


Systèmes d'exploitation des ordinateurs

Mme Mona LAROUSI & Mme Leila BACCOUCHE

Université Virtuelle de Tunis

Introduction aux systèmes d'exploitation

Objectifs

1. Rappels sur le matériel
 2. Notions de systèmes d'exploitation
 3. historique des systèmes d'exploitation
 4. Les principaux systèmes d'exploitation
 5. Le logiciel dans un ordinateur
 6. TD1
-
-
- 

1. Rappels sur le matériel

1.1. [Architecture simplifiée d'un ordinateur](#)

1.2. [la carte mère](#)

1.3. [L'unité centrale](#)

1.1 Architecture simplifiée d'un ordinateur

Commençons par préciser ce que nous entendons par le terme "ordinateur", car les ordinateurs ne sont pas tous pareils. Un ordinateur dans le jargon des utilisateurs est associé au PC (personnel Computer) qui désigne l'ordinateur personnel. Toutefois, un ordinateur peut être une machine beaucoup plus puissante qu'un simple PC. De nos jours un ordinateur est constitué d'une partie matérielle et d'une partie logicielle. Même s'ils n'ont pas tous le même aspect, tous les ordinateurs comportent les mêmes éléments de base à savoir :

- Une unité pour effectuer les traitements, également appelée unité centrale ou processeur,
- Une unité pour contenir les programmes à exécuter qui est le lieu de travail dans un ordinateur communément appelée mémoire centrale,
- Des périphériques de stockage permanent pour y enregistrer les travaux effectués en mémoire centrale tel que le disque dur,
- Des dispositifs pour entrer et récupérer des données appelés périphériques d'entrée-sortie : un écran, une souris, un clavier, un lecteur de disquettes et un lecteur de CD-ROM ou DVD-ROM.

On trouve également une horloge et plusieurs bus afin de permettre à ces unités de communiquer. Le processeur est un composant qui a besoin d'une horloge pour contrôler la cadence des opérations qu'il exécute.

L'unité centrale et la mémoire centrale sont internes à l'ordinateur et sont disposées sur une carte appelée carte mère qui est en fait un circuit imprimé à base de silicium. Les autres

composants sont externes, quoique le disque dur soit généralement placé à l'intérieur mais il peut également fonctionner en étant externe.

La figure suivante explicite la structure d'un ordinateur d'un point de vue système :

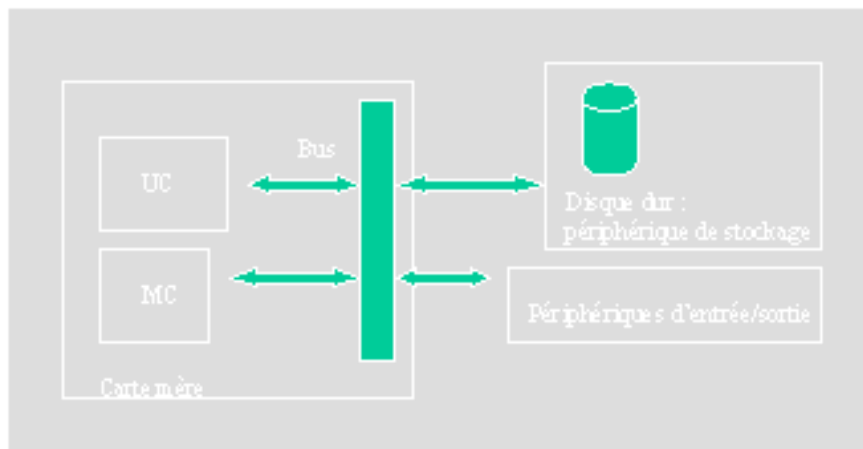


Fig. 1. Structure simplifiée d'un ordinateur



1.2 La carte mère

Du point de vue physique, [la carte mère](#) est une plaque de résine contenant à l'intérieur et sur les deux faces une fine couche de cuivre sur laquelle est imprimé le circuit imprimé. La couche de cuivre contient les fils électriques qui relient les composants. La carte mère supporte les principaux éléments d'un PC, sans elle un ordinateur ne peut pas fonctionner. On y trouve les éléments suivants :

- [Le microprocesseur](#)
- La mémoire vive RAM (Random Access Memory). Elle représente le lieu de travail dans un ordinateur à savoir qu'un programme stocké sur le disque dur est chargé en mémoire centrale où ses instructions seront accédées une à une pour être exécutées par le processeur. La RAM est une mémoire volatile c'est-à-dire que son contenu serait perdu en cas de coupure d'électricité. Elle est constituée d'un ensemble de puces (circuits intégrés) assemblées en [barrettes](#) de 32, 64 ou 128 MO dites barrettes SIMM ou DIMM.
- La mémoire morte ROM (Read Only memory). Elle contient les programmes du BIOS qui gèrent le chargement du système et les entrées-sorties. On distingue plusieurs

puces ROM tel que la PROM (Programmable ROM) et EPROM (Erasable Programmable ROM)

---> La hierarchie des mémoires.

- L'horloge qui permet de cadencer le fonctionnement du processeur, du bus. Sa fréquence caractérise la carte mère. Elle est généralement très inférieure à celle du processeur (de l'ordre de quelques centaines de MHz).
- Un ensemble de bus : un bus est un ensemble de fils de cuivre incrustés dans la carte mère qui permettent de véhiculer l'information. Le bus se caractérise par le nombre de fils qui le composent. Si le nombre de fils est de 64, on parle alors de bus 64 bits. Il est également caractérisé par sa fréquence de fonctionnement.
- Le "chipset" ou "jeu de composants" **soudé** sur la carte mère. Le chipset régit tous les échanges au sein du PC en aiguillant les données sur les différents bus de la carte mère.

Les périphériques sont des dispositifs qui sont assez lents par rapport à l'unité centrale, cette dernière ne peut donc pas rester à l'écoute pour savoir si le périphérique est prêt à lui envoyer une donnée ou à la recevoir. Un composant appelé contrôleur est associé à chaque périphérique et gère le dialogue avec l'unité centrale. La mémoire centrale également a une fréquence faible par rapport à l'unité centrale, on lui associe donc un contrôleur. On distingue les contrôleurs suivants présents sur la carte mère :

- Les différents contrôleurs d'entrée/sortie (disque, mémoire, clavier, moniteur)
- Des slots : emplacements pour connecter des cartes d'extension. On peut ainsi connecter une carte réseau avec ou sans fil, une carte modem, une carte son, carte TV, une carte graphique.
- des connecteurs d'alimentation, USB, de clavier, de disque etc.

Exemple de connecteur : connecteurs d'extension .

La figure n° 2 qui suit présente une unité centrale de PC découverte. On peut voir au fond la carte mère et les cartes d'extension. La figure n° 3 présente la carte mère d'un ordinateur de type PC.



Fig. 2. Vue des différentes cartes d'un PC

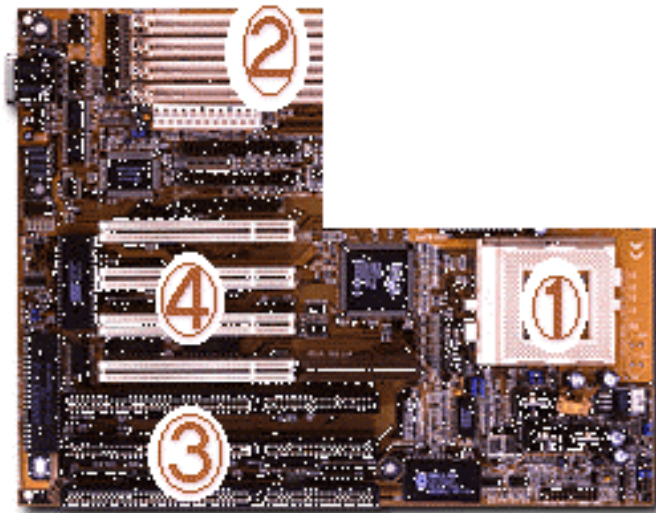


Fig. 3. Exemple de carte mère

Exercice [1](#) sur les composants de la carte mère.

Exercice [2](#) sur les composants de la carte mère.

Exercice [3](#) sur les composants de la carte mère.



1.3 L'unité centrale

On utilise souvent le même terme pour désigner l'unité centrale et le boîtier de l'unité centrale. Il s'agit en réalité d'un abus de langage et l'unité centrale est un circuit intégré qui réalise les

traitements et les décisions. Elle est également appelée microprocesseur. Elle se compose d'une unité de commande et de contrôle UCC, d'une unité arithmétique et logique UAL, de registres, d'une horloge et d'un bus interne qui relie ces unités.

- Les registres sont des zones mémoires internes au processeur destinées à accueillir les données, les instructions et les résultats.
- L'UCC recherche les instructions, les décode et en supervise leur exécution par l'UAL.
- L'UAL réalise les traitements qu'ils soient arithmétiques ou logiques.
- Le bus interne permet à ces unités et aux registres de communiquer entre eux.
- L'horloge rythme le processeur. A chaque top d'horloge le processeur effectue une instruction, ainsi plus l'horloge a une fréquence élevée, plus le processeur effectue d'instructions par seconde (MIPS: Millions d'instruction par seconde). Par exemple un ordinateur ayant une fréquence de 1 GHz (1000 MHz) effectue 1000 millions d'instructions par seconde.

La figure qui suit présente une architecture élémentaire d'unité centrale.



Fig. 4. Structure simplifiée de l'unité centrale

De nos jours d'autres composants sont intégrés au processeur tels que :

- Une unité flottante pour le calcul des opérations sur les nombres réels.
- La mémoire cache : c'est une mémoire de petite taille, à accès plus rapide que la mémoire principale. Elle permet au processeur de se "rappeler" les opérations déjà effectuées auparavant. En effet, elle sert à conserver les données ou les instructions fréquemment utilisées par le processeur. De la sorte il ne perd pas de temps à recalculer

des choses qu'il a déjà faites précédemment. La taille de la mémoire cache est généralement de l'ordre de quelques centaines de KO. Ce type de mémoire résidait sur la carte mère, sur les ordinateurs récents ce type de mémoire est directement intégré dans le processeur.

- Les unités de gestion mémoire servent à convertir des adresses logiques en des adresses réelles situées en mémoire.



2. Notions de systèmes d'exploitation

Avant l'avènement des systèmes d'exploitation, la conception d'un programme nécessitait la connaissance parfaite du mode de fonctionnement de la machine. Le programmeur devait gérer au moindre détail le placement des programmes en mémoire en utilisant directement les adresses, l'exécution au niveau du processeur à savoir le chargement et la terminaison des programmes, l'affichage sur l'écran des résultats, etc. Cette tâche étant très complexe, peu de programmes pouvaient être développés.

Il y a quelques années, on a ressenti le besoin de dissocier la programmation de la machine utilisée afin d'atteindre un grand nombre d'utilisateurs. Ainsi, on a développé une couche de logiciel pour enrober le matériel et le présenter aux programmeurs comme une machine virtuelle plus facile à comprendre et à utiliser. Le système d'exploitation est une couche de logiciel. La **fonction** du système d'exploitation est de masquer la complexité du matériel et de proposer des instructions plus simples à l'utilisateur

Le système d'exploitation est un gestionnaire de ressources, c'est-à-dire qu'il contrôle l'accès à toutes les ressources de la machine, l'attribution de ces ressources aux différents utilisateurs et la libération de ces ressources lorsqu'elles ne sont plus utilisées. À ce titre, tous les périphériques comme la mémoire, le disque dur ou les imprimantes sont des ressources. Le processeur également est une ressource.

3. Historique des systèmes d'exploitation

Partant du fait qu'un système d'exploitation est conçu pour une machine bien particulière, il est logique que les systèmes d'exploitation évoluent avec les générations d'ordinateurs. Sans les progrès de l'optique et de la chimie il n'y aurait pas de circuit intégré ni de processeur, indispensables au micro-ordinateur moderne. Il a fallu attendre les circuits intégrés et le premier véritable ordinateur pour disposer d'un véritable système d'exploitation. A l'origine on a commencé par développer du logiciel pour automatiser certaines tâches, pour compiler des programmes et pour réaliser les entrées-sorties. Ce n'est qu'avec la multiprogrammation qu'on a réellement senti le besoin d'un système d'exploitation capable de prendre en charge la machine et d'en dispenser le programmeur.

Cette section commence par rappeler les premières réalisations de machines, ensuite nous présentons pour chaque génération d'ordinateurs l'état d'avancement des systèmes d'exploitation.

⇒ [Sommaire de la section :](#)

- ✓ [Premières réalisations](#)
 - ✓ [La première génération \(1945-1955\)](#)
 - ✓ [La deuxième génération \(1955-1960\)](#)
 - ✓ [La troisième génération \(1960-1970\)](#)
 - ✓ [La quatrième génération \(1971- \)](#)
 - ✓ [La cinquième génération \(début des années 90\)](#)
 - ✓ [Les gammes d'ordinateurs](#)
-

3.1 Premières réalisations

L'introduction des cartes perforées par la mécanographie, a encouragé les premières machines à calculer. L'avènement de l'électronique a permis à l'informatique d'atteindre le développement qu'on lui connaît actuellement.

Le mathématicien, penseur et écrivain Blaise Pascal met au point en 1642 (à l'âge de 19 ans) une machine, qu'il appelle Pascaline, capable d'effectuer l'addition et la soustraction, pour aider son père qui était percepteur de taxes. Les calculs s'effectuaient en base 10 à l'aide d'un mécanisme à roues dentées.

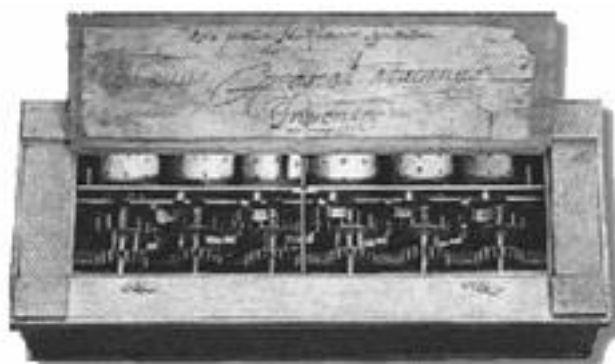


Fig. 5 . La machine à calculer de Pascal de l'extérieur et de l'intérieur

Au début des années 1800, vint la notion d'automates avec le métier à tisser (tissage des pulls) de Joseph-Marie Jacquard qui introduisit l'utilisation des cartes perforées. La carte perforée permettait aux machines de tisser les pulls en alternant les pelotes de différentes couleurs automatiquement grâce au dessin reproduit sur la carte.

Charles BABBAGE (mathématicien anglais à l'université de Cambridge) conçoit vers 1830 sa machine analytique, capable d'enchaîner des opérations arithmétiques de façon autonome. Cette première réalisation était entièrement mécanique. Elle comportait une mémoire, une unité de calcul et utilisait les cartes perforées pour les entrées/sorties.



Fig. 6 . La machine de Charles Babbage

En savoir plus sur les premières réalisations

En 1847, Georges BOOLE (mathématicien anglais) élabore la logique algébrique moderne, laquelle servira ultérieurement à l'élaboration des langages informatiques actuels.

En 1939, John Vincent Atanasoff, avec l'aide de son étudiant Clifford Berry de l'université d'Iowa, achève la mise au point de la première machine à calculer utilisant un système binaire, la ABC (Atanasoff-Berry Computer, complétée en 1942), destinée à résoudre des équations différentielles. Ce calculateur fonctionnait avec des lampes, et était capable de résoudre des équations à 29 variables. Sa fréquence d'horloge était de 60Hz et il effectuait 1 multiplication à la seconde.

En 1970, Alan Turing a introduit la notion d'algorithme et est connu pour avoir proposé une machine (virtuelle) capable de résoudre tout problème pouvant être mis sous forme d'algorithme.

3.2 La première génération (1945-1955) : Prototypes d'ordinateurs à base de tubes électroniques à vide

Le plus connu des ordinateurs non mécaniques est l'ENIAC (Electronic Numerical Integrator And Computer) mis au point de 1943 à 1946 par John Mauchly et J. Presper Eckert de l'Université de Pennsylvanie. Il pèse 30 tonnes, occupe quelques dizaines de m², comporte 18 000 tubes à vide, et nécessite 140 KW d'énergie. Il est capable d'effectuer 5000 additions à la seconde. Il avait été conçu dans le but d'effectuer des calculs pour l'armée américaine pendant la guerre, mais fut terminé trop tard. Il servit après la guerre aux calculs pour la bombe H, et fut utilisé jusqu'en 1955. Commandé par l'armée des États-Unis en 1943 pour effectuer les calculs de balistique, il remplaçait 200 personnes chargées auparavant de calculer les tables de tir.

L'ENIAC effectue ses calculs en décimal. Les données sont lues au moyen d'un lecteur de cartes perforées, et les résultats sont fournis sur cartes perforées ou imprimés avec une machine à écrire électrique.

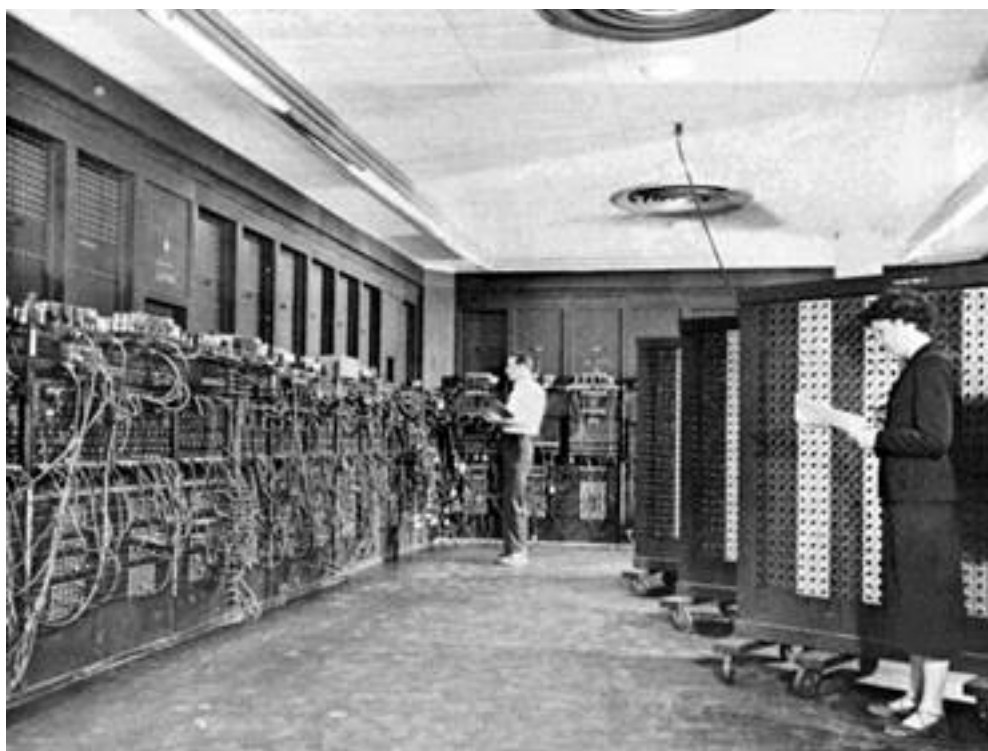


Fig. 7. L'ENIAC

En savoir plus sur cette génération

L'une des figures les plus marquantes de l'évolution des ordinateurs fut sans conteste John Von

Neumann. Mathématicien de génie, John Von Neumann participa à l'élaboration de la théorie des ensembles, à la conception de la bombe H, et fut le précurseur de l'intelligence artificielle et des sciences cognitives. Il définit pour la première fois le concept de calculateur à programme enregistré, qui régit tous les types actuels d'ordinateurs sous le terme générique de "machines de Von Neumann " (sauf dans de rares cas de parallélisme).

La proposition de Von Neumann présente l'ordinateur comme une machine dotée d'une unité centrale pour effectuer les calculs et d'une mémoire centrale pour y ranger les données et le programme à exécuter. L'unité centrale est composée d'une unité de commande et d'une unité arithmétique et logique. Par ailleurs, dans sa proposition, Von Neumann adopte le système binaire à la place du décimal.

Pour la première fois, le programme n'est pas câblé, mais est stocké en mémoire comme les données. Les travaux de Von Neumann furent utilisés pour la construction du premier ordinateur à programme enregistré : la machine EDVAC en 1947 par Eckert et Mauchly à l'université de Pennsylvanie, l'EDSAC en 1949 par Maurice Wilkes de l'université de Cambridge. L'EDSAC avait une mémoire centrale totale de 4096 mots de 40 bits. D'autres calculateurs l'ont suivi tel que l'UNIVAC (département du recensement américain) en 1951. En 1953 IBM met en circulation son premier ordinateur électronique basé sur les travaux de Von Neumann : le 701.

Bien entendu, aucun de ces calculateurs ne disposait de système d'exploitation. Par ailleurs, un groupe restreint de personnes concevait, construisait, programmait, utilisait et effectuait la maintenance de la machine.

3.3 La deuxième génération (1955-1960): Les transistors et le traitement par lots

En 1947, W. Shockley, W. Brattain et J. Bardeen, ingénieurs chez Bell, testent avec succès le transistor ouvrant ainsi la voie à la révolution des semi-conducteurs. En 1954, Texas Instruments met au point le transistor avec jonction à base de silicium. Il est plus petit, moins coûteux et dissipe moins d'énergie qu'une lampe à vide. Les transistors remplacent progressivement les tubes à vide. Ils permettent de concevoir des ordinateurs moins encombrants. L'emploi de mémoires de masse comme les bandes et les disques magnétiques se généralise.

Les principaux évènements

En 1955, des chercheurs des laboratoires Bell annoncent la création de TRADIC, le premier ordinateur composé en totalité de transistors. C'est pour cette raison que la deuxième génération commence en 1955.

Le transistor a permis à l'ordinateur de gagner en fiabilité et par conséquent d'en augmenter les ventes. De nombreuses constructions voient le jour. Par ailleurs, des langages de programmation comme Fortran (FORmula TRANslator) et LISP (premier langage de l'Intelligence Artificielle par John McCarthy), sont mis au point. Le premier disque dur est commercialisé par IBM, le RAMAC 305 (Random Access Method of Accounting and Control). Il peut stocker 5 MO de données.

Le traitement par lots

Le traitement par lot consiste à enchaîner automatiquement les travaux. L'idée était de collecter un ensemble de travaux, de les enregistrer sur bande magnétique, puis de faire lire la bande par l'ordinateur chargé des calculs. Les résultats étaient récupérés sur une autre bande pour être imprimés. Chaque programme de la bande était lu, exécuté et ensuite l'ordinateur passait automatiquement au suivant. Cet automatisme était possible grâce à un programme résident en mémoire appelé **moniteur** système qui permettait de passer le contrôle d'un programme à un autre.

En savoir plus sur cette génération

Les systèmes d'exploitation sont apparus avec les premiers vrais ordinateurs c'est-à-dire avec l'apparition du transistor. Désormais la machine devient un ordinateur et on dissocie le programmeur du constructeur de cette dernière.

Le traitement par lots est né dans les centres de calculs où plusieurs programmeurs transcrivaient sur cartes perforées leurs programmes écrits en fortran, assembleur et autres. Des opérateurs étaient chargés de donner les cartes à lire à l'ordinateur, de récupérer les résultats et de les communiquer aux programmeurs. Ils devaient également charger le compilateur adéquat qui lui aussi se trouvait sur bande magnétique. L'existence de ces opérateurs était due à des raisons de sécurité, en effet les ordinateurs de l'époque valant plusieurs milliers de dollars, ils étaient conservés dans des pièces à air conditionné où seuls les opérateurs étaient autorisés à pénétrer.

Il y avait un second avantage au traitement par lot à savoir que les opérateurs pouvaient trier les travaux et regrouper les programmes nécessitant le même compilateur ensembles.

Voyons à présent de plus près le moniteur. Celui-ci consiste en un programme dont on commande l'exécution en utilisant des directives pour indiquer par exemple quel est le premier programme à exécuter, quel compilateur utiliser. Le caractère \$ était alors adopté pour signaler une directive au moniteur (programme batch).

La firme IBM à l'époque permettait le traitement par lot grâce à deux machines : un ordinateur simple le 1401 pour gérer les cartes et le 7094 pour effectuer les calculs. Les systèmes d'exploitation s'appelaient Fortran Monitor System FMS et IBSYS.

3.4 La troisième génération (1960-1970) : Les circuits intégrés et la multiprogrammation

Le premier circuit intégré est développé par Jack Kilby en 1958 à Texas instruments. Le principe consiste à fabriquer dans un même bloc de semi-conducteur le plus réduit possible, un maximum de fonctions logiques, auxquelles l'extérieur pourrait accéder grâce à des connexions réparties tout autour du circuit.

En 1959, Robert Noyce met au point à la société Fairchild un circuit intégré imprimé sur une surface de silicium. De 1958 à nos jours, le concept de circuit intégré s'est extraordinairement développé. Son application la plus réputée a été le microprocesseur.

Les principaux évènements

La troisième génération d'ordinateurs est très riche en événements, en effet elle a vu la création de la société Intel, la naissance du système Unix et la mise en place du réseau de communication Arpanet ancêtre d'Internet.

En 1968, Robert Noyce et Gordon Moore, fondent la compagnie, Intel, contraction de Integrated et d'Electronics.

Cette génération a été marquée par la naissance du système Unix en 1969. Son ancêtre Multics (MULTiplexed Information and Computing System) a été conçu et développé par des chercheurs des laboratoires Bell et d'autres du MIT (Massachusetts Institute of Technology). Ce dernier avait des ambitions grandioses qu'il ne put jamais atteindre.

En 1969, deux chercheurs Ken Thompson et Dennis Ritchie décidèrent d'en réécrire une version simplifiée en assembleur qui s'est avérée fonctionnelle. Elle reçut le nom de UNICS (UNiplexed Information and Computing System, Uniplexed en opposition à MULTiplexed) puis fut très vite rebaptisé UNIX.

Unix tournait sur le PDP-7 et occupait 16 KO en mémoire. UNIX fut réécrit en langage C en 1973 ce qui a permis de le porter sur de nombreuses plate-formes.

La multiprogrammation

Cette génération introduit la notion de multiprogrammation puisque le processeur peut se partager entre plusieurs tâches. La multiprogrammation trouve ses origines dans l'idée suivante : un programme peut nécessiter une arrivée de donnée à partir du clavier ou à partir d'une bande qui peut ne pas être disponible. Dans tous les cas de figure, c'est une instruction d'entrée-sortie et son exécution est très lente vu la différence de vitesse entre le périphérique et le processeur. Au lieu que celui-ci reste inactif dans l'attente de la réalisation de cette dernière, le système d'exploitation fait basculer le processeur vers un autre programme et l'exécute. Le processeur peut ainsi avoir un rendement de 100%.

La multiprogrammation est un concept logiciel et non matériel. Il ne s'agit pas d'un nouveau modèle de processeur capable de réaliser plusieurs tâches en même temps, mais d'un système d'exploitation multiprogrammé, qui permet de gérer l'exécution de plusieurs programmes au niveau du processeur et de la mémoire.

Les systèmes d'exploitation de l'époque offraient les fonctionnalités d'allocation du processeur et de gestion des files d'entrée-sortie. La présence de plusieurs programmes en mémoire nécessite de nouveaux contrôles pour les protéger entre eux, ainsi le système d'exploitation gérait en plus l'allocation de la mémoire.

En savoir plus sur cette génération

Vers le milieu des années 60, IBM lance la gamme de machines SYSTEM/360 (avec le système d'exploitation OS/360) qui permet de gérer les entrées-sorties et d'effectuer des calculs en même temps. SYSTEM/360 est une famille de six ordinateurs compatibles entre eux et de quarante périphériques capables de travailler ensemble.

La société DEC met en vente en 1965, le premier mini-ordinateur ayant connu un certain succès commercial le PDP-8. La société HP commercialise sa gamme de mini-ordinateurs HP-2000 puis 3000.

En 1966, Seymour Cray élabore le super calculateur CDC 6600, un ordinateur multiprocesseurs capable d'exécuter trois millions d'instructions à la seconde.

Le langage Pascal est inventé en 1968 par Niklaus WIRTH.

Pour en revenir à l'histoire, les supports de stockage de masse sont largement exploités à cette époque et les ordinateurs 1401 disparaissent puisque le système d'exploitation charge les programmes directement à partir du disque. Avec l'avènement des disques, un nouveau mode de fonctionnement s'est développé : le spooling ou spool. Le spool (Simultaneous Peripheral Operations On Line) consiste, lorsque la mémoire est saturée, à placer les programmes non encore exécutés sur le disque dur dans une zone dite zone de spool. Dès que la mémoire se libère le système d'exploitation sélectionne un programme et le charge en mémoire centrale.

Après le traitement par lots, plusieurs modes d'exploitation des ordinateurs ont vu le jour :

- Le **temps partagé** variante de **la multiprogrammation**. Plusieurs utilisateurs peuvent se connecter à une même machine. Le processeur est partagé équitablement entre les tâches des différents utilisateurs. Des systèmes à temps partagé ont été développés pour la gamme 360 de IBM.
- Le **temps réel** pour le contrôle des systèmes temps réel : processus industriels, systèmes avioniques, radars, centrales nucléaires. Un système temps réel est un système dont l'exécution doit tenir compte de la survenue d'événements extérieurs auxquels il doit réagir en respectant certains délais, afin de garantir la fiabilité du système.

3.5 La quatrième génération (1971-) : Les micro-ordinateurs et les systèmes d'exploitation actuels

Cette génération est apparue vers les années 70 et est de loin la plus mouvementée et celle qui a le plus marqué la micro-informatique. Elle ne comporte pas de limite dans le temps, en effet aucun composant n'est venu révolutionner le monde de l'informatique depuis. De nouveaux micro-ordinateurs et systèmes sont sans cesse mis sur le marché. Pour cette raison, certains pensent que nous sommes encore en quatrième génération.

On la doit à la Technologie MOS (Metal Oxyde Semiconductor). Cette technologie permet de fabriquer des transistors plus petits et plus rapides. L'apparition des circuits LSI (Large Scale Integration) et VLSI (Very Large Scale Integration) a permis le développement des microprocesseurs. Le processeur occupe une surface de quelques mm² et son prix chute considérablement. On peut désormais avoir son ordinateur personnel à bon prix. Certaines fonctionnalités du système d'exploitation sont intégrées dans le silicium grâce à la technologie VLSI (la traduction des adresses par exemple).

Cette génération compte des centaines de fabricants de matériel informatique.

1.3.5.1 Les principaux événements

Le premier microprocesseur, le Intel 4004 apparaît en 1971. Conçu par Ted Hoff, ingénieur chez Intel, le 4004 est un microprocesseur capable de traiter des données de 4 bits. Il contient 2300 transistors et peut exécuter 60.000 opérations par seconde (fréquence de base de 108 KHz). Sa puissance était égale à celle de l'ENIAC.

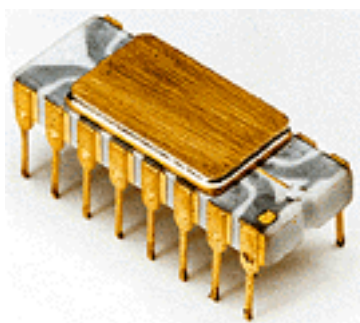


Fig. 8. Le processeur 4004

En 1974, Gary Kildall (directeur de Digital Research) écrit CP/M (Control Program for

Microcomputers) le premier système d'exploitation pour micro-ordinateur, ancêtre de MS-DOS.

En 1975, la société Microsoft est créée par Bill Gates et Paul Allen. Ils commencent par écrire un interpréteur BASIC pour l'ordinateur Altair.

Le 1^{er} Avril 1976, deux amis, Steve Wozniak et Steve Jobs fondent officiellement l'Apple Computer Company afin de commercialiser l'apple I. Jobs choisit le nom Apple car il pensait à la pomme comme au fruit parfait, et il voulait que Apple fut la compagnie parfaite.

La plupart des systèmes d'exploitation voient le jour et évoluent durant cette génération. Ils sont pour la plupart multitâches, multi-utilisateurs et gèrent la mémoire virtuelle.

En savoir plus sur cette génération

Le premier micro-ordinateur, le Kenback 1 (de Kenback Corporation), voit le jour en 1971. Il intégrait le processeur 4004 et avait une mémoire de 256 octets. Suivi du Micral en 1973, fabriqué par la société R2E (Réalizations Etudes Electroniques en France). Il était muni du processeur 8008 de Intel et capable d'adresser 16 KO de mémoire. En 1975, Le Altair 8800 est présenté par ED. Roberts (MITS) et basé sur le processeur 8080.

Le produit phare d'apple, l'ordinateur Apple I était basé sur la puce MOS 6502 (MOS Technologies) de puissance 1 MHz et capable d'adresser 64 KO de mémoire, alors que la plupart des autres micro-ordinateurs étaient construits à base d'Intel 8080. Il était plus simple au démarrage grâce à un programme intégré en ROM. L'Apple I n'était qu'une carte-mère, (la plupart des Apple 1 ont été équipés ensuite d'un caisson en bois, cf. figure n° 13). Cette première machine ne fut pas un succès. L'Apple II, sorti en 1977 fût le premier succès de la société grâce à son interface graphique. Le macintosh de la société sera commercialisé en 1984 (figure n° 14).



Fig. 9. L'apple I dans un caisson en bois



Fig. 10. Le Macintosh

En 1977, deux étudiants de l'Université de Californie à Berkeley, Bill Joy et Chuck Haley, proposent leur première version d'UNIX BSD (Berkeley Software Distribution) qui comporte de nombreuses améliorations par rapport à la version AT&T.

L'ordinateur Alto est construit en 1974, au centre de recherche Xerox. Il utilise le langage objet Smalltalk, une souris comme outil de pointage et peut être raccordé avec d'autres ordinateurs Alto.

En 1978, Intel 8086 présente le premier processeur à architecture x86, qui permettait d'adresser 1 Mo de mémoire. La société DEC présente son ordinateur 32 bits, le VAX 11/780 (Virtual Address eXtension.) le premier modèle de "supermini". Il présentait suffisamment de ressources pour supporter des applications qui étaient jusqu'ici réservées aux gros calculateurs. La série des VAX fonctionne avec le système d'exploitation VMS.

En 1979, Motorola présente le 68000 (le premier d'une grande famille) un processeur 32 bits. Motorola nomma son processeur 68000 car il contenait 68000 composants, il équipera le Macintosh en 1984 ainsi que les Atari et Amiga.

Le 12 août 1981, IBM dévoile son "Personal Computer" une machine 16 bits bâtie sur le processeur Intel 8088 avec 16 KO de mémoire RAM et capable d'adresser jusqu'à 1MO de mémoire physique (centrale). En 1982, Le microprocesseur Intel 80286, sera au cœur de la fameuse machine AT d'IBM. En 1983 IBM lance un nouveau micro, le XT, avec un disque dur intégré de 10 MO.



Fig. 11. L'IBM PC

Les processeurs Motorola à partir du 68010 pouvaient gérer la mémoire virtuelle, qui est une technique où la mémoire centrale est étendue de manière virtuelle par de l'espace sur le disque dur.

L'année 1982, verra la création de Sun (Stanford University Network) Microsystems, en plein cœur de la Silicon Valley, par des universitaires issus de Berkeley et Stanford. Sun invente le concept de station de travail, intermédiaire entre le PC et le mini-ordinateur. Unix sera leur système d'exploitation.

Les processeurs Risc voient le jour. Les plus célèbres fabricants sont : SUN avec l'architecture SPARC, Motorola avec le 88000, IBM avec RISC6000, Motorola-IBM-Apple avec le PowerPC, HP avec HP-PA, Digital avec Alpha, MIPS (devenu Silicon Graphics) avec les R_x000.

