre de commandes mémorisées dans l'historique
$TERM	Mode du terminal courant. Ce mode doit correspondre à la configuration du terminal (par exemple Putty quand on se connecte à distance)
$PS1 $PS2	Chaînes utilisées pour créer le prompt
$TMOUT	Temps d'inactivité avant une déconnexion automatique du shell
$LOGNAME $USER	Nom de l'utilisateur connecté
$EDITOR	Editeur utilisé par défaut
$SHELL	Shell par défaut de l'utilisateur

On peut rendre une variable globale à l'aide de la commande *export variable*. On peut combiner la commande export et l'assignation d'une valeur : *export var=val*.

On peut lister les variables en utilisant la commande *set*. Cette commande liste également les fonctions "exportées" d'un shell à l'autre (les fonctions sont exportées sous une forme semblable aux variables). La commande *export* utilisée seule listera uniquement les variables exportées.

IX.2 Expressions

Une chaîne de caractères peut être exprimée sous plusieurs formes

- Entre apostrophes (') : la chaîne sera prise telle qu'elle sans aucune modifications et sera considérée comme un mot unique

- Entre guillements (") : lorsqu'une variable apparaît, elle est remplacée par sa valeur. Les caractères $, ` et \ doivent être précédés d'un \ et certaines séquences composées d'un \ et d'une lettre représenteront des caractères spéciaux (retour de chariot, tabulation, ...). La chaîne sera considérée comme un mot unique.

- sans aucun délimiteur : les variables seront remplacées par leurs valeurs et chaque mot séparé par un espace correspondra à un mot séparé.

- Entre *backquotes* (`) : les variables dans la chaîne seront remplacées par leur valeur. La chaîne sera ensuite exécutée comme une commande et la sortie (stdout) de cette commande sera retournée comme résultat. Par exemple A=`ls` sauvera dans la variable A le texte retourné par la commande ls.

On peut découper une chaîne de caractères contenue dans une variable en utilisant

```
${variable:offset}
${variable:offset:longueur}
```

qui permettront de récupérer à partir du caractère en position offset et un maximum de longueur caractères (jusqu'à la fin si longueur est absent).

IX.3 Expressions numériques

Par défaut, le shell travaille sur des chaînes de caractères. Si on désire effectuer des opérations sur des nombres, il est nécessaire de prendre quelques précautions :

- une variable peut être déclarée numérique en la déclarant comme telle

```
declare -i var
```

Lorsque la variable est numérique, elle permettra de déclencher automatiquement des traitements numériques.

- On peut forcer une évaluation numérique d'une expression en utilisant *$((expression))*

```
B=$(($A+1))
```

IX.4 Tests

Le shell permet d'effectuer différents tests. Ces tests retourneront un statut de 0 (Ok) s'il est vrai et 1 (erreur) s'il est faux. Les tests sont prévus pour être utilisés dans des expressions conditionnelles.

```
[ test ]
```

Les tests suivants sont possibles :

-e fichier	OK si le fichier existe
-f fichier	OK si le fichier existe et est un fichier de données
-d fichier	OK si le fichier existe et est un répertoire
-r fichier	OK si le fichier existe et est accessible en lecture
-w fichier	OK si le fichier existe et est accessible en écriture
-x fichier	OK si le fichier existe et est accessible en exécution
-s fichier	OK si le fichier existe et est de taille non nulle
-n chaîne	OK si la chaîne est non vide
-z chaîne	OK si la chaîne est vide
chaîne = chaîne	OK si les chaînes sont égales
chaîne != chaîne	OK si les chaînes sont différentes
chaîne < chaîne	OK si la seconde chaîne est après la première en ordre alphabétique
chaîne > chaîne	OK si la seconde chaîne est avant la première en ordre alphabétique
val1 -eq val2	OK si les deux valeurs numériques sont égales
val1 -ne val2	OK si les deux valeurs numériques sont différentes
val1 -lt val2	OK si val1 est inférieur à val2
val1 -gt val2	OK si val1 est supérieur à val2
val1 -le -val2	OK si val1 est inférieur ou égal à val2
val1 -ge val2	OK si val1 est supérieur ou égal à val2
! test	OK si test est faux (error)
test1 -a test2	OK si test1 et test2 sont tous les deux vrais (OK)
test1 -o test2	OK si test1 ou test2 est vrai (OK)

IX.5 Expressions conditionnelles

En Shell, les différentes expressions conditionnelles se basent sur le statut de retour de commandes. Une valeur de 0 indique que la commande s'est terminée avec succès et correspond à une valeur vraie. Une valeur différente de 0 indique que la commande ne s'est pas terminée correctement et représente une valeur fausse.

IX.5.a Expressions court-circuit

Deux commandes séparées par un ";" seront exécutées l'une après l'autre et le statut de la dernière d'entre elles retourné.

Si les commandes sont séparées par "&&", la seconde ne sera exécutée que si le résultat de la première est vrai.

Si les commandes sont séparées par "||", la seconde ne sera exécutée que si le résultat de la première est faux.

