Système d'exploitation

Dr. FOKO SINDJOUNG Miguel Landry
Institut Universitaire de Technologie Fotso Victor de Bandjoun
Université de Dschang

Avril 2021

Table des matières

Objectifs

Ce cours a pour objectif d'initier les étudiants aux notions de Système d'Exploitation (SE). De ce fait, il sera question tout au long de cette unité d'enseignement de :

1. Définir la notion de système d'exploitation
2. Donner le rôle du SE dans un système informatique
3. Décrire les éléments constitutifs d'un SE
4. Décrire de façon détaillée le principe de fonctionnement des SE
5. Prendre en main les SE Windows et Linux

Chapitre 1

Introduction aux Systèmes d'Exploitation (SE)

1.1 Objectifs

Ce chapitre introductif vise à outiller l'étudiant aux notions de bases relatives aux systèmes d'exploitation. Ainsi, après l'avoir achevé, l'étudiant devra être cappable de :

- Définir la notion de système d'exploitation
- Connaitre les fonctionnalités d'un système informatique
- Donner le rôle et la composition d'un système d'exploitation
- Donner l'historique de l'évolution des systèmes d'exploitation
- Maitriser le mode de fonctionnement des différentes différents SE apparus au fil du temps

La charge horaire destinée à ce chapitre est départie comme suit :

1. Cours magistral : 5h
2. Travaux dirigés : 2h
3. Travail personnel de l'étudiant : 1h

1.2 Concepts fondamentaux

Pour pouvoir parler de Système d'exploitation, il faut tout d'abord définir quelques termes techniques. La première chose à savoir est : qu'est-ce qu'un ordinateur ? Il est évident que nous ne rentrerons pas dans les détails ici. Un cours d'architecture des ordinateurs vous permettrait d'appréhender cette notion en détails, mais nous pouvons en donner ici une définition (simpliste et) simplifiée.

Ordinateur : Machine permettant de stocker des informations et d'effectuer des calculs (beaucoup plus rapide que l'humain). Cet ordinateur regroupe un certain nombre de composants que nous appelerons matériel (hardware). Parmi ceux-ci, nous trouvons **clavier, souris, écran, mémoire, disques, . . .**

Ce matériel, sans logiciels (software) n'est qu'un amas de féraille et de composants électroniques.

Les **logiciels** sont des séquences d'instructions que l'on appelle souvent programmes. Un programme est l'implantation d'un ensemble de méthodes de résolution de problèmes dans un langage compris par l'ordinateur. La compréhension de l'ordinateur n'est en fait que la séquence d'actions résultant du programme (écrire sur l'imprimante, sur le disque, a' l'écran, . . .). La méthode de résolution est ce que l'on appelle l'algorithme.

Le **système d'exploitation** d'un ordinateur est un programme servant d'interface entre le matériel et les utilisateurs. Son but est de rendre le maniement de l'ordinateur facile et de proposer une utilisation efficace de celui-ci (il n'est par exemple, pas question pour l'utilisateur de savoir quelle va être l'organisation du fichier sur le disque lorsqu il le sauve).

1.3 Fonctionnalités d'un système informatique

En se limitant au seul point de vue d'un utilisateur d'un système informatique, les fonctions devant être assurées par celui-ci peuvent être résumées de la façon suivante :

— **Gestion et conservation de l'information**. Le système informatique doit permettre à tout utilisateur de créer, conserver, retrouver ou détruire les objets sur lesquels celui-ci désire effectuer des opérations.

— **Préparation, mise au point et exploitation de programmes**.

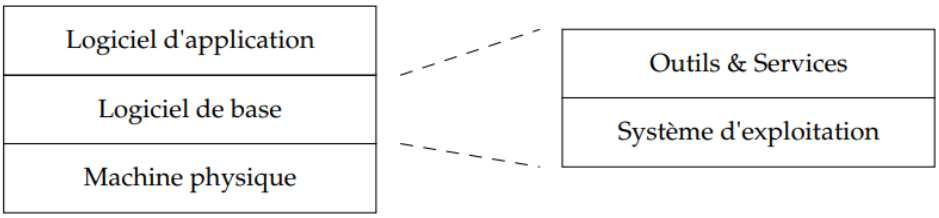


Figure 1.1 – *Structure logicielle d'un système informatique*

La figure 1.1 précise l'organisation logicielle d'un système informatique avec d'une part le logiciel d'application (traitement de textes, gestionnaire de bases de données, compilateurs, etc.) et d'autres parts le logiciel de base livré avec la machine.