Cette génération a également vu la naissance des différents formats de disquettes : la 8 pouces en 1971 par IBM, la 5 pouces ¼ en 1976 par Shugart Associates et la fameuse 3 pouces ½ en 1980 par Sony.

En 1991, Linus Torvald, 21 ans, étudiant en licence universitaire d'informatique en Finlande,

présente la première version de Linux, un système d'exploitation gratuit et incluant les sources, qui va révolutionner l'informatique au 21^{ème} siècle.

En 1993, Intel présente le Pentium l'aîné d'une famille de processeurs 32 bits. En 1995 Sun Microsystems présente le langage orienté objet, JAVA.

3.6 La cinquième génération (début des années 90) : Les systèmes parallèles et répartis

On observe depuis quelques années une tendance vers les systèmes multiprocesseurs. Les raisons ayant conduit au développement de tels systèmes sont entre autres un besoin de plus grandes capacités et de performances, une quête de machines plus fiables, et un partage de ressources.

On distingue deux types d'architectures multiprocesseurs : les systèmes dits parallèles et ceux dits répartis ou distribués.

Un système informatique réparti est composé de plusieurs sites chacun comportant une ou plusieurs machines reliées à travers un réseau de communication. Il comporte également des ressources matérielles telles que les imprimantes, les disques durs et autres qui peuvent ainsi être partagées par tout le système.

Un système d'exploitation réparti fournit et contrôle l'accès aux différentes ressources du système. Il doit offrir une transparence aux utilisateurs, ceux-ci doivent pouvoir accéder à toute ressource distante comme si elle était locale. La figure n° 16 illustre un exemple de système réparti.

Fig. 12. Schéma de système réparti

Dans un système réparti, le calcul peut être accéléré mais d'une manière transparente à l'utilisateur à savoir qu'il ignore si une partie de son traitement a été exécutée sur un autre site. Outre l'accélération du calcul, ces systèmes offrent l'avantage de partager des ressources. Un

disque de grande capacité peut être utilisé par plusieurs sites, de même qu'une imprimante, ce qui permet de réaliser des économies. Un campus universitaire est typiquement un exemple de système réparti.

Un système parallèle est un système d'exploitation pour une machine parallèle qui est composée de plusieurs processeurs. Le nombre des processeurs peut atteindre plusieurs milliers. Des exemples de machines parallèles sont le CDC 6600, le premier super ordinateur suivi de la famille des CRAY (CRAY Research). Egalement la Paragon de Intel, la Connection machine, l'Hypercube à base des processeurs transputers de Inmos, le Butterfly de BBN, le RP3 de IBM.

3.7 Les gammes d'ordinateurs

On distingue principalement 3 gammes d'ordinateurs, chacune ayant ses propres systèmes d'exploitation :

- **Le micro-ordinateur** : on y trouve le compatible PC (ordinateur personnel), le Macintosh, le portable et la station de travail : le Macintosh est apparu en 1984 et c'est un des premiers ordinateurs pilotés par une souris et dotés d'une interface utilisateur graphique (après l'alto de Xerox). Le Macintosh ne peut supporter que le système du Macintosh Mac-OS, qui a atteint la version X. Il se caractérise par une interface graphique très sophistiquée qui a de loin devancée le PC. En revanche le PC peut supporter plusieurs systèmes d'exploitation tels que Ms-Dos, diverses versions d'Unix et de Windows. A noter qu'il existe des centaines de fabricants de matériel informatique qui construisent des ordinateurs compatibles PC, mais que seule la société Apple fabrique des Macintosh.

La plupart des constructeurs d'ordinateurs proposent un modèle de leur gamme en portable. Le processeur Intel centrino a été spécialement conçu pour les portables. Il leur permet en outre d'être légers.

La station de travail est une sorte d'ordinateur personnel mais bien plus puissant et disposant généralement d'une carte graphique haut de gamme. La puissance est obtenue grâce à l'adjonction de processeurs ou par l'utilisation de modèles de processeurs à architecture différente tels que les processeurs RISC par exemple le processeur SPARC (SUN), PowerPC (IBM), Alpha(DEC). Nous citons les stations SUN, HP, IBM, DEC, MIPS. On distingue les stations de travail sur plate forme Intel (plus économiques) et les stations de travail sur architecture RISC qui restent les plus performantes en calcul 3D par exemple.

La station de travail étant une machine chère, elle est souvent exploitée par plusieurs utilisateurs qui s'y connectent pour exécuter des logiciels qui nécessitent des quantités de mémoire et des performances de calcul en virgule flottante considérables dans des domaines tels que les effets spéciaux de films, les animations 3D, l'ingénierie et les applications scientifiques.

- **Les minis ordinateurs** : ce sont des serveurs multiprocesseurs auxquels on peut connecter (grâce à un réseau local) plusieurs terminaux qui peuvent être géographiquement éloignés. On les utilise généralement pour les systèmes bancaires, de réservation d'avion, les assurances etc. Dans les agences, les terminaux permettent de se connecter au serveur et d'exécuter les applications qui s'y trouvent. Généralement les terminaux ne disposent ni de processeur ni de mémoire de stockage. Tout le travail et la sauvegarde sont effectués sur le serveur. Ces mini-ordinateurs ont souvent leur système propriétaire fourni par le constructeur. Nous citons les systèmes OS400 et AIX pour les ordinateurs IBM, VMS pour les machines de type VAX (DEC) etc. Toutefois certains mini supportent des systèmes d'exploitation tels que Unix et Windows 2000 mais dans une version plus évoluée que celle pour PC.
- **Les super calculateurs** : Cette gamme désigne les machines comportant un très grand nombre de processeurs et réservées à des calculs scientifiques longs et complexes. A titre d'exemple nous citons les centres de recherche spatiale qui les utilisent pour simuler de manière intensive les programmes destinés aux futures navettes. Ces calculateurs également ont un système d'exploitation propriétaire fourni par le constructeur. Parmi les super-calculateurs, on retrouve les CRAY, Les SP de IBM, La Connection machine (16 000 processeurs), la Paragon de Intel (2 000 processeurs) etc.

4. Les principaux systèmes d'exploitation

L'évolution des micro-ordinateurs a été encouragée par l'apparition des systèmes d'exploitation adéquats. Les systèmes d'exploitation actuels sont multi-utilisateurs. Ils intègrent une gestion des réseaux, et permettent la protection des utilisateurs entre eux. Nous présentons ci-dessous une liste des principaux systèmes sur le marché dans un ordre chronologique (approximatif). Une liste exhaustive est fournie dans ??

VMS

VMS est la propriété de DEC. Il est apparu avec le premier ordinateur VAX de digital. VMS (Virtual Memory System), est un des premiers systèmes à appliquer la mémoire virtuelle.

CP/M

L'histoire des systèmes d'exploitation pour PC commence avec CP/M créé par Gary Kildall en 1974. A cette époque, chaque ordinateur était livré avec son propre système d'exploitation, étroitement dépendant du matériel utilisé. L'idée de CP/M (Control Program for Microcomputers), une première à l'époque, consistait à créer un système d'exploitation pouvant fonctionner sur les machines de plusieurs constructeurs. C'est d'ailleurs avec CP/M que le Bios apparaît, cette couche logicielle basse permettant d'interfacer matériel et système d'exploitation. CP/M est porté sur presque toutes les plates-formes en vue de l'époque.

MS-DOS

MS-DOS est le plus connu des premiers systèmes d'exploitation pour PC. Ses concepteurs ne se doutaient pas du succès qu'il aurait. Il est mono-utilisateur et mono-tâche. On a du greffer des couches logicielles pour répondre aux évolutions matérielles et aux demandes des utilisateurs. Ms-Dos a été rapidement supplanté par les systèmes Windows.

Mac OS

C'est le système d'exploitation de la firme Apple. Il a été livré pour le Macintosh en 1984.

C'est un des premiers systèmes à utiliser la souris et une interface graphique avec plusieurs fenêtres.

La version actuelle est la X (prononcer dix). Mac OS X se distingue par un noyau Darwin qui est un open source. Mac OS est un des principaux rivaux des Windows.

OS/2

En 1987, IBM propose le PS/2, plus puissant que le PC avec un nouveau système d'exploitation OS/2 (copropriété d'IBM et de Microsoft). Celui-ci est multitâche.

Il est renommé OS/2 Warp Server à partir de la version 3.

NetWare

NetWare de Novell est le premier système d'exploitation réseau 32 bits pour PC. Il intègre un serveur de fichiers et d'impression.

Unix

Unix étant distribué gratuitement à ses tous débuts, il a donné naissance à de nombreuses versions : Les versions les plus connues à ce jour sont Unix SYSTEM V (évolution de la version initiale d'AT&T et Bell) et Unix BSD. Il fonctionne aussi bien sur PC que sur les mini-ordinateurs.

Les principaux Unix du marché sur Intel sont : Open Server et Unixware de SCO (Santa Cruz Operation), Solaris (Sun Microsystems), BSD (Berkeley), Caldera OpenLinux. Cependant trois Unix dominent le monde des serveurs : HP/UX, Sun Solaris, IBM AIX.

Linux

Linux a pris des parts de marché aux Unix, à Novell Netware et à Windows NT-2000 serveur. Il s'est imposé dès la fin du 20^{ème} siècle. Linux est multi-utilisateurs, multi-tâches, stable et gratuit.

Principales distributions de Linux : RedHat (la plus appréciée des administrateurs de serveurs),

MandrakeSoft (plus facile ou assistée pour débutants), Suze (allemande), Debian, Caldera (devenue payante), Turbolinux (plus connue en Asie).

La famille des Windows :

Microsoft propose en 1992 Windows 3.10 et Windows pour Workgroups 3.11 dont les mots clés sont Multifenêtres et Multitâches coopératif. En 1993, on voit apparaître la première version de Windows NT 3.1 suivie en 1994 par NT 3.5.

L'année 1995, verra la sortie du fort célèbre [Windows 95](#) ("Et soudain le monde devient plus beau"). En 1996, Windows NT 4 avec deux versions station de travail et [Serveur](#). Ensuite, Windows Terminal Server : un système qui simule un environnement multi-utilisateurs et prend en charge la connexion de plusieurs terminaux. En 1998 Windows 98.

En 2000, Microsoft commercialise Windows 2000 professionnel et serveur, Windows Millenium, suivi de Windows XP familial et serveur.

Windows 2003 (initialement baptisé .NET) sort en 2003.

5. Le logiciel dans un ordinateur

La définition correcte d'un ordinateur inclut la machine elle-même, le système d'exploitation et les logiciels. En effet, si l'on fait l'acquisition d'un ordinateur c'est généralement pour une utilisation immédiate [vu la somme qu'il faut débours]. Il est donc nécessaire de se munir de logiciels ou d'applications à exécuter sur son ordinateur.

A présent, il s'agit de situer le système d'exploitation parmi le logiciel qui se trouve être un terme familier pour désigner un programme. Pour comprendre cela, le logiciel dans un ordinateur se répartit en 2 catégories :

- Les programmes fondamentaux qui permettent le fonctionnement de l'ordinateur (dits programmes système)
- Les programmes destinés aux utilisateurs (logiciels et applications)

La figure n° 17 présente en détail les différents types de logiciels dans un ordinateur.

Fig. 13. Types de logiciel dans un ordinateur

La partie matérielle contient au plus bas niveau les circuits intégrés, les périphériques etc, en somme les composants matériels de l'ordinateur. Au dessus, on trouve une couche de logiciel pour contrôler ces périphériques et exécuter les instructions en langage machine. Ce logiciel est considéré comme faisant partie du matériel par les constructeurs.

⇒ Sommaire de la section :

✓ Les programmes système

✓ Les logiciels et les applications

5.1 Les programmes système

La deuxième couche constitue les programmes systèmes. Le système d'exploitation est le programme fondamental des programmes système. Il est généralement livré avec un ensemble d'outils également nécessaires pour pouvoir exploiter l'ordinateur. Nous citons les éditeurs de texte nécessaires afin de taper un fichier de commandes ou un fichier dans un langage de programmation donné, les compilateurs pour compiler ces fichiers. Il est également nécessaire de disposer d'un interpréteur de commandes qui comme son nom l'indique transforme des commandes simples dictées par l'utilisateur en des ordres mieux élaborés pour le système d'exploitation. Des utilitaires système sont également fournis tel que la compression ou la vérification du disque. Le système d'exploitation est un logiciel protégé. Un utilisateur peut écrire son propre compilateur, mais il ne peut pas modifier un système de gestion d'interruptions ou de mémoire et en proposer un autre pour le système d'exploitation (à moins qu'il ne dispose du code source du système).

Les outils systèmes se trouvent au dessus du système d'exploitation car ils y font appel. Nous verrons par la suite que tout ce qui s'exécute sur un ordinateur doit être géré par le système d'exploitation. Souvent toute cette couche est vue comme le système d'exploitation car elle est fournie ensemble.

5.2 Les logiciels et les applications

La dernière couche est composée des applications et des logiciels. Elle est entièrement indépendante du système d'exploitation mais elle repose bel et bien au dessus.

Un logiciel donné fait appel au système d'exploitation afin d'ouvrir un fichier par exemple. Un même logiciel ne peut pas être compatible avec toutes les plates-formes et il existe une version par système d'exploitation.

1. Il existe une différence fondamentale entre l'application et le logiciel. Les applications sont destinées à être utilisées par un public de non spécialistes en informatique. On distingue les applications bancaires, de réservation d'avion, les jeux, les applications de gestion, de comptabilité, les applications du domaine industriel etc. Vouloir tout citer devient un défi. Toutefois, le logiciel peut être structuré en deux catégories : le logiciel d'exploitation et le logiciel de développement.

TD système d'exploitation N°1

1. Quelles sont les deux principales fonctions d'un système d'exploitation ?
2. Que fait l'UC quand il n'y'a aucun programme à exécuter ?
3. Quelle est la caractéristique commune aux déroutements, aux interruptions, aux appels au superviseur et aux appels aux sous-programmes ?
4. Qu'est ce qui différencie les déroutements, les interruptions et les appels au superviseur des appels aux sous-programmes ?
5. Parmi les instructions suivantes, lesquels doivent être privilégiées (à savoir lesquels ne peuvent être exécutés qu'en mode superviseur) ?
 - a. changement des registres de gestion de mémoire
 - b. écriture du compteur de programmes
 - c. lecture de l'horloge
 - d. réglage de l'horloge
 - e. changement de la priorité du processeur
6. Un système d'exploitation peut mettre en œuvre un périphérique d'E/S de mémoire. Les opérations d'E/S du périphérique provoquent la lecture ou l'écriture de l'emplacement de mémoire correspondant. Quel est l'inconvénient de ce type de périphérique ? doit-il être accessible aux utilisateurs ou simplement aux administrateurs système ?
7. Les applications suivantes sont-elles des applications par lot ou des applications interactives ?
 - a. traitement de texte
 - b. production de relevés bancaires mensuels
 - c. calcul du nombre pi jusqu'à un million de chiffres après la virgule

- 8. Pourquoi un ordinateur doit-il démarrer en mode superviseur lors de sa première mise sous tension**
- 9. Quelle est la taille maximale d'un programme d'initialisation de niveau 1 au début du disque dur, sachant que le profil binaire valide est de 2 octets et la taille du secteur est de 512 octets ?**

Architecture des systèmes d'exploitation

Objectifs

1. Les tâches d'un système d'exploitation
 2. L'interface de programmation
 3. Modèles de systèmes d'exploitation
 4. TD2
-



1 - Les tâches d'un système d'exploitation

L'existence d'une multitude de systèmes d'exploitation peut laisser penser qu'ils sont tous différents. Sachant que les systèmes sont conçus pour une gamme d'ordinateurs, il est plus judicieux de dire que les systèmes d'exploitation sont différents d'une gamme d'ordinateurs à l'autre. Ils se distinguent par l'interface qu'ils proposent et les algorithmes et stratégies qu'ils appliquent.

Le principal objectif d'un système est de gérer les composants de l'ordinateur. On retrouve par conséquent quatre tâches qui correspondent à la gestion du disque dur, de la mémoire centrale, du processeur et des périphériques.

⇒ Sommaire de la section :

- ✓ La gestion de la mémoire secondaire
 - ✓ La gestion de la mémoire centrale
 - ✓ La gestion du processeur
 - ✓ La gestion des entrées/sorties
 - ✓ La gestion du réseau
 - ✓ L'interface du système d'exploitation
-

1.1 La gestion de la mémoire secondaire

Le disque dur est un support auxiliaire qui a pour objectif de conserver de manière permanente les programmes exécutés en mémoire centrale (celle-ci étant volatile). Les programmes sont stockés sous forme de fichiers sur le disque dur et organisés en répertoires. La gestion de la mémoire secondaire inclut ainsi la gestion des fichiers c'est pour cela que l'on parle de **SGF** : Système de Gestion des Fichiers. Le système de fichiers doit offrir des primitives afin de créer, copier, lire, supprimer... ces fichiers. Il doit également gérer l'espace occupé par les fichiers ainsi que l'espace libre. Pour finir il doit prendre en charge le partage et la protection des fichiers dans un environnement multi-utilisateurs.

1.2 La gestion de la mémoire centrale

La mémoire centrale est un espace de taille importante organisé en mots (ensemble d'octets) et destiné à accueillir les données à traiter par l'unité centrale. L'unité centrale charge les instructions à exécuter dans les registres du processeur à partir d'adresses en mémoire centrale. De même après exécution, les résultats sont placés en mémoire centrale.

Il s'agit ici de gérer l'allocation de cette mémoire aux programmes (attribution, libération de mémoire), les règles d'adressage et de veiller à ce que les programmes en mémoire ne puissent pas interférer entre eux. Le système doit prendre en charge la mémoire virtuelle et offrir tous les mécanismes nécessaires à sa mise en œuvre. La mémoire virtuelle est une technique grâce à laquelle on peut considérer la mémoire centrale comme de taille infinie. En réalité, les programmes sont stockés sur disque dur et ils sont chargés en mémoire centrale au fur et à mesure que cela est possible.

1.3 La gestion du processeur

La principale tâche du système d'exploitation concerne l'allocation du processeur aux processus. Il s'agit de décider quel processus s'exécute à un moment donné, à quel moment interrompre le processus, quel sera le suivant, et de quoi il a besoin comme ressources pour son exécution. Le système d'exploitation doit gérer deux types de processus : les siens et ceux des utilisateurs.

Le système d'exploitation gère également les conflits dus à la concurrence et leurs solutions, en effet les processus peuvent utiliser des variables en commun. Si un premier processus utilise une variable donnée X et s'il la modifie et est ensuite interrompu, un second processus ayant besoin de la valeur de X doit y trouver la dernière valeur obtenue par le premier processus et non une valeur intermédiaire.

Le système doit par ailleurs offrir des primitives pour assurer la communication entre les processus ainsi que leur synchronisation. Pour finir le système d'exploitation doit parer aux erreurs et effectuer la correction des situations d'interblocage entre processus en fournissant les mécanismes adéquats.

1.4 La gestion des entrées/sorties

L'existence d'entrées-sorties dans un programme introduit de nombreux problèmes dus notamment à la différence de vitesse entre les périphériques et l'unité centrale, le risque d'erreur de programmation qui peut entraîner le blocage de l'unité centrale. Ainsi lors de la conception d'un système d'exploitation, une attention particulière est accordée aux entrées-sorties et généralement cela se traduit par la conception d'un système d'entrée-sortie qui permet de protéger les entrées-sorties et d'en interdire généralement l'accès direct aux programmeurs (mais cela dépend du système d'exploitation). De la sorte on évite de monopoliser l'unité centrale pendant le déroulement de l'entrée-sortie, et on assure un partage efficace des ressources entre les programmes.

1.5 La gestion du réseau

Les systèmes d'exploitation modernes intègrent d'autres caractéristiques qui trouvent leurs origines dans les évolutions des systèmes informatiques. L'interconnexion des machines par des réseaux locaux, constitue une des majeures évolutions. Les systèmes d'exploitation actuels prennent en charge cet aspect dans la mesure où ils offrent un partage des fichiers utilisateurs, une protection pour ces fichiers, l'identification des machines et des utilisateurs connectés au réseau etc. Nous ne traiterons pas cette partie car cet cours étudie les systèmes d'exploitation centralisés. Les lecteurs intéressés peuvent consulter les références bibliographiques suivantes (21), (24).

1.6 L'interface du système d'exploitation

L'interaction entre les systèmes d'exploitation et l'utilisateur se fait généralement de deux manières. La première est simple d'utilisation, conviviale et est destinée aux utilisateurs peu expérimentés ou bien ceux désirant simplement interroger le système. Le dialogue se fait à travers des commandes telles que "dir" ou "ls" pour visualiser le contenu d'un répertoire... Un programme système appelé interpréteur de commandes offre de nombreuses commandes outre "dir" ou "ls" afin d'interroger le système. Récemment, les systèmes d'exploitation ont adopté les fenêtres multiples disposant de propriétés graphiques, ce qui rend cet échange encore plus convivial. En sus le système d'exploitation offre une autre forme de dialogue adaptée aux programmeurs. Elle consiste en une série d'appels système correspondant à des fonctions qui permettent par exemple l'affichage du contenu d'un répertoire, la création d'un répertoire, l'allocation d'une zone de la mémoire principale, la création d'un processus.

2 - L'interface de programmation

⇒ Sommaire de la section :

✓ Les appels système

✓ Les modes superviseur et utilisateur

✓ Exécution d'un appel système

2.1 Les appels système

Certains traitements ne peuvent pas être exécutés et contrôlés directement par un programme utilisateur car le système ne le permet pas. Le programmeur doit faire explicitement appel aux services du système d'exploitation pour accéder à certaines ressources. À ces fins, le système d'exploitation propose une interface de programmation, c'est-à-dire qu'il permet d'accéder à un certain nombre de fonctionnalités qu'il exécute pour l'utilisateur. Les appels système sont l'interface proposée par le système d'exploitation pour accéder aux différentes ressources de la machine.

Pour chaque composant de l'ordinateur le système d'exploitation propose des **appels système**. Seuls ces appels systèmes peuvent être utilisés, en effet le reste des primitives du système est protégé et leur usage direct est interdit aux utilisateurs. Les appels système sont généralement classés en quatre catégories :

§ Gestion des processus

§ Gestion des fichiers

§ Communication et stockage d'informations

§ Gestion des périphériques

Par exemple, il est nécessaire de faire appel au système d'exploitation pour créer un fichier sur le disque dur et cela via un appel système comme **createfile** sous Windows 2000. Il est logique d'interdire au programme utilisateur d'effectuer lui-même la création pour les raisons suivantes :

§ Le système d'exploitation doit vérifier si l'utilisateur dispose du droit d'écriture dans le répertoire, autrement il refusera de créer le fichier.

§ Il doit également vérifier si un fichier du même nom n'existe pas déjà

§ La création du fichier nécessite l'attribution de quelques secteurs du disque dur pour l'enregistrement des données du fichier. L'utilisateur n'a aucune connaissance des zones libres

appel

du disque.

2.2 Les modes superviseur et utilisateur

Afin de protéger l'exécution du système d'exploitation de celles des programmes utilisateurs, les processeurs actuels proposent deux modes de fonctionnement :

§ Le mode utilisateur dans lequel les programmes utilisateurs sont exécutés. Un programme utilisateur qui perd le contrôle peut anéantir des données d'un autre programme ou même nuire au système d'exploitation. Dans ce mode certaines instructions sont interdites, de cette manière on contraint les programmes à faire appel au système d'exploitation pour certaines opérations.

§ Le mode protégé ou superviseur (également appelé mode noyau) est réservé à l'exécution des primitives du système d'exploitation. Dans ce mode le processeur peut exécuter toutes les instructions. Les appels système s'exécutent dans ce mode.

En n'autorisant l'accès aux différentes ressources de la machine qu'aux programmes s'exécutant en mode protégé, le système d'exploitation protège ses ressources et contraint les programmes utilisateurs à faire appel à lui pour accéder aux ressources de la machine.

2.3 Exécution d'un appel système

Lorsqu'un programme effectue un appel système, son exécution en mode utilisateur est interrompue et le système prend le contrôle en mode superviseur. En effet un appel système provoque en fait une interruption logicielle et il suffit alors de programmer la machine pour que la routine correspondant à l'interruption fasse partie du système d'exploitation. Un appel système est exécuté en mode noyau même si le programme ayant demandé son exécution est exécuté en mode utilisateur.

Fig. 14. Exécution d'un programme utilisateur par le système d'exploitation

Les appels système sont des fonctions, leur exécution donne en retour un résultat (un descripteur de fichier pour un appel système concernant les répertoires et fichiers). Si l'appel échoue, le résultat vaut -1.

3 - Modèles de systèmes d'exploitation

Le système d'exploitation se présente sous la forme d'un ensemble de primitives, chacune mettant en oeuvre une partie du système. Lorsqu'un utilisateur donne un ordre au système d'exploitation, ce dernier doit appeler plusieurs primitives, chacune exécutant la partie qui lui incombe. Prenons l'exemple d'une requête d'ouverture de fichier. Nous pouvons aisément imaginer les primitives invoquées : la recherche de l'existence du fichier à travers les informations maintenues par le système d'exploitation, la vérification de l'existence du fichier en mémoire centrale, le chargement des blocs constituant le fichier à partir du disque dur.

On distingue plusieurs manières de structurer ces primitives. Par exemple, le système d'exploitation peut autoriser que ces dernières s'appellent entre elles sans aucune restriction, ou bien il peut interdire à des fonctions de gestion de la mémoire d'appeler des fonctions de gestion des entrées/sorties.

On distingue trois principales manières d'organiser les primitives du système à travers les différents types de systèmes d'exploitation. La section suivante présente les différents modèles de systèmes d'exploitation.

⇒ [Sommaire de la section :](#)

- ✓ [Les systèmes monolithiques](#)
 - ✓ [Les systèmes à couches](#)
 - ✓ [Le modèle client-serveur](#)
-

3.1 Les systèmes monolithiques

Cette approche est réputée pour son désordre, car elle ne possède pas de structure bien définie. Le terme monolithique désigne un seul niveau. Le système d'exploitation est constitué d'un gros noyau et de programmes système, une collection de fonctions chacune pouvant appeler l'autre à tout moment. Le noyau s'exécute en mode superviseur et les programmes en mode utilisateur. On peut illustrer le système d'exploitation (voir la figure n° 15) comme un programme principal qui appelle plusieurs procédures chacune correspondant à un appel système, et qui utilise des procédures utilitaires pour assister ces procédures.

Fig. 15. Un système d'exploitation à structure monolithique

Ainsi, n'importe quelle procédure du noyau peut en appeler n'importe quelle autre et l'implantation de nouveaux services est très délicate : une erreur à un endroit du noyau peut entraîner un dysfonctionnement à un autre endroit qui, a priori, n'a rien à voir avec le premier.

Le système MS-DOS appartient à cette catégorie, en effet, il a été conçu par des personnes qui ne se doutaient pas du succès qu'il aurait. Le fait que les procédures ne soient pas protégées les unes des autres explique les blocages fréquents qui peuvent avoir lieu avec ce système.

Les premières versions de Unix utilisaient un modèle structuré en noyau et en modules s'exécutant en mode utilisateur. Le noyau quant à lui était à structure monolithique. Toutes les fonctionnalités (gestion du matériel, des processus, gestion du réseau et des systèmes de fichiers) étaient regroupées dans un seul module volumineux. Linux également possède un noyau monolithique. Il présente toutefois l'avantage d'être modulaire à savoir que le noyau est fragmenté en plusieurs modules ce qui à l'avantage de regrouper ensemble les procédures destinées à la gestion d'un même composant.

3.2 Les systèmes à couches

Un système est structuré en couches lorsqu'il est composé de plusieurs niveaux, chacun s'appuyant sur celui qui lui est immédiatement inférieur, et que l'intérieur de chaque niveau ou couche est masqué par rapport aux autres.

Le masquage des informations est important dans la mesure où il protège les couches entre elles dans un premier temps et dans un second temps donne une liberté aux programmeurs ; on parle alors de système modulaire.

Chaque couche est implémentée en appelant les primitives offertes par la couche immédiatement inférieure. L'intérieur de ces primitives, à savoir les structures de données, le code et les appels que cette couche fait, est masqué.

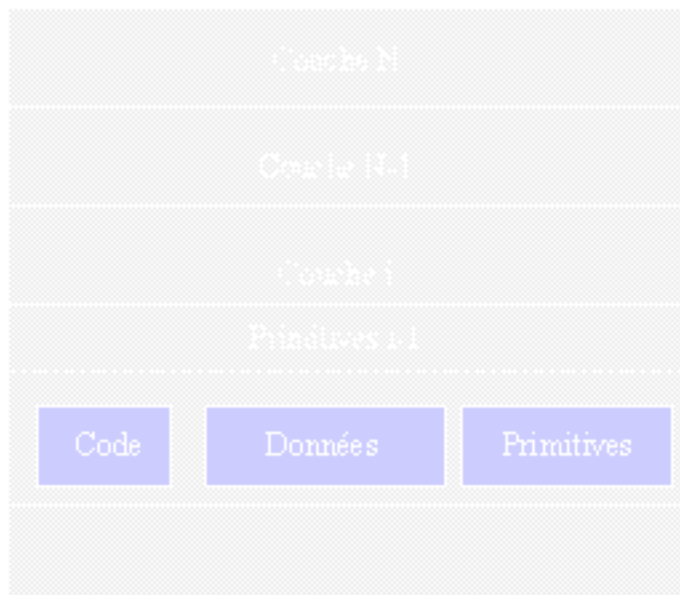


Fig. 16. Structure d'un système d'exploitation en couches

Cette approche présente toutefois divers inconvénients qui ont fait qu'elle a été délaissée par la suite. Le principal inconvénient est qu'un appel système est transmis de couche en couche avant d'être traité, ce qui ralentit son traitement. L'autre difficulté est qu'il faut définir de manière adéquate les couches et les frontières entre elles. On peut effectuer un découpage logique, par exemple, une couche implémente la gestion de la mémoire, une autre

l'ordonnancement des processus, etc.

Le système THE développé par E. W Dijkstra et ses élèves aux Pays-bas, est un des premiers systèmes à couches. Il porte le nom de l'institut où il a été développé Technische Hogescholl d'Eindhoven. MULTICS, ancêtre d'Unix, était basé sur la structure en couches, et était composé de 6 couches concentriques.

3.3 Le modèle client-serveur

Il est issu d'une tendance des systèmes d'exploitation actuels à réduire le système à un noyau minimal. Pour éviter les problèmes posés par les gros noyaux ci-dessus présentés, une solution consiste à réduire le noyau à quelques procédures essentielles (**ordonnancement des tâches, primitives d'entrées/sortie**) et à reporter tous les autres services dans des programmes utilisateurs. Lorsque le noyau est extrêmement réduit, on le qualifie de micro noyau.

Dans ce cas, le micro noyau est conçu de manière complètement différente. En effet, tout ce qui sort du cadre strict de la gestion de matériel, de processus, de mémoire et de la communication inter processus ne fait plus partie du micro noyau. L'avantage qui en découle est naturellement une modularité extrême. Il est alors possible d'implémenter toutes les autres fonctionnalités en dehors du noyau, ce qui a pour effet de les placer dans le mode utilisateur et de réduire les blocages.

Dans le modèle client-serveur, une grande partie des programmes du système est déplacée dans une couche qui repose sur le micro-noyau et qui se présente sous l'aspect de processus clients et de processus serveurs. Tous les appels système ou presque sont désormais des processus clients. Le noyau gère la communication entre les processus clients et serveurs. Par exemple pour créer un fichier un processus client va envoyer un message au processus serveur qui se trouve être le système de gestion des fichiers.

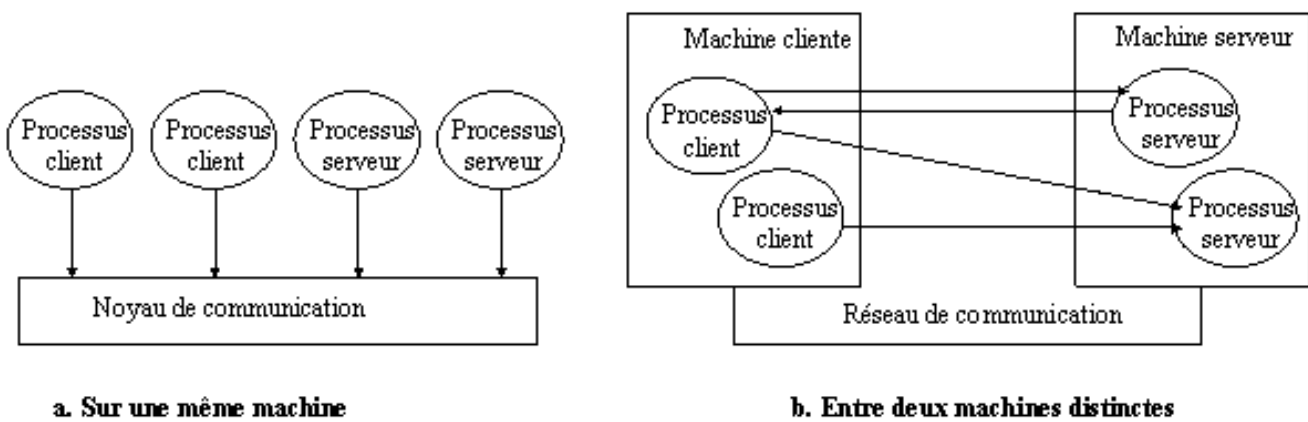


Fig. 17. Exemple de modèle client/serveur

Ce modèle rend les systèmes portables. Il a permis la naissance des systèmes distribués où le processus ne se soucie pas de savoir si sa requête (son service) sera traitée localement ou non. Autre point positif dans ce modèle, un dysfonctionnement dans l'un des modules n'affecte pas les autres. Les systèmes Windows NT et 2000 appliquent le modèle client-serveur tout en ayant un noyau structuré en couches. Le noyau (appelé Exécutif par Microsoft) contient un gestionnaire d'objet, un moniteur de références de sécurité, un gestionnaire de processus, une unité de gestion des appels de procédures locales, un gestionnaire de mémoire, un gestionnaire d'entrées/sorties et une autre partie située au niveau le plus bas et appelée noyau par Microsoft.

Certaines versions actuelles de Unix (compatible POSIX et X/OPEN) ont une architecture en micro-noyau qui implémente les fonctions de base, basée sur le micro-noyau Mach.

Si l'on veut résumer schématiquement l'évolution des différents systèmes du point de vue de leur architecture de conception, nous pouvons représenter le modèle monolithique comme un gros bloc, le modèle à couches comme un bloc décomposé en sous blocs. Quant au modèle client serveur, il est obtenu en retournant de 90 ° le bloc à couches et en séparant les appels systèmes du reste du système. Ils peuvent ainsi être placés sur des machines différentes, la communication se faisant à travers le noyau. Ce dernier peut avoir une structure monolithique ou à couches.