Les deux dernières formes sont appelées "court-circuit" car la seconde ne sera exécutée que si nécessaire (pour effectuer un ET ou un OU logique entre les résultats). Si le résultat de la première commande suffit à déterminer le résultat, la seconde n'est pas exécutée.

IX.5.b if...fi

La syntaxe d'une condition "simple" est la suivante :

```
if commande ; then
   commandes à exécuter
fi
```

Si la commande retourne une valeur vraie (statut 0), les commandes seront exécutées. Souvent, on utilisera un des tests décrits ci-dessus.

Il est possible de prévoir une alternative :

```
if test ; then
   commandes à exécuter vrai
else
   commandes à exécuter faux
fi
```

Il est possible de chaîner les conditions à l'aide de l'instruction elif

```
if test ; then
   commandes à exécuter
elif test2 ; then
   commandes à exécuter
...
else
   commandes à exécuter
fi
```

On peut utiliser autant de elif que l'on désire.

IX.5.c case...esac

Il est possible de tester une chaîne avec des valeurs multiples et d'exécuter des commandes différentes selon la valeur de la chaîne testée.

```
case chaîne in
   pattern)
      commandes à exécuter
      ;;
   pattern)
      commandes à exécuter
      ;;
esac
```

Le pattern peut contenir un * pour remplacer un nombre quelconque de caractères. Souvent, on utilisera d'ailleurs un pattern ne contenant que * en dernier lieu pour le cas par défaut.

La chaîne sera composée de un seul mot.

IX.5.d while...done

Il est possible de répéter des commandes tant qu'une condition est vraie.

```
while commande1 ;
do
   liste de commandes
done
```

La liste de commandes sera exécutée tant que la commande1 retournera une valeur vraie. La commande while dans son entièreté (jusqu'au done) sera considérée comme une commande unique et on peut préciser des redirections après le mot done qui seront appliquées à l'ensemble du while.

IX.5.e for...done

Il est possible d'exécuter une ou des commandes sur une série de mots ou de fichiers. Pour cela, on utilisera la forme suivante :

```
for var in liste de mots ;
do
   liste de commandes
done
```

La liste de mot peut soit être une série de chaînes, soit un fileglob. Dans le second cas, le fileglob sera remplacé par la liste des fichiers correspondant. Ainsi, on peut utiliser une commande telle que

```
for a in *.c ; do echo $a ; done
```

La variable *var* contiendra le mot en cours de traitement.

IX.5.f break/continue

Il est possible de sortir d'une boucle while ou for à l'aide de la commande *break*. Si cette dernière est rencontrée, l'exécution du script continuera à la commande qui suivra le done de fin de boucle.

Il est également possible de passer directement à l'itération suivante en utilisant la commande *continue*. On revient au test pour une boucle while ou on passe au mot suivant pour un for.

IX.6 Autres commandes

IX.6.a exit

La commande exit permet de terminer le script immédiatement. Si elle est suivie d'un nombre, ce dernier sera le statut de retour du script.

IX.6.b shift

La commande shift permet de décaler les paramètres de un rang. $2 sera copié dans $1, $3 dans $2 etc...

IX.6.c set

La commande set suivie d'une série de mots sauvera les mots en question dans $1, $2, $3, ...

IX.6.d read

La commande read permet de lire des mots de stdin. La commande sera suivie d'un ou plusieurs noms de variables. Le premier mot sera sauvé dans la première variable, le second dans la seconde variable, ... et la dernière variable contiendra le reste de la ligne. Si une seule variable est fournie, elle contiendra donc la totalité de la ligne lue.

```
read var1 var2 remain
```

Si la commande read a pu lire une ligne, elle retourne un statut OK sinon elle retournera une valeur fausse (ce qui permet de l'utiliser comme condition d'un while).

```
while read line ;
do
   commandes
done < fichier
```

permet de lire le contenu d'un fichier ligne par ligne.

IX.7 Constructions spéciales

IX.7.a Sous-shell

Si une ou des commande est placée entre parenthèses, elle sera exécutée dans un shell à part qui se terminera une fois la commande terminée. La valeur de retour de ce sous-shell sera celle de la commande qu'il aura exécuté.

IX.7.b heredoc

Si on désire afficher plusieurs lignes, plutôt que répéter le mot clé echo, on peut utiliser le "document here".

```
cat << EOF
première ligne
deuxième ligne
...
EOF
```

Le mot EOF utilisé ici peut être n'importe quel mot. Il doit impérativement apparaître seul et en début de ligne pour indiquer la fin de la commande.

Le texte présent dans la ligne sera utilisé comme stdin de la commande exécutée. Si on désire utiliser une redirection de stdout, on la placera avant le << EOF.

IX.7.c Fonctions

Pour définir une fonction, on utilisera la syntaxe suivante :

```
mafonc () {
      commandes de la fonction
```

```
}

mafonc par1 par2
```

Lorsque l'on appelle la fonction, les paramètres seront copiés dans $1, $2, ... La fonction doit être définie AVANT d'être utilisée.

Dans une fonction, on peut utiliser la commande *return* suivie d'une valeur pour quitter la fonction. La valeur sera utilisée comme statut de la commande qui a appelé la fonction (0 veut dire OK/Vrai).