1.3.1 Fonctions d'un système d'exploitation

Le système d'exploitation est un des éléments clef d'un système informatique. Il reprend à son compte les deux fonctions précédentes en y ajoutant des nouvelles fonctions liées à la bonne gestion de la machine physique :

— **Structuration de l'information** (sous forme de fichiers) en vue de sa conservation et de sa modification

— **Transfert des données** entre les éléments constituants du système informatique (unité centrale, périphériques d'impression ou de lecture, modem, etc.)

— **Gestion de l'ensemble des ressources** pour offrir à tout utilisateur un environnement nécessaire 'a l'exécution d'un travail

— **Gestion du partage des ressources**. Le système doit répartir les ressources dont il dispose entre les divers usagers en respectant la règle d'équité et en empêchant la famine. En particulier, il doit réaliser un ordonnancement des travaux qui lui sont soumis et éviter les interblocages
— **Extension de la machine hôte**. Le rôle du système est ici de simuler une machine ayant des caractéristiques différentes de celles de la machine réelle sur laquelle il est implanté. Chaque utilisateur dispose alors d'une machine virtuelle munie d'un langage étendu permettant l'exécution et la mise au point des programmes au moyen d'outils plus facilement utilisables que ceux dont est dotée la machine câblée.

1.3.2 Aspects internes

La diversité des tâches à remplir et des matériels utilisés a pour conséquence une grande variété des aspects externes des systèmes :

— Les systèmes destinés à la conduite de processus industriels (chimie industrielle, cracking, central téléphonique, guidage de fusée, surveillance médicale ou monitoring , etc.) ;
— Les systèmes gérant les bases de données (réservations de places, gestion de stock, gestion de comptes bancaires, documentation automatique, etc.) ;
— Les systèmes destinés à la création et l'exécution de programmes qui peuvent être subdivisés en plusieurs classes selon :
  — Le degré d'interaction entre l'utilisateur et ses programmes (traitement par trains ou conversationnel)
  — Le mode de partage des ressources (mono ou multiprogrammation) ;
  — Les possibilités offertes par le langage étendu (accès à un ou plusieurs langages) ;

Malgré cette grande diversité, les systèmes comportent entre eux des parties très ressemblantes, voire identiques, et il serait donc très utile de pouvoir dégager celles-ci afin de profiter de certaines études partielles dans l'élaboration de portions plus complexes. C'est ce souci de rentabilisation qui a conduit à une conception modulaire des systèmes et de leurs différents constituants, technique généralisable à tout logiciel développé sur une machine quelconque.

1.4 Historique des systèmes d'exploitation

L'historique est un moyen agréable de présenter les principaux concepts en partant de l'absence de S.E. pour arriver aux systèmes répartis.

1.4.1 Les systèmes monoprogrammés

Sur les premiers ordinateurs il n'existe pas de S.E. à proprement parler. L'exploitation de la machine est confiée à tour de rôle aux utilisateurs ; chacun disposant d'une période de temps fixe. C'est une organisation en porte ouverte.

Au début des années 50, on voit apparaître le premier programme dont le but est de gérer la machine : c'est le moniteur d'enchaînement des tâches. Cet embryon de

S.E. a la charge d'enchaîner l'exécution des programmes pour améliorer l'utilisation de l'unité centrale (U.C.). Il assure également des fonctions de protection (vis à vis du programme en cours d'exécution), de limitation de durée et de supervision des entrées/sorties. Pour réaliser ces opérations, le moniteur est toujours présent en mémoire. Il est dit résident.

La fin des années 50 marque le début du traitement par lots (batch processing). Une machine prépare les données en entrée (lecture des cartes perforées à cette époque) tandis que la machine principale effectue le travail et qu'une troisième produit le résultat. Il existe donc un parallélisme des tâches entre lecture, exécution et impression. Les opérations d'E/S ne sont plus réalisées par la CPU, ce qui libère du temps de calcul

1.4.2 Les systèmes multiprogrammés

La multiprogrammation arrive au début des années 60. Elle est caractérisée par la présence simultanée en mémoire de plusieurs programmes sans compter le S.E. lui-même. Cette caractéristique s'explique de la manière suivante : l'exécution d'un programme peut être vue comme une suite d'étapes de calcul (les cycles d'U.C.) et d'étapes d'E/S (les cycles d'E/S) comme le montre la figure 1.2. Sur un système monoprogrammé, la CPU est donc inutilisée durant les cycles d'E/S.