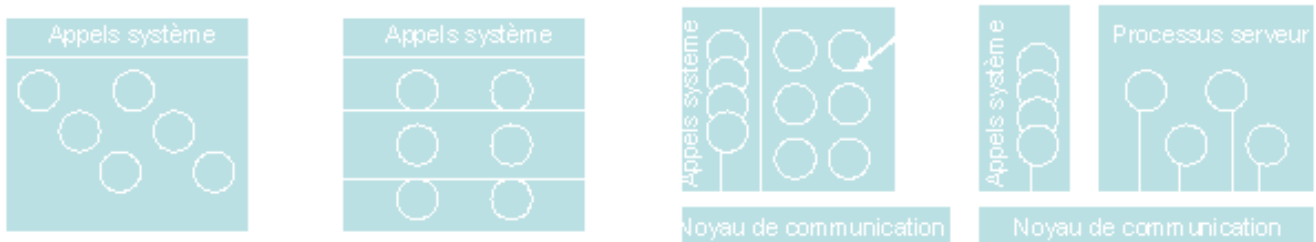


Figure 18 : évolution des systèmes d'exploitation



TD système d'exploitation N°2

1. Sur un système doté de n UC, quel est le nombre maximum de processus pouvant se trouver dans les états prêt, exécution et bloqué ?
2. Sur un système doté de n UC, quel est le nombre minimum de processus pouvant se trouver dans les états prêt, exécution et bloqué ?
3. Quel est le principal avantage de la multiprogrammation ?
4. Quel est le principal inconvénient qui découle d'un abus de la multiprogrammation ?
5. Le pourcentage d'attente des E/S, w , d'un processus est le pourcentage de temps pendant le quel ce processus attend l'achèvement des E/S, lors de l'exécution dans un environnement de monoprogrammation. Sur un système qui a recours à l'ordonnancement à tourniquet avec n processus, tous ayant le même pourcentage d'attente des E/S, quel est le pourcentage de temps d'inactivité de l'UC en termes de w .
6. Pour chacune des transitions suivantes entre les états des processus, indiquez si la transition est possible. Si c'est le cas, donnez un exemple d'un élément qui pourrait en être à l'origine.
 - a. En exécution – prêt
 - b. En exécution – bloqué
 - c. En exécution – permuté-bloqué
 - d. Bloqué – En exécution
 - e. En exécution – terminé
7. soit un système avec n processus, combien existe-t-il de manières d'ordonner ces processus.
8. sur un système recourant à l'ordonnancement à tourniquet, quelle serait la conséquence de l'introduction d'un même processus à deux reprises dans la liste des processus
9. avec les processus du tableau ci-dessous dessinez un schéma illustrant leurs exécutions à l'aide de :
 - f. l'algorithme FCFS

g. #9; l'algorithme SJF

h. l'algorithme SRT

i. l'algorithme à tourniquet (quantum = 2)

j. l'algorithme à tourniquet (quantum = 1)

<i>Processus</i>	<i>Temps d'arrivée</i>	<i>Temps de traitement</i>
<i>A</i>	<i>0,000</i>	<i>3</i>
<i>B</i>	<i>1,001</i>	<i>6</i>
<i>C</i>	<i>4,001</i>	<i>4</i>
<i>D</i>	<i>6,001</i>	<i>2</i>

10. pour les processus de l'exercice 9, quel est le temps moyen de rotation (arrondi au centième) si l'on utilise

k. #9; l'algorithme FCFS

l. #9; l'algorithme SJF

m. l'algorithme SRT

n. l'algorithme à tourniquet (quantum = 2)

o. l'algorithme à tourniquet (quantum = 1)

11. pour les processus de l'exercice 9, quel est le temps d'attente de chaque processus (arrondi au centième) si l'on utilise

p. #9; l'algorithme FCFS

q. #9; l'algorithme SJF

r. l'algorithme SRT

s. l'algorithme à tourniquet (quantum = 2)

t. l'algorithme à tourniquet (quantum = 1)

12. pour les processus de l'exercice 9, quel est le débit si l'on utilise

u. #9; l'algorithme FCFS

v. #9; l'algorithme SJF

w. l'algorithme SRT

x. l'algorithme à tourniquet (quantum = 2)

y. l'algorithme à tourniquet (quantum = 1)

13. avec les processus du tableau ci-dessous dessinez un schéma illustrant leurs exécutions utilisant l'ordonnancement de priorité. Un nombre de priorité élevé correspond à une priorité plus importante :

z. préemptif

aa. Non préemptif

<i>Processus</i>	<i>Date d'arrivée</i>	<i>Cycle</i>	<i>Priorité</i>
<i>A</i>	<i>0,000</i>	<i>4</i>	<i>3</i>
<i>B</i>	<i>1,001</i>	<i>3</i>	<i>4</i>
<i>C</i>	<i>2,001</i>	<i>3</i>	<i>6</i>

D

3,001

5

5

14. pour les processus de l'exercice 13, quel est le temps de rotation (arrondi au centième) de chaque processus si l'on utilise

bb. préemptif

cc. Non préemptif

15. pour les processus de l'exercice 13, quel est le débit moyen ?

16. Simulez l'exécution, avec un ordonnancement "le plus court d'abord", les 5 travaux (notés de A à E) qui ont des temps d'exécution respectifs de 2, 4, 1, 1 et 1 unité(s) et qui sont soumis aux instants 0, 0, 3, 3 et 3. Calculez l'attente moyenne dans ce contexte, puis dans un contexte où tous les travaux sont soumis en même temps.

17. Cinq travaux, notés de A à E, demandent à être exécutés au même instant. Leurs temps d'exécution respectifs sont estimés à 10, 6, 2, 4 et 8 minutes. Leurs priorités, déterminées de manière externe, sont respectivement 3, 5, 2, 1 et 4. Déterminez le temps moyen d'attente pour chacun des algorithmes d'ordonnancement suivants : "tourniquet", "avec priorités", "premier arrivé premier servi", "le plus court d'abord". Pour le premier mécanisme d'ordonnancement, on considère que le SE est multiprogrammé et que le temps processeur est équitablement réparti entre les différents travaux. Pour les autres, on suppose que chaque travail est exécuté jusqu'à ce qu'il se termine. Dans tous les cas, les travaux n'effectuent pas d'E/S.

18. sur un système utilisant des files multiniveaux à retour, un processus entièrement tributaire l'UC a besoin de 14 secondes pour s'exécuter. Si la première file utilise un quantum de temps à 2 et qu'à chaque niveau, le quantum de temps augmente de 5 unité. Combien de fois sera interrompu le travail et dans quelle file se trouvera-t-il lorsqu'il s'achèvera ?

19. sur un système Unix , un appel système permet d'augmenter ou de réduire la valeur de la priorité associée à un processus. Après l'augmentation de la valeur de la priorité des processus, on observe que le système semble s'exécuter plus lentement pourquoi ?

20. le choix d'un algorithme d'ordonnancement non préemptif est-il un choix judicieux pour

un système interactif. Expliquez brièvement pourquoi.

21. A quel degré les algorithmes suivants favorisent-ils les processus tributaires de l'UC ?

dd. FCFS

ee. SJF

ff. #9; SRT

gg. RR

hh. MFQ

La gestion des processus

Objectifs

1. Introduction aux processus
 2. Description d'un processus
 3. Ordonnement de processus
 4. La synchronisation de processus : le problème des accès concurrents
 5. Les interblocages
 6. La communication inter-processus
 7. TD3
-



3 - Introduction aux processus

Un processus est l'entité créée par le système d'exploitation pour l'exécution d'un programme.

Rappels sur la multiprogrammation

La multiprogrammation permet au processeur de se partager entre tous les processus actifs. Ainsi, quand le processeur atteint une instruction d'entrée-sortie (les périphériques ayant une vitesse de transfert et d'exécution bien inférieure à celle du processeur), il interrompt le programme en cours sitôt les paramètres de l'entrée/sortie transmis au périphérique et passe à l'exécution d'un autre programme. La multiprogrammation nécessite le partage du processeur entre les programmes mais également le partage de toutes les ressources disponibles de l'ordinateur (mémoires, périphériques...).

Un programme a une existence statique, il est stocké sur le disque puis chargé en mémoire afin d'être exécuté. Le processus en revanche a un contact direct avec le processeur en effet c'est l'entité exécutée par le processeur. Le processus est créé par le système d'exploitation ou l'utilisateur au moment où l'exécution du programme doit commencer, comme illustré à travers la figure n° 19. Une fois le processus terminé, il est supprimé par le système d'exploitation. Toutefois un seul programme peut nécessiter plusieurs processus pour son exécution.

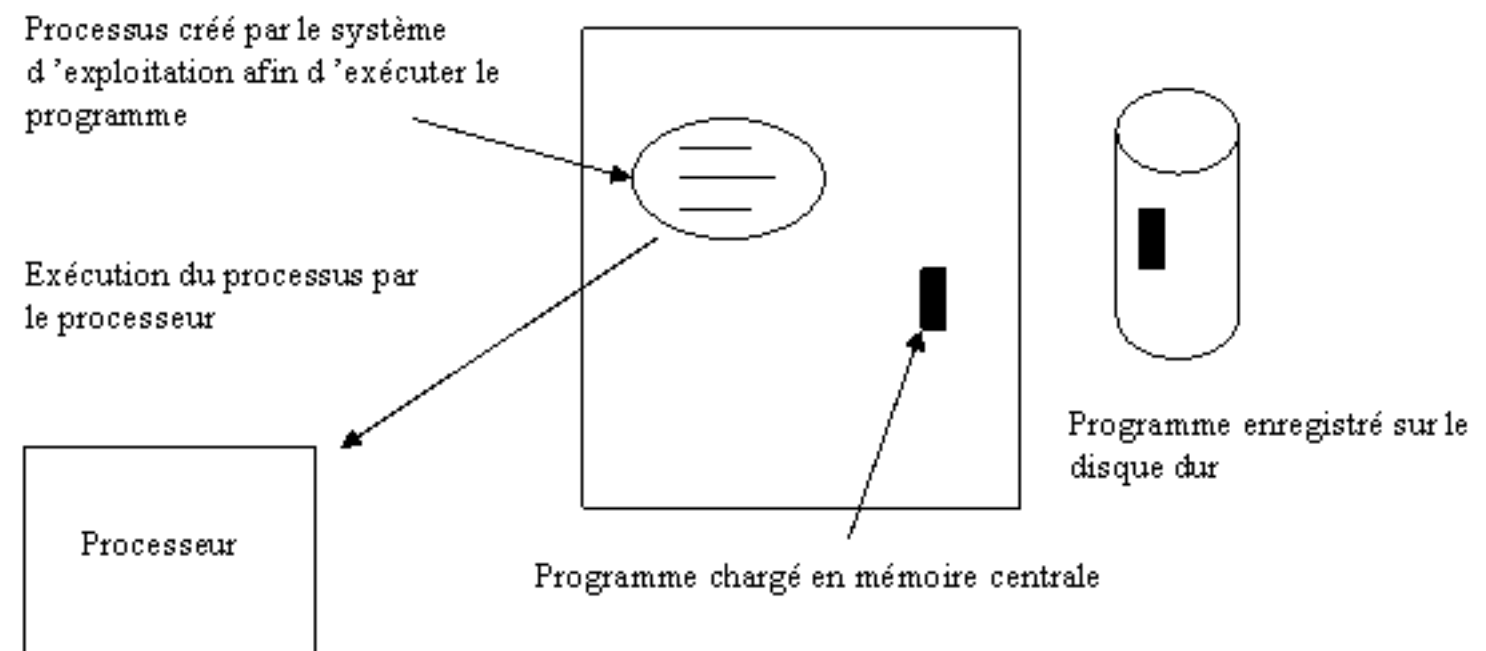


Fig. 19. Un programme et son processus

La gestion des processus consiste à décider quel processus s'exécute à un moment donné, à quel moment interrompre le processus, quel sera le suivant, et de quelles ressources il a besoin pour son exécution. Le système d'exploitation manipule deux types de processus : ceux du système et ceux des utilisateurs.

Les fonctionnalités du système d'exploitation en matière de gestion de processus sont :

§ La création, suppression et interruption de processus

§ L'ordonnancement des processus afin de décider d'un ordre d'exécution équitable entre les utilisateurs tout en privilégiant les processus du système.

§ La synchronisation entre les processus ainsi que la communication

§ La gestion des conflits d'accès aux ressources partagées

§ La protection des processus d'un utilisateur contre les actions d'un autre utilisateur.

2 - Description d'un processus

⇒ Sommaire de la section :

- ✓ Etats d'un processus
 - ✓ Structure de l'espace mémoire d'un processus
 - ✓ Structures de données pour la gestion des processus
-

2.1 Etats d'un processus

Un processus est une suite d'instructions, il a des données en entrée et en sortie. Du fait qu'il s'exécute et qu'il peut être interrompu, il est décrit par un état courant (en attente, suspendu, terminé.)

Un processus peut passer par plusieurs états avant de finir son exécution. On distingue les quatre états suivants :

§ En exécution : le processus est en cours d'exécution au niveau du processeur

§ prêt : le processus est prêt et pourra s'exécuter dès qu'il sera sélectionné par le système d'exploitation.

§ En attente : le processus est en attente ou bloqué car il attend des données ou la libération d'une ressource afin de poursuivre son exécution. Pendant ce temps le processeur est attribué à un autre processus.

§ Terminé : le processus a terminé son exécution.

Plusieurs processus peuvent se trouver dans l'état prêt ou en attente, le système d'exploitation les place alors dans une file d'attente (une par état) au niveau de laquelle ils vont attendre leur tour. Le système d'exploitation sélectionne toujours le processus en tête de la file et l'extrait. C'est le principe des files d'attente en algorithmique: on rejoint une file d'attente par la queue et on la quitte par la tête comme toute file d'attente au niveau d'un guichet.

Six transitions peuvent avoir lieu entre ces états, elles sont illustrées dans la figure n° 20 :

§ La transition 1 a lieu quand le processus ne peut plus poursuivre son exécution car il a besoin d'une ressource non disponible par exemple : il passe à l'état en attente.

§ La transition 2 a lieu quand le processus a terminé le temps imparti par le système d'exploitation pour son exécution. Un processus ne s'exécute pas forcément jusqu'à la fin car le système d'exploitation a d'autres processus à exécuter. Cette transition a également lieu si un processus plus urgent doit prendre la main (bien entendu parmi ces processus urgents, se

trouvent tous les processus du système d'exploitation). Nous verrons plus loin dans ce chapitre que quand le système ne distingue pas entre les processus en utilisant une priorité, ils les exécute de manière équitable en allouant à chacun la même durée appelée *quantum*. Nous rappelons que l'interruption d'un processus parce que sa tranche de temps s'est écoulée, a été introduite par la notion de temps partagé.

§ La transition 3 signale que le système d'exploitation a sélectionné un processus pour l'exécuter.

§ La transition 4 indique que le processus a fini son exécution.

§ La transition 5 a lieu sitôt que le processus n'a plus de raison d'être bloqué; par exemple les données deviennent disponibles. Le processus passe à l'état prêt.

§ La transition 6 a lieu quand l'événement attendu par le processus ne peut se réaliser. Il est donc inutile de faire patienter davantage ce processus, autant le terminer. Un cas d'interblocage en est une parfaite illustration.

Fig. 20. Les différents états d'un processus

Lorsqu'un processus est interrompu suite à l'exécution d'une instruction d'entrée-sortie, expiration du quantum ou exécution d'un processus plus urgent, on parle d'interruption logicielle.

2.1.1 Fonctionnement d'une interruption logicielle

Nous rappelons qu'une interruption est provoquée par un signal généré sur occurrence d'un événement qui peut être interne (lié au processus) ou externe et indépendant. Lorsqu'une interruption est générée, le processus en cours d'exécution est interrompu. Il quitte le processeur et un gestionnaire d'interruption est chargé dans les registres du processeur et s'exécute pour traiter l'interruption. Dans un premier temps il est nécessaire de connaître quelle interruption a eu lieu.

Une fois le signal de l'interruption reconnu, le gestionnaire d'interruption accède à une table appelée table des vecteurs d'interruptions et y recherche l'adresse du programme associé à exécuter. Ce programme est appelé routine d'interruption.

Une fois l'interruption traitée, le système charge un autre processus à partir de la file d'attente et l'exécute.

2.2 Structure de l'espace mémoire d'un processus

L'espace mémoire utilisé par un processus est divisé en plusieurs zones. On trouve en particulier le segment de code, le segment de données, la pile et le tas.

Le segment de code

Le segment de code représente le programme à exécuter. Il est toujours placé dans des zones fixes de la mémoire, c'est-à-dire au début de la zone disponible (puisque'il n'est pas amené à augmenter de taille durant l'exécution du processus).

Le segment de données

Au dessus du segment de code se trouve le segment de données. Ce segment est traditionnellement composé d'un segment de données initialisées (les variables globales et statiques) et d'un segment de données non initialisées qui est créé dynamiquement. Ce segment est amené à croître décroître suite à l'allocation dynamique de variables.

La pile

Pour stocker les données obtenues en cours d'exécution, le système utilise un segment pile. Ce segment est généralement situé en haut de l'espace d'adressage et il croît vers les adresses basses (pile renversée)

On utilise souvent les piles lors des appels de fonction. Le système empile le nom de la fonction et les paramètres à passer à la fonction puis les différentes variables locales de la fonction. Une fois la fonction terminée le système dépile toutes les données qu'il a placées pour l'appel de la fonction. Il a à nouveau accès aux données qu'il avait stockées avant l'appel de la fonction et peut poursuivre l'exécution du programme.

2.3 Structures de données pour la gestion des processus

Les systèmes d'exploitation manipulent deux structures de données principales pour gérer les processus créés sur une machine : la table des processus et le bloc de contexte d'un processus.

2.3.1 Contexte d'un processus

Le contexte d'un processus également appelé bloc de contexte ou de contrôle est une structure de données qui décrit un processus en cours d'exécution. Ce bloc est créé au même moment que le processus et il est mis à jour en grande partie lors de l'interruption du processus afin de pouvoir reprendre l'exécution du processus ultérieurement.

Le contexte d'un processus comporte les informations suivantes :

- § Le compteur ordinal : adresse de la prochaine instruction à exécuter par le processeur
- § Les contenus des registres généraux : ils contiennent les résultats calculés par le processus
- § Les registres qui décrivent l'espace qu'il occupe en mémoire centrale (l'adresse de début et de fin par exemple)
- § Le registre variable d'état qui indique l'état du processus
- § D'autres informations telles que la valeur de l'horloge, la priorité du processus,

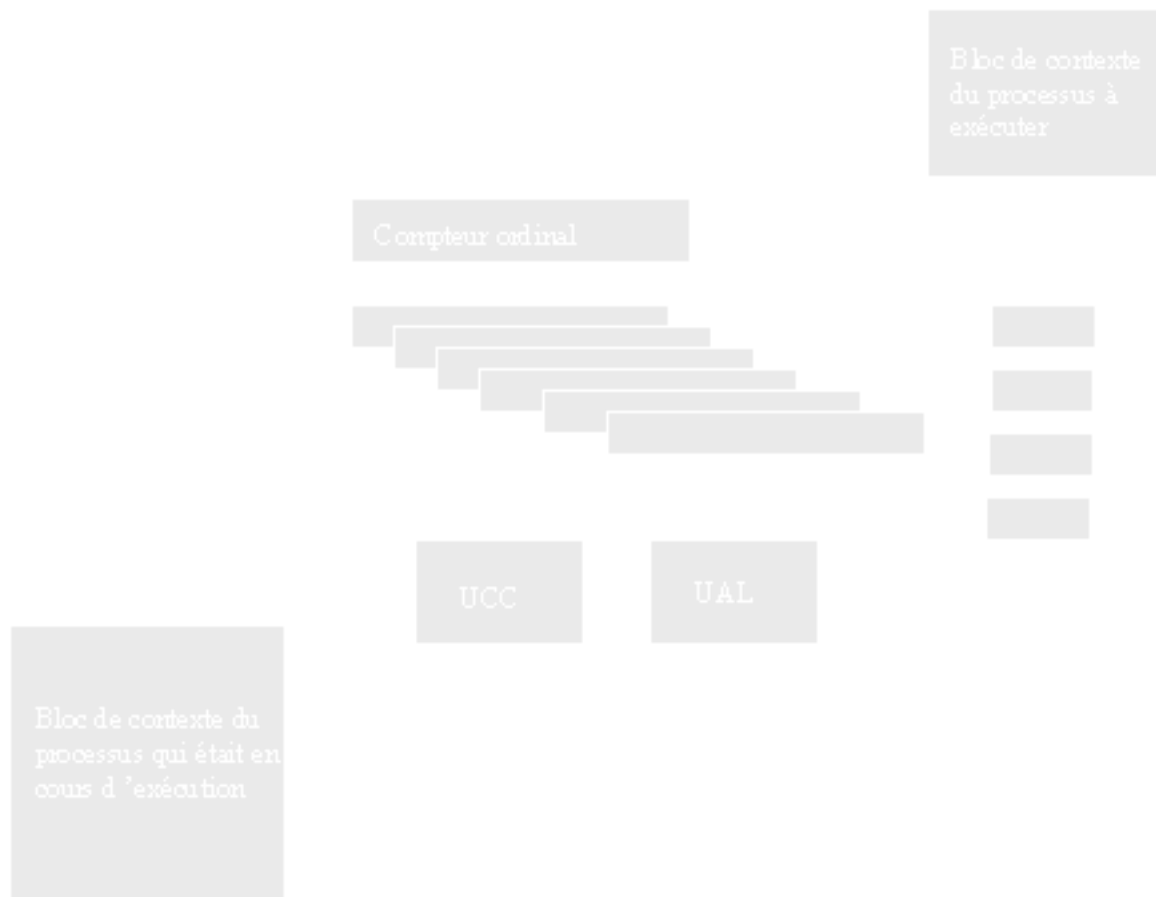


Fig. 21. Schéma d'une commutation de contexte

L'opération qui consiste à sauvegarder le contexte d'un processus et à copier le contexte d'un autre processus dans l'unité centrale s'appelle changement ou commutation de contexte (context switching en anglais). La durée d'une commutation de contexte varie d'un constructeur à un autre, elle reste toutefois très faible.

2.3.2 La table des processus

Le système d'exploitation manipule une structure de données qui lui permet de conserver d'autres informations sur les processus : la table des processus.

La table des processus contient toutes les informations indispensables au système d'exploitation pour assurer une gestion cohérente des processus. Elle est stockée dans l'espace mémoire du système d'exploitation, ce qui signifie que les processus ne peuvent pas y accéder.

Elle comporte une entrée par processus qui rassemble toutes les informations concernant un processus (même si le processus n'est pas en mémoire), aussi bien celles concernant son exécution au niveau du processeur mais également des informations sur les fichiers qu'il

manipule (nécessaires au SGF) ainsi que des informations sur son occupation mémoire. Quant aux informations sur l'occupation mémoire on y trouve par exemple des pointeurs sur les différents segments code, données et pile.

Pour limiter la taille de la table, de nombreux systèmes reportent toutes les informations qui ne sont pas absolument indispensables à la gestion des processus dans l'espace mémoire de ceux-ci, dans le bloc de contrôle.



3 - Ordonnement de processus

L'ordonnement de processus s'articule autour de 2 composantes :

§ La file d'attente des processus : elle regroupe les descripteurs des processus prêts. C'est à partir de cette file que l'ordonnanceur choisit le processus à exécuter.

§ L'algorithme d'ordonnement à appliquer : il organise les descripteurs de processus dans un ordre qui reflète la stratégie appliquée.

L'ordonnement des processus (on parle également d'ordonnement de l'unité centrale) permet d'optimiser le temps de réponse : moyenne des temps d'exécution. Cela est perceptible essentiellement au niveau des délais d'obtention des résultats. Si le système d'exploitation n'applique aucun algorithme d'ordonnement, on se retrouve avec un système monoprocesseur et les processus courts sont pénalisés par l'exécution de processus assez longs.

⇒ [Sommaire de la section :](#)

- ✓ [L'ordonnanceur](#)
 - ✓ [Algorithmes d'ordonnement](#)
 - ✓ [L'ordonnement FIFO : First In First Out](#)
 - ✓ [L'ordonnement circulaire : le tourniquet \(Round Robin\)](#)
 - ✓ [L'ordonnement avec priorités](#)
 - ✓ [L'ordonnement selon le plus court d'abord \(SJF Shortest Job first\)](#)
 - ✓ [Simulez votre exemple](#)
-

3.1 L'ordonnanceur

Le système d'exploitation applique un programme pour gérer l'ordre d'exécution des processus, appelé *scheduler*, *dispatcher* ou *ordonnanceur*. L'ordonnanceur dispose de plusieurs algorithmes d'ordonnancement qui correspondent à différentes politiques d'organisation des processus : équitable, avec priorité, plus court d'abord. Il agit sur la file des processus prêts et l'organise en appliquant un algorithme donné. Il faut souligner à ce stade que l'ordonnanceur ne change pas de politique d'ordonnancement. Chaque système d'exploitation applique un algorithme choisi par les concepteurs.

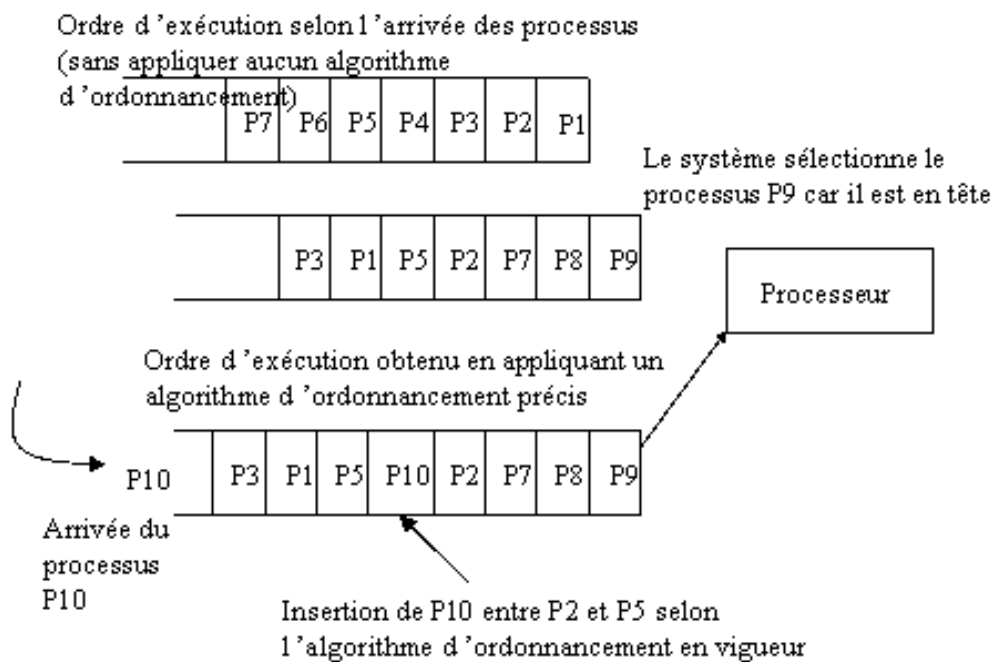


Fig. 22. Le fonctionnement de l'ordonnanceur

L'ordonnanceur est sollicité à la fin de l'exécution de chaque processus. Il s'exécute lui-même et sélectionne un processus pour l'exécution. Généralement il sélectionne celui qui est en tête de la file car les processus sont déjà dans l'ordre. En effet, à chaque création de processus, le processus créé doit rejoindre la file des processus prêts et c'est lors de son insertion que l'ordonnanceur applique son algorithme.

3.2 Algorithmes d'ordonnancement

Actuellement, on distingue de nombreuses approches pour l'ordonnancement, les plus connues sont : l'ordonnancement circulaire (tourniquet), à priorité, plus court d'abord. Il existe par ailleurs des algorithmes spécifiques.

Soient les processus suivants décrits par une durée d'exécution estimée en millisecondes.

Processus	Durée d'exécution
P1	3
P2	5
P3	2
P4	6
P5	4

Table 1. 5 processus et leurs temps d'exécution

Pour chacun des algorithmes suivants : FIFO, tourniquet, priorité, nous calculons le temps de réponse et le temps d'attente. Le temps de réponse est la moyenne des dates de fin d'exécution. Le temps d'attente est la moyenne des délais d'attente pour commencer une exécution.

3.3 L'ordonnancement FIFO : First In First Out

La politique FIFO agit de la manière suivante : le premier processus arrivé est le premier servi. Il n'y a aucun effort à faire de la part de l'ordonnanceur, il suffit d'insérer le processus à la fin de la file.

La figure n° 23 illustre l'ordonnancement des 5 processus de la table 1 avec FIFO :

Fig. 23. Ordonnancement avec FIFO

Temps de réponse $TR = (3+8+10+16+20)/5 = 57/5 = 11.4$ ms

Temps d'attente $TA = (3+8+10+16)/5 = 37/5 = 7.4$ ms

Appliquer la politique FIFO revient à faire de la monoprogrammation en quelque sorte, pourtant cette politique est souvent retenue du fait de sa simplicité et de sa faible consommation en temps processeur.

---> Visualisez un exemple d'ordonnancement avec l'algorithme [FIFO](#)

3.4 L'ordonnancement circulaire : le tourniquet (Round Robin)

Le premier processus de la file est exécuté durant une période précise fixée par le système, appelée quantum. Une fois le processeur alloué, ce processus s'exécute jusqu'à ce que sa tranche de temps soit écoulée ou qu'il se bloque. Il est alors interrompu et mis à la fin de la file d'attente (fonctionnement circulaire). L'ordonnanceur retire le processus suivant et l'exécute lui aussi durant la même période. Les processus sont placés dans la file selon leur ordre d'arrivée.

Le tourniquet est une technique des systèmes à temps partagé puisque chaque processus s'exécute pendant la même durée à chaque fois. Le quantum est de l'ordre de quelques dizaines de millisecondes

Fig. 24. Ordonnancement circulaire

$$\text{Temps de réponse } TR = (6+11+17+18+20)/5 = 72/5 = 14.4 \text{ ms}$$

$$\text{Temps d'attente } TA = ((8)+(2+7+4)+(4)+(6+5+3)+(8+5))/5 = 52/5 = 10.4 \text{ ms}$$

3.5 L'ordonnancement avec priorités

Cette stratégie permet de distinguer entre les processus sur la base de leurs priorités. Si l'utilisateur ou le système d'exploitation exprime une priorité pour les processus (une priorité est un nombre) il faut que le processus de plus haute priorité soit exécuté le premier. L'algorithme d'ordonnancement avec priorité classe les processus avec un ordre décroissant de leur priorité.

Chaque processus s'exécute jusqu'à la fin sinon la priorité n'exprime aucune urgence. A signaler que les processus du système d'exploitation ont la priorité la plus haute.

Si les processus ci-dessus présentés avaient respectivement les priorités 1 3 2 5 4, ils seraient exécutés selon l'ordre suivant :

Fig. 25. Ordonnancement basé sur la priorité

Temps de réponse $TR = (6+10+15+17+20)/5 = 68/5 = 13.6 \text{ ms}$

Temps d'attente $TA = (6+10+15+17)/5 = 48/5 = 9.6 \text{ ms}$

Résolution du problème de la famine

Lorsque plusieurs processus ont la même priorité, le système d'exploitation affecte une file d'attente pour chaque priorité. Au sein d'une même file, il n'est pas logique d'exécuter un processus en entier, autrement il serait prioritaire par rapport aux autres de cette même file. Souvent l'algorithme du tourniquet est appliqué au niveau de chaque file.

Le problème de la famine peut se poser s'il existe plusieurs processus de haute priorité. Ils monopoliseront l'unité centrale et la file des processus de faible priorité ne sera servie que très rarement.

Parmi les solutions qui existent, la plupart décrémentent la priorité du processus où après

exécution d'un quantum ou à chaque impulsion d'horloge. Chaque processus est exécuté durant un quantum, ensuite sa priorité est décrétementée de 1, et il rejoint la file de priorité inférieure par la queue.

P7

Fig. 26. Ordonnement avec files multiples

3.6 L'ordonnancement selon le plus court d'abord (SJF Shortest Job First)

Cette politique compare les processus sur la base de leur durée d'exécution et exécute le plus court en premier. Le SJF est appliqué dans les systèmes qui régissent les processus industriels où une chaîne d'exécution doit se terminer le plus rapidement possible.

Fig. 27. Ordonnancement selon le plus court d'abord

Temps de réponse TR = $(2+5+9+14+20)/5 = 50/5 = 10$ millisecondes

Temps d'attente TA = $(2+5+9+14)/5 = 30/5 = 6$ millisecondes

---> Visualisez un exemple d'ordonnancement avec l'algorithme [SJF](#)

3.7 Simulez votre exemple selon le plus court d'abord (SJF Shortest Job First)

---> Vous pouvez simuler votre propre exemple d'ordonnement avec l'algorithme de votre choix [ICI](#)



4 - La synchronisation de processus : le problème des accès concurrents

Un accès concurrent est une situation dans laquelle deux processus partagent une ressource (logicielle ou matérielle). Si la ressource est accessible en mode partagé, généralement c'est le cas des accès en lecture, les deux processus s'exécutent sans aucun problème. Là où les choses se corsent c'est quand il y a un accès exclusif à la ressource et c'est souvent le cas des accès en écriture. Voyons à présent une situation d'accès concurrent afin d'explicitier le problème qui se pose :

Soit deux processus P1 et P2, chacun ayant un fichier à imprimer. Soit une imprimante gérée par un processus appelé démon qui vérifie périodiquement s'il y a des fichiers à imprimer.

Le démon inspecte une variable de type tableau qui contient les fichiers à imprimer et 2 variables entières *prochain* et *libre*. La variable *prochain* indique le numéro du prochain fichier dans le répertoire et la variable *libre* celle du premier emplacement libre où un processus peut déposer un fichier.

P1 lit la variable *libre* et y trouve la valeur 5 et est ensuite interrompu. P2 prend la main et lit la valeur de libre qui est toujours à 5 et met son fichier à cet emplacement ensuite il incrémente *libre* qui passe à 6.

Quand P1 reprend la main, dans ses registres il trouve la valeur de libre à 5, il met son fichier à l'emplacement 5 et écrase celui de P2 qui ne sera jamais imprimé.

L'origine du problème vient de l'interruption de P1 et du fait qu'il ne réexécute pas l'instruction au niveau de laquelle il a été interrompu, en effet une instruction interrompue est reprise et continuée sur la base des valeurs des registres sauvegardés dans le bloc de contexte du processus.

On pourrait penser qu'une solution à ce problème est d'empêcher les interruptions d'avoir lieu. C'est d'ailleurs le cas et nous l'étudierons plus loin dans ce chapitre lorsque nous présenterons les différentes solutions à ce problème.

⇒ Sommaire de la section :

- ✓ Notion de section critique et d'exclusion mutuelle
 - ✓ Solutions logicielles au problème des accès concurrents
 - ✓ Solutions matérielles au problème des accès concurrents
-

4.1 Notion de section critique et d'exclusion mutuelle

Une section critique est un ensemble d'instructions comportant un conflit d'accès. Ces instructions utilisent des variables ou des ressources partagées par d'autres processus.

On rentre en section critique, par une section d'entrée qui permet de mettre en ouvre une condition et on la quitte par une section de sortie. Si cette section est exécutée en entier, les problèmes sont résolus. Or, un processus peut être interrompu à tout moment notamment au milieu de sa section critique.

Si une ressource a été accédée par un premier processus et qu'il est interrompu, aucun autre processus ne peut y accéder tant qu'elle n'a pas été libérée par le premier processus. Il faut donc attendre que le processus reprenne son exécution et qu'il libère la ressource et quitte la section critique.

On parle d'exclusion mutuelle quand un seul processus à la fois a le droit de rentrer en section critique (la condition d'entrée ne permet qu'à un seul processus de passer).

La figure n° 28 qui suit, schématise l'exécution de trois processus par l'algorithme du tourniquet en tenant compte des différentes sections critiques. La variable x, y et z sont partagées par les deux premiers processus. La variable x est utilisée par le 3ème processus en lecture. C'est une section critique car elle ne doit pas être modifiée en même temps par les autres processus.

4.2 Solutions logicielles au problème des accès concurrents

les sémaphores et les moniteurs sont les deux principales solutions utilisées pour résoudre le problème des accès concurrents : les moniteurs sont relativement difficiles à manipuler pour un débutant.

4.2.1 Les Sémaphores

Un sémaphore est un objet du langage de programmation ou du système, de type entier. Sa valeur initiale est le nombre de processus autorisés à rentrer en section critique. On distingue des sémaphores binaires qui peuvent prendre les valeurs 0 ou 1 et les sémaphores n-aires. Les sémaphores binaires sont utilisés pour réaliser de l'exclusion mutuelle. Quant aux sémaphores n-aires, ils ont pour rôle de spécifier un nombre maximal d'accès en lecture à une ressource.

Conjointement au sémaphore, deux primitives indivisibles P et V, mettent en œuvre les sections d'entrée et de sortie. Elles permettent de décrémenter et d'incrémenter le sémaphore. Leur origine provient des termes hollandais Proberen et Verhogen en effet l'inventeur des sémaphores, Mr E. W. Dijkstra, est hollandais. Certains emploient une métaphore pour ces primitives, ainsi P signifierait **Puis-je** et V **Vas-y**.

P(S) permet de prendre le sémaphore et équivaut à exécuter les instructions suivantes :

```

Si S > 0      Alors s = s - 1

                Sinon s'endormir

Finsi
  
```

Si le sémaphore est binaire, en l'initialisant à 1 on ne permet qu'une seule exécution de la section critique à la fois.

V(S) permet de libérer le sémaphore et un processus bloqué s'il y en a

```

Si un processus est bloqué sur S
  
```

Alors le libérer

Sinon $s = s + 1$

Finsi

Le fait de vérifier s'il existe des processus en attente du sémaphore avant d'incrémenter sa valeur permet de respecter l'ordre dans lequel les processus se sont bloqués au niveau de l'accès à cette section critique.

Un processus a typiquement les instructions suivantes :

faire

Instructions

P(S)

Section critique

V(S)

Instructions

Fin

Un processus bloqué est placé dans la file des processus en attente, une fois libéré il passe dans celle des prêts.

4.2.2 Le problème du producteur/consommateur résolu avec les sémaphores

Le producteur et le consommateur sont deux processus cycliques qui manipulent un tampon. Celui-ci consiste en une zone de mémoire organisée comme un ensemble de cases pouvant accueillir plusieurs objets à raison de un par case.

Producteur	Consommateur
...	...
produire(objet);	retirer(case, objet);
déposer(case, objet);	consommer(objet);
...	...

La solution à une case étant triviale, nous détaillons la solution à N cases. La gestion du tampon implique que:

§ si le tampon est vide, le consommateur ne peut rien retirer

§ si le tampon est plein, le producteur ne peut rien déposer

§ il faut empêcher que le dépôt et le retrait se chevauchent.

Sachant qu'il faut bloquer les processus tantôt sur un tampon vide, tantôt sur un tampon plein, ce problème nécessite deux sémaphores n-aires. On utilise ainsi, *pleins* et *vides* initialisés à 0 et n. *Pleins* indique le nombre de cases pleines et *vides* celui des cases vides.

Avant d'accéder au tampon pour y déposer ou retirer un objet, il faut également disposer d'un accès exclusif. Un troisième sémaphore *exmut* est utilisé pour l'accès au tampon. Les indices tête et queue sont manipulés respectivement lors du dépôt et du retrait d'objets. Ils sont protégés grâce au sémaphore d'exclusion mutuelle *exmut* et ne sont pas traités séparément pour des raisons de simplification.

Le programme écrit en langage C suivant, présente une solution à ce problème. Les appels à P et V sont remplacés par les appels système *Down* et *Up* en langage C :

```
#define N 20
typedef int semaphore ;
semaphore exmut = 1 ;
semaphore vides = N ;
```

semaphore pleins = 0 ;

```
void producteur(void)
{
    int objet ;

    while (true)
    {
        produire_objet(&objet) ;

        down(&vides) ;          /* décrémenter le nombre de
                                cases vides */

        down(&exmut) ;         /* pour ne pas accéder au
                                tampon en même temps */

        déposer_objet(objet) ;

        up(&exmut) ;           /* libérer l'accès au tampon */

        up(&pleins) ;          /* incrémenter le nombre de
                                cases pleines */

    }
}
```



```
void consommateur(void)
{
    int objet ;

    while (true)
    {
        down(&pleins) ;           /* décrémenter le nombre de
                                   cases pleines */

        down(&exmut) ;           /* pour ne pas accéder au
                                   tampon en même temps */

        retirer_objet(&objet) ;

        up(&exmut) ;             /* libérer le sémaphore */

        up(&vides) ;             /* incrémente le nombre de
                                   cases vides */

        Consommer_objet(objet) ;

    }
}
```

L'ordre des P(S) dans chacun des programmes doit être réalisé avec prudence. Prenons le cas du producteur, il peut se bloquer avec une mémoire tampon pleine. L'instruction `down (&vides)` sera bloquante puisque *vides* vaut 0. Si le processus prend le sémaphore *exmut* avant

de se bloquer, le consommateur ne pourra pas accéder au tampon et le nombre de *vides* ne pourra jamais être incrémenté.

Les sémaphores *vides* et *pleins* assurent que le producteur s'arrête quand la zone tampon est pleine et que le consommateur s'arrête quand elle est vide : ce n'est pas de l'exclusion mutuelle. Celle-ci est réalisée avec le sémaphore *exmut*.

Une dernière remarque pour expliquer pourquoi l'objet est produit et consommé en dehors de la section critique. Durant toute la période où un sémaphore est pris, on pénalise les autres processus, il vaut mieux écourter la section critique en retirant les instructions qui ne posent pas de problème d'accès concurrent.

La programmation avec les sémaphores est risquée. Il suffit d'inverser deux appels consécutifs de P pour bloquer le programme à jamais. On distingue d'autres solutions telles que les moniteurs.

4.2.3 Les moniteurs

Un moniteur est un ensemble de procédures, de variables et de structures de données regroupées dans un module spécial et gérées par le compilateur (Certains compilateurs ne gèrent pas les moniteurs. Ils sont ainsi incapables de distinguer les procédures du moniteur des autres procédures). Désormais, il suffit au programmeur de reporter la section critique dans des procédures du moniteur. Une fois les procédures définies, elles pourront être appelées par les processus sans qu'il n'ait connaissance ni accès à la structure interne des procédures. Le compilateur vérifie avant d'exécuter le processus si un autre processus est actif dans le moniteur. Etant donné que cette vérification est reportée sur le compilateur, on écarte le risque d'erreur engendré par l'inversion des P et V pour les sémaphores.

Les moniteurs utilisent deux primitives Wait et Signal afin de bloquer les processus sur la réalisation d'une condition et pouvoir ensuite les réveiller.

Le programme ci-dessous exprime une solution à l'aide des moniteurs à un problème dans lequel deux processus s'exécutent en parallèle. Le premier processus dépose des objets dans une structure de données de capacité N. Le second retire de cette structure.

```
Monitor nom_moniteur
```

```
Condition plein, vide ;
```

```
Int cpt ;
```

```
Void mettre()
```

```
{
```

```
    If cpt == N wait(plein) ; /* la limite de dépôt est de N */
```

```
    Mettre_objet() ;
```

```
    Cpt++ ;
```

```
}
```

```
Void retirer()
```

```
{
```

```
    if cpt == 0 wait(vide) ; /* ne pas retirer s'il n'y pas d'objet */
```

```
    retirer_objet() ;
```

```
    cpt- -;
```

```
    if (cpt == N-1) signal(plein)
```

```
}
```

```
cpt = 0;
```

```
end monitor;
```

```
void .....()  
{  
    .....  
    nom_moniteur.mettre;  
    .....  
}
```

```
void .....()  
{  
    .....  
    nom_moniteur.retirer;  
    .....  
}
```

4.3 Solutions matérielles au problème des accès concurrents

On distingue principalement deux solutions : la première consiste à désarmer les interruptions pendant toute la section critique et de les réarmer à la fin de celle-ci. La seconde est une instruction élémentaire TS qui permet de lire et d'écrire le contenu d'un mot mémoire de manière indivisible (TS signifie : lire en vue de tester et positionner une variable)

La solution basée sur les interruptions est risquée car un programmeur peut oublier de réarmer les interruptions ce qui bloquerait le système. C'est une méthode intéressante mais à n'appliquer qu'entre processus systèmes.



5 - Les interblocages

On parle d'interblocage pour un ensemble de processus quand chacun d'eux est dans l'attente d'un évènement de la part d'un autre processus. On imagine aisément une situation avec les sémaphores si les opérations P et V ne sont pas rapprochées ou sont inversés. Les interblocages sont fréquents quand les processus manipulent plusieurs ressources. En cas d'interblocage, les ressources ne seront jamais libérées, et aucun traitement par d'autres processus sur les dites ressources ne pourra être entrepris.

Les interblocages sont peu fréquents dans un PC pour cela certains systèmes ne proposent pas de solutions et supposent que l'utilisateur redémarrera sa machine afin de mettre fin à l'interblocage si jamais il a lieu. On distingue deux approches pour résoudre le problème :

1. L'approche préventive : elle empêche les interblocages d'avoir lieu ainsi tous les efforts sont conjugués afin que les processus ne se retrouvent pas interbloqués.
2. L'approche de détection et guérison : plutôt que de consacrer tous les efforts à la prévention, le système laisse les interblocages avoir lieu, ensuite il les détecte et il corrige.

Une dernière approche dite de l'autruche consiste à ignorer l'interblocage et à laisser l'utilisateur redémarrer sa machine. Elle est valable pour un PC où ce dernier est le seul à utiliser sa machine.

6 - La communication inter-processus

Les processus nécessitent de communiquer entre eux afin d'échanger des données ou des résultats. Un programmeur peut décomposer son programme en traitements plus ou moins indépendants et créer un processus afin d'exécuter chaque traitement. Au fur et à mesure de l'exécution, les processus auront besoin de communiquer afin de s'échanger leurs résultats. On recense essentiellement deux techniques de communication :

§ l'échange de messages

§ le partage de mémoire ou la communication par mémoire partagée

⇒ Sommaire de la section :

 [Communication par envoi de messages](#)

 [Communication par mémoire partagée](#)

6.1 Communication par envoi de messages

Les processus échangent explicitement des messages entre eux qu'ils transmettent à travers un système de communication. Si les processus se trouvent sur un même ordinateur, c'est le système d'exploitation qui se charge de faire parvenir un message au processus destinataire. En revanche, si nous sommes dans un environnement distribué, le message est transmis par le système d'exploitation à un protocole de communication qui se charge de le faire transiter par un réseau physique. Arrivé à destination, le message est transmis au système d'exploitation qui le délivre au processus destinataire.

L'envoi et la réception de messages se font au moyen d'appels à des primitives telles que `send` et `receive`. Celles-ci peuvent être utilisées pour mettre en œuvre une liaison directe qui n'existe qu'entre le processus émetteur et le processus récepteur ou pour faire communiquer plusieurs processus grâce à l'utilisation d'une boîte aux lettres.

Lorsqu'il s'agit d'une liaison directe le programmeur précise le processus auquel il veut transmettre dans les paramètres du `send`.

```
Send(proc1,&message,....)
```

En revanche, le recours à la boîte aux lettres permet la communication entre processus dont on ignore l'identifiant. Par ailleurs ceci permet à plusieurs processus de communiquer.

Lorsqu'il s'agit d'utiliser une boîte aux lettres, le `send` contient l'identifiant de la boîte aux lettres. Tout processus ayant vent de cet identifiant peut y placer et en retirer des messages. La boîte aux lettres peut être vue comme une file d'attente de messages. Ils y sont stockés dans l'ordre de leur arrivée et délivrés en respectant le même ordre.

```
Send(bal, &message,....)
```

```
Receive(bal, &message,....)
```



6.2 Communication par mémoire partagée

Dans un programme donné, lorsque l'on souhaite modifier une variable V par plusieurs procédures, celle-ci est passée en paramètres par adresse à chaque procédure : on peut alors dire que les procédures communiquent entre elles au moyen de la variable V .

Transposons cet exemple pour deux processus, la même variable V ne peut être modifiée par les deux processus pour une raison de visibilité car généralement chaque processus s'exécute dans un espace d'adressage séparé. Les données d'un processus P_1 sont modifiées à l'intérieur de son segment de données et donc inaccessibles à un autre processus P_2 . Le programmeur doit recourir à une technique explicite de communication entre les processus afin d'échanger entre eux les valeurs de la variable V .

La communication par mémoire partagée permet aux processus de partager une zone mémoire dans laquelle ils placent les variables en commun. Le système d'exploitation leur fournit de la sorte une zone mémoire à laquelle ils vont se rattacher pour partager ces variables.



TD système d'exploitation N°3

Exercice 1

Simulez l'exécution de 2 processus P1 et P2 sur une machine dont le gestionnaire de processus possède les caractéristiques suivantes :

les priorités des processus varient dynamiquement de 1 à 5, où 5 dénote la plus forte priorité : Quand un processus ne consomme pas entièrement son quantum, sa priorité est décrétementée ;

les quanta varient dynamiquement de 1 à 5 unités de temps : Quand un processus ne consomme pas entièrement son quantum, son quantum est incrémenté ;

les priorités varient de manière inversement proportionnelle aux quanta : Quand la priorité d'un processus est décrétementée, son quantum est incrémenté ;

le temps de commutation de contexte est de 1 unité de temps ;

le temps d'exécution de l'algorithme d'ordonnancement est négligeable.

Les processus ont les caractéristiques suivantes :

Ils sont soumis en même temps ;

Les quanta initiaux sont de 2 pour P1 et de 5 pour P2 ;

Les priorités respectives initiales sont 2 et 3 ;

Chacun des deux processus requiert 16 unités de temps, E/S non incluses ;

P1 lance une E/S au 4ème et 10ème instants de son exécution, qui durent toutes les deux 1 unité de temps ;

P2 lance une E/S au 3ème et 11ème instants, lesquelles durent respectivement 7 et 3 unités de temps.

Nota bene : L'ordonnanceur décrit n'est pas un "bon" ordonnanceur. Il vaudrait mieux, pour se rapprocher des critères d'un bon ordonnanceur, que le quantum d'un processus n'ayant pas consommé son dernier quantum soit décrémenté et non pas incrémenté. De cette façon, les processus interactifs obtiendraient une forte priorité et un petit quantum tandis que les processus de type "traitement par lot" auraient une priorité faible et un quantum élevé.

Exercice 2

Considérer une variante de l'algorithme RR dans laquelle des entrées dans la queue de processus prêts est implémentées avec des pointeurs aux PCB.

Quel est l'effet de mettre deux pointeurs à un même processus dans la queue de processus prêts.

Quels sont les principaux avantages/ désavantages de ce type d'algorithmes.

Peut-on obtenir même résultat sans dupliquer les pointeurs.

Exercice 3

On s'intéresse ici à l'utilisation par cinq processus, P_1 , P_2 , P_3 , P_4 , P_5 , de quatre types de ressources, RA , RB , RC et RD . Le nombre maximal d'unités de chaque type de ressources est : 4 pour RA , 8 pour RB , 4 pour RC et 5 pour RD . On considère le système dans l'état représenté par les deux tableaux ci-dessous, qui indiquent respectivement le nombre de ressources de chaque type couramment allouées par chaque processus (a), et le nombre maximal de ressources de chaque type

utilisées par chaque processus (b).

(a) Ressources allouées (b) Besoins maximaux

	RA	RB	RC	RD			RA	RB	RC	RD
P_1	1	2	2	2		P_1	3	6	2	5
P_2	0	2	0	0		P_2	2	2	1	0
P_3	2	0	1	1		P_3	2	2	1	1

<i>P 4</i>	1	1	1	1		<i>P 4</i>	3	4	2	2
<i>P 5</i>	0	1	0	0		<i>P 5</i>	0	3	3	4

1. L'état courant est-il sûr (*safe*), "risqué" (*unsafe*) ou correspond-il à un interblocage ? Utilisez l'algorithme du banquier pour justifier votre réponse.

2. Le processus *P 5* fait une demande pour 1 ressource de type *R D* . Cette demande doit-elle être satisfaite ?

Exercice 4

On souhaite modifier la solution du problème des philosophes dîneurs de manière à pouvoir changer le nombre de philosophes en cours d'exécution (le menu demeure inchangé). Il vous est demandé d'écrire deux fonctions `Ajoute_philosophe()` et `Retire_philosophe()` qui gèrent les aspects de synchronisation de ce problème.

N.B.

- L'ajout ou le retrait d'un dîneur ne sont évidemment pas décidés par un autre

Philosophe mais par un superviseur externe, par exemple si on observe que les ressources sont sous utilisées ou sur-utilisées ;

- On supposera que les aspects sordides de gestion des ressources (en particulier la redéfinition du nombre de philosophes, `N`, et des macros `LEFT` et `RIGHT`) sont traités par des fonctions `Ajout()` et `Retrait()` qu'il ne vous est pas demandé d'écrire, mais que vos fonctions devraient

appeler ;

- Les ajouts/retraits se font à la position $N - 1$.

Le système de gestion des fichiers

Objectifs

1. Les supports de stockage
 2. Les tâches d'un SGF
 3. L'organisation des fichiers sur le disque
 4. La gestion de l'espace libre sur le disque
 5. La gestion des fichiers dans un environnement multi-utilisat
 6. La cohérence d'un SGF
 7. TD4
-



1 - Les supports de stockage

Cette section commence par rappeler la structure des supports magnétiques en exposant la structure des disques d'abord, les disquettes ensuite. Nous donnons ensuite, un aperçu sur le codage des données sur ces supports.

⇒ Sommaire de la section :

- ✓ La mécanique d'un disque dur
 - ✓ Structure d'un plateau
 - ✓ Le codage des données
 - ✓ Le formatage d'un support magnétique
 - ✓ Le partitionnement du disque
-

1.1 La mécanique d'un disque dur

Un disque dur classique (décrit par la figure n° 22) est constitué des composants suivants :

- Les plateaux : un plateau est le support destiné à accueillir les données. Les plateaux ont une forme circulaire et ils sont empilés les uns au dessus des autres. Le diamètre d'un plateau indique la taille du disque et il est généralement de 3 pouces $\frac{1}{2}$ ou 5 pouces $\frac{1}{4}$ (pour les PC).
- Les têtes de lecture/écriture : Sur chaque plateau, on peut écrire les données sur les deux faces. Il faut ainsi associer une tête de lecture/écriture à chaque face. Les têtes sont reliées à un bras qui permet de les déplacer simultanément. Chaque face du plateau est constituée d'un ensemble de pistes. La tête se déplace ainsi d'une piste à l'autre.
- Un moteur rotatif : il permet de faire tourner les plateaux à une vitesse autour de 7200 tr/min (environ 200 km/h) voire davantage. Quand la machine est mise sous tension, le moteur entreprend un mouvement de rotation permanent et il se crée une dépression sous chaque tête l'obligeant à s'éloigner du plateau de quelques millièmes de centimètre. On parle de crash disque si la tête atterrit sur le disque en cours de rotation.
- Des positionneurs de têtes : ils permettent de positionner la tête de lecture/écriture à l'emplacement souhaité.

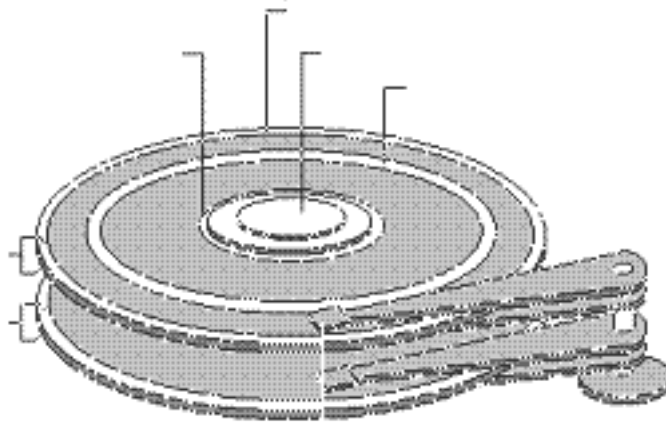


Fig. 29. Structure d'un disque dur

L'ensemble de ces composants est enfermé dans un boîtier étanche à l'air et à la poussière. Tout comme le microprocesseur, les disques durs sont assemblés dans des pièces spéciales. On distingue aujourd'hui deux types de disques: les disques SCSI et IDE auxquels on associe un contrôleur SCSI ou IDE. Les disques SCSI sont plus robustes et plus performants mais également plus chers.

1.2 Structure d'un plateau

La surface de chaque plateau est divisée en pistes concentriques numérotées à partir de zéro en commençant à partir de la piste la plus à l'extérieur. Chaque piste est divisée en secteurs (parts) dont le nombre varie d'un disque à l'autre.

Quand le bras se déplace, à un instant donné, toutes les têtes sont positionnées sur le même numéro de piste. Il est plus judicieux d'écrire un fichier sur plusieurs plateaux (puisque les têtes sont positionnées) que de remplir entièrement un plateau avant de passer à un autre. On définit ainsi la notion de cylindre qui est l'ensemble des pistes superposées. Le cylindre n°2 concerne toutes les pistes n° 2 appartenant à tous les plateaux. Pour adresser un secteur, il faut spécifier le plateau, la face, le cylindre et le secteur.

1.3 Le codage des données

Chaque support de stockage magnétique, est recouvert d'une fine couche magnétique destinée au stockage des informations. Ce sont les variations d'un champ magnétique qui permettent de lire et d'écrire sur le support. Selon les principes de l'électromagnétisme, lorsque un courant électrique traverse un corps conducteur, il génère un champ magnétique et vice-versa, lorsque un corps conducteur est traversé par un champ magnétique variant, il induit un courant électrique.

Pour l'écriture, on envoie un courant électrique à travers la tête de lecture/écriture (plus précisément dans l'électroaimant qu'elle porte), qui va produire un champ magnétique dont le résultat sera la modification de la surface du plateau. La lecture est réalisée par le phénomène inverse, à savoir qu'un courant induit est obtenu à la détection d'une variation de champ magnétique.

1.4 Le formatage d'un support magnétique

Il existe deux types de formatage : un formatage de bas niveau opéré à l'usine et un formatage de haut niveau que l'utilisateur peut effectuer autant de fois qu'il le souhaite.

1.4.1 Le formatage de bas niveau

Il est réalisé en usine et il consiste à tracer les pistes et les secteurs sur les différents plateaux. Lorsqu'un disque est formaté, des informations supplémentaires sont créées au niveau de chaque secteur. Elles permettent de l'identifier et de le décrire. Une piste n'est pas divisée en secteurs de 512 octets, mais en zones de plus grande taille incluant le secteur de données. La figure ci-dessous illustre le format d'un secteur :

Fig. 30. Format d'un secteur

1.4.2 Le formatage de haut niveau

Le formatage de haut niveau consiste à organiser les pistes et les secteurs d'une manière compréhensible par le système d'exploitation. Nul n'ignore qu'une disquette formatée sous Dos ne peut être lue sous Unix. Cela est plus clair à présent car chaque système organise ses pistes et ses secteurs à sa manière et crée ses propres structures de données.

L'unité d'allocation d'un fichier est le secteur, or la taille d'un secteur étant réduite le système procède à travers l'allocation d'un cluster (également appelé bloc) constitué d'un ensemble de secteurs. Plus le disque est grand, plus le cluster sera grand. Sa taille varie entre 4KO et

environ 64 KO. Le système renumérote les secteurs du disque en affectant à chaque quadruplet (n°secteur, n°piste, N) plateau, n° face) un numéro de cluster.

Par ailleurs la FAT, structure utilisée par Dos pour gérer les blocs d'un disque, est créée à ce moment là.

1.5 Le partitionnement du disque

Le partitionnement d'un disque dur consiste à le diviser en partitions. La partition est une zone du disque qui peut être considérée comme un disque logique à part. On parle également de lecteur logique. Chaque partition peut recevoir un système d'exploitation différent, pour cela il suffit que ce disque soit amorçable et que le programme d'amorçage du système soit placé dans le secteur de boot.

La commande externe DOS FDISK permet de partitionner un disque dur en plusieurs lecteurs et de spécifier pour chacun la taille et le type de système de fichiers. On distingue des logiciels de partitionnement tel que *Magic partition* qui a le mérite, en plus du partitionnement, de pouvoir restructurer la taille d'une partition en la réduisant ou en l'augmentant. La plupart des systèmes d'exploitation fournissent un administrateur de disques avec lequel on peut créer et gérer des partitions.

Une fois le disque partitionné, chaque partition doit être formatée pour le système d'exploitation qui va la gérer. Une partition est destinée à recevoir un système d'exploitation, des logiciels ou des fichiers. L'utilisateur peut choisir le système de fichiers qu'il souhaite à condition qu'il soit compatible avec le système installé sur la machine. Les plus connus sont les systèmes FAT16 et FAT32. Une partition peut avoir un des systèmes de fichiers suivants :

- FAT16 : Le système d'exploitation Ms-Dos utilise le système FAT16, c'est-à-dire que les numéros de clusters s'écrivent sur 16 bits. Les partitions formatées avec FAT16 ont une taille maximale de 2 GO.
- FAT32 est pris en charge par Windows 95 et les versions qui ont suivis. Les numéros de clusters s'écrivent sur 32 bits et les partitions peuvent atteindre 2 TO. Avec les nouveaux disques, la limite des 2 GO est vite devenue complètement absurde.
- NTFS fourni avec NT (NT File System) est conçu pour gérer des disques avec une capacité supérieure à 400 MO. Il autorise les noms de fichiers longs. Quand un volume est formaté avec NTFS, il est possible de choisir la taille d'un cluster : 512 octets, 1024 ou 2048 octets.

Ces systèmes de fichiers ne sont pas compatibles avec Unix. Linux utilise les partitions linux swap et linux native :

- *Linux Native* sert à stocker les fichiers et les répertoires. L'utilisateur peut en créer plusieurs s'il le souhaite.
- *Linux Swap* est utilisée par la mémoire virtuelle. Elle sert de complément à la mémoire centrale lorsque celle-ci est pleine. Sa taille est égale à 2 ou 3 fois celle de la mémoire centrale.

2 - Les tâches d'un SGF

§ Le stockage des fichiers sur le disque dur

§ La gestion de l'espace libre sur le disque dur

§ La gestion des fichiers dans un environnement multi-utilisateurs

§ La donnée d'une interface conviviale pour manipuler les fichiers (commandes et des attributs simples pour manipuler et décrire les fichiers)

§ La donnée d'utilitaires pour le diagnostic, la récupération en cas d'erreurs, l'organisation des fichiers

3 - L'organisation des fichiers sur le disque

⇒ Sommaire de la section :

- ✓ Le concept de fichier
 - ✓ Le concept de répertoire
 - ✓ Techniques d'allocation des blocs sur le disque
 - ✓ La création d'un fichier par le système d'exploitation
 - ✓ Structures de données pour les fichiers et les répertoires
-

3.1 Le concept de fichier

Un fichier est une unité de stockage logique mise à la disposition des utilisateurs pour l'enregistrement de leurs données : c'est l'unité d'allocation. Le système d'exploitation établit la correspondance entre le fichier et le système binaire utilisé lors du stockage de manière transparente pour les utilisateurs. Dans un fichier on peut écrire du texte, des images, des calculs, des programmes... Les fichiers sont généralement créés par les utilisateurs. Toutefois certains fichiers sont générés par le système ou certains outils comme les compilateurs. Afin de différencier les fichiers entre eux, chaque fichier a un ensemble d'attributs qui le décrivent. Parmi ceux-ci on retrouve : le nom, l'extension, la date et l'heure de sa création ou de sa dernière modification, la taille, la protection. Certains attributs sont indiqués par l'utilisateur, d'autres sont complétés par le système d'exploitation.

Du point de vue du SGF, le fichier est une collection de blocs.

3.2 Le concept de répertoire

Un répertoire est une entité créée pour l'organisation des fichiers. Imaginez un courrier remis en vrac dans une institution comportant des milliers de personnes. Chacun devra tout passer en revue afin de trouver son courrier. Par ailleurs pour pouvoir être retrouvé chaque fichier doit porter un nom unique. Avec la multitude des fichiers créés, le système d'exploitation a besoin d'une organisation afin de structurer ces fichiers et de pouvoir y accéder rapidement. Cette organisation est réalisée au moyen de répertoires également appelés catalogues ou directory. Un répertoire est lui-même un fichier puisqu'il est stocké sur le disque et est destiné à contenir des fichiers.

De nos jours, tous les systèmes de fichiers adoptent une structure hiérarchique. Toutefois, les premiers systèmes utilisaient des structures plate ou à deux niveaux.

Du point de vue du SGF, un répertoire est un fichier qui dispose d'une structure logique : il est considéré comme un tableau qui possède une entrée par fichier. L'entrée du répertoire permet d'associer au nom du fichier qui est un nom externe, les informations stockées en interne par le SGF. Chaque entrée peut contenir des informations sur le fichier (nommées attributs) ou faire référence à (pointer sur) des structures qui contiennent ces informations.

Fig. 31. Structure logique d'un répertoire

La figure n° 31 décrit ces structures. Certaines informations sont données, d'autres calculées

par le système d'exploitation. Que les informations soient conservées au niveau de l'entrée ou dans une structure annexe, l'entrée du répertoire doit permettre d'accéder au stockage du fichier sur le disque dur.

3.3 Techniques d'allocation des blocs sur le disque

On distingue trois manières d'organiser les blocs d'un fichier : contiguë, chaînée, indexée, chacune offrant des avantages et mais également des inconvénients.

§ Allocation contiguë : le fichier est enregistré sur des blocs consécutifs sur le disque

§ Allocation chaînée : on divise le fichier en plusieurs blocs qu'on enregistre à des endroits espacés et on les relie par des liens.

§ Allocation indexée : les blocs sont indépendants mais on conserve dans une structure statique ou dynamique (bloc index) les numéros des blocs appartenant au fichier.

3.3.1 Allocation contiguë

C'est le mode d'allocation le plus logique et le plus simple à gérer. Pour chaque fichier à enregistrer, le système recherche une zone suffisamment grande pour accueillir le fichier. Le fichier sera constitué de plusieurs blocs contigus. L'entrée d'un répertoire dans un système qui applique l'allocation contiguë a la structure suivante :

Fig. 32. Description d'un fichier dans un système qui applique l'allocation contiguë

Le principal avantage de cette méthode est la rapidité lors de l'accès. Il suffit pour lire les différents secteurs d'attendre qu'ils passent sous la tête de lecture/écriture : l'accès est séquentiel.

Cette méthode a été utilisée il y a quelques années, elle ne l'est pratiquement plus de nos jours. En effet elle présente un grand nombre d'inconvénients :

§ La difficulté de prévoir la taille qu'il faut réserver pour le fichier : un fichier est amené à augmenter de taille, il faut par conséquent prévoir de l'espace libre après le dernier secteur alloué.

§ La perte d'espace sur le disque : Si nous prévoyons trop d'espace, le fichier risque de ne pas l'utiliser en entier. En revanche, si nous allouons trop peu d'espace, le fichier risque de ne pas pouvoir être étendu.

§ La fragmentation externe : c'est l'espace perdu en dehors des fichiers. On peut effacer des données ou supprimer des fichiers ce qui libère des secteurs sur le disque. Au fil de l'utilisation, il peut se créer un grand nombre de petites zones dont la taille ne suffit souvent pas pour allouer un fichier mais dont le total correspond à un espace assez volumineux.

3.3.2 L'allocation chaînée

L'allocation chaînée consiste à allouer des blocs chaînés entre eux aux fichiers. Un fichier peut désormais être éparpillé sur le disque puisque chaque bloc permet de retrouver le bloc suivant. D'un point de vue pratique, le numéro du premier bloc du fichier doit être sauvegardé dans l'entrée du répertoire et le chaînage est réalisé de la manière suivante : à la fin de chaque bloc, on réserve quelques octets pour y écrire le numéro du bloc suivant; c'est comme si les blocs étaient chaînés entre eux.

Les avantages de cette méthode sont l'élimination de la fragmentation externe puisque tous les blocs peuvent être alloués. Par ailleurs, elle ne nécessite pas de structure de données spéciale.

En revanche, les inconvénients ici aussi sont multiples :

§ L'accès au fichier est séquentiel. On doit toujours commencer le parcours à partir du début du fichier même si on a récemment chargé les blocs.

§ La perte d'un chaînage entraîne la perte de tous le reste du fichier. Pire encore, il suffit qu'une valeur soit modifiée dans un pointeur pour qu'on se retrouve dans une autre zone de la mémoire.

3.3.3 L'allocation indexée

Tous les inconvénients de l'allocation chaînée peuvent être résolus d'une manière simple : il suffit de retirer les pointeurs des blocs et de les placer dans une structure de données gardée en mémoire centrale ainsi l'information sur les numéros de blocs peut être obtenue à tout moment. La plupart des systèmes actuels appliquent ce mode. MS-DOS utilise la FAT pour y conserver

les chaînages entre les blocs. Unix utilise le I-Node pour y conserver les 10 premières adresses et les pointeurs vers les autres adresses. Windows NT utilise la MFT. Les systèmes d'exploitation appliquent généralement l'allocation indexée ou une combinaison des deux : chaînée et indexée.

3.4 La création d'un fichier par le système d'exploitation

Un fichier et un répertoire sont tous les deux créés en suivant les mêmes étapes qui sont les suivantes :

§ La création d'une structure de données pour décrire le fichier. Tout fichier doit être décrit afin que le système puisse le connaître et le reconnaître. C'est une sorte de recensement qui permet au système d'exploitation d'organiser au mieux les fichiers. Les attributs des fichiers sont sauvegardés dans cette structure de données. Il est évident qu'un fichier et un répertoire se seront pas décrits par la même structure de données.

§ La création du fichier proprement dit. Cela consiste à allouer au fichier un certain nombre de blocs sur le disque selon sa taille. Le contenu d'un bloc sera différent selon qu'il s'agit d'un fichier ou d'un répertoire. Un répertoire est censé accueillir des fichiers et des sous-répertoires, en revanche un fichier est constitué de plusieurs données.

La différence essentielle entre un fichier et un répertoire réside à ce niveau. Les blocs d'un fichier vont contenir des données sans aucune structure particulière. Les blocs d'un répertoire ont une structure bien précise, en effet ils contiennent des noms et des attributs de fichiers et de sous-répertoires (cf. figure n° 33).

Fig. 33. Structure des fichiers et des répertoires

3.5 Structures de données pour les fichiers et les répertoires

Une structure de données au sens algorithmique est un regroupement de types utilisés pour décrire un objet. La structure de données d'un fichier ou d'un répertoire contient dans un premier temps le nom du fichier. Selon le système d'exploitation, les attributs et les informations sur les blocs seront placés dans la même structure de données ou organisés ailleurs.

Nous présentons à présent comment les systèmes MS-Dos, Unix et NT décrivent les fichiers et les répertoires.

3.5.1 Le système d'exploitation MS-DOS

Le répertoire racine

MS-DOS utilise une structure de données : le répertoire racine afin de contenir les fichiers et les répertoires. Le répertoire racine est enregistré sur le disque dur à un emplacement fixe.

Chaque disque formaté avec MS-DOS contient un répertoire racine et des sous-répertoires. Tous ces répertoires ont la même structure et chaque entrée d'un répertoire a une taille de 32 octets et a la structure suivante :

Fig. 34. Une entrée de répertoire dans MS-DOS

Cette organisation est très intéressante et prévenante. En ne mettant que le premier numéro, on ne limite pas la taille du fichier. MS-DOS associe un tableau où chaque indice contient la valeur du prochain bloc associé au fichier.

La FAT

Ce tableau est connu sous le nom de table d'allocation des fichiers FAT (File Allocation Table). MS-DOS ne gère pas individuellement les blocs, il alloue aux fichiers des clusters qui sont constitués d'un ensemble de secteurs. Selon la taille du disque à gérer, un cluster sera composé d'un secteur pour une disquette 3" ½, de deux secteurs pour une disquette de extra haute densité, de 32, 64 secteurs ou plus pour un disque de quelques Gigaoctets. Dans l'exemple de la figure n° 35, l'entrée du

répertoire indique que le fichier commence au cluster numéro 2. En consultant ce tableau, nous retrouvons la suite des numéros de clusters qui composent le fichier *prog.c* : 5 6.