L'idée de base est d'utiliser ces temps d'attente pour exécuter un autre programme. Ce programme doit n´ecessairement être déjà présent en mémoire afin d'éviter l'E/S de chargement puisque justement on cherche à utiliser les temps morts d'E/S. La réalisation pratique de cette idée nécessite :

— Des unités matérielles capables d'effectuer des E/S de manière autonome (libérant ainsi la C.P.U pour d'autres tâches) ;
— Des possibilités matérielles liées à la protection de la mémoire et/ou à la réimplantation du code pour éviter qu'une erreur d'un programme influence le déroulement d'un autre

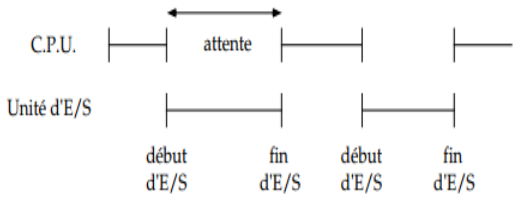


Figure 1.2 – *Cycle de CPU cycle d'E/S*

Dans les années 60/70 les premiers systèmes en temps partagé (time sharing) sont disponibles. Ces systèmes sont directement liés à l'utilisation interactive des machines au moyen de terminaux vidéo. Ce mode d'utilisation impose un temps de réponse acceptable puisque les utilisateurs attendent devant leur terminaux. Pour garantir un bon temps de réponse moyen, le temps d'exécution de la CPU est découpé en tranches appelées des quanta. Ces quanta sont allouées aux programmes en cours d'activité. Le temps d'exécution de la CPU est donc partagé entre les programmes utilisateurs. Si le nombre d'utilisateurs n'est pas trop important et, sachant qu'un utilisateur moyen passe 90% de son temps à réfléchir et seulement 10%

à exécuter une action, le temps de réponse reste acceptable et chaque utilisateur à l'impression d'avoir sa propre machine. Sur un plan matériel, le temps partagé est basé sur les possibilités suivantes :

- Les programmes sont tous en mémoire ; le temps partagé implique donc la multiprogrammation
- Le matériel doit permettre l'interruption d'un programme au bout de son quanta de temps pour passer la CPU à autre programme
- Les temps de commutation d'un programme vers un autre doit être aussi faible que possible car durant cette étape la CPU est utilisée par le S.E. au détriment des programmes utilisateurs.

Les systèmes répartis se développent durant les années 80. Dans cette organisation, les données mais aussi les programmes sont réparties sur plusieurs machines connectées par un réseau. Les problèmes sont plus complexes puisqu'ils couvrent la communication, la synchronisation et la collaboration entre ces machines, mais ceci est une autre histoire... et un autre cours

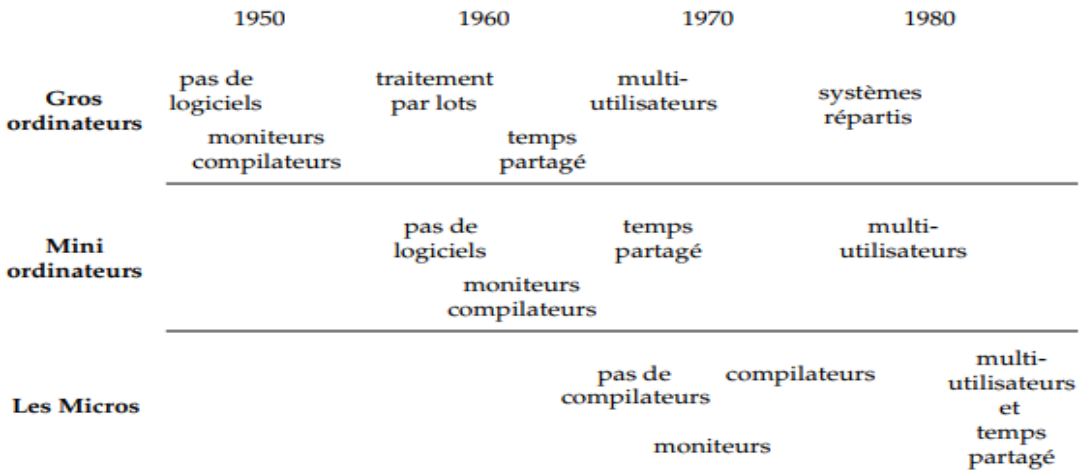


Figure 1.3 – *Evolution des S.E.*

1.5 Structure Interne des systèmes d'exploitation

1.5.1 Conception descendante et structure en couche

Dans cette section, le terme de langage sera utilisé dans le sens suivant : un langage définit des objets (et les mécanismes permettant de les créer), des actions (ou primitives) permettant de manipuler ces objets et des règles de composition de ces actions.