						Numéro du cluster	Valeur
Soit le fichier prog.c décrit par l'entrée suivante dans le répertoire racine						2	5
						3	0
	Prog.c	2		4	0
						5	6
						6	FFFFh
						7	8
						8	FFFFh
						9	0
						10	0
						11	0
						12	0
						13	0
						14	0
						15	0
						16	0
						17	0

Fig. 35. Structure de la FAT

Par mesure de sécurité, deux copies de la FAT sont enregistrées près du répertoire racine.

3.5.2 Le système d'exploitation Unix

Lorsqu'un disque est formaté sous Unix, le système de fichier est créé à ce moment là et les structures de données du système sont alors allouées : le bloc de démarrage (secteur de boot), le superbloc et la table des I-Nodes. Le système écrit les blocs de données par la suite.

Le secteur de boot est le premier secteur du disque. Sur les secteurs voisins le système inscrit le super bloc suivi des I-Nodes. A la suite, le système écrit les blocs de données. Unix utilise les blocs

logiques et ceux physiques. Un bloc physique est un secteur ou un ensemble de secteurs. Un bloc logique correspond à la notion de cluster vu précédemment. La taille d'un bloc logique diffère d'une version de Unix à une autre, mais elle est généralement de 1 ou 4 KO.

Le super bloc

Il s'agit d'une structure de données qui décrit le système de fichiers. On y distingue les caractéristiques suivantes :

- La taille du système de fichiers
- Le nombre de blocs libres ainsi que la tête de la liste des blocs spéciaux
- Des informations sur le nombre des I-Nodes et sur ceux libres

La table des I-Nodes

Lorsqu'un fichier est ouvert, son numéro d'I-Node est chargé dans une entrée de la Table des I-Nodes. Cette structure de données comporte un nombre d'entrées fixe. Le premier I-Node est réservé à la gestion des blocs défectueux. L'I-Node 2 est celui de la racine et il est toujours chargé en mémoire au démarrage du système. Lorsqu'un fichier est détruit, son numéro d'I-Node est marqué. Il peut être attribué à un nouveau fichier au bout d'un certain temps.

...

...

Fig. 36. Structure d'un disque Unix

Le système Unix utilise une structure de données bien particulière afin de décrire les fichiers : c'est le nœud d'index appelé en anglais I-Node. L'I-Node est une structure conçue afin d'alléger le répertoire et d'en éliminer les attributs du fichier ainsi que les informations sur l'emplacement des données. L'entrée d'un répertoire sous Unix a la structure suivante et occupe 16 octets:

Fig. 37. Structure d'une entrée de répertoire sous Unix

Structure d'un I-Node

Un nœud d'index est constitué d'attributs décrivant le fichier ou le répertoire et d'adresses de blocs contenant des données, il occupe 64 octets. Cette structure permet au système de disposer d'un certain nombre de données sur le fichier :

- la taille,
- l'identité du propriétaire, du groupe et les droits d'accès au fichier,
- les dates de création, de dernière consultation et de dernière modification,
- le nombre de références existant pour ce fichier dans le système,
- l'adresse de blocs physiques. Ces blocs physiques contiennent soit les données du fichier, soit l'adresse d'autres blocs physiques (on parle alors de bloc d'indirection).

On peut remarquer au niveau de la figure n° 31, que l'I-Node ne contient aucune identité pour le fichier. La référence à un fichier se fait par l'intermédiaire du répertoire associé au fichier et auquel le système associe un entier, l'index d'identité.

Si nous supposons que la taille d'un bloc est de 1 KO, un fichier sous Unix peut avoir la taille maximale suivante: $10 * 1\text{KO} + 256 * 1\text{KO} + 256 * 256 * 1\text{KO} + 256 * 256 * 256 * 1\text{KO}$ ce qui nous donne plus de 16 GO. En réalité, la taille réelle maximum d'un fichier est inférieure à cette valeur car les pointeurs utilisés pour le déplacement au sein du fichier sont des entiers signés donc limités.

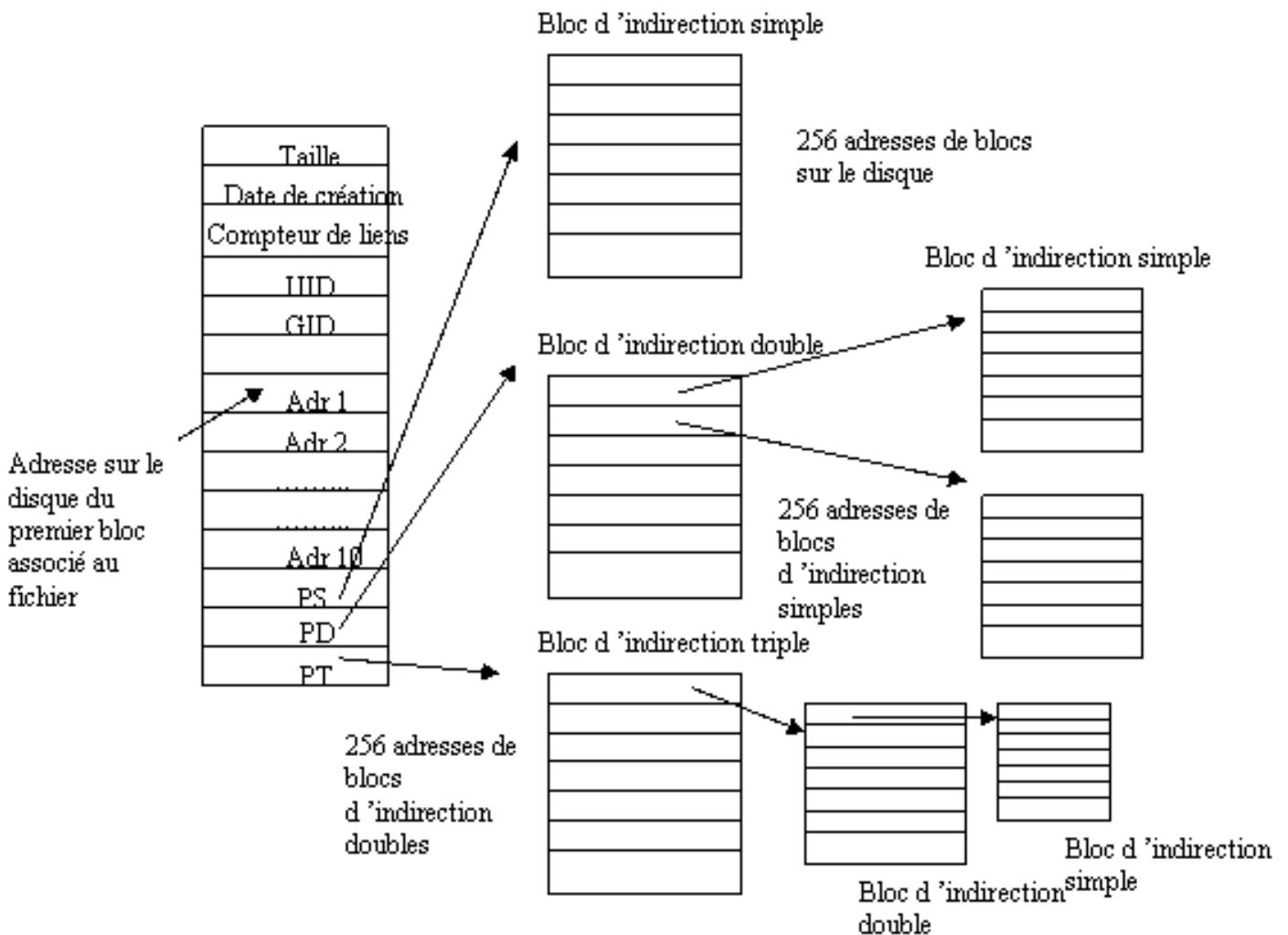


Fig. 38. Structure d'un I-Node

3.5.3 Le système d'exploitation Windows NT

Le système d'exploitation Windows NT propose plusieurs systèmes de gestion de fichiers : un nouveau système de fichiers qui lui est propre : **NTFS** NT File System et les systèmes FAT16 et FAT32.

NTFS utilise une structure de données organisée en table nommée Master File Table MFT pour gérer les fichiers (table des fichiers maître). Elle contient des informations détaillées sur les fichiers et les répertoires. Lorsque ces derniers ont une taille réduite, elle contient les fichiers et les répertoires eux-mêmes.

Les 16 premières entrées de la MFT sont réservées au système. La première entrée de la MFT est appelée descripteur et elle contient des informations sur la table. La seconde est une copie miroir de la première. La troisième entrée contient le journal de récupération en cas d'erreur. Il permet de restaurer les données et de reconstituer la MFT.

Chaque entrée peut contenir 2KO d'informations. Cette valeur assez élevée permet de stocker dans une

entrée un fichier si sa taille est inférieure à 1,5 KO (en supposant que les 500 octets restants sont utilisés pour décrire le fichier). L'avantage de cette organisation est l'accès direct au fichier.

Quand les fichiers sont volumineux, l'entrée inclue un pointeur vers des clusters contenant les données.

Les fichiers sont décrits par NTFS à l'aide des attributs suivants :

- Le nom du fichier
- Un entête de description
- Un descripteur de sécurité
- Les données

Fig. 39. Structure d'une entrée de la MFT

Lorsqu'il s'agit d'un répertoire, le champ *données* décrit un ensemble de fichiers et conserve pour chaque fichier le nom du fichier et le numéro de l'entrée où il est placé.

Les grands répertoires quant à eux ont dans leur champ *données* un pointeur vers un cluster qui stocke les noms des fichiers ainsi que leur numéros.

Le chargement d'un fichier par le système d'exploitation

Que ce soit à travers une commande ou à travers un programme utilisateur, lorsqu'un ordre est émis afin de charger un fichier, l'appel système d'ouverture de fichier est lancé et le système prend la main pour rechercher le fichier dans ses structures de données et le charger du disque à partir de l'emplacement trouvé.

Nous détaillons à présent les étapes déployées pour retrouver tous les blocs d'un fichier et d'un répertoire avec UNIX.

Le schéma du premier accès au fichier est le suivant :

- Accéder au répertoire
- Parcourir le répertoire à la recherche du nom du fichier
- Une fois le fichier trouvé, chercher l'information sur l'emplacement du premier bloc

En réitérant le schéma de parcours de la racine, on accède aux différents répertoires et finalement au fichier.

Un répertoire est enregistré dans un bloc sur le disque dur et un même répertoire peut nécessiter plusieurs

blocs. Il faut tous les charger et les parcourir afin d'y rechercher le fichier ou les sous-répertoires.

La section suivante schématise l'accès au fichier */tp/prog.c*.

Le système commence par accéder au répertoire racine (qui est chargé avec le système) à la recherche du répertoire TP. Une fois localisé, il accède à son I-Node pointé par la table des I-Nodes.

Il en est de même pour le fichier *prog.c*, une fois son entrée localisée dans les blocs du répertoire TP, on obtient le numéro de l'I-Node qui le décrit.

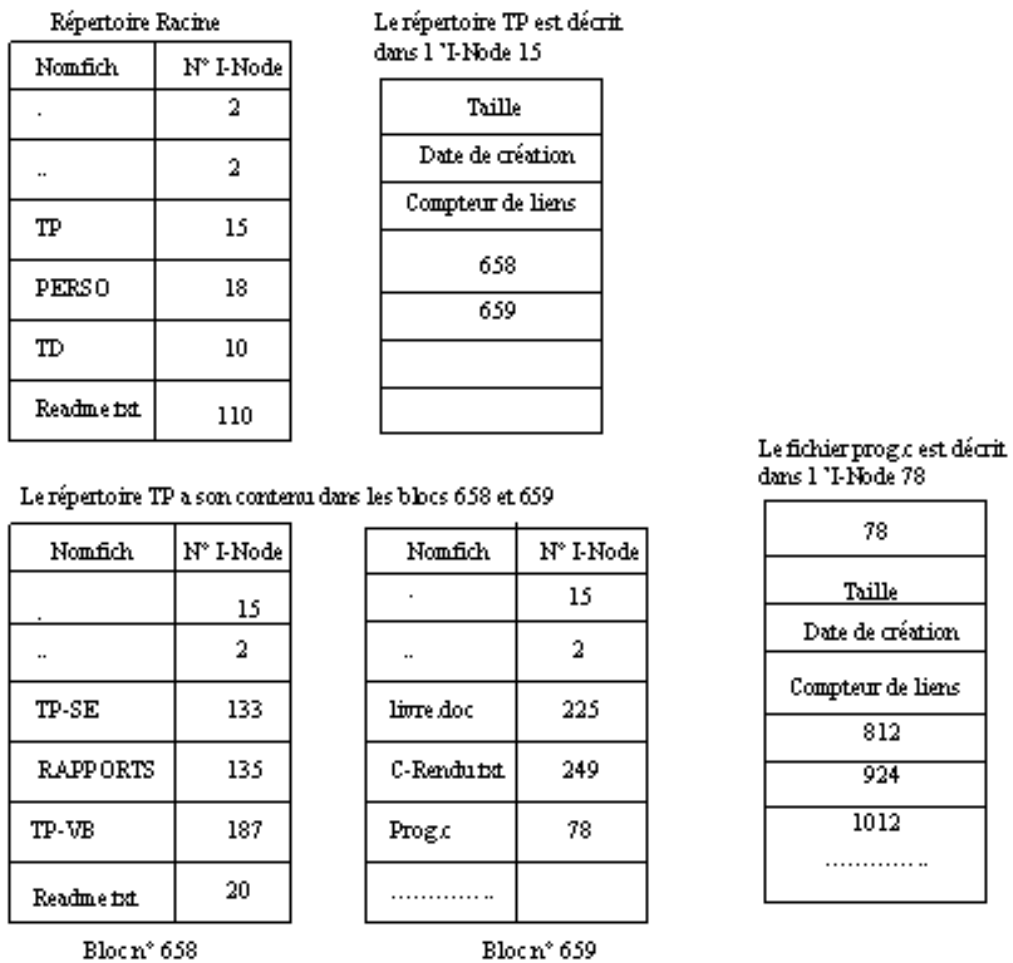


Fig. 40. L'accès à un fichier avec UNIX

4 - La gestion de l'espace libre sur le disque

Les systèmes d'exploitation utilisent essentiellement deux approches pour mémoriser l'espace libre : une statique et une dynamique. L'approche statique utilise une structure de données de taille fixe, le système doit ainsi prévoir la taille nécessaire pour le cas extrême celui où tout le disque dur est vide ou presque.

En revanche dans l'approche dynamique, on a la possibilité de rajouter des éléments pour mémoriser l'espace libre au fur et à mesure que celui-ci augmente.

La structure dynamique utilise généralement une liste chaînée et l'approche statique une table de bits :

- La liste chaînée est constituée d'éléments chacun mémorisant des numéros de blocs libres. Chaque élément que nous appelons bloc spécial (par l'usage qu'on en fait) a une taille de 1K par exemple et peut contenir N adresses de blocs libres. Si les numéros de blocs s'écrivent sur 2 octets, chaque bloc peut contenir les adresses de 512 autres blocs libres.

Au delà de 512 blocs libres, on alloue un autre bloc spécial qu'on rattache au précédent en mettant son adresse dans la dernière adresse de ce bloc. Un pointeur étant une adresse, c'est comme si les blocs étaient chaînés entre eux.

Fig. 41. Mémorisation de l'espace libre à l'aide de blocs spéciaux

Essayons à présent de calculer pour des blocs de 1 KO le nombre de blocs spéciaux pour un disque de 20 MO et un autre de 9 GO :

Soit un disque de 20 MO, soit 20 fois 1024 KO, il y a $20 \cdot 1024$ blocs. Le plus grand numéro de blocs $25 \cdot 2^{10}$ nécessite 15 bits. Il faut 2 octets pour représenter chaque bloc. Un bloc peut contenir 512 adresses ou 511 si on compte le chaînage avec les blocs qui suivent. Il faut donc au maximum $20 \cdot 1024 / 512 = 40$ blocs soit une occupation mémoire de 40 KO.

Dans un disque de 9 GO, on ne peut considérer des blocs de 1 KO. Prenons à titre d'exemple la taille d'un bloc à 4 KO. 9 GO équivaut à $9 \cdot 1024 \cdot 1024$ KO soit $9 \cdot 1024 \cdot 256$ blocs ce qui nous donne le plus grand numéro qui nécessite $4 + 10 + 8$ bits pour son écriture. Les numéros de blocs s'écrivent ainsi sur 32 bits. Un bloc de 4 KO peut contenir 1024 adresses. Il faut ainsi $9 \cdot 1024 \cdot 256 / 1024$ blocs spéciaux soit 2304 blocs spéciaux ou environ 9 MO.

Cette organisation présente plusieurs avantages :

- avec un seul bloc spécial chargé en mémoire on peut allouer 1023 nouveaux blocs aux fichiers,
- on n'utilise un nouveau bloc spécial que si on ne trouve pas de place dans les précédents
- quand le disque est plein, les 9 MO sont ramenés à quelques KO.

Il existe une implémentation légèrement différente mais très intéressante de ces blocs spéciaux dans laquelle un numéro de bloc libre est associé au nombre de blocs libres qui le suivent.

- La seconde approche statique utilise une structure de tableau. Le système du macintosh (mac OS) utilise un tableau appelé table de bits. Chaque entrée est un espace de 2 ou 4 octets où chaque bit référence un bloc. S'il est libre le bit est à 1 sinon à 0. Un disque de N blocs nécessite une table de N bits.

Fig. 42. Mémorisation de l'espace libre à l'aide d'une table de bits

Pour le disque de 20 MO, il faut $20 \cdot 1024$ bits soit $20 \cdot 1024 / 8$ octets = 20/8 blocs de 1 KO soit 2 blocs et demi donc 3 blocs.

Pour le disque de 9 GO, il faut $9 \cdot 1024 \cdot 256$ bits soit $9 \cdot 1024 \cdot 256 / 8$ octets = $9 \cdot 32$ KO soit 72 = $9 \cdot 8$ blocs de 4 KO.

La table de bits n'est pas chargée en entier en mémoire centrale en effet, elle peut être chargée bloc par bloc puisque sa largeur est organisée en fonction de la taille d'un bloc. Toutefois, on peut charger plusieurs blocs de la table de bits, n'y trouver aucun emplacement libre et devoir en charger d'autres, ce qui n'est pas le cas des blocs spéciaux.

5 - La gestion des fichiers dans un environnement multi-utilisat

⇒ Sommaire de la section :

- ✓ La problématique
 - ✓ La sécurité des fichiers
 - ✓ Le partage des fichiers
-

5.1 La problématique

Hormis MS-DOS, la plupart des systèmes actuels sont des systèmes d'exploitation multi-utilisateurs. Le partage de ressources telles que la mémoire, le processeur, l'imprimante et d'autres encore introduit des problèmes de sécurité, de cohérence et de confidentialité pour les utilisateurs. Ce besoin se fait ressentir essentiellement pour l'accès à la mémoire centrale et aux données. Quand des programmes sont placés en mémoire à une certaine adresse, si le système d'exploitation ne fournit aucune protection particulière, un programmeur expérimenté peut espionner voire détruire les données d'un utilisateur. Il peut en outre décider de s'octroyer la plus grande priorité pour l'exécution de ses programmes.

Quant à l'accès aux données, il s'agit d'empêcher un utilisateur de lire ou de modifier les fichiers d'un autre utilisateur si celui-ci ne lui en donne pas le droit. Un système d'exploitation multi-utilisateurs doit intégrer la notion de droit d'accès à un fichier.

Par ailleurs, certains utilisateurs peuvent souhaiter partager des programmes ou des données entre eux afin de les exploiter et de les mettre à jour ensemble. On parle alors de partage de fichiers et cela introduit un problème de **cohérence**. Il est évident qu'il n'existe qu'une seule copie du fichier. Si un premier utilisateur modifie le fichier, les modifications doivent être perçues par tous les utilisateurs qui partagent ce fichier. Par ailleurs, on ne peut autoriser deux utilisateurs à écrire en même temps.

On distingue plusieurs manières de protéger un système et ses utilisateurs. Mais l'aspect le plus mis en évidence est la protection entre utilisateurs par l'attribution de mots de passe et de droits d'accès aux fichiers.

Le nombre d'utilisateurs qui ont accès à un serveur est considérable. Un système d'exploitation doit disposer d'une organisation afin de différencier ces utilisateurs. Généralement on définit des groupes pour privilégier certains utilisateurs et on définit des droits d'accès pour chaque fichier en fonction de la catégorie de l'utilisateur.

5.2 La sécurité des fichiers

Les principaux types d'accès à un fichier sont : la lecture, l'écriture ou l'exécution. Le fait d'indiquer des droits d'accès revient à indiquer quels utilisateurs sont autorisés à lire, écrire ou exécuter un fichier.

Le système d'exploitation UNIX manipule 3 catégories d'utilisateurs : le propriétaire, le groupe auquel appartient le propriétaire et les autres (ceux qui sont considérés quelconques car n'appartenant pas au groupe ou inconnus). Unix code les droits d'accès sur 9 bits à raison de 3 pour le propriétaire, 3 pour le groupe et 3 pour les autres. Ce nombre est sauvegardé dans l'I-Node qui décrit le fichier.

Fig. 43. Un exemple de droits d'accès à un fichier avec UNIX

5.3 Le partage des fichiers

Plusieurs utilisateurs peuvent avoir besoin de travailler sur un même fichier. Le système doit tout en conservant une seule version du fichier donner aux utilisateurs l'impression d'avoir un accès direct au fichier. On distingue deux manières d'implémenter les fichiers partagés :

- placer le fichier dans un répertoire auquel tous les utilisateurs accèdent. Cette solution est rarement utilisée car les utilisateurs aiment avoir leurs fichiers dans leur répertoire. Par ailleurs on est contraint d'autoriser de la sorte l'accès au répertoire du propriétaire ce qui n'est guère agréable pour ce dernier.
- Créer une sorte de lien vers le fichier matérialisé par un nouveau fichier de type spécial portant le même nom que le fichier partagé (mais qu'on peut modifier). Ce dernier est placé dans le répertoire de l'utilisateur et pointe vers le fichier en question.

C'est la seconde solution qui est retenue pour la suite de cette section. Elle transforme la structure du système de fichiers de celle d'un arbre vers celle d'un graphe acyclique orienté ce qui pose quelques problèmes de répétitions des noms de fichiers lors du parcours. Nous n'en discuterons pas davantage mais les lecteurs peuvent obtenir plus de détail sur ce genre de situation dans les ouvrages de théorie des graphes.

5.3.1 Les liens symbolique et physique d'unix

Dans le système Unix, du fait que l'I-Node ne contient pas de nom de fichier, un même fichier peut être référencé sous différents noms dans plusieurs répertoires. Ainsi pour tout utilisateur désirant partager un fichier, on crée un fichier appelé lien qui est placé dans son répertoire.

Unix offre deux types de liens : les liens physiques et ceux symboliques. Quand le lien est symbolique, si l'on examine l'entrée au niveau du répertoire, on y trouve le numéro d'un I-Node différent de celui du fichier. L'examen du contenu du bloc pointé par cet I-Node nous donne le chemin d'accès vers le fichier réel. Avec les liens physiques, on retrouve le numéro de l'I-Node dans l'entrée du répertoire.

Avec les liens symboliques, seul le propriétaire a un pointeur sur l'I-Node. Cette solution présente les inconvénients suivants : trop de parcours (répertoire, I-Node du fichier lien, bloc qui contient le chemin pour enfin accéder à l'I-Node), mémoire (un I-Node et un bloc supplémentaires), mais elle

permet de mémoriser des chemins d'accès à des fichiers situés sur d'autres ordinateurs.

5.3.2 La suppression d'un fichier partagé

On distingue plusieurs problèmes au niveau de la suppression. Peut-on supprimer un fichier si des utilisateurs y accèdent encore ? Un propriétaire n'est-il pas libre de supprimer son fichier quand il le désire ? la réponse à ces questions est différente selon le système d'exploitation et le type de lien.

Avec Unix, quand il s'agit d'un lien symbolique, les liens demeurent dans les répertoires quand on détruit un fichier. La destruction d'un lien symbolique n'affecte pas le fichier. En revanche, si un fichier est détruit l'utilisateur ne sera informé que lors du prochain accès au fichier. Par contre s'il s'agit d'un lien physique, on ne peut supprimer le fichier et son I-Node en laissant ce numéro apparaître dans plusieurs répertoires.

Pour ce faire Unix propose le compteur de liens qui est un champ inclus dans l'I-Node afin de comptabiliser le nombre de liens au fichier. Il renseigne sur le nombre de répertoires qui possèdent ce fichier partagé (bien entendu, on ne peut pas mémoriser les répertoires qui y accèdent, mais juste leur nombre). Supprimer le fichier par un utilisateur qui n'en est pas propriétaire équivaut à supprimer l'entrée au niveau du répertoire. Il en est de même pour le propriétaire si le compteur n'est pas à 1. Le fichier n'est supprimé que si son compteur atteint la valeur 1.

6 - La cohérence d'un SGF

Un système de fichiers est cohérent s'il est capable de restituer à l'utilisateur ses fichiers et ses répertoires dans l'état où il les a laissés. Cela sous-entend qu'aucun fichier n'est ni déplacé d'un répertoire à un autre ni perdu, et que le contenu des fichiers est correct et n'a pas été altéré. Si l'on venait par erreur à détruire le SGF, la restauration des organisations de fichiers et de répertoires pourrait carrément être impossible.

Parmi les utilitaires d'un système d'exploitation, on trouve notamment un utilitaire de vérification et de correction des erreurs au niveau du système de fichiers tel que **scandisk** pour Windows (automatiquement lancé après un redémarrage ou manuellement à la demande de l'utilisateur).

Du point de vue pratique, ces utilitaires examinent les structures de données du système d'exploitation et corrigent les erreurs qu'ils peuvent y trouver. La vérification concerne les blocs appartenant aux fichiers ou libres, et l'état des répertoires. Ainsi ils vont parcourir les différents répertoires et la FAT ou bien la MFT ou encore les I-Nodes. La raison pour laquelle l'information restituée n'est pas la même qu'initialement, est que le système peut tomber en panne ou se bloquer au milieu d'une opération de création de I-Node, ou d'écriture de numéros de blocs libres etc.

Nous présentons dans ce qui suit les deux utilitaires employés par Unix pour vérifier la cohérence de son SGF. Ils se basent sur la vérification des blocs et des I-Nodes. En ce qui concerne les blocs, cet utilitaire parcourt les structures de gestion de blocs et construit deux tableaux. Dans le premier tableau, l'utilitaire indique pour chaque case i le nombre de fois où le bloc numéro i est référencé dans un fichier donc occupé. Dans le second, le nombre de fois où il apparaît parmi les blocs libres (en consultant la liste des blocs spéciaux chaînés). Quand le SGF est cohérent, chaque bloc apparaît une fois au maximum soit dans le premier tableau soit dans le second.

Les tableaux qui suivent illustrent plusieurs cas d'incohérence :

Numéro du bloc	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
----------------	---	---	---	---	---	---	---	---	---	---	----	----	----	----	----	----

Nombre de fois où le bloc a été trouvé occupé	1	1	0	1	0	2	1	1	1	0	0	1	1	1	0	0
Nombre de fois où le bloc a été trouvé libre	0	0	0	0	2	0	0	0	0	1	1	1	0	0	1	1

On distingue quatre cas d'incohérence schématisés à travers les blocs 2, 4, 5, 11 :

- le bloc 2 est à 0 partout, il n'est ni libre ni occupé : la correction se fait en le rajoutant aux blocs libres de manière à pouvoir l'allouer ultérieurement à un fichier.
- Le bloc 4 est à (0,2), il est plusieurs fois libre : cette situation ne peut avoir lieu si le système d'exploitation utilise une table de bits pour référencer les blocs libres. En effet la case ne peut contenir que les valeurs 0 ou 1. En revanche, avec la liste chaînée des blocs spéciaux, cette situation peut se produire. On la corrige en retirant une occurrence de ce numéro de bloc de la liste chaînée.
- Le bloc 5 est à (2,0). Il est utilisé par plus d'un fichier. Commençons par expliciter ce cas de figure. Un même bloc utilisé par plus d'un fichier nous fait penser qu'il est peut-être partagé. Or avec Unix, les blocs sont décrits dans un I-Node qui est unique; même si un fichier est partagé, c'est le numéro de l'I-Node qu'on risque de trouver plusieurs fois et non celui du bloc : il s'agit réellement d'un cas d'incohérence. La solution est délicate et nécessite l'allocation d'un nouveau bloc libre, la copie du bloc 5 dans celui-ci et l'insertion du nouveau bloc parmi les numéros des blocs de l'un des fichiers.
- Le bloc 11 est à (1,1), à la fois libre et occupé. C'est la situation occupée qui prime et on corrige en retirant le numéro du bloc de ceux libres.

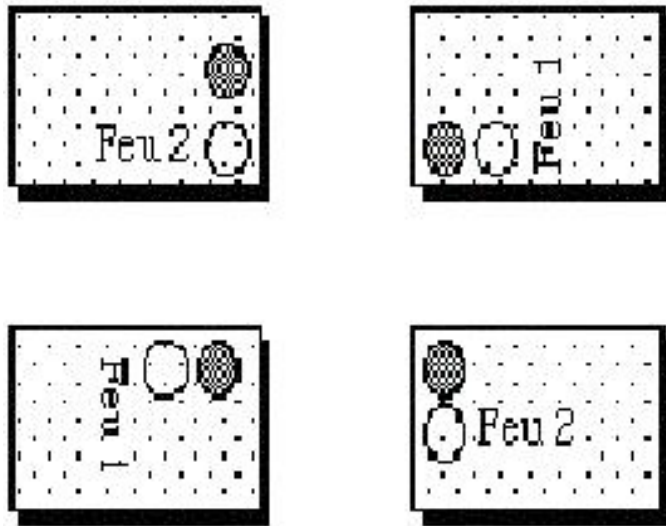
Le second utilitaire vérifie les I-Nodes et utilise un tableau de compteurs qu'il applique aux fichiers et non plus aux blocs. Son rôle consiste à parcourir tous les répertoires en partant de la racine et à construire un tableau où le contenu d'une case est le nombre de fois où un I-Node est référencé. Lorsqu'un fichier est partagé par un lien physique son numéro d'I-Node apparaît dans un autre répertoire. Pour vérifier la cohérence, on compare la valeur obtenue avec le compteur de liens dans chaque I-Node :

- Si le compteur de liens est supérieur à la valeur comptée par l'utilitaire (bien entendu on se fie à la valeur comptée), il y a une incohérence mais cette situation n'est pas grave. Lorsqu'on aura supprimé tous les accès à l'I-Node, le compteur ne sera pas à zéro, et l'I-Node ne sera jamais supprimé. On corrige en affectant la valeur trouvée par l'utilitaire à celle du compteur.
- Si le compteur de liens est inférieur à la valeur trouvée par l'utilitaire, cela veut dire qu'il existe plus d'utilisateurs qui accèdent au fichier que ne signale le compteur de liens. L'I-Node risque d'être supprimé alors que des utilisateurs utilisent encore le fichier. Si cette situation n'est pas corrigée en affectant au compteur de liens la valeur trouvée par l'utilitaire, quand le compteur atteindra zéro, le fichier sera supprimé ainsi que son I-Node. Plus tard ce numéro d'I-Node sera attribué de nouveau à un fichier et les utilisateurs risquent de pointer vers le nouveau fichier.

TD système d'exploitation N°4

1. Le carrefour ou le problème de la gestion des feux de circulation.

La circulation au carrefour de deux voies est réglée par des signaux lumineux (feu vert/rouge). On suppose que les voitures traversent le carrefour en ligne droite et que le carrefour peut contenir au plus une voiture à la fois.



On impose les conditions suivantes :

- toute voiture se présentant au carrefour le franchit en un temps fini,
- les feux de chaque voie passent alternativement du vert au rouge, chaque couleur étant maintenue pendant un temps fini.

Les arrivées sur les deux voies sont réparties de façon quelconque. Le fonctionnement de ce système peut être modélisé par un ensemble de processus parallèles :

- un processus P qui exécute la procédure Changement de commande des feux;
- un processus est associé à chaque voiture;
- le traversée du carrefour par une voiture qui circule sur la voie i ($i = 1, 2$) correspond à l'exécution d'une procédure $Traversee_i$ par le processus associé à cette voiture.

On demande d'écrire le programme `Changeement` ainsi que les procédures `Traversee1` et `Traversee2`.

Remarque :

Le processus P doit attendre que la voiture engagée dans le carrefour en soit sortie avant d'ouvrir le passage sur l'autre voie.

2. L'application simple.c (Annexe 1)

- lit le flux de caractères arrivant sur l'entrée standard
- sépare les chiffres et les lettres
- effectue l'opération appropriée en fonction du type de caractère, à savoir :
 - sommer les chiffres
 - réaliser un spectre de fréquence pour les lettres
- affiche le résultat obtenu

Pour cela, le programme est composé de trois entités :

- un distributeur (en charge de la répartition des caractères) ;
- un additionneur (opérant sur les chiffres) ;
- un compteur (opérant sur les lettres).

Adapter simple.c pour que les fonctions d'additionneur et de compteurs soient assurées par des processus enfants d'un processus père qui assure la fonction de distributeur. Les communications entre les processus se font par :

par mémoire partagée.

/*****

**

** Version initiale.

** -----

** Ce programme se compose de trois fonctions. La fonction de desitribution

** est chargée de lire les caractères en provenance de l'entrée standard,

** d'envoyer les chiffres à la fonction chiffre (par l'intermédiaire d'un

** appel de fonction classique) et les lettres à la fonction lettre (aussi par

** l'intermédiaire d'un appel de fonction).

** Une fois la fin de fichier atteinte, les fonctions renvoient leurs

** résultats à la fonction de distribution qui affiche ensuite les résultats.

**

**

*****/

#include <assert.h>

#include <ctype.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

/*

** Somme des chiffres.

*/

int chiffre(int car) {

```
static int somme = 0 ;

if (car != EOF) {

/* Somme des chiffres. */

somme += (car - '0') ;

} else {

/* Renvoi du résultat. */

return somme ;

}

return 0 ;

}

/*

** Fréquence des lettres.

*/

int* lettre(int car) {

/* On suppose que le tableau est correctement initialisé. */

static int frequence['z' - 'a' + 1] ;

if (car != EOF) {

/* Incrément des fréquences. */

frequence [ tolower ( car ) - 'a' ] ++ ;

} else {

/* Renvoi du résultat. */
```

```
return frequence ;

}

return NULL ;

}

/*

** Distribution des caractères et affichage des résultats.

*/

void distributeur() {

int car, *frequence, i, somme ;

/* Distribution des caractères. */

do {

car = getchar() ;

if (isalpha(car) || (car == EOF)) {

frequence = lettre ( car ) ;

}

if (isdigit(car) || (car == EOF)) {

somme = chiffre(car);

}

} while(car != EOF);

/* Lecture et affichage de la somme. */

printf("Somme : %d", somme);

/* Lecture et affichage des frequences. */
```

```
for(i = 'a' ; i <= 'z' ; i ++ ) {  
  
printf(" ; %c : %d", i, frequence[i - 'a']) ;  
  
}  
  
printf("\n");  
  
}  
  
/*  
  
** Lancement de la fonction de distribution.  
  
*/  
  
int main(int argc, char* argv[]) {  
  
/* Lancement du travail. */  
  
distributeur();  
  
/* Tout s'est bien terminé. */  
  
exit(0);  
  
}
```


La gestion de la mémoire centrale

Objectifs

1. Traitement d'un programme
 2. Rappels sur le déroulement d'une instruction
 3. La liaison d'adresses
 4. Les stratégies d'allocation de la mémoire
 5. Gestion de la mémoire appliquant le va et vient
 6. La mémoire virtuelle
 7. Les systèmes paginés
 8. Les systèmes segmentés
 9. TD5
-



1 - Traitement d'un programme

Tout programme écrit dans un langage de programmation donné suit la suite logique suivante : il est tapé, ensuite enregistré ensuite compilé et pour finir exécuté. La phase de compilation est de plus en plus transparente aux utilisateurs dans les nouveaux environnements de développement. En effet, elle est automatiquement lancée même si le programmeur demande directement l'exécution du programme. Tout programme est traduit durant la phase de compilation en instructions machine exécutables par l'unité centrale. Un traducteur traduit un programme source en programme objet (exemple interpréteur ou compilateur).