En particulier, tout langage définit une machine capable de l'interpréter : les instructions de cette machine représentent l'ensemble des primitives du langage, sa mémoire permet de représenter les objets et son mécanisme d'exécution est celui défini par les règles d'interprétation du langage.

Désirant résoudre un problème, une démarche habituelle consiste à décomposer celui-ci en une succession de plusieurs sous-problèmes que l'on espère résoudre plus

aisément. On essaie donc dans un premier temps de définir une machine M_0 dont les primitives rendront la résolution du problème plus facile. Le problème initial (supposé résolu par la machine M_0) se transforme donc en la réalisation de cette machine M_0 sur la machine disponible M. On va alors définir pour cela une machine M_1, etc. jusqu'à l'obtention d'une machine M_n facilement réalisable sur M.

La puissance de cette méthode ne réside pas dans la seule simplification du problème à chaque niveau, mais résulte aussi du processus d'abstraction consistant à focaliser l'étude sur les aspects essentiels du problème, concrétisés par la spécification d'une machine, c'est à dire, en fait, celle de son interface.

Lorsque la réalisation d'une machine M_i utilise l'interface d'une machine M_j, on dit que M_i dépend de M_j. En fait, cette relation de dépendance ne porte que sur l'interface et non sur la réalisation interne. On peut alors décrire la structure d'un système par un graphe dont les nœuds représentent les machines, et les arcs, les relations de dépendance.

Dans la figure 1.4, le schéma (a) représente la structure résultant de la méthode de conception descendante. Celui-ci peut être généralisé en (b) si l'on autorise chaque machine à utiliser les primitives de toute autre machine de niveau inférieur. Enfin, si la seule contrainte est d'avoir un graphe sans circuit, on obtient le schéma (c) où les machines sont classées par niveau d'abstraction, chacune d'elles n'utilisant que les machines de niveau inférieur.

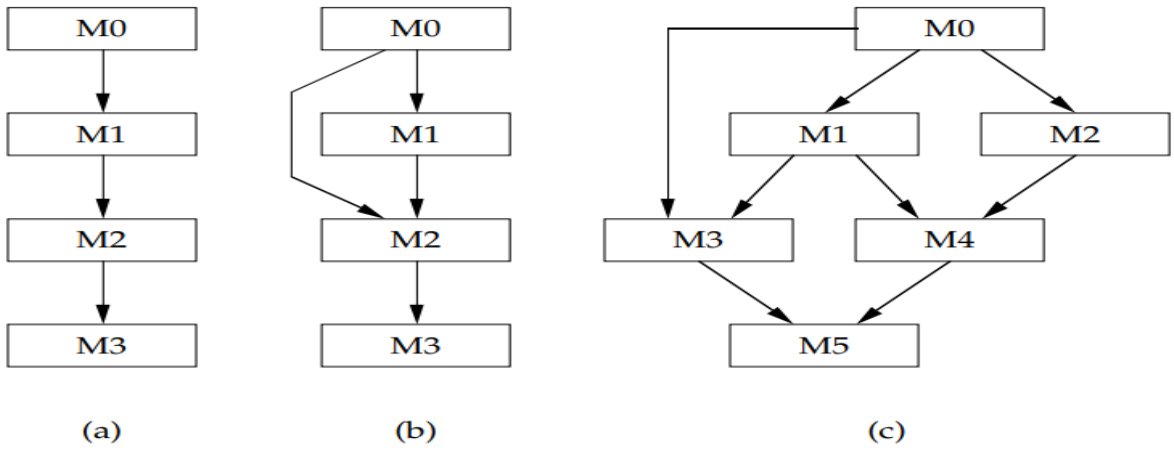


Figure 1.4 – *Décomposition hiérarchiquee*

En réalité, la méthode de conception descendante est rarement utilisable à l'état pur. Plusieurs facteurs sont à prendre en considération : l'expérience du concepteur, l'existence de machines déjà réalisées, la difficulté de fixer à l'avance les spécifications détaillées des interfaces, qui utilisent en fait le résultat d'expérimentation sur des réalisations partielles. En tout état de cause, l'indépendance introduite par l'abstraction procure à la structure hiérarchique par niveau plusieurs avantages :

— Indépendance pour la conception. Description totale du comportement d'une machine par les spécifications de son interface.

— Indépendance pour la modification. Les modifications dans la réalisation d'une machine n'altèrent en rien celles qui l'utilisent si les spécifications d'interface demeurent inchangées.

— Indépendance pour la mise au point. Son interface ayant été spécifiée, une machine M peut être mise au point indépendamment de celles qui l'utilisent ; réciproquement, une machine M étant réalisée, les machines utilisant M peuvent être mises au point indépendamment de M.

1.5.2 Notion d'objet

La décomposition hiérarchique que l'on vient de présenter répond mal à certains aspects de structuration des systèmes. En particulier, la description de collections d'éléments ayant des caractéristiques communes, la création ou la destruction dynamique d'éléments seront facilitées grâce à la notion d'objet, nouvel outil de structuration permettant d'exprimer des concepts importants tels que : désignation, liaison, type, protection.

Un objet est concrétisé par une représentation : représentation externe à laquelle a accès l'utilisateur et représentation interne qui est celle concernant le système. Cette représentation des objets ainsi que la façon d'y accéder font appel à des fonctions d'accès.

Un autre concept permet de regrouper un ensemble d'objets ayant des caractéristiques communes : c'est la notion de classe. Les opérations et fonctions d'accès associées à une classe sont applicables à chaque objet de la classe. Il peut advenir que l'on désire définir des ensembles d'objets ayant à la fois des propriétés communes avec une classe déjà existante et des propriétés particulières. La notion de sous-classe permet d'y parvenir.

Une sous-classe hérite des propriétés associées à la classe mère, auxquelles s'ajoutent des propriétés spécifiques. Cette sous-classe pourra à son tour être considérée comme classe mère d'un autre ensemble, etc. On obtient ainsi une hiérarchie de classes. Citons quelques classes bien connues, et la ressource physique associée dont elles constituent une abstraction :

— Les fichiers ==> mémoire secondaire
— Les flots ==> organes périphériques
— Les processus ==> processeur
— La mémoire virtuelle ==> mémoire physique.

Les fonctions d'accès associées à une classe quelconque permettent, entre autre, de créer ou de supprimer des objets de cette classe et donc d'en faire varier dynamiquement le nombre. Une fois créé, un objet dispose d'un état représenté par ses propres données qui peuvent varier dans le temps.

1.5.3 Interfaces et spécifications

Comme nous l'avons vu dans les deux paragraphes précédents, une interface est associée, soit à une machine abstraite, soit à une classe d'objets. Elle comporte trois types d'information :

— Des structures de données
— Des procédures
— Des règles d'utilisation des données et procédures exprimant des restrictions :
  — Restrictions d'accès aux données (lecture seule autorisée...) ;

— Restrictions sur l'ordre des procédures ;
— Contraintes de simultanéité dans l'exécution des procédures ou l'accès aux données.

A l'heure actuelle, aucune solution satisfaisante n'a encore été proposée pour exprimer les spécifications d'une interface ou les contraintes d'utilisation, si ce n'est le recours au langage naturel.

Deux méthodes sont utilisées pour prendre en compte, dans la spécification, les éventuelles erreurs :

— Chaque procédure comporte un paramètre supplémentaire (code d'erreur) modifiable par la procédure. La valeur finale de cette variable constitue un compte-rendu interprétable à un niveau supérieur.
— Á chaque cause d'erreur est associée une procédure de traitement spécifique. En cas d'erreur, un mécanisme que nous détaillerons ultérieurement (déroutement), déclenche automatiquement la procédure correspondante.

Dans tous les cas, le traitement d'une erreur consistera à revenir à un état stable du système où l'exécution puisse reprendre normalement, en perdant le moins d'information possible. Des deux méthodes de prise en compte d'erreur présentées plus haut, la seconde nécessite un mécanisme supplémentaire, mais elle sera préférable à la première pour deux raisons essentielles :

— Sécurité : le caractère systématique de déroutement en cas d'erreur est supérieur au test de code qui peut être omis, provoquant ainsi une propagation d'erreur.
— Clarté : le fait de pouvoir associer une procédure particulière à chaque cause d'erreur permet de séparer clairement le traitement des situations normales de celui des situations exceptionnelles.