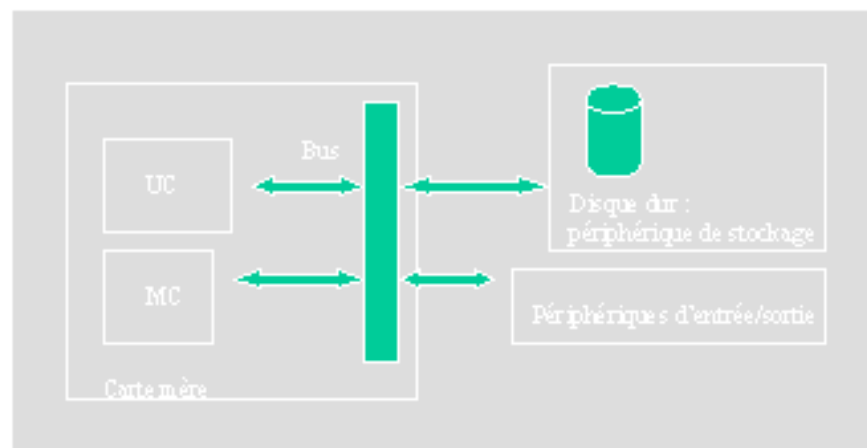


Fig. 44. Cycle de vie d'un programme

Les traitements entrepris par chaque étape sont décrits ci-dessous et illustrés à travers la figure n° 55 :

- compilation : étape de vérification de la cohérence du programme. Elle comprend trois sous-étapes : la vérification syntaxique, lexicale et sémantique. Ainsi le compilateur vérifie que toute variable utilisée est déclarée, que les points virgules sont mis correctement, que les structures du langage sont utilisées correctement, par exemple si le programme est écrit en langage C il n'utilise pas de then après un if.
- Lors de la compilation, à chaque objet (variable, fichier,..) est associé un nom intermédiaire qui possède une adresse logique. Cette étape génère un fichier objet.
- Edition des liens : cette étape établit le lien entre le programme et les différents fichiers concernés. Parmi ceux-ci on trouve, les fichiers d'entête (extension h) les

fichiers contenant les autres parties du programme (dans le cas où il est réparti entre plusieurs fichiers) les bibliothèques du langage (dans le cas où des fonctions sont appelées). Cette étape génère un fichier exécutable.

- Chargement du programme : le fichier exécutable ainsi que les bibliothèques du langage et/ou du système sont chargés en mémoire centrale à une adresse bien précise.

Durant chacune de ces étapes, certains objets sont créés au fur et à mesure.

Le fichier exécutable obtenu est une suite d'instructions machines possédant chacune une adresse logique. Chaque adresse logique doit être traduite en une adresse physique, pour que celle-ci ait une correspondance réelle, une fois le fichier chargé en mémoire. Nous signalons toutefois que cette conversion d'adresses peut se faire durant l'étape d'édition des liens ou celle de chargement.

Nous allons à présent discuter du déroulement de l'instruction au niveau du processeur.

Quelles adresses sont manipulées, logiques ou physiques ? Comment le processeur demande-t-il une instruction en précisant son adresse dans le bus d'adresses ?

2 - Rappels sur le déroulement d'une instruction

Dans un premier temps il est nécessaire de rappeler les différentes étapes entreprises par le processeur pour exécuter une instruction.

- Examen de la valeur du compteur ordinal,
- Extraction de l'instruction dont l'adresse se trouve dans le compteur ordinal, c'est-à-dire l'envoi de cette adresse à travers le bus adresses, l'extraction de l'instruction placée à cette adresse de la mémoire centrale et son renvoi à travers le bus données,
- Chargement de l'instruction dans le registre instruction
- Décodage de l'instruction par l'UCC afin d'y identifier les opérateurs et les opérandes.
- Recherche des opérandes (si elles ont été récemment utilisées, elles peuvent être dans le cache du processeur). Le cas échéant, le processeur doit se procurer les adresses de ces opérandes et reprendre les étapes d'extraction de la mémoire.

3 - La liaison d'adresses

On appelle liaison d'adresses la correspondance entre une adresse logique et une adresse physique. Cette étape de liaison peut être réalisée durant l'édition des liens mais dans la plupart des cas elle est effectuée au moment de l'exécution. Le processeur transmet des adresses physiques sur le bus adresses.

Certains systèmes utilisent un composant matériel interne au processeur afin de convertir les adresses, appelé MMU Memory Management Unit (Unité de gestion de mémoire). On distingue l'adressage absolu et l'adressage relatif.

⇒ [Sommaire de la section :](#)

✓ [Adressage absolu](#)

✓ [Adressage relatif](#)

3.1 Adressage absolu

Quand une adresse est absolue, elle ne nécessite aucun traitement pour obtenir une adresse physique. Les adresses physiques sont utilisées telles quelles par l'unité centrale car les adresses logiques et physiques sont équivalentes. On ne peut pas déplacer le programme en mémoire centrale. En revanche dans ce cas, le programme doit toujours être chargé à la même adresse, faute de quoi il ne pourra pas s'exécuter, à moins d'effectuer une nouvelle étape d'édérations des liens. Les programmes correspondants sont dits non relogeables.

3.2 Adressage relatif

C'est un adressage qui est relatif à la première adresse attribuée au programme exécutable. Toutes les adresses doivent être converties ou traduites en partant de cette adresse. On peut ainsi changer de zone mémoire d'une exécution à l'autre. Un tel programme est dit relogeable.

La conversion des adresses est liée au mode d'allocation appliqué : contiguë ou non. En mode contiguë, une adresse relative qui vaut 100 doit être considérée comme $100 +$ l'adresse de début du fichier généré par l'éditeur de liens. En mode non contiguë, le programme est partagé entre plusieurs zones dispersées dans la mémoire et le mécanisme de gestion de la mémoire doit trouver la zone et ensuite l'adresse.

Après avoir introduit et expliqué les notions d'adresses et de conversion, nous présentons les stratégies de gestion de la mémoire centrale.

4 - Les stratégies d'allocation de la mémoire

On recense essentiellement deux modes de gestion de la mémoire centrale : le mode contigu et celui non contigu. Selon le mode de gestion de la mémoire qui est appliqué, lorsqu'un programme est chargé en mémoire centrale à partir du disque, le programme sera placé dans une seule zone (allocation contiguë) ou réparti entre plusieurs zones (allocation non contiguë). Les avantages et les inconvénients sont les mêmes que ceux étudiés pour la gestion de la mémoire auxiliaire.

Par ailleurs la zone de mémoire allouée à un programme est de taille limitée, or tout programme est amené à augmenter de taille lors de son exécution. En effet, des résultats sont calculés et des variables peuvent être créées dynamiquement. On distingue alors des systèmes qui utilisent des zones de taille fixe et d'autres qui permettent au programme de s'étendre sur l'espace avoisinant si celui-ci est libre. Le mode d'allocation contiguë n'est plus appliqué de nos jours, nous n'en donnerons qu'un rapide aperçu en guise d'historique.

⇒ [Sommaire de la section :](#)

- ✓ [Allocation contiguë en mémoire centrale](#)
 - ✓ [Allocation non contiguë en mémoire centrale](#)
-

4.1 Allocation contiguë en mémoire centrale

Une partition est un emplacement de la mémoire de taille limité, qui est décrit par une adresse de début et une adresse de fin. On distingue les partitions de taille fixe et celles de taille variable.

4.1.1 Les partitions de taille fixe

Dans le cas des partitions fixes, la mémoire est découpée par l'administrateur au démarrage du système.

Fig. 45.Découpage de la mémoire en partitions de taille fixe

Chaque processus est supposé occuper la plus petite partition qui peut le contenir. Tout espace inutilisé dans la partition est perdu car on ne cumule pas des espaces appartenant à des partitions voisines.

On distingue plusieurs stratégies pour le choix du processus et celui de la partition, parmi lesquelles nous citons :

§ Stratégie de la première zone libre (First Fit) : la première partition ayant une taille suffisante

§ Stratégie du meilleur ajustement (best fit) : la plus petite partition de taille suffisante

§ Stratégie préventive du worst fit : qui alloue une grande partition plutôt qu'une petite afin de garder des petites zones libres. En effet, les petits programmes sont plus fréquents que ceux de grande taille.

Le principal inconvénient de l'allocation par partition de taille fixe est la saturation, un programme peut rapidement occuper l'ensemble de la partition qui lui est allouée et l'exécution du programme serait ainsi terminée pour faute d'espace supplémentaire. La seule issue dans une telle situation est le déplacement du programme sur une autre partition.

4.1.2 Les partitions de taille variable

Lorsque les partitions sont de taille variable, leur nombre ainsi que leur taille varient dynamiquement au fur et à mesure que les processus sont créés ou terminés.

Si un processus nécessite davantage de mémoire car il en a alloué dynamiquement pour certaines variables, le système d'exploitation peut augmenter la taille de sa partition en lui attribuant des partitions voisines s'il en existe. Le cas échéant, le recours est le même que pour les partitions fixes, à savoir que le programme est déplacé ou bien des processus voisins sont déplacés.

il est possible de créer des partitions plus grandes en déplaçant des programmes vers le haut ou le bas de la mémoire. Ce traitement est appelé compactage de la mémoire, toutefois son usage n'est pas recommandé en raison du temps processeur qu'il nécessite.

Exemples d'allocation avec des partitions de taille fixe

Exemple d'allocation avec l'algorithme d'allocation mémoire : first fit lorsque l'algorithme d'ordonnancement est [fifo](#).

Exemple d'allocation avec l'algorithme d'allocation mémoire : first fit lorsque l'algorithme d'ordonnancement est [SJF](#).

Exemple d'allocation avec l'algorithme d'allocation mémoire : first fit lorsque l'algorithme d'ordonnancement est [à priorités](#).

Exemple d'allocation avec l'algorithme d'allocation mémoire : best fit lorsque l'algorithme d'ordonnancement est [fifo](#).

Exemple d'allocation avec l'algorithme d'allocation mémoire : best fit lorsque l'algorithme d'ordonnancement est [SJF](#).

Exemple d'allocation avec l'algorithme d'allocation mémoire : best fit lorsque l'algorithme d'ordonnancement est [à priorités](#).

Exemple d'allocation avec l'algorithme d'allocation mémoire : worst fit lorsque l'algorithme d'ordonnement est [SJF](#).

Simulez votre [Exemple](#).

Exemples d'allocation avec des partitions de taille variable

Exemple d'allocation avec l'algorithme d'allocation mémoire : first fit lorsque l'algorithme d'ordonnement est [fifo](#).

Exemple d'allocation avec l'algorithme d'allocation mémoire : first fit lorsque l'algorithme d'ordonnement est [SJF](#).

Exemple d'allocation avec l'algorithme d'allocation mémoire : first fit lorsque l'algorithme d'ordonnement est [à priorités](#).

Exemple d'allocation avec l'algorithme d'allocation mémoire : best fit lorsque l'algorithme d'ordonnement est [fifo](#).

Exemple d'allocation avec l'algorithme d'allocation mémoire : best fit lorsque l'algorithme d'ordonnement est [SJF](#).

Exemple d'allocation avec l'algorithme d'allocation mémoire : best fit lorsque l'algorithme d'ordonnement est [à priorités](#).

Exemple d'allocation avec l'algorithme d'allocation mémoire : worst fit lorsque l'algorithme d'ordonnement est [SJF](#).

Simulez votre [Exemple](#).

4.2 Allocation non contiguë en mémoire centrale

C'est le mode d'allocation qui est appliqué par les systèmes actuels, ainsi un fichier peut être chargé à des adresses dispersées en mémoire. La correspondance entre les adresses est réalisée au cours de l'exécution.

La mémoire peut être allouée par zones de taille fixe ou variable. Quand toutes les zones ont la même taille, on parle de **page** et de systèmes paginés. Quand leur taille peut varier, on parle de **segments** et de systèmes segmentés. On peut combiner les deux modes: des segments composés de pages.

5 - Gestion de la mémoire appliquant le va et vient

La taille de la mémoire centrale est limitée et très faible comparée à celle du disque dur. La mémoire centrale est fréquemment saturée, le système évacue alors certains programmes de la mémoire et les place sur une zone bien précise du disque (dite zone de swap, elle ne contient que les processus délogés de la mémoire centrale). Ensuite il charge d'autres programmes à partir du disque en mémoire centrale et les exécute.

On a vu apparaître plusieurs techniques qui ont donné naissance aux fondements de la gestion actuelle des mémoires centrales : la première est le va et vient (swapping en anglais)

Cette technique s'est répandue avec le temps partagé où il fallait exécuter les processus de plusieurs utilisateurs. Quand un processus a fini son quantum de temps et qu'il existe plus de processus que la mémoire ne peut contenir, il est déplacé sur le disque par un processus appelé swapper. Il faut à ce stade préciser que le processus est transféré en entier. C'est l'ordonnanceur en collaboration avec le swapper qui est chargé de le ramener une fois que son exécution approche. Pour l'utilisateur ceci est transparent.

La question qui se pose à présent est : le processus une fois ramené peut-il être placé n'importe où ? Cela dépend du mécanisme de traduction d'adresses. Si les adresses sont converties à l'exécution, le MMU déterminera les nouvelles adresses physiques. Autrement le processus devra être placé à la même adresse.

6 - La mémoire virtuelle

Le swapping ne résout pas le problème de certains programmes trop grands pour être contenus en mémoire centrale. On a ainsi vu apparaître un nouveau mode de gestion de la mémoire basé sur la notion de mémoire virtuelle. Elle représente le second mode appliqué afin d'exécuter des programmes volumineux (en plus du swapping).

On parle de mémoire virtuelle car elle a une taille irréaliste, bien plus grande que la mémoire physique et elle permet aux programmes d'avoir des tailles plus grandes que celle de la mémoire centrale, tout simplement. Ce n'est pas un grand espace virtuel que vont se partager les processus mais la possibilité pour chaque processus d'occuper un espace d'adressage très grand.

La taille des programmes, des données et de la pile peut dépasser la mémoire disponible, le système d'exploitation garde en mémoire les parties qui sont utilisées et stocke le reste sur le disque. On voit ainsi que la mémoire virtuelle s'inspire du swapping. En supposant qu'une partie du disque dur est intégrée à la mémoire centrale, on peut disposer d'un espace d'adresses logiques plus grand.

L'espace d'adressage du processus est désormais un espace d'adressage virtuel dont les adresses varient de 0 à une adresse maximale fixe qui est la même pour tous les processus. La taille de la mémoire virtuelle est fixée par la taille des registres, la plupart des machines ont des registres de 32 bits donc les adresses logiques sont entre 0 et 2^{32} soit un espace d'adressage de 4 Go. La conversion des adresses virtuelles en adresses physiques est souvent réalisée par le MMU.

La mémoire virtuelle est indépendante du mode de gestion paginé ou segmenté qui est appliqué. Dans un système paginé, l'espace virtuel sera partagé en pages et dans un système segmenté, il sera décomposé en segment.

7 - Les systèmes paginés

L'ensemble de la mémoire logique (l'espace d'adressage) d'un programme est décomposé en pages numérotées à partir de 0. La mémoire centrale ou physique est décomposée en cases numérotées de la même manière. L'espace d'adressage logique d'un programme ne devient concret qu'une fois placé en mémoire centrale. Sa taille effective représente l'ensemble des adresses logiques utilisées, ainsi l'adresse 10234 ne fera partie de cet espace que si un objet y est placé..

Quand le système n'applique pas de mémoire virtuelle, on a le même nombre de cases et de pages. Un programme est chargé du disque dur en mémoire centrale quand le système affecte une case pour contenir la page. Généralement les pages et les cases ont la même taille.

Il faut souligner que les pages sont successives et les cases également mais deux pages successives ne sont pas forcément placées dans des cases successives.

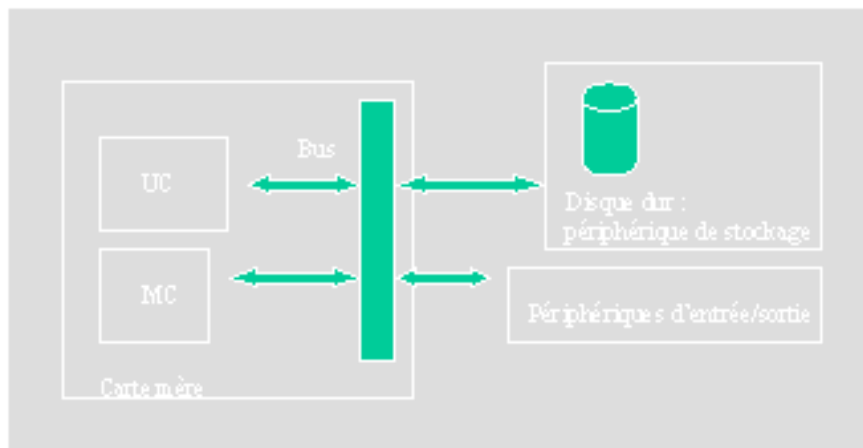


Fig. 46. Placement de pages en mémoire centrale

Dans la figure n° 57, on observe que le programme P2 constitué de 5 pages a uniquement ses deux premières pages chargées et qu'elles ne sont pas voisines en mémoire centrale.

- ✓ [La table des pages](#)
 - ✓ [Conversion des adresses](#)
 - ✓ [Fonctionnement de l'unité de gestion de la mémoire: Le MMU](#)
 - ✓ [La pagination multi-niveaux](#)
 - ✓ [La mémoire associative](#)
 - ✓ [Inconvénients des systèmes paginés](#)
-

7.1 La table des pages

Le système d'exploitation utilise une structure de données pour décrire les pages d'un processus. Une entrée de la table contient les informations suivantes :

Bit de présence	Bit de modification	Bits de protection	Bit de référence	Adresse sur le DD	Numéro de la case mémoire
-----------------	---------------------	--------------------	------------------	-------------------	---------------------------

Fig. 47. Une entrée de la table des pages d'un processus

§ Le bit de présence indique si la page est chargée en mémoire ou non

§ Les bits de protection définissent le mode d'accès à la page en lecture ou en écriture

§ Le bit de modification permet d'économiser une recopie sur le disque si la case va être allouée à une autre page.

§ Le bit de référence indique que la page placée dans cette case a été référencée c'est-à-dire accédée, cette information sert aux algorithmes de remplacement de pages.

La table des pages contient une entrée pour chaque page de l'espace d'adressage. Si le système d'exploitation applique la mémoire virtuelle, la plus grande adresse virtuelle est 2^{32}

Soit des pages de 4 KO, une page peut contenir 4096 adresses. Le nombre des pages est égal à la taille de la mémoire virtuelle divisé par la taille d'une page. Soit $4\text{ G0}/4\text{ KO} = 2^{20}$ environ 1 million de pages.

Une table des pages comportant un million d'entrée est une structure de données gigantesque dont la taille sera pénalisante lors du parcours. Par ailleurs, même si la table n'est remplie qu'en partie, la structure de tableau étant à taille fixe, il faut réserver l'espace pour le million éventuel de pages à la création de la table des pages. Nous rappelons à ce stade que chaque programme a sa propre table des pages. Fort heureusement, On distingue plusieurs manières d'organiser cette table et nous verrons plus loin qu'elle peut être décomposée en blocs et chargée au fur et à mesure. Nous reviendrons plus tard sur cette organisation de la table des pages. Nous allons dans la section suivante présenter une première utilisation de cette table par le MMU.

7.2 Conversion des adresses

Une adresse logique AL est donnée sous la forme suivante (x,y) , x indique la page et y le déplacement au sein de cette page. X est obtenu en divisant AL par la taille d'une page et y est le reste de cette division.

Le calcul de l'adresse physique AP se fait de la manière suivante : le système d'exploitation commence par trouver l'emplacement de la page x en mémoire centrale (la case correspondante), ensuite il effectue un déplacement de y par rapport à la première adresse dans cette case, en effet les pages et les cases sont de même taille.

Dans la figure n° 48, nous prenons l'adresse logique 9035 et nous l'exprimons sous la forme (x, y) précédemment présentée. $9035 \text{ div } 4 \text{ KO} = 9035 \text{ div } 4096 = 2$ cette adresse se trouve donc dans la page n° 2. Le déplacement est égal à $9035 \text{ modulo } 4096 = 843$ donnant le couple $(2,843)$. Il suffit de trouver l'adresse de la case contenant la page n° 2 et d'y effectuer un déplacement de 843 pour obtenir l'adresse physique.



Fig. 48. Exemple de conversion d'une adresse

7.3 Fonctionnement de l'unité de gestion de la mémoire: Le MMU

Le MMU est le composant matériel chargé de la conversion des adresses logiques en adresses physiques. La plupart des processeurs CISC disposent d'une unité de gestion de la mémoire. Un composant réservé à la conversion est en effet plus rapide que l'exécution d'un programme de conversion. Par ailleurs, la conversion des adresses est sollicitée à une fréquence très élevée dans un ordinateur.

Il y a quelques années ce composant se trouvait sur la carte mère, à présent, si jamais il existe, il est intégré au processeur, ce qui rend la conversion plus rapide.

Le fonctionnement du MMU est le suivant : l'adresse logique est donnée sur 32 bits, il la considère comme un numéro de page et un déplacement au sein de cette page. Ensuite il utilise la table des pages afin d'obtenir le numéro de la case dans laquelle est chargée cette page. Le numéro de la page est utilisé comme indice dans cette table des pages. Enfin il concatène ce numéro de case avec le déplacement, afin d'obtenir l'adresse physique (si la case et la page ont la même taille ce qui est généralement le cas, le déplacement n'a pas à être modifié). Pour savoir si l'adresse demandée existe en mémoire ou non, à savoir que la page associée est chargée, le MMU utilise le bit de présence dans la table des pages (en fait l'entrée correspondant à la page est chargée au niveau du MMU et examinée).

Lorsque le bit de présence indique que la table n'est pas chargée en mémoire centrale, le MMU génère un défaut de page qui, comme son nom l'indique, signale au système d'exploitation qu'il doit charger la page en question afin de terminer la conversion de l'adresse. On a un défaut de page au début de l'exécution, avant que la page ne soit chargée ou bien si elle a été retirée de la mémoire car il n'y a plus assez de place.

Fig. 49. Fonctionnement du MMU

Le système d'exploitation appelle une procédure pour obtenir un emplacement de case libre. Si ce numéro lui est rendu, il obtient, grâce à la table des pages, l'adresse sur le disque dur de la page à charger. Le cas échéant si aucune case n'est libre, il procède à un remplacement de page, à savoir qu'il déroule un algorithme qui lui indique quelle page retirer afin d'en réquisitionner la case.

7.4 La pagination multi-niveaux

Le principal inconvénient des systèmes paginés est la taille gigantesque de la table des pages. De plus, chaque programme a sa propre table qui doit être chargée en mémoire, autrement le système d'exploitation n'a pas accès au bit de présence ni à l'adresse sur le disque des pages. La solution a été proposée avec la pagination multi-niveaux où la table des pages est décomposée en petites tables de taille raisonnable qui sont chargées au fur et à mesure que le système en a besoin. On peut souligner deux intérêts à cette organisation : partant du fait qu'une adresse n'appartient à l'espace d'adressage d'un programme que si elle est utilisée par le programme, les tables sont allouées au fur et à mesure que les adresses qu'elles comportent sont utilisées. Par ailleurs on ne charge à partir du disque que les tables dont on se sert, ainsi on charge à la demande ce qui évite d'avoir toutes les tables en mémoire centrale. On dit alors que la table des pages est elle-même paginée.

La figure n° 50 suivante illustre comment organiser en 2 niveaux des pages virtuelles écrites sur 20 bits.

Fig. 50. Exemple de pagination à deux niveaux

La table du premier niveau comporte 1024 entrées et permet d'adresser autant de tables de pages ($1024 = 2^{10}$). Ainsi les 10 premiers bits de l'adresse logique sont considérés comme index dans cette table.

Une fois qu'on a accédé à l'entrée adéquate, on trouve l'adresse de la case mémoire où se trouve la table des pages (de second niveau). Les 10 autres bits sont alors utilisés comme index dans cette table et l'entrée nous délivre la case mémoire associée à la page virtuelle. Les cases ayant une taille de 4 KO soit 2^{12} , on retrouve les 32 bits de l'adresse logique ($10 + 10 + 12$).

Prenons l'exemple de l'adresse virtuelle 5269875 que l'on écrit 506973 en hexadécimal. Les 12 bits de poids faible donnent un déplacement de 973. Le nombre 506 en hexadécimal s'écrit sur 12 bits en binaire $(010100000110)_2$, si nous retenons les 10 bits correspondants à l'index dans la seconde table, nous obtenons un numéro de page = 100000110 soit l'entrée 262 dans la table de deuxième niveau et les 2 bits restants $(01)_2$ l'entrée n° 1 dans la table de premier niveau.

La pagination à 2 niveaux ne résout pas définitivement le problème de la taille de la table des pages. C'est pour cette raison que les systèmes d'exploitation utilisent des tables à plusieurs niveaux généralement 3 ou 4. Il va de soi que le nombre de niveaux est décidé par le matériel, donc lié au MMU utilisé pour gérer la pagination. Certains processeurs SPARC utilisent une pagination à 3 niveaux.

La pagination aisée a commencé avec les processeurs MOTOROLA 68000 plus précisément 68030 où le nombre de niveaux est paramétrable (entre 1 et 4). Il est également possible pour le système d'exploitation de fixer la taille des pages, donc le nombre de bits pour le déplacement, ainsi que le nombre de bits pour chaque niveau.

7.5 La mémoire associative

Cette mémoire est destinée à accélérer la pagination. Les constructeurs ont proposé la mémoire associative afin d'y conserver les adresses physiques, ce qui évite de passer par la table des pages. En effet on a constaté que les programmes font souvent référence aux mêmes adresses ou à des adresses voisines (principe de la localité dans le temps, voir la section 6.4) La mémoire associative un composant matériel qui associe les adresses physiques aux adresses virtuelles sans passer par la table des pages. Ce composant intégré au MMU dispose d'un petit nombre d'entrées (entre 8 et quelques centaines) et une entrée est semblable à celle d'une table des pages. La différence est que la recherche dans ce tableau se fait en parallèle sur toutes les entrées, donc en une seule fois. Cette mémoire est un cache spécial de traduction d'adresses appelé TLB (tampon de traduction anticipé). La gestion de ce cache est associative (on peut utiliser n'importe quelle entrée), c'est pour cela que l'on parle de mémoire associative.

Si la page n'est pas trouvée, le MMU a recours à la table des pages (ce qui constitue une perte de temps, mais toute solution présenté des avantages et des inconvénients) et une fois le numéro de la case trouvé, il extrait une entrée de la mémoire associative et y place la nouvelle page référencée. Cette mémoire est très intéressante pour le cas où l'unité centrale est en train d'exécuter une boucle (n'oublions pas que le MMU traduit les adresses plusieurs fois pour une même instruction). L'idéal serait d'avoir une grande mémoire associative mais on est hélas limité par les capacités du composant matériel.

7.6 Inconvénients des systèmes paginés

Les systèmes paginés présentent les inconvénients suivants :

§ La fragmentation interne : Elle désigne l'espace non utilisé au sein d'une page. La fragmentation interne est une des principales caractéristiques des systèmes paginés. En effet, le système doit espacer les adresses de code, de pile... afin que ces derniers puissent croître. En conséquence, des adresses sont inutilisées au sein d'une page. Par ailleurs, une page placée en mémoire occupe la case en entier, même si elle ne comporte que 2 adresses affectées.

§ La taille de la table des pages est énorme : Nous avons expliqué plus haut que la table des pages comporte un million d'entrées. Si chaque entrée nécessite 4 octets, 4 MO de la mémoire centrale sont nécessaires afin de charger cette table. Or tout programme a sa propre table des pages. La pagination multi-niveaux résout en partie le problème puisqu'elle charge les tables au fur et à mesure que le système d'exploitation s'en sert.

§ Le partage de code est difficile à mettre en œuvre dans un système paginé : Le contenu d'une page est généralement inconnu du programmeur, il lui est donc difficile de partager une page.

§ La protection des objets est également délicate à réaliser pour les mêmes raisons de transparence. Le programmeur ignore l'emplacement des objets à protéger, son seul recours est de protéger une page entière ou plutôt un ensemble de pages (en utilisant les bits de protection au niveau de la table des pages) car le contenu d'une page lui est inconnu.

8 - Les systèmes segmentés

⇒ Sommaire de la section :

- ✓ La segmentation
 - ✓ Motivation de la segmentation
 - ✓ La table des segments
 - ✓ Conversion des adresses
 - ✓ Inconvénients des systèmes segmentés
-

8.1 La segmentation

A la différence des systèmes paginés, la mémoire ici est allouée par partitions de tailles variables appelés segments. L'ensemble des segments est généré à la compilation. Un segment a une taille qui peut varier (croître ou décroître) au cours de l'exécution. Il comporte des adresses entre 0 et sa taille actuelle.

Voyons à présent l'intérêt des systèmes segmentés. Prenons l'exemple courant de la phase de compilation. Le compilateur génère :

A la différence des systèmes paginés, la mémoire ici est allouée par partitions de tailles variables appelés segments. L'ensemble des segments est généré à la compilation. Un segment a une taille qui peut varier (croître ou décroître) au cours de l'exécution. Il comporte des adresses entre 0 et sa taille actuelle.

§ des adresses pour les variables contenues dans la table des symboles,

§ des adresses de constantes contenues dans la table des constantes,

§ des adresses pour la pile

§ des adresses pour le code

Chaque ensemble d'adresses est placé en mémoire dans une partition à part, les adresses concernant le code sont dans une partition, celles de la pile sont dans une autre zone.

Supposons à présent que le système adopte une taille fixe pour chacune de ces zones. Les partitions étant placées consécutivement dans l'espace d'adressage logique, il est évident que si on ne prévoit pas suffisamment d'adresses pour une zone donnée, la compilation va s'arrêter alors qu'il reste des adresses à côté des autres zones. Par ailleurs si on espace énormément les zones, l'espace entre elles est perdu. Pour cette raison dans les systèmes paginés on a de la fragmentation interne due à l'espace réservé pour les adresses générées par le compilateur. La solution est proposée par l'adoption des segments qui sont des partitions à taille variable.

Tout comme les systèmes partagés, un segment est chargé du disque vers la mémoire centrale pour être placé à une adresse précise pouvant le contenir car de taille suffisante. Si le segment

a besoin de croître, il le fera sur l'espace avoisinant. Dans un espace d'adressage donné il n'y a aucune restriction sur le nombre de segments. La différence entre un segment et une page est très nettement illustrée dans la figure n° 51 qui suit. On y observe que dans chaque segment les adresses démarrent à 0 et qu'il n'y a aucun lien entre les segments logiques d'un programme contrairement aux pages qui elles, se suivent.



Fig. 51. Placement de segments en mémoire centrale

L'intérêt de ce mode de gestion est que le segment ne s'élargit que si l'adresse est affectée. A titre d'exemple nous citons la taille du segment de pile qui s'élargit et rétrécit en fonction des appels de fonctions.

Le segment est une solution efficace mais il doit être géré par le programmeur. Généralement il contient des objets d'un seul type : procédures ou variables ou pile..ce qui a pour avantage de séparer les données et les programmes dans des espaces indépendants. Le fait qu'il soit géré par le programmeur rend son utilisation délicate mais les avantages sont multiples.

8.2 Motivation de la segmentation

La segmentation a été développée pour pallier les inconvénients de la pagination. Néanmoins, elle présente de nombreux avantages, comparée à la pagination :

§ Le premier avantage concerne la protection, le programmeur sait ce que contient son segment, il peut décider d'en interdire l'accès en écriture afin de le protéger. Il peut également décider de le partager pour que plusieurs processus puissent l'exécuter. Dans les systèmes paginés, ces derniers sont difficiles à mettre en œuvre car on ignore le contenu d'une page.

§ En cas de modification du programme, seules les adresses appartenant au segment ayant été modifié le seront. Le rajout de code à une procédure entraîne le décalage des adresses de ce segment mais les autres segments ne sont pas affectés.

§ La segmentation apporte une solution à la fragmentation interne caractéristique des systèmes paginés puisque le segment ne contient que l'espace des adresses réellement affectées.

8.3 La table des segments

C'est la structure de données qui décrit tous les segments d'un processus donné. Elle comporte une entrée par segment au niveau de laquelle nous trouvons, la base du segment, sa taille mais également d'autres informations utiles pour la conversion et le remplacement. Ainsi un bit de présence est nécessaire, également un bit de référence et un bit de modification.

Une entrée de la table des segments décrit à l'indice i le segment numéro i par la donnée de deux valeurs :

Base B	Taille T
--------	----------

La base est l'adresse de début du segment et la taille la longueur ou la taille du segment. Une adresse (s,d) sera convertie en $B + d$.

8.4 Conversion des adresses

Le compilateur génère désormais des adresses appartenant à des segments. A la différence des systèmes paginés, l'adresse logique a la forme (s,d) : numéro du segment et adresse dans le segment (on parle d'espace logique a deux dimensions). La procédure de conversion suit le cheminement suivant : on commence par retrouver l'emplacement du segment en mémoire centrale grâce à une table des segments. Ensuite au sein du segment, on effectue un déplacement de d pour accéder à l'adresse physique. Ces étapes sont illustrées à travers la figure n° 52.

Chaque programme dispose de sa table des segments. De même que dans un système paginé, on vérifie d'abord si le segment est en mémoire. En revanche, On distingue un test supplémentaire qu'il faut réaliser, à savoir qu'on vérifie que le déplacement est bien inférieur à la taille du segment autrement la conversion retournerait une adresse contenue dans un autre segment qui peut ne pas appartenir à ce programme.

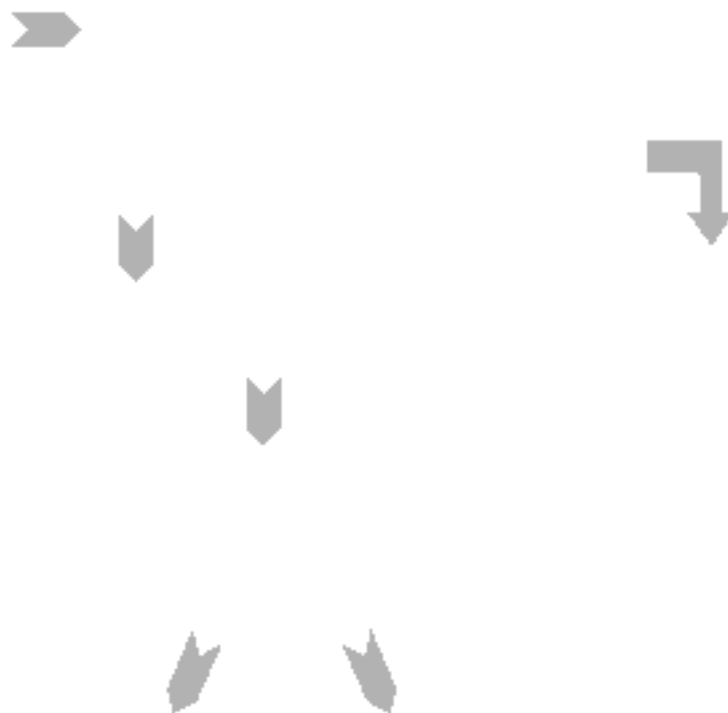


Fig. 52. Conversion d'une adresse dans un système segmenté

On peut passer d'un système segmenté à un système paginé si la taille T est la même pour tous les segments, on a alors des pages de taille T .