Les méthodes de conception que nous venons de présenter et les concepts ou outils d'abstraction qui leur sont associés permettent, on l'a vu, grâce à la modularité ainsi acquise, de diviser et subdiviser un système informatique en plusieurs parties indépendamment modifiables ou même interchangeables. Cet aspect s'avère primordial, car les systèmes élaborés à l'heure actuelle sont de plus en plus importants et de plus en plus complexes. Si bien que leur réalisation est confiée à plusieurs personnes, à divers services, voire à plusieurs équipes. La cohérence du tout ne peut être facilement obtenue qu'à la condition d'avoir préalablement clairement d´efini les spécifications d'interface et l'arborescence des classes d'objets manipulés,... mais seulement cela.

D'autre part, certaines portions du système peuvent être conçues de différentes façons en utilisant différentes stratégies. Dans ces conditions, les concepteurs devront tester celles-ci sur des critères de rapidité, d'optimisation d'occupation, d'utilisation de ressources, etc. Le respect des contraintes d'interfaçage autorisera la mise au point et l'évaluation de performance de ces parties par simple substitution sans affecter le reste du système.

A propos de ce cas de figure, les contraintes s'avèrent très strictes, la substitution d'un module par un autre module n'étant possible qu'à la condition que les autres modules n'y accèdent qu'en utilisant son interface. En particulier, un programme appelant un module ne devra en aucune façon exploiter des renseignements sur la réalisation interne du module. Une méthode efficace pour parvenir à ce but a été

proposée par Parnas : laisser les programmeurs d'un module dans l'ignorance de la réalisation des autres.

1.5.4 Les composantes d'un système d'exploitation

L'étude des composantes permet de fixer les rôles de chaque couche logicielle et les rapports entre ces couches. Nous allons par la suite distinguer les modules suivants (voir la figure 1.5) :

- **Le gestionnaire d'interruptions** récupère les interruptions matérielles et logicielles et applique le traitement approprié qui varie sur la cause de ces interruptions.
- **Les pilotes de périphériques (drivers)** gèrent l'échange des données avec les périphériques. Chaque pilote connaît son périphérique et cache son mode d'utilisation aux couches supérieures du système. Ces drivers utilisent les interruptions car le dialogue asynchrone entre CPU et unités externes s'effectue au moyen des interruptions. En d'autres termes, le pilote envoie des ordres à son périphérique qui répond au bout d'un temps non défini par le biais d'une interruption.
- **Le système d'E/S** masque les drivers de périphériques. Il offre des fonctions d'E/S qui, bien qu'étant de bas niveau, ne distinguent pas de manière explicite la nature du périphérique. Ces E/S sont réalisées à partir (ou vers) des zones de la mémoire appelés des tampons (buffer). L'allocation de ces tampons passe donc par le gestionnaire de la mémoire centrale.
- **La gestion de la mémoire centrale** répond aux demandes d'allocation et de libération de zones mémoire. Dans une première approche, la mémoire virtuelle peut être vue comme une extension de la mémoire centrale qui est temporairement rangée sur disque. Ce déplacement d'une partie de la mémoire implique :
  - Le retour à la demande des informations utiles et non présentes en mémoire centrale (c'est une opération d'E/S) ;
  - La sauvegarde sur disque des informations présentes mais inutilisées.
- **Le système de gestion des fichiers (SGF)** offre toutes les primitives nécessaires à la création, destruction, modification des fichiers se trouvant en mémoire secondaire.
- **La gestion des processus** répartit la ou les CPU entre les tâches qui en ont besoin. Ces tâches consomment de la mémoire et exploitent des fichiers.
- **Les processus utilisateur** (dont les interpréteurs de commande sont un exemple particulier) utilisent le S.E. en lui adressant des requêtes en bonne et due forme. Ces requêtes permettent, au choix :
  - De lancer, de figer ou de tuer d'autres processus
  - D'exploiter ou de modifier des fichiers
  - D'allouer de la mémoire, etc.

1.6 Travaux dirigés

Exercice 1 *Question à réponses ouvertes*

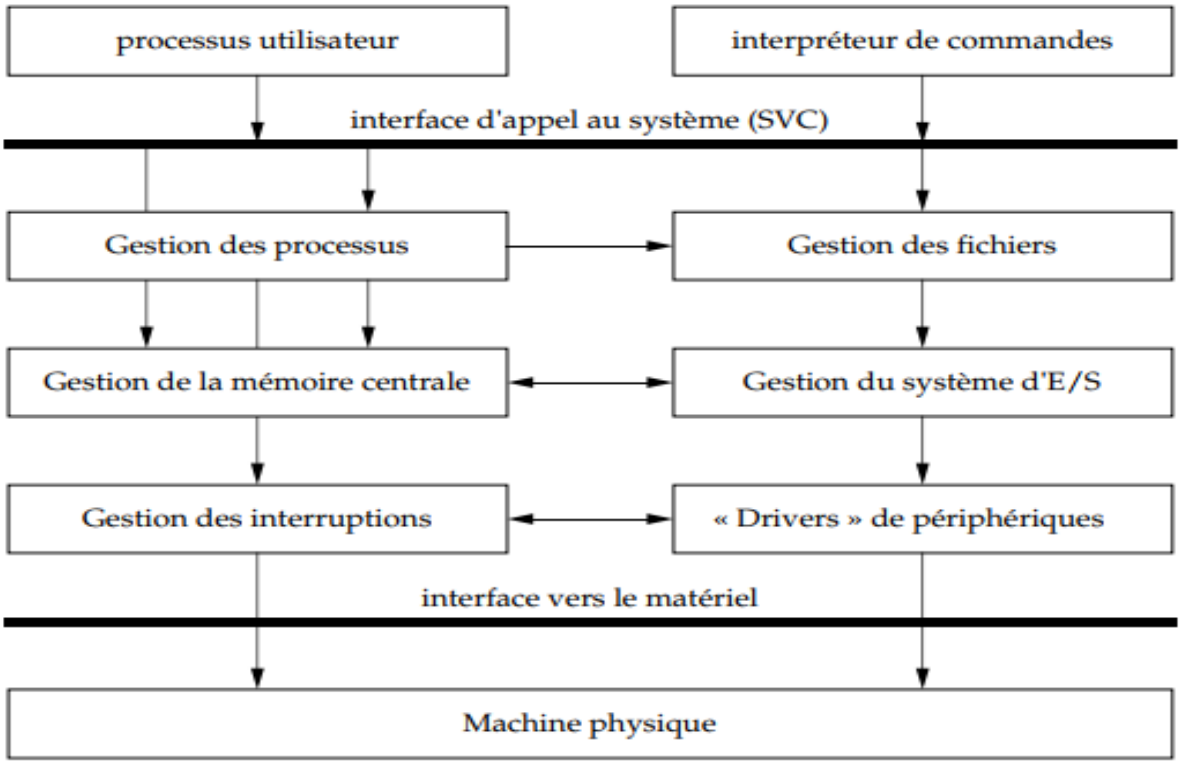


Figure 1.5 – *Les composantes et la structure d'un système*

1. *Définir les termes et expressions suivants : ordinateur, logiciel, système d'exploitation*
2. *Quelles sont les principales fonctions qu'un système informatique doit assurer ?*
3. *Présentez et expliquez en vos propres mots les principales fonctions d'un système d'exploitation*
4. *Faire une étude de l'évolution des systèmes d'exploitation. L'étude devra être présentée dans un tableau ou vous ferrez ressortir les points forts et les points faible de chaque génération de système.*
5. *En vous refférant à la figure 1.5, présentez le rôle d'un SE, et dire en quoi chacun de ces rôles consiste.*
6. *Qu'est-ce qu'un SE temps réel ? Citez cinq exemples avec leurs éditeurs respectifs.*
7. *Présentez, exemples à l'appuie les différents niveaux de langage de programmation à votre connaissance.*

Exercice 2 *Organisation d'un système d'exploitation On considère un ordinateur dont les organes périphériques sont un lecteur de cartes (1000 cartes par minute) et une imprimante (1000 lignes par minute). Un travail moyen est défini par :*

- *Lire 300 cartes*
- *Utiliser le processeur 1 min*
- *Imprimer 500 lignes*

— *Temps de réflexion 3,2 min.*

On suppose que tous les travaux soumis par les usagers ont des caractéristiques identiques à celles de ce travail. On définit deux mesures de performances du système :

— *Le débit moyen D des travaux : nombre de travaux exécutés pendant 1 heure*
— *Le rendement h de l'unité centrale : fraction du temps total d'utilisation de l'U.C. pendant lequel elle exécute du travail utile (autre que gestion des périphériques)*