8.5 Inconvénients des systèmes segmentés

Les systèmes segmentés présentent certains inconvénients qui sont toutefois moins nombreux que ceux des systèmes paginés. La première difficulté réside dans le choix des segments qui est à la charge du programmeur. Un minimum de connaissances en programmation structurée est donc requis.

Par ailleurs, on retrouve la fragmentation dans sa forme externe. Il va de soi qu'elle ne peut pas être interne puisqu'un segment ne peut contenir que les adresses qui sont réellement affectées. L'espace perdu entre les segments placés en mémoire appelé miettes représente la fragmentation externe.

TD système d'exploitation N°5

Exercice 1

On considère une machine M dans dont la mémoire centrale est partitionnée en 5 partitions de tailles respectives de 100, 500, 200, 300 et 600 K. On suppose que nous avons 4 processus que nous voulons exécuter sur la machine M dans l'ordre P1, P2, P3, P4 et qui ont une taille respective de 212, 417, 112 et 426 k.

Comment seront placés ce processus en mémoire centrale dans le cas où l'algorithme est

- 1) First-fit
- 2) Best-fit
- 3) Worst-fit

Exercice 2

On considère la table de segment suivante

Segment	Base	Longueur
0	219	600
1	2300	14
2	90	100
3	1327	580
4	1952	96

Quelles sont les adresses physiques pour les adresses logiques suivantes

- 1) 0,430
- 2) 0,10
- 3) 2,500

4) 3,400

5) 4,112

Exercice 3

soit un espace adresse logique de huit pages de 1024 mots chacune, en correspondance avec une mémoire physique de 32 cadres de pages

Combien de bits existent-ils dans l'adresse logique

Combien de bits existent-ils dans l'adresse physique

Exercice 4

Soit un système de pagination avec une table de pages stockée en mémoire

1) Si une mémoire prend 200 ns, combien de temps prend une référence mémoire paginée

2) Si on ajoute des registres associatifs et que 75% de toutes les références à la table de pages se trouvent dans les registres associatifs, quel est le temps effectif d'accès à la mémoire en supposant que trouver une entrée dans la table de pages dans les registres associatifs ne prend pas de temps si l'entrée se trouve là dedans.

Exercice 5

Certains systèmes à va-et-vient tentent d'éliminer la fragmentation externe au moyen de compactage.

Supposez qu'un ordinateur de 1M de mémoire utilisateur effectue un compactage toutes les secondes. S'il faut $\frac{1}{2}$ microseconde pour copier un octet et si la taille moyenne des zones libres est égale à 0.4 fois celle des segments, quelle fraction du temps, quelle fraction du temps processeur faut-il pour le compactage ?

Exercice 6

Soit la chaîne de référence suivante

1,2,3,4,2,1,5,6,2,1,2,3,7,6,3,2,1,2,3,6

on se propose de calculer le nombre de défaut de page avec les algorithmes FIFO, de l'horloge et LRU.

On rappelle qu'initialement aucune page n'est chargée en mémoire. La mémoire centrale est composée de 5 cases.

Schématisez à chaque étape le contenu de la liste de remplacement et précisez pour chaque algorithme le nombre de défaut de pages et les pages qui en sont responsable.

Exercice 7

On dispose d'un ordinateur dont le système de mémoire virtuelle compte 3 cadres de pages (*frames*) pour un espace virtuel de 8 pages (numérotées de 0 à 7).

On suppose que les trois cadres sont initialement vides et que les pages sont appelées dans l'ordre suivant au cours de l'exécution des programmes partageant l'accès au processeur :

2-6-3-4-6-5-6-7-4-5-6-5-4-3-4-6-3-2

Indiquez tout au long de la séquence d'exécution quelles pages sont présentées dans un cadre de la mémoire physique et le nombre de fautes de page lorsque l'algorithme de remplacement de pages est :

1. l'algorithme de remplacement optimal, (6)

2. l'algorithme L R U.(10)

Exercice 8

On suppose que la **mémoire** (non paginée) d'un ordinateur présente des partitions d'espace libre de 100K, 500K, 200K, 400K et 600K (dans cet ordre). Comment serait alloué l'espace mémoire si des demandes étaient faites par des processus pour des partitions (contiguës) de 50K, 125K, 440K, 320K et 130K (dans cet ordre)

(a) avec un algorithme de *best-fit* ?

(b) avec un algorithme de *first-fit* ?

(c) avec un algorithme de *worst-fit* ?

Quel algorithme utilise ici la mémoire de la façon la plus judicieuse ? Pourquoi ?

Exercice 9

On considère un système disposant de 16 MB de mémoire réelle non paginée dont 4 MB sont

occupés par le système d'exploitation. On suppose que l'exécution de chaque processus se

compose d'une seule giclée d'UCT (possiblement décomposée en séquences plus courtes, par suite des activités du répartiteur de bas niveau) suivie d'une période pendant laquelle sont effectuées des E/S, après quoi le processus termine son exécution en libérant sa partition. On suppose de plus que les processus n'attendent pas avant d'effectuer leurs E/S (par exemple, ils utilisent tous un périphérique différent).

Le tableau suivant donne un exemple de séquence de tâches pour le système :

instant (ms)	Processus	Taille Giclée d'UCT	Durée E/S		
0	A	6 MB	200 ms	200 ms	
50	B	3 MB	300 ms	300 ms	
100	C	5 MB	200 ms	300 ms	
150	D	1 MB	300 ms	200 ms	
300	E	3 MB	100 ms		300 ms
500	F	4 MB	200 ms	100 ms	
500	G	1 MB	200 ms	100 ms	

La première colonne donne l'instant de soumission d'un processus (au répartiteur de haut niveau). La seconde et troisième colonnes donnent le nom du processus et la taille de la partition mémoire qu'il demande au système. La quatrième colonne donne la durée de la giclée d'UCT et la cinquième colonne donne la durée de l'exécution des E/S pour ce processus.

Par exemple, la demande d'exécution pour le processus *C* est soumise à $t = 100$ ms au répartiteur de

haut niveau. Lorsque ce processus sera chargé en mémoire, il y occupera un espace adresse de 5

NB. Sa giclée d'UCT durera 200 ms (éventuellement décomposée en plusieurs morceaux, par suite de l'action du répartiteur de bas niveau), puis *C* fera une demande d'E/S dont l'exécution occupera 300 ms, après quoi il terminera son exécution en libérant sa partition.

Question

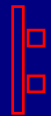
Donnez les états d'occupation de la mémoire aux différentes étapes de traitement de ces processus jusqu'à complétion de leur exécution, sous les hypothèses suivantes :

- Les partitions sont de taille variable ;
- Le gestionnaire de la mémoire utilise un algorithme de *Best Fit* pour l'allocation des trous ;
- Le répartiteur de haut niveau fonctionne selon PAPS ;
- Le répartiteur de bas niveau utilise l'algorithme du tourniquet avec un quantum de 100 ms ;
- Les changements de contexte sont instantanés.

Conception de Systèmes de Gestion de la Mémoire Centrale

Objectifs

1. Conception de systèmes paginés
 2. Conception des systèmes segmentés
 3. La segmentation avec pagination
 4. TD6
-



1 - Conception de systèmes paginés

⇒ Sommaire de la section :

- ✓ Allocation et libération de cases
 - ✓ Remplacement de pages
 - ✓ A propos de chargement de pages
-

1.1 Allocation et libération de cases

Le système d'exploitation peut utiliser une structure statique ou dynamique pour conserver les numéros de cases libres. La structure statique sera une table de bits, où chaque bit représentera une case. Il faut parcourir la table à la recherche de bits consécutifs pour un processus qui nécessite 3 cases par exemple, autrement il sera alloué de manière non contiguë. Cette table aura au maximum la taille de la mémoire centrale divisée par la taille d'une page. Exemple : pour gérer 128 MO il faut $128 * 1024$ bits soit 4 KO. La position du bit donne le numéro de la case qui, une fois multiplié par la taille d'une case, donne l'adresse de début de la case. Cette organisation statique est donc assez satisfaisante.

En adoptant une structure dynamique, il faut lier entre eux les numéros de cases libres. Cette liste peut être conservée dans un ordre trié afin que les processus ne soit pas trop éparpillés en mémoire.

En d'autres termes, étant donné que les cases ont la même taille que les pages, il n'est pas nécessaire d'effectuer une recherche minutieuse de la case la plus optimale à allouer. Certes, des optimisations peuvent être effectuées pour réaliser de l'allocation contiguë autant que possible mais ce traitement reste simple.

1.2 Remplacement de pages

La mémoire étant de taille faible, elle peut rapidement être saturée. Pour cette raison, on a souvent recours au remplacement de pages, d'autant plus qu'on applique la mémoire virtuelle qui introduit un nombre très élevé de pages. Nous rappelons qu'une page est remplacée si le processeur réclame une information située dans une page qui se trouve être non chargée en mémoire centrale et qu'il n'existe aucune case libre pour y charger cette dernière. Assez souvent il peut y avoir confusion entre défaut de page et remplacement de page. Le défaut de page peut entraîner un remplacement si le système d'exploitation ne trouve aucune case de libre.

Lorsqu'il ne reste plus de cases libres, le système d'exploitation va repérer une case et examiner son bit de modification. Une valeur à 1 indique que cette page a été utilisée en écriture. Il va ainsi la sauvegarder sur le disque et charger la page qui fait défaut à l'emplacement de cette case. Si le bit de modification vaut 0, la page est chargée dans cette case en écrasant l'ancienne page. Le fait d'utiliser de la mémoire virtuelle augmente le nombre de défauts de pages puisque les deux tailles de mémoire ne sont pas comparables. Nous verrons plus loin que le système peut anticiper et remplacer des pages avant d'atteindre l'état de saturation de la mémoire dans l'optique d'éviter ce que l'on appelle l'écroulement du système.

Il est clair que le choix de la page à enlever de la mémoire peut influencer les performances du système. Prenons le cas d'une page appartenant à une boucle qui est sélectionnée par le système. Sitôt la page retirée et sauvegardée sur le disque, il faudra de nouveau la charger en mémoire. On aurait pu gagner du temps en choisissant une page plus appropriée, par exemple une page non utilisée récemment.

1.2.1 Principe d'un algorithme de remplacement

On distingue plusieurs stratégies de remplacement, mais elles se rejoignent toutes au niveau du principe et des structures de données.

Il faut préciser que ces algorithmes sont très coûteux car ils nécessitent que les numéros de pages en mémoire soient organisés sous la forme d'une liste chaînée. Il est impensable d'adopter une structure de données statique.

Cette liste appelée liste des pages contient des éléments chaînés simplement ou doublement. Chaque élément contient un numéro de page. Le système d'exploitation peut éventuellement inclure dans la structure d'un élément de la liste le numéro du programme auquel la page appartient. En effet il peut utiliser une liste de remplacement globale pour tous les programmes en mémoire ou construire une liste par programme. L'algorithme de remplacement organise la liste de telle sorte que les éléments en tête désignent les pages à retirer. Il est sollicité à deux reprises : pour le remplacement d'une page et chaque fois qu'une page est utilisée.

Le remplacement d'une page consiste à retirer la page qui est en tête de liste. En revanche selon la stratégie appliquée, chaque fois qu'une page est utilisée l'algorithme peut être amené à la changer de position au sein de la liste. Il l'emmènera vers la queue afin d'éviter qu'elle ne soit retirée.

1.2.2 Remplacement local ou global ?

Il peut sembler implicite que l'algorithme de remplacement de page s'applique au niveau local et que le système d'exploitation exécute un algorithme par programme. Ceci est tout à fait concevable puisqu'il suffit de sauvegarder un pointeur sur la liste des pages qu'utilise l'algorithme. Il semble également logique qu'on ne retire pas les pages d'un autre programme pour continuer à exécuter celui qui nécessite le chargement en cours.

En réalité On distingue deux modes de remplacement : local et global. Dans le mode global, on peut supprimer les pages d'un autre programme pour exécuter le processus qui a provoqué le défaut de page.

Dans le mode local, chaque programme a un nombre de cases fixes qui lui sont allouées et les concepteurs doivent décider de la valeur de ce nombre. Le nombre de cases peut être choisi de manière équitable ou en fonction de la taille du programme. Toutefois, si on alloue peu de cases au programme, son nombre de défauts de pages va augmenter et ralentir son exécution, et le cas contraire, il va pénaliser un second programme qui restera en attente de case libre alors que le premier en a plus que nécessaire.

Le système Windows NT applique le mode de remplacement local. Le système Unix quant à lui applique un remplacement global. En règle générale, le remplacement au niveau global

donne de meilleurs résultats et c'est la méthode la plus courante, mais il est plus difficile à mettre en oeuvre.

On distingue plusieurs algorithmes de remplacement qui implémentent diverses stratégies. On peut retirer une page choisie au hasard mais au risque de la recharger immédiatement après, ce qui entraîne une perte de temps inutile. Nous présentons dans les paragraphes ci-dessous plusieurs stratégies de remplacement. Certaines sont simples d'autres plus recherchées et visent à retirer la page qui pénalise le moins l'exécution.

1.2.3 L'algorithme de remplacement de page optimal

Cet algorithme se base sur un principe très simple, mais impossible en pratique à mettre en oeuvre. L'idée est d'associer à chaque page le nombre d'instructions à exécuter avant de la référencer. Par exemple, on peut supposer que la page n°1 sera appelée après que la page n° 0 soit exécutée en entier. Si les pages ont une taille de 4 KO et qu'une instruction s'écrit sur 4 octets, une page contient environ 1000 instructions. On associe ainsi la valeur 1000 à la page n° 1, la valeur 2000 à la page n° 2 et la valeur 10000 à la page n° 10. Lorsque le système est amené à effectuer un remplacement, en retirant la page de plus grand numéro, on est assuré de retirer une page dont les instructions ne seront exécutées prochainement. Malheureusement cette supposition s'écroule rapidement dès lors qu'une page contient une ou plusieurs boucles ou qu'on y effectue un appel de procédure. Le système ne peut pas connaître avec précision l'instant où les différentes pages seront référencées à moins d'exécuter une fois le programme.

La solution est apportée par l'utilisation d'un simulateur qui exécute le programme et mémorise au bout de combien d'instructions chaque page est référencée.

L'algorithme optimal est impossible à mettre en oeuvre, il est toutefois utilisé pour l'évaluation des performances des algorithmes de remplacement.

--->Simuler l'algorithme [optimal](#).

1.2.4 L'algorithme de remplacement d'une page selon FIFO

La politique FIFO peut être appliquée au remplacement d'une page. La page à remplacer est celle qui a été chargée en mémoire depuis le plus long temps. Le remplacement concerne la page en tête et lors d'un chargement on met les pages au fur et à mesure en dernière position.

La figure n° 53 représente des pages selon FIFO.

Cet algorithme consomme très peu de temps processeur mais il n'est pas appliqué car il n'a aucune considération pour les pages. Ainsi il peut retirer une page en cours d'utilisation.

Fig. 53. Remplacement de pages avec FIFO

--->Simuler l'algorithme [FIFO](#).

1.2.5 L'algorithme de la seconde chance

Cet algorithme est une version améliorée de FIFO, à savoir qu'il en conserve les avantages en occupant un temps processeur faible et en pallie les inconvénients en donnant une seconde chance à la page en tête. Selon FIFO même si la page 2 est réutilisée, elle ne change pas de position dans la liste et sera enlevée au prochain remplacement. Avant de retirer la page en tête, cet algorithme examine son bit de référence en accédant à la table des pages. Une valeur du bit à 1 indique que la page a été utilisée récemment. Elle est retirée et placée en queue et son bit R réinitialisé.

1.2.6 L'algorithme de l'horloge

Cet algorithme est identique au précédent sauf qu'il utilise une structure de liste chaînée circulaire. Cette organisation évite que les pages utilisées soient déplacées vers la queue. Il suffit de déplacer le pointeur tête sur l'élément suivant pour que la page qui était en tête soit considérée en queue.

Fig. 54. Remplacement de page avec l'algorithme de l'horloge

1.2.7 L'algorithme de remplacement d'une page non récemment utilisée NRU

L'algorithme NRU (Not Recently Used) utilise les bits de *Référence* et de *modification*. Si une page n'a pas été référencée, le bit de référence sera à 0. Par contre si la page l'a été, afin de différencier les pages récemment utilisées des autres, les systèmes d'exploitation utilisent les interruptions horloge afin de remettre ce bit à 0. Ainsi entre 2 interruptions horloge, les pages qui ont été utilisées (accédées en lecture ou écriture) auront leur bit à 1 et ne seront pas choisies par l'algorithme en cas de remplacement à effectuer.

L'algorithme tient aussi compte du bit de modification car une page peut être référencée ensuite modifiée et ensuite R remis à zéro car elle est restée quelques temps non utilisée. Si jamais cette page doit être enlevée de la mémoire, elle doit forcément être sauvegardée sur le disque. Dans ce cas, il vaut mieux choisir une page qui a été non référencée et non modifiée qu'une page qui a été non référencée et modifiée.

Les pages sont réparties en 4 classes :

§ Classe 0 : non référencée non modifiée

§ Classe 1 : non référencée modifiée

§ Classe 2 : référencée non modifiée

§ Classe 3 : référencée modifiée

L'algorithme NRU choisit la classe non vide de plus petit numéro. Cet algorithme a des

performances assez satisfaisantes.

1.2.8 L'algorithme de remplacement de la page la moins récemment utilisée LRU

La stratégie LRU (Least Recently Used) est basée sur la supposition suivante : une page qui a été utilisée plusieurs fois récemment va encore l'être à l'avenir et une qui ne l'a pas été ne le sera pas. Supposition qui n'est pas très plausible mais qui se tient si on considère le cas des boucles, des variables et des tableaux.

LRU place en tête de liste les numéros des pages les plus utilisées et en fin celles qui le sont moins. Chaque fois qu'une page est référencée dans une instruction, il faut mettre à jour la liste LRU, à savoir rechercher le numéro de page et mettre l'élément correspondant en début de liste d'où le coût associé à l'algorithme.

Cet algorithme est en théorie le meilleur pour implémenter le remplacement des pages puisqu'il retire la page la moins pénalisante et celle qui risque fort peu d'être utilisée à nouveau. C'est également le plus coûteux étant donné les mises à jour à chaque accès à une page. Néanmoins il est utilisé pour la gestion de la mémoire cache où il fournit de bonnes performances vu la faible taille de cette dernière.

Les opérations sur les listes chaînées étant très coûteuses, les concepteurs ont eu recours au matériel pour implanter cette approche. Le matériel doit mémoriser une matrice n fois n avec n le nombre de cases. La matrice est initialisée à 0. Lorsqu'une page i est référencée, le matériel met à 1 la rangée $n^\circ i$ et ensuite à 0 la colonne i . C'est assez surprenant mais la page la moins récemment référencée est celle dont la rangée a la plus petite valeur binaire.

Soit la chaîne de référence aux pages suivante (ordre dans lequel les pages sont accédées) 0 2 1 3 1 2 3. Les tableaux ci-dessous illustrent les étapes 1 et 7 (tableau 1 pour l'étape 1 et le tableau 2 pour l'étape 7)

--->Simuler l'algorithme [LRU](#).

1.2.9 L'algorithme du vieillissement

C'est une amélioration logicielle de l'algorithme LRU en vu de le rendre moins coûteux. A chaque entrée d'une page dans la table des pages, on rajoute un champ compteur qui est incrémenté de 1 quand une page est référencée. En réalité, R est rajouté au compteur à chaque interruption de l'horloge et R est remis à 0 ensuite.

Quand il se produit un défaut de page, si on remplace la page dont le compteur est le plus faible on risque d'avoir la situation suivante : une page accédée à maintes reprises il y a quelques temps sera considérée plus récente qu'une page accédée moins fréquemment et plus récemment. La solution est proposée avec un mécanisme de vieillissement des compteurs.

Le champ compteur est décalé d'un bit vers la droite avant d'y rajouter R. La valeur de R est rajoutée au bit de poids fort donc à gauche (en fait cela revient à le placer). Si l'on regarde de près le principe de l'algorithme, on remarque qu'en comptant à partir de la gauche, si le bit de poids fort (p) est à 0, la page n'a pas été référencée depuis le dernier top, si le bit de poids (p-1) est à 0 également cela veut dire que la page n'a pas été référencée depuis 2 tops d'horloge etc. Il est donc logique de retirer la page qui a la plus petite valeur du compteur.

La figure n° 55 ci-dessous, donne les valeurs des bits R après 5 tops d'horloge et les valeurs des compteurs :

	Top0	Top1	Top2	Top3	Top4
Bit R pour les 5 pages	101011	110010	110101	100010	011000

Page0	10000000	11000000	11100000	11110000	01111000
Page1	00000000	10000000	11000000	01100000	10110000
Page2	10000000	01000000	00100000	00010000	10001000
Page3	00000000	00000000	10000000	01000000	00100000
Page4	10000000	11000000	01100000	10110000	01011000
Page5	10000000	01000000	10100000	01010000	00101000

Fig. 55. Evolution de la valeur du compteur dans l'algorithme du vieillissement

1.2.10 Vous pouvez [simuler](#) votre propre exemple.

1.3 A propos de chargement de pages

On recense deux modes appliqués par les systèmes d'exploitation pour le **chargement** des pages :

- Le chargement à la demande : le système d'exploitation attend qu'un défaut de page se produit pour aller charger cette page.
- Le chargement de l'espace de travail : c'est une sorte de pagination anticipée qui vise à réduire le nombre de défauts de pages. Elle se base sur la prédiction des numéros de pages à charger.

Le préchargement se base sur 2 principes de localité :

- La **localité dans le temps** suppose que les pages qui ont été référencées dernièrement vont l'être à nouveau dans le futur. C'est vérifiable pour les boucles, les tableaux, les procédures récursives...
- La **localité dans l'espace** suppose que les références des pages sont souvent voisines, c'est-à-dire que le programme utilise des adresses consécutives.

Quand un processus est interrompu et qu'il est déplacé sur le disque dur par ce qu'il ne reste plus de place en mémoire, le système d'exploitation peut sauvegarder les numéros de pages utilisées et en appliquant le principe de la localité connaître les pages qui risquent de l'être. Ainsi l'espace de travail pourra être chargé avec le processus quand il sera ré exécuté.

2 - Conception des systèmes segmentés

⇒ Sommaire de la section :

- ✓ Allocation et récupération des segments
- ✓ Récupération de segments
- ✓ Remplacement de segment

2.1 Allocation et récupération des segments

Dans les systèmes paginés, le principal traitement est basé sur les politiques de remplacement de pages. Nous allons voir que dans les systèmes segmentés, c'est l'allocation des segments qui représente le traitement le plus complexe.

Tout d'abord, en ce qui concerne l'étape de recherche d'une zone mémoire libre, dans un système paginé il suffit de trouver une case libre alors que dans un système segmenté, il faut rechercher un segment de taille suffisante. On suppose donc que les structures gérant l'espace libre en mémoire ne sont pas les mêmes puisqu'il faut connaître la taille de chaque zone.

2.1.1 Structure de données pour la gestion des segments libres en mémoire

On distingue plusieurs manières de gérer les espaces libres en mémoire centrale, les plus connues étant la liste chaînée et la table de bits introduites lors de la présentation de la gestion du disque dur. La table de bits ne peut s'appliquer pour des systèmes segmentés car on a besoin de connaître les adresses de début et de fin du segment en plus de son occupation. Par ailleurs, une structure de table c'est-à-dire statique n'est pas envisageable car on ignore le nombre de segments en mémoire. Initialement c'est un grand espace qui est fractionné au fur et à mesure que des segments logiques y sont placés et libérés.

Quant à la liste chaînée, cette solution est celle retenue et elle peut avoir diverses implémentations, elle peut être triée par ordre de taille ou bien par contiguïté des adresses et bien entendu par ordre FIFO. Elle peut regrouper les segments libres et ceux occupés, ou bien on peut utiliser deux listes séparées, une pour les segments libres et une pour les segments occupés (liste utile pour le remplacement de segments).

Chaque organisation présente des avantages et des inconvénients :

§ Le choix d'une liste unique évite de retirer un élément de la liste des libres et de le placer dans celle des occupés et vice-versa mais au prix d'un parcours de toute la liste à chaque fois.

§ Le fait d'avoir deux listes permet de trier celles des segments libres et de trouver rapidement un segment de taille suffisante.

On ne peut dire qu'un choix est meilleur que l'autre, chacun ayant un temps processeur plus court au moment de l'allocation du segment ou au moment où il est rendu au système.

2.1.2 Algorithmes d'allocation de segment

Nous avons étudié dans la section concernant l'allocation contiguë différentes politiques pour le choix de la zone mémoire à allouer à un programme. Nous citons FIFO (première zone libre), meilleur ajustement (best fit). Tous ces algorithmes permettent d'allouer des segments et d'autres encore telle l'allocation par subdivision binaire qui alloue la première zone de taille puissance de 2 suffisante.

Le choix d'un algorithme est conditionné par l'ordre des segments dans la liste. Il est évident qu'un tri par ordre croissant permet d'appliquer facilement le best-fit (des études ont prouvé que ses résultats sont les meilleurs).

2.2 Récupération de segments

Dans un système segmenté la procédure de récupération de segment est plus complexe en effet à chaque fois qu'on récupère une zone libre, on insère un élément la décrivant dans la liste et on examine les voisins afin de voir s'il y a lieu d'effectuer une fusion pour obtenir un espace plus grand plutôt que des miettes réparties à travers la mémoire.

Fig. 56. Récupération de segments

Lorsqu'un segment SL est libéré, une fois l'emplacement au niveau duquel on va l'insérer déterminé, On distingue quatre traitements possibles :

§ Le segment SL a une adresse de début égale à l'adresse de fin du segment le précédent P.

L'adresse de fin de P reçoit la valeur de l'adresse de fin de SL

§ Le segment SL a une adresse de fin égale à l'adresse de début du segment le suivant S dans la liste. L'adresse de début de S reçoit la valeur de l'adresse de fin de SL.

§ L'adresse de début et de fin de SL correspondent avec les segments S et P. Les segments S et P doivent être fusionnés en un seul. Ainsi l'adresse de fin de P prend la valeur de l'adresse de fin de S et S est retiré de la liste.

§ Ni l'adresse de début ni celle de fin de SL ne concorde avec celles des segments voisins. Il est simplement inséré dans la liste.

2.3 Remplacement de segment

Le remplacement de segments également est très différent du remplacement de pages. Tout d'abord, il n'affecte pas tous les segments. Certains segments sont plus importants que d'autres et ne doivent pas être enlevés tels que les segments de code par rapport à certains segments de données. Il en est pratiquement de même dans les systèmes paginés, sauf que ceci est réalisé de manière transparente car on ne sait pas quelles pages occupe le segment code.

Par ailleurs contrairement aux pages, le remplacement peut concerner plusieurs segments. Il faut souligner à ce stade que la structure de la liste des segments conditionne le choix de l'algorithme de remplacement. On dispose d'une liste des segments en mémoire, appliquer FIFO revient à retirer le premier ou les n premiers segments si cela ne suffit pas. En revanche, avec une liste triée selon la taille, on peut connaître celui qui dispose d'une taille suffisante et ne retirer qu'un seul segment.

3 - La segmentation avec pagination

⇒ Sommaire de la section :

✓ Conversion des adresses

✓ La segmentation avec pagination du 386

3.1 Conversion des adresses

Une adresse logique a la forme (S, P, D) un numéro de segment, un numéro de page et un déplacement. Le numéro de segment permet de retrouver son adresse de début grâce à la table des segments. Le numéro de page permet en accédant à la table des pages de savoir si la page est en mémoire et auquel cas dans quelle case. Il faut toutefois préciser que chaque segment dispose de sa propre table des pages puisqu'il constitue un espace d'adressage à part, reste à trouver la table des pages pour un segment donné. Il est clair qu'elle doit être indiquée ou pointée au niveau de l'entrée décrivant le segment dans la table des segments. Fréquemment, la table des pages est située au début du segment. La figure n° 57 qui suit, illustre le déroulement de la conversion d'adresse.

Fig. 57. Accès à une adresse dans un système segmenté et paginé

Il peut arriver que la pagination à deux niveaux soit utilisée afin d'éviter des tables de pages trop volumineuses.

3.2 La segmentation avec pagination du 386

Cette section présente la gestion mémoire du processeur 80386.

L'Intel 80386

Le processeur 80386 applique également la segmentation et la pagination en offrant une unité de pagination et une de segmentation. Il a toutefois un fonctionnement plus complexe que Multics.

Le 386 peut adresser 16K segments. Chaque segment peut atteindre une taille de 2^{32} octets soit environ un milliard de mots (un mot = 32 bits). Les pages ont une taille de 4 KO.

Le 386 utilise deux tables pour décrire les segments et les pages. La première TDG (table des descripteurs globaux) décrit tous les programmes. La seconde TDL (table des descripteurs locaux) décrit les segments d'un programme.

Un numéro de segment s'écrit sur 16 bits organisés de la manière suivante : les 13 premiers bits indiquent son numéro réel, le bit suivant si c'est un segment local ou global, et les deux derniers sa protection.

Une fois l'appartenance du segment identifiée, il est extrait de la table. Une entrée d'une table de descripteurs a la structure suivante :

Fig. 58. Un descripteur de segment du processeur 386 sur 8 octets



TD système d'exploitation N°6

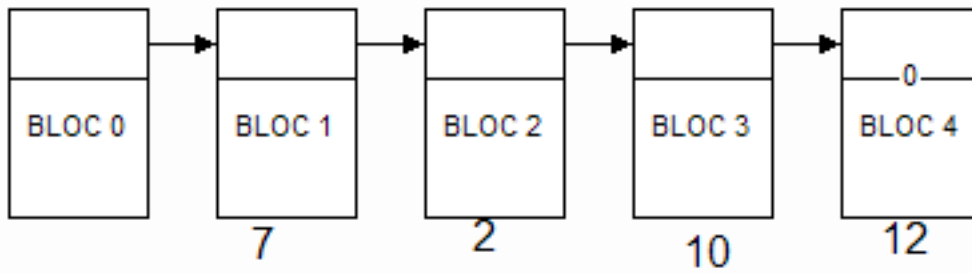
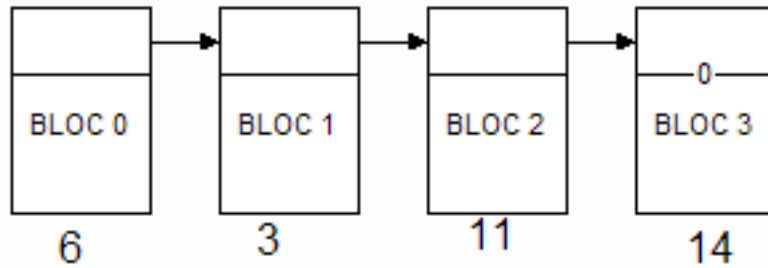
- 1) Expliquer le traitement entrepris suite à un défaut de page
- 2) Ecrivez les algorithmes d'allocation et de libération de cases
- 3) Ecrivez les algorithmes d'insertion et de retrait des pages pour chacun des algorithmes FIFO et LRU
- 4) Pourquoi la segmentation et la pagination sont-elles quelques fois combinées dans un seul schéma
- 5) Quand un segment peut-il appartenir à l'espace d'adressage de deux processus différents
- 6) Calculer les adresses logiques des adresses physiques suivantes : 1035, 4093, 4098, 6000. le mode de gestion est la pagination et les pages ont une taille de 4KO.

Numéro de la case	Numéro de la page contenu dans la case
0	4
1	6
2	15

- 7) On considère un système de mémoire secondaire (disque dur) présentant 160 pistes numérotées en ordre de l'intérieur vers l'extérieur de 0 à 159. On suppose que la tête de lecture/écriture se trouve placée à la verticale de la piste 50, que sa position précédente était en 80 et que des requêtes arrivent pour des accès aux pistes suivantes :

100 153 40 140 20 110 55 60 (dans cet ordre)

- Quel serait le déplacement total (en nombre de piste) de la tête de lecture/écriture après avoir accédé à toutes ces pistes si l'algorithme de planification des déplacements de la tête est :
 - le plus petit déplacement d'abord (*SSTF*),
 - un balayage en va-et-vient (*SCAN*),
 - un balayage circulaire (*C-SCAN*)..
- 8)** Laquelle de ces étapes doit être réalisée avant l'autre : le partitionnement ou le formatage d'un disque ?
- 9)** On considère un fichier texte 2500000 caractères ASCII (y compris les caractères de fin de ligne et de fin de fichier). Suite à un malencontreux accident, le inode de ce fichier est corrompu et le contenu des pointeurs indirects double et triple est détruit. Quelle proportion du contenu de ce fichier a-t-elle été perdue par suite de cet accident ?
- 10)** Avec UNIX, quelle taille maximale peut avoir un fichier, sachant que les blocs sont de 1Ko et que les adresses des blocs sont sur 4 octets ?
- 11)** On peut mémoriser l'espace libre sur le disque au moyen d'une liste des blocs libres ou d'une table de bits. Les adresses sur le disque requièrent D bits pour un disque de B blocs dont F sont libres, quelle est la condition nécessaire pour que la liste des blocs libres soit plus petite que la table de bits ? si D vaut 16 bits, exprimer la réponse sous d'un pourcentage de l'espace du disque qui doit être libre
- 12)** Soit un disque de 9.1 GO, si ce disque est formaté en utilisant des clusters de 32 KO, quelle quantité de mémoire serait réservée pour mémoriser l'espace libre ? Faites le calcul pour un système qui utilise un vecteur de bits et un autre qui emploie une liste chaînée de blocs spéciaux.
- 13)** Dessinez la FAT pour la figure ci-dessous.

FICHER A**FICHER B**

- 14)** Est-il intéressant de mémoriser la première partie de chaque fichier UNIX dans le même bloc que son nœud d'information ?
- 15)** Imaginer un système qui utilise une liste de blocs spéciaux afin de gérer l'espace libre. Si par mégarde on perd le pointeur sur cette liste, le système d'exploitation peut-il la reconstruire ? comment ?
- 16)** Lorsqu'un fichier est supprimé, les numéros des blocs sont-ils simplement rajoutées à ceux libres ou bien le contenu de chacun des blocs est-il effacé ?
- 17)** Soit un système qui désire autoriser 4990 personnes à lire un fichier et en interdire l'accès à 10 autres comment procéder ?

Le système des entrées-sorties

Objectifs

1. Fonctionnement des périphériques
 2. Entrées-Sorties synchrones et asynchrones
 3. Les entrées-sorties DMA (par accès direct à la mémoire)
 4. Schéma d'exécution d'une entrée-sortie
 5. Le sous-système d'entrée-sortie
 6. Etude de cas : le périphérique disque
-



1 - Fonctionnement des périphériques

Nous rappelons dans cette section les composants des périphériques, ainsi que leur fonctionnement de base. Ceci permettra dans un premier temps de discerner entre les parties mécaniques et celles électroniques dans un périphérique. Dans un second temps cela nous aidera à introduire certains termes relatifs au système d'entrée/sortie.

⇒ Sommaire de la section :

- ✓ Structure d'un périphérique
 - ✓ Les types de périphériques
 - ✓ Les contrôleurs de périphériques
 - ✓ Interaction entre le système d'exploitation et le contrôleur
-

1.1 Structure d'un périphérique

Un périphérique est composé du périphérique lui-même et de composants électroniques pour le commander donc d'une partie mécanique et d'une autre électronique. A titre d'exemple, dans un disque dur les plateaux, les têtes de lecture/écriture, le bras ainsi que le moteur constituent la partie mécanique. La partie électronique est appelée contrôleur de périphérique. On l'appelle ainsi car elle contrôle le périphérique à la place du processeur. Le contrôleur est une carte composée de circuits qu'on relie au périphérique. Sans contrôleur, le processeur devrait se charger lui-même de tous les périphériques. Un même contrôleur peut gérer plusieurs périphériques de même type.

Nous avons introduit au chapitre 1 l'utilité du contrôleur. Nous rappelons que la différence de vitesse qui existe entre le processeur et les périphériques en est à l'origine. Etant donné que le processeur ne peut pas rester à l'écoute pour savoir si le périphérique est prêt à lui envoyer une donnée ou à la recevoir, un contrôleur est associé à chaque périphérique et gère le dialogue avec l'unité centrale.

1.2 Les types de périphériques

Si nous essayons d'établir une classification des périphériques existants outre celle de périphériques d'entrée et de sortie, nous réalisons que la plupart des systèmes (tel Unix) recensent deux types de périphériques : les périphériques bloc et les périphériques caractère. C'est la possibilité de faire ce classement qui permet d'écrire des pilotes de périphériques composés de fonctions identiques, ce qui permet de masquer le support d'entrée-sortie. Néanmoins certains périphériques n'appartiennent à aucune de ces catégories ce qui rend cette classification quelque peu incomplète.

Les périphériques bloc effectuent un adressage par bloc, c'est-à-dire la quantité minimale manipulée est le bloc. Le disque dur est un périphérique bloc. Ils permettent également d'utiliser un bloc indépendamment des autres. Cette dernière précision est très pertinente en effet elle écarte l'imprimante de cette catégorie car on ne peut écrire un bloc en plein milieu d'une page, l'accès étant séquentiel. Il en est de même pour les lecteurs de bandes magnétiques. En revanche dans un disque dur on peut lire et écrire un bloc quelle que soit sa position. Certains périphériques semblent être des périphériques bloc du fait qu'ils manipulent des blocs, mais ils ne le sont pas réellement à cause de leur utilisation séquentielle.

Les périphériques caractère manipulent un flot de caractères. Ils ne permettent pas d'adresser une zone directement puisque l'accès est séquentiel. Nous citons les terminaux et les imprimantes, on n'écrit pas au milieu de la page ou de l'écran sans commencer à partir du début.

Certains périphériques ne rentrent dans aucune de ces catégories. L'horloge ne manipule ni bloc ni caractère.

1.3 Les contrôleurs de périphériques

En réalité, le processeur ne communique pas directement avec les périphériques : ceux-ci sont reliés à des contrôleurs de périphériques et c'est avec eux que le processeur dialogue. Par exemple si le processeur veut écrire une donnée sur le disque dur, il le demande au contrôleur de disque dur et c'est ce dernier qui se débrouille pour effectivement satisfaire la demande du processeur. Le processeur transmet alors la donnée à écrire au contrôleur, qui la stocke et la transmet au disque dur le moment venu.

Ce relais de l'information par des contrôleurs permet déjà de s'abstraire des spécificités des périphériques : une partie de ces spécificités n'est connue que du contrôleur. Cela permet par exemple de développer des périphériques très différents les uns des autres sans qu'il y ait de problème majeur pour les insérer dans un ordinateur. Cela permet aussi de renouveler les périphériques d'un ordinateur sans avoir à réinstaller le système.

On distingue plusieurs contrôleurs dans un ordinateur, les plus connus sont le contrôleur de disque, le contrôleur de mémoire RAM, de mémoire cache, le contrôleur de carte vidéo, le contrôleur de clavier, d'écran, le contrôleur de bus.

Afin de cerner de près le rôle d'un contrôleur outre de faire fonctionner les parties mécaniques, nous allons nous pencher sur un exemple de périphérique : le disque dur.

On se propose d'écrire un bloc sur le disque. Le contrôleur est le seul à connaître la position à laquelle se trouve le bras (sur quel cylindre). Avant de commander le déplacement du bras, il doit calculer le nombre de cylindres qu'il faut parcourir afin de se positionner sur le cylindre au niveau duquel le bloc doit être écrit. Lors de la présentation du SGF nous avons expliqué qu'un secteur de 512 octets comprend en réalité moins d'octets pour les données puisque certains sont réservés, notamment des octets pour écrire le code correcteur d'erreur. Ce code est calculé par le système et fourni au contrôleur afin qu'il l'inscrive à la suite des données.

Lors de la lecture de ce bloc, c'est le contrôleur qui vérifie la validité du secteur lu en utilisant ce code et en le comparant à un code qu'il calcule lui-même. En cas d'erreur, le contrôleur fera une nouvelle tentative de lecture puisqu'il dispose de tout ce qui est nécessaire pour détecter l'erreur.

Prenons à présent l'exemple d'un moniteur (écran), le contrôleur va générer des signaux pour diriger le faisceau d'électrons vers l'endroit où va être affiché chaque caractère. Sans contrôleur, c'est au programmeur de gérer le tube cathodique ou le déplacement du bras dans un DD. Par ailleurs, lorsqu'une ligne est entièrement remplie le contrôleur doit opérer un retour à la ligne après un passage à la ligne suivante. Le nombre de caractères affichable n'est pas un paramètre connu du système d'exploitation mais du contrôleur. De même pour l'écran lorsque ce dernier s'est rempli, le contrôleur devra gérer le décalage des lignes vers le haut.

Nous concluons de ces exemples que le contrôleur effectue principalement les tâches suivantes :

§ recevoir les requêtes d'entrée-sortie à effectuer

§ effectuer l'entrée-sortie en plusieurs étapes si cela est nécessaire

§ traduire les ordres du système en des signaux compréhensibles par le périphérique : identifier les caractéristiques du périphérique, effectuer les calculs nécessaires à la réalisation de l'entrée/sortie.

§ corriger les éventuelles erreurs.

Pour ce faire la carte du contrôleur se compose d'un circuit intégré pour la commande électronique, de registres destinés à accueillir les commandes à exécuter et à stocker temporairement les données avant leur transfert en mémoire centrale et vice-versa.

1.4 Interaction entre le système d'exploitation et le contrôleur

Le système d'exploitation dialogue avec le contrôleur à travers le bus. Il lui transmet les paramètres de l'entrée/sortie et ce dernier les stocke dans ses registres. Le contrôleur une fois qu'il a reçu l'information, génère les signaux adéquats, en effet le rôle du contrôleur est de transformer les signaux compris par l'unité centrale en signaux électriques compris par le périphérique et vice-versa.

Le système d'exploitation place des commandes et des paramètres dans les registres, par exemple : read, write. Une fois que la commande est acceptée par le contrôleur, il peut exécuter la commande : le processeur n'a plus besoin d'intervenir et peut s'occuper d'une autre tâche. Une fois que le contrôleur a fini, le processeur reprend le contrôle et le système d'exploitation prend la main (nous verrons plus tard comment le processeur est informé). Il va aller tester les résultats du déroulement de l'entrée-sortie à travers le contenu des registres.



2 - Entrées-Sorties synchrones et asynchrones

Un processus qui demande une entrée-sortie n'est pas automatiquement bloqué, en effet cela dépend du type de l'entrée-sortie : on distingue les entrées/sorties synchrones et celles asynchrones.

Les entrées-sorties sont par défaut synchrones mais il est possible pour le programmeur d'utiliser des entrées-sorties asynchrones. Une entrée-sortie est asynchrone si le processus continue son exécution après l'appel de l'instruction d'entrée-sortie. Le processus est informé de la fin de l'entrée-sortie par le positionnement d'un drapeau ou par l'exécution d'une routine système. Le fait de programmer avec des entrées-sorties asynchrones ne change en rien le mode de fonctionnement de l'entrée-sortie. Il y a toujours une interruption à la fin qui est générée par le périphérique. C'est quand le système d'exploitation va prendre en compte cette interruption que le traitement va changer, au lieu de déplacer le processus de la file *bloqué* vers celle *prêt*, le système d'exploitation va exécuter une routine ou positionner un drapeau.

3 - Les entrées-sorties DMA (par accès direct à la mémoire)

Cette technique a été inventée au temps où les processeurs étaient lents pour accélérer le transfert. En fait une fois que le périphérique a terminé l'entrée-sortie (par exemple un bloc lu du disque dur), le transfert en mémoire centrale se fait à travers le processeur par octets ou par mots.

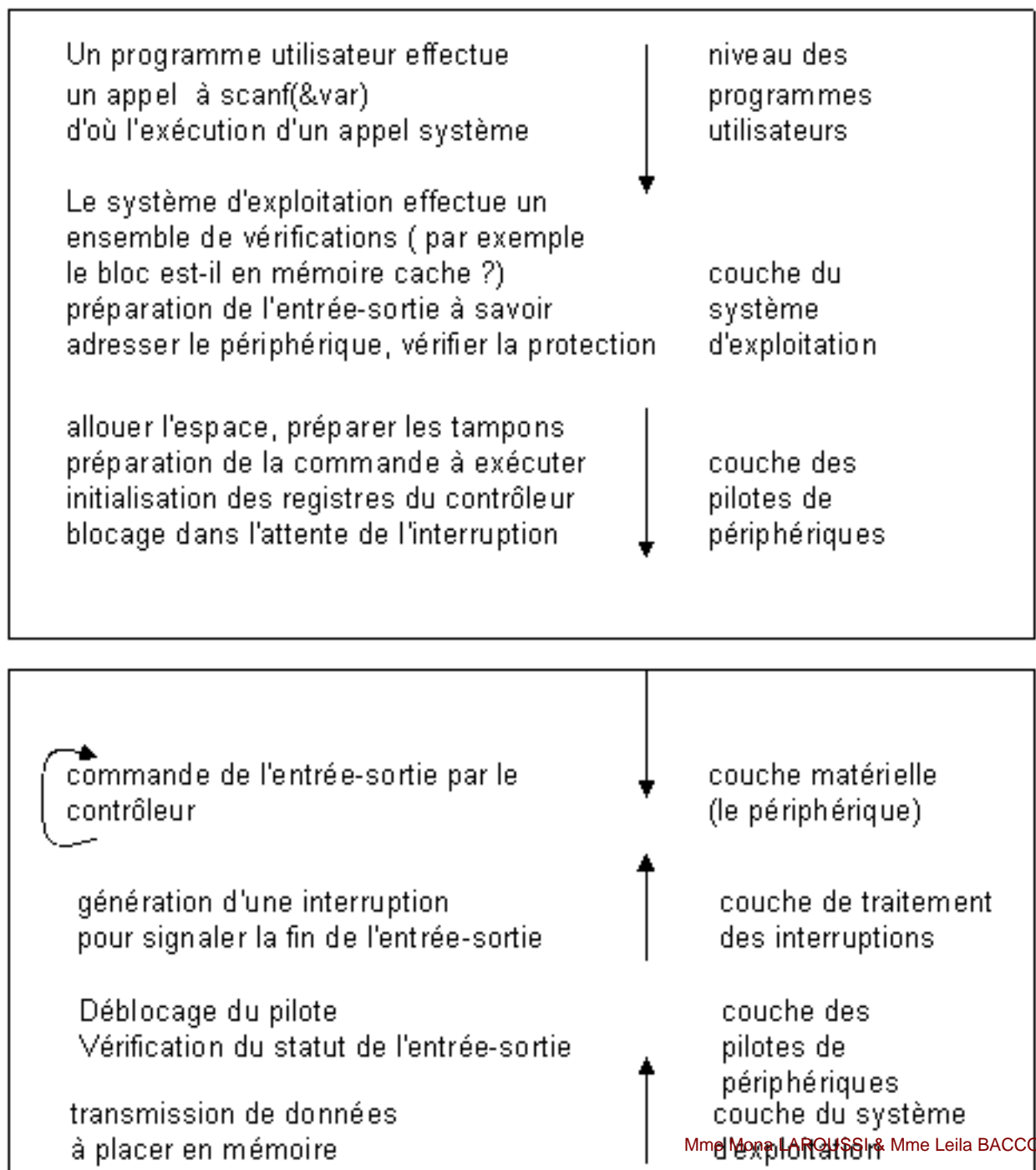
Certains contrôleurs (les plus rapides parmi les périphériques bloc) sont capables de faire un accès direct à la mémoire (Direct Memory Access), ce qui décharge le processeur du transfert des données. Le fait qu'ils soient rapides permet aux périphériques de générer une interruption après le transfert d'un bloc entier plutôt que de le faire après quelques octets.

Le contrôleur DMA vole cette autorité suprême qui revient au processeur (on parle de vol de cycle). Il prend le contrôle du bus et y place les données à transférer de/vers la mémoire. Lorsqu'il a fini, il génère une interruption pour informer le processeur. Ce mode est asynchrone puisque le processeur peut vaquer à d'autres tâches pendant le déroulement de l'entrée-sortie.

Actuellement elle est utilisée pour le rafraîchissement de la mémoire centrale. C'est également un mode d'entrée/sortie qui est offert aux programmeurs.

4 - Schéma d'exécution d'une entrée-sortie

La figure ci-dessus illustre le schéma d'exécution d'une entrée/sortie. Elle met en évidence les différentes couches du système d'exploitation qui sont sollicitées :



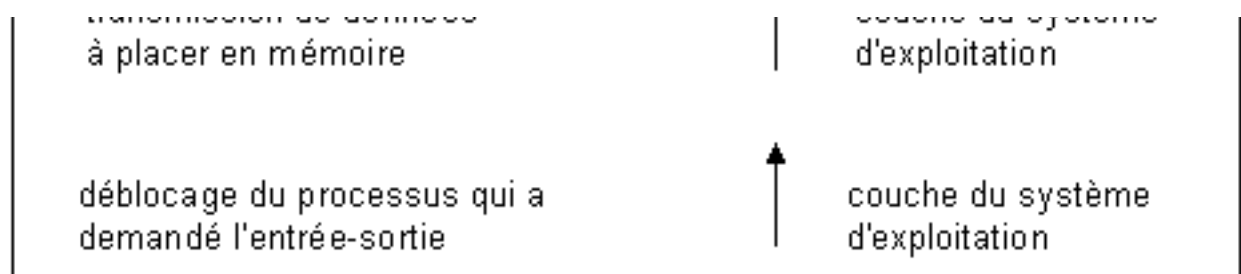


Fig. 59. Exécution d'une entrée/sortie

On remarque qu'il existe :

- une couche pour traiter les interruptions qui est la plus proche du matériel
- une couche constituée des pilotes de périphériques qui est une couche dépendante des périphériques
- une couche indépendante du matériel qu'on appellera couche système

Le système d'entrée-sortie est formé des couches *pilote* et *système d'exploitation*. Il est difficile de tracer une frontière bien définie entre les différentes couches, ce qu'il faut retenir c'est que le contrôleur est piloté par le pilote de périphériques. Ce dernier comprend tout le code nécessaire et que ce code est dépendant du matériel. La couche système comprend le code indépendant du matériel, elle a pour rôle de fournir une interface uniforme aux logiciels.

5 - Le sous-système d'entrée-sortie

⇒ Sommaire de la section :

✓ La couche système

✓ Les pilotes de périphériques

ENTRER

5.1 La couche système

Le principal objectif de cette couche est de fournir une interface uniforme à la couche au dessus à savoir les programmes utilisateurs en leur cachant le lien avec le matériel.

Après avoir effectué certaines vérifications comme cité auparavant, une requête d'entrée-sortie va être construite par la création d'un IOCB (input output control block). C'est une structure qui contient des informations décrivant l'entrée-sortie parmi lesquelles :

§le type d'opération lecture ou écriture,

§ce qu'on appelle la longueur de l'entrée-sortie c'est-à-dire le nombre de caractères à transférer par exemple,

§l'adresse d'une zone tampon en mémoire destinée à accueillir les données. On voit ici que cette couche effectue les allocations mémoire nécessaires à l'entrée-sortie,

§d'autres informations complémentaires.

L'IOCB est passé de couche en couche en effet, il contient les informations concernant la requête d'entrée-sortie. Ces informations sont complétées au fur et à mesure.

La plupart des systèmes interdisent aux programmes des utilisateurs d'effectuer directement des opérations d'entrées-sorties (pour des raisons de non blocage du processeur, et pour faciliter la programmation). C'est le système d'exploitation qui réalise les entrées-sorties. Pour ce faire, il est le seul à connaître l'adresse du périphérique. Les utilisateurs utilisent un nom logique qui est traduit par le système d'exploitation à l'aide d'une table de noms. Le système d'exploitation transmet au pilote de périphérique le numéro du périphérique, ainsi que l'IOCB afin de démarrer le traitement de l'entrée-sortie.

5.2 Les pilotes de périphériques

On distingue généralement un pilote par type de périphérique. Il est à noter que désormais, la plupart des pilotes sont réalisées par le fabricant plutôt que par les concepteurs du système, de part le trop grand nombre de périphériques différents. Ces pilotes sont chargés par le système au démarrage et sont exploités de la même façon que les autres, en ce qui concerne la partie gérant le matériel.

Tout comme le système d'exploitation est le seul à connaître l'adresse du périphérique, à son tour le pilote est le seul à connaître les paramètres à ajuster c'est-à-dire les caractéristiques du périphérique.

Chaque périphérique est décrit dans une table à travers un descripteur. Celui-ci contient l'adresse du périphérique, le vecteur d'interruption qui lui est associé, et la description de toutes ses caractéristiques. Les IOCB sont chaînés entre eux et rattachés à la table. Nous verrons plus tard que le fait de disposer d'une liste chaînée permet de choisir l'ordre de chaînage.

Un pilote de périphérique est constitué de routines qui s'appellent les unes les autres. Le rôle du pilote est décrit ci-dessous :

§ traduire la requête à partir de l'IOCB en termes concrets. Prenons le cas d'un pilote de disque, il doit déterminer où se trouve le bloc à lire par exemple (sans rentrer dans les détails) la position du bras, le cylindre concerné, afin d'initialiser les registres du contrôleur. Il prépare le travail du contrôleur.

§ Lancer le contrôleur et se mettre en attente de la fin de l'entrée-sortie c'est donc un appel bloquant. Si l'entrée-sortie nécessite plusieurs transferts, c'est au pilote de commander le transfert suivant.

§ Une fois l'entrée-sortie terminée, le périphérique génère une interruption. La procédure de traitement des interruptions est lancée. Elle va identifier la source de l'interruption, elle saura donc quelle est l'unité qui a généré l'interruption et par conséquent le traitement à entreprendre et le pilote associé est réveillé. Il vérifie le statut de l'entrée-sortie. Il traite les erreurs non résolues par le contrôleur, par exemple il effectue une nouvelle tentative. Si l'entrée-sortie ne s'est toujours pas réalisée, il informe la couche système qui informe à son tour le processus

appelant.

Pour finir, le système d'exploitation va débloquent le processus qui a demandé l'entrée-sortie et le faire passer de la file des processus bloqués dans ceux prêts.

L'entrée-sortie a fini d'être traitée, les programmes des utilisateurs peuvent reprendre.

6 - Etude de cas : le périphérique disque

Le disque dur le périphérique le plus intéressant du point de vue des programmes fournis par les pilotes. La section suivante présente certains aspects spécifiques aux disques durs, tel que l'entrelacement des blocs, ainsi que quelques traitements gérés par le pilote du disque notamment les algorithmes qui concernent l'accès au disque dur, la prise en compte des erreurs etc.

⇒ Sommaire de la section :

- ✓ L'entrelacement des blocs
 - ✓ Notion de cache disque
 - ✓ La prise en compte des erreurs
 - ✓ Ordonnancement des requêtes d'entrées-sorties
-

6.1 L'entrelacement des blocs

Lorsque le contrôleur effectue une entrée-sortie DMA, il met un certain temps pour transférer le secteur lu vers la mémoire. Or durant ce temps, le disque continue de tourner. Si le contrôleur ne peut effectuer une entrée et une sortie en parallèle, le secteur suivant ne peut être lu durant le transfert vers la mémoire. Si le temps de transfert vers la mémoire est égal au temps de lecture d'un secteur du disque, il faut deux rotations du disque afin de lire toute une piste. Le secteur de numéro suivant ne doit passer sous la tête de lecture qu'une fois que le contrôleur a fini le transfert du secteur précédent. Une solution serait que les numéros de secteurs ne soient pas consécutifs, afin que le secteur suivant ne défile sous la tête qu'une fois la fin du transfert.

On appelle entrelacement le fait de sauter des numéros de secteurs afin que le contrôleur puisse les lire dans un ordre normal. Cette numérotation spéciale est réalisée au formatage du disque en tenant compte du facteur d'entrelacement. Un facteur d'entrelacement de 1 désigne le saut d'un secteur à chaque fois. Ce facteur est lié au temps nécessaire au contrôleur pour effectuer son transfert vers la mémoire. Il est clair que si le contrôleur met plus de temps, il faut prévoir un facteur de 2.

Fig. 59. Numérotation des secteurs en fonction du facteur d'entrelacement

La figure n° 59 illustre un disque avec les numéros de secteurs numérotés de 0 à 11. Sans entrelacement après un tour complet, le contrôleur n'aura pu lire que les secteurs n° 0 2 4 6 8 10. Au tour suivant, il lira les secteurs 1 3 5 7 9 11. Avec un facteur simple, le contrôleur laissera passer un secteur et lira le suivant. Avec un facteur d'entrelacement double, il laissera

passer deux secteurs pour lire le secteur suivant.

6.2 Notion de cache disque

Certains concepteurs ont eu l'idée d'appliquer le principe du cache pour le contenu d'une piste. Quand les requêtes d'entrées-sorties sont très nombreuses, il devient fréquent d'avoir des accès à des secteurs voisins donc appartenant à une même piste. Le principe du cache pour les pistes est de copier le contenu de toute la piste dans le cache. Ainsi des requêtes futures pour des secteurs de cette piste éviteront d'accéder au disque. Le cache fait partie du contrôleur.

De nos jours les contrôleurs sont tellement rapides, qu'en un seul accès ils peuvent recopier toute la piste dans leur zone tampon.

6.3 La prise en compte des erreurs

Les erreurs, comme nous l'avons expliqué auparavant doivent être traitées par le composant le plus proche du périphérique, donc par le contrôleur, sinon le pilote doit s'en charger.

Parmi les erreurs qui peuvent survenir, nous citons :

§ Les erreurs de positionnement du bras : le contrôleur peut se tromper dans les paramètres qu'il utilise pour accéder à une piste donnée. Quand il se rend compte qu'il y a eu erreur, par exemple qu'il n'est pas positionné sur le cylindre en question, le contrôleur corrige tout seul en réessayant. S'il n'y arrive pas, le pilote va envoyer de nouveaux paramètres afin de déplacer le bras à la position extrême et recommencer.

§ Les erreurs de programmation : si le pilote demande au contrôleur d'accéder à une piste ou à un secteur qui n'existe pas le contrôleur doit être capable de déceler l'erreur et de la signaler.

§ La rencontre d'un bloc endommagé. Le contrôleur doit ne pas l'utiliser ou en utiliser un autre (si c'est un contrôleur intelligent), en se référant à une liste de bloc endommagés qu'il doit avoir en mémoire. Celle-ci est une copie de celle dont dispose le système.

§ Les erreurs de lecture : elles peuvent être dues à la présence d'une poussière sur la surface du plateau. Elles sont décelées grâce au calcul du code correcteur d'erreur et corrigées en réessayant l'opération de lecture. Le code correcteur d'erreur (en anglais ECC Error Correcting Code) est un code calculé et inscrit par le contrôleur en même temps qu'il inscrit le secteur. Souvenez-vous qu'un secteur de 512 octets occupe réellement plus d'octets (voir chapitre 3 section 3.1.6.1) . En effet, il est accompagné du numéro du secteur, du ECC et autres...Lors de toute opération de lecture, le code est recalculé et comparé à celui inscrit avec le secteur. Si les deux codes ne concordent pas, le contrôleur essaye de corriger le secteur en se basant sur la valeur du code. Le cas échéant il tente une nouvelle lecture, sinon il marque le secteur comme défectueux

6.4 Ordonnancement des requêtes d'entrées-sorties

En ce qui concerne l'accès au disque dur, il y a de nombreuses optimisations qui peuvent être réalisées afin de réduire les temps d'accès et ceci est du ressort du pilote.

Quand on doit effectuer un transfert de ou vers le disque dur, le **temps total cumulé** dépend du temps nécessaire pour positionner la tête de lecture/écriture sur le bon cylindre (appelé **temps de recherche**) et du temps passé à attendre que le secteur passe sous la tête afin de lire les octets (appelé **temps de rotation**, en moyenne la moitié d'un tour) et enfin le **temps pour le transfert**.

Le temps de recherche est celui qui peut être optimisé de manière logicielle. Le contrôleur du disque dispose d'une seule unité, les requêtes de transfert seront donc chaînées dans la même liste. Parmi les tâches du pilote on trouve l'ordonnancement des requêtes dans cette liste.

Il est clair que si les requêtes sont traitées dans l'ordre d'arrivée le bras risque de se déplacer inutilement entre plusieurs cylindres ce qui rajoute du temps de recherche. Prenons un exemple :

La tête est positionnée sur le cylindre n° 10, on souhaite lire les secteurs appartenant aux pistes 1, 9, 8, 26, 13, 23. Nous allons comparer les temps de déplacements selon plusieurs politiques d'ordonnancement connues : FIFO, plus court déplacement, ascenseur.

7.6.4.1. FIFO

Comme l'indique son nom, la requête arrivée en premier est exécutée en premier. Pour les requêtes aux pistes citées plus haut, le bras va effectuer un déplacement qui va couvrir $9 + 8 + 1 + 18 + 13 + 10 = 59$ cylindres.

--->Simuler l'algorithme [FIFO](#).

7.6.4.2. Plus court déplacement ou plus court temps de recherche

Cet algorithme s'appelle en anglais SSTF (Shortest Seek Time First). Le contrôleur sert la requête qui est la plus proche de celle qui vient d'être traitée. Si l'on ordonnance ces requêtes

en essayant de minimiser les déplacements par rapport à la position courante, on obtient un accès dans cet ordre 10, 9, 8, 13, 23, 26, 1 soit un déplacement qui couvre $1 + 1 + 5 + 10 + 3 + 25 = 45$ cylindres, ce qui représente une amélioration notable par rapport à FIFO.

Cet algorithme peut provoquer des situations de famine si la plupart des requêtes se situent dans un même espace et qu'il existe une ou deux requêtes plus loin, celles-ci ne seront traitées que très tard. Par conséquent, cet algorithme n'est pas équitable.

--->Simuler l'algorithme [PCD](#).

7.6.4.3. Ordonnement selon l'algorithme de l'ascenseur

Cet algorithme est utilisé également pour les ascenseurs dans les grands buildings, où on sert les requêtes en maintenant un seul sens de déplacement (vers le haut ou le bas pour les ascenseurs) vers l'extérieur où vers l'intérieur (pour les disques) jusqu'à ce qu'il n'y ait plus de requêtes dans le sens choisi. Cet algorithme garantit une certaine équité. On dit qu'il applique un balayage de bout en bout et est connu sous le nom de SCAN.

Le pilote mémorise un bit de sens (haut ou bas) qui indique la direction que le contrôleur doit privilégier. Ainsi, il traite toutes les requêtes dans un sens, puis, quand il n'y a plus de requête dans ce sens (le bras est donc vers une extrémité), on inverse le bit et on traite les requêtes situées de l'autre côté du bras. On est donc sûr de traiter rapidement toutes les requêtes

Pour cet exemple, supposons que le sens de déplacement du bras était vers l'intérieur. Servir les requêtes 1, 9, 8, 13, 26, 23 en partant de la 10 va donner l'ordre suivant 10 13 23 26 9 8 1 soit un déplacement de $3 + 10 + 3 + 17 + 1 + 7 = 41$ cylindres.

La différence n'est pas très significative mais l'avantage de cet algorithme est qu'il est indépendant du nombre des requêtes et équitable. Le déplacement du bras est égal à maximum deux fois le nombre de cylindres. Donc deux fois le temps nécessaire pour aller du cylindre du centre vers celui le plus à l'extérieur.

--->Simuler l'algorithme [SCAN](#).

On distingue deux variantes à cet algorithme.

§ La première connue sous le nom de C-SCAN consiste à parcourir les cylindres toujours dans le même sens. Après le dernier cylindre, le bras retourne au premier. Le temps de réponse s'en trouve un petit peu amélioré.

§ La seconde n'est possible qu'avec certains contrôleurs. En effet, certains précisent le secteur qui se trouve sous la tête. Si on a deux requêtes sur un même cylindre, on peut alors commencer par traiter la requête qui passera en premier sous la tête.

----> Vous pouvez simuler votre propre exemple.

1. Bach, Maurice J. *Conception du système Unix*. Paris : Masson, London : Prentice Hall, 1993. 497 p. ISBN 2-225-81596-8.
2. Beauquier, Joffroy ; Berard, Béatrice. *Systèmes d'exploitation : Concepts et Algorithmes*. 4^{ème} tirage. Paris : Ediscience International, 1994. 541 p. ISBN 2-84074-025-7.
3. Bourne, Steve. *Le système Unix*. Paris : Interéditions, 1985. 38 p. ISBN 2-7296-0014-0.
4. Communication Micro-informatique Interfaces. *Musée de la micro-informatique*. [En ligne]. 2002 [réf. de 01-2002]. Disponible sur Internet : < <http://www.cmi-visavoy.com> >
5. Custer, Helen. *Au cœur de Windows NT*. 6^{ème} tirage. Les Ulis : Microsoft Press, 1996. 383 p. ISBN 2-84082-001-3.
6. Dubois, philippe. *Le Musée d'Histoire Informatique*. [En ligne]. 2002 [réf. de 01-2002] Disponible sur Internet : < <http://mo5.com/MHI/index.php> >
7. Duffet, B. *Historique des systèmes d'exploitation et des réseaux Micro-informatique sur PC* [En ligne]. 2002 [réf. de 01-2002] Disponible sur Internet <<http://mapage.noos.fr/bduffet/technique/chrono.htm> >
8. GendNetClub. *Le portail des gendarmeries et forces de polices francophones Attributs des fichiers, gestions matérielles et système avec MS Dos*. [En ligne]. 2000 [réf. de 06-2002] Disponible sur Internet : <<http://www.gendnetclub.com/index.htm> >
9. Giguere, C. *VDN Informatique Articles sur les concepts informatiques & vulgarisation*. [En ligne]. 1997- 2002 [réf. de 01-2002] Disponible sur Internet : <<http://membres.lycos.fr/cgiguere/vdn/> >
10. Goguey, Eric. *Dictionnaire de l'informatique*. [En ligne]. 1999-2001 [réf. de 06-2002] Disponible sur Internet <http://www.dicofr.com/chronologie/>
11. Griffiths, Michael ; Vayssade, Michel. *Architecture des systèmes d'exploitation*. Paris : Hermes, 1990. ISBN 2-86601-127-9.
12. Guillier, François. *Histoire de l'informatique*. [En ligne]. 1996-2002 [réf. de 01-2002] Disponible sur Internet : <<http://www.histoire-informatique.org/> >
13. Hennessy, John L. ; Patterson David A. *Architecture des ordinateurs, Une approche quantitative*. 2^{ème} édition, International THOMPSON Publishing, 1996.
14. IEEE. *IEEE home*. [En ligne]. 2002 [réf. de 09-2002] Disponible sur Internet : < <http://www.ieee.org/> >
15. Krakowiak, Sacha. *Principes des systèmes d'exploitation des ordinateurs*. Paris : Dunod, 1987. 486 p. ISBN 2-04-018623-8.
16. McKusik, Marshall Kirk et al. *The design and implementation of the 4.4 BSD Unix operating system*. USA : Addison-Wesley, 1996. ISBN 0-201-54979-4.

17. Mueller, Scott. *Le PC Architecture Maintenance et Mise à Niveau*. 5^{ème} édition. Paris : Campus Press, 2002. 1344 p. ISBN 2-7440-1303-X.
18. Pillou, Jean-François. *CCM Vulgarisation Informatique*. [En ligne]. 1999-2002 [réf. de 09-2002] Disponible sur Internet : <www.commentcamarche.net/pc>
19. RIFFLET, Jean-Marie. *La programmation sous Unix*. 3^{ème} édition. Paris : Edisciences International, 1995. 630 p. ISBN 2-84074-013-3.
20. Ritchie, dennis. *Dennis Ritchie Home page*. [En ligne]. 2002 [réf. de 09-2002] Disponible sur Internet <<http://cm.bell-labs.com/cm/cs/who/dmr/> >
21. Silberschatz, Abraham ; Galvin Peter B. *Principes des systèmes d'exploitation*. 4^{ème} édition. Paris : Addison Wesley, 1994. 772 p. ISBN 2-87908-078-9.
22. Solomon, D.A. ; Russinovich, M.E. *Inside Microsoft Windows 2000*. 3^{ème} édition. Microsoft Press, 2000. ISBN 0-7356-1021-5.
23. Tanenbaum, Andrew. *Minix information sheet*. [En ligne]. 1996 [réf. de 09-2002] Disponible sur Internet : <http://www.cs.vu.nl/~ast/minix.html>
24. Tanenbaum, Andrew. *Systèmes d'exploitation : systèmes centralisés systèmes distribués*. Paris : Dunod, London : Prentice Hall , 1994. 795 p. ISBN 2-10-004554-7.
25. Tanenbaum, Andrew. *Architecture de l'ordinateur*. 4^{ème} édition. Paris : Dunod, 2001. 638 p. ISBN 2-10-005158-X.
26. The Open Group. *The Open Group Working for interoperability*. [En ligne]. 1995-2002 [réf. de 09-2002] Disponible sur Internet : <<http://www.opengroup.org/>>
27. The Open Group. *The UNIX System Home Page*. [En ligne]. 1995-2002 [réf. de 09-2002] Disponible sur Internet : <<http://www.unix-systems.org/>>
28. Thibaut, bernard. *Informatique*. [En ligne]. 2000-2001 [réf. de 09-2002] Disponible sur Internet : <<http://www.alphaquark.com/> >
29. ThinkQuest Inc. *Le microprocesseur*. [En ligne]. 1995-2002 [réf. de 01-2002] Disponible sur Internet : <<http://library.thinkquest.org/C008227F/microprocesseur.htm> >
30. Tischer, Michael ; Jennrich, Bruno. *La bible PC Programmation Système*. Paris : Micro-Application, 1997. 1632 p. ISBN 2-74429-0544-8.
31. Unix System Laboratories. *UNIX SYSTEM V/386 Release 3.2 Manuel de référence du gestionnaire système*. Unix Press. London : Prentice Hall, Paris : Masson, 1991. ISBN 2-225-82542-4.

32. Unix System Laboratories. *UNIX SYSTEM V/386 Release 4 Manuel de référence du programmeur*. Unix Press. London : Prentice Hall, Paris : Masson, 1992. ISBN 2-225-82599-8.
33. Yahoo! Inc. Hachette Multimédia / Hachette Livre. *Yahoo encyclopédie*. [En ligne]. 2001 [réf. de 09-2002] Disponible sur Internet : <<http://fr.encyclopedia.yahoo.com/> >
34. Yunès, Jean-Baptiste. *A propos du cours systèmes*. [En ligne]. 2002 [réf. de 09-2002] Disponible sur Internet : <http://verif.liafa.jussieu.fr/~yunes/systemes/>