1. ***Porte ouverte et moniteur d'enchaînement des tâches***
   On suppose d'abord que les périphériques sont gérés par l'U.C. Calculer h et D dans les hypothèses de fonctionnement suivantes :
   (a) *Le système est exploité en porte ouverte ; durée de session : 15 min*
   (b) *Le système est exploité avec un moniteur d'enchaînement séquentiel des travaux.*
2. ***Traitement par lots***

   On suppose maintenant que les périphériques sont gérés par un ordinateur séparé, qui constitue une bande magnétique d'entrée à partir des cartes et liste sur l'imprimante le contenu d'une bande magnétique de sortie. L'ordinateur principal est alimenté par la bande d'entrée et produit la bande de sortie ; on néglige la durée de lecture et d'écriture des bandes. Le temps de permutation des bandes d'un ordinateur à l'autre est de 5 min dans chaque sens ; on suppose qu'une bande regroupe une fournée de 50 travaux ; on possède au moins trois bandes.
   (a) *On suppose que le rythme de soumission des travaux est suffisant pour occuper l'ordinateur central à plein temps. Calculez les valeurs de h et de D.*
   (b) *Etablir la planification de la constitution des trains de travaux et calculer le temps d'attente moyen d'un usager (entre soumission du travail et réception résultats). On admettra que les travaux arrivent à un rythme régulier, que le temps de constitution d'une fournée est de 10 min et que le temps de distribution des résultats est aussi de 10 min.*
3. ***Utilisation des cannaux d'E/S***

   Les périphériques sont maintenant gérés par un canal d'entrée-sortie. Le système est mono-programmé et le moniteur d'enchaînement permet à l'U.C. d'exécuter le traitement d'un travail parallèlement à la lecture du suivant et à l'impression du précédent.
   (a) *Calculer dans ces conditions h et D*
   (b) *Même question si le travail moyen lit 1200 cartes et imprime 1500 lignes pour 1 min d'utilisation d'U.C.*
4. ***Utilisation des tampons d'E/S***

Les entrées-sorties sont maintenant gérées avec un tampon en mémoire centrale (buffer). Le travail moyen est celui défini à la question 3 pour une fournée de 50 travaux

- *On suppose qu'une carte et une ligne d'impression occupent respectivement 80 et 100 octets. Quelle est la taille minimale nécessaire des tampons de lecture et d'impression pour que l'U.C. soit utilisée à son rendement maximal ? Quel est alors le débit des travaux ?*
- *Le rythme d'arrivée des travaux et la taille du tampon de lecture sont ceux calculés en a), et la taille du tampon d'écriture est de 2 méga-octets. Quel est le rendement de l'U.C ?*

Exercice 3 *Installation d'un système d'exploitation Linux/Ubuntu*

1. *Téléchargez et installez une version de Linux/Ubuntu sur votre machine*
2. *Décrivez toutes les étapes que vous avez suivi pour la précédente installation*
3. *Après avoir défini le sigle BIOS, donnez la combinaison de touches du clavier à valider pour accéder à celui de votre machine*
4. *Décrivez les informations que l'on retrouve sur le BIOS de votre machine*

1.7 Conclusion

Dans ce chapitre, il était question de présenter les concepts fondamentaux nécessaires à la compréhension des S.E. C'est ainsi qu'après avoir définis quelques mots clefs liés aux SE, nous avons situé la place de ces derniers dans un système informatique, par la suite nous avons présenté leur structure interne, puis nous avons présenté leur évolution au fil du temps. Nous avons bouclé le chapitre par une fiche de travaux dirigés qui avait pour objectifs d'une part de mesurer le degré de compréhension des apprenants relativement aux objectifs énoncés au début du chapitre, et d'autres part de permettre aux apprenants d'étendre leurs connaissances sur les généralités liées aux SE qui n'ont pas été abordées dans le chapitre.

Chapitre 2

Interruptions, déroutements et appels système

2.1 Objectifs

Avant d'entrer plus en détail dans l'étude des éléments constitutifs d'un SE, nous jugeons utile de présenter le mode d'exécution des programmes, car ce mode impacte significativement sur la gestions des ressources matérielles par le SE. Ainsi, à la fin de ce chapitre, l'apprenant devra :

— Comprendre le mode d'exécution des programmes sur une machine
— Définir et comprendre le mécanisme des interruptions
— Donner le rôle des interruptions
— Savoir à quoi renvoie les notions d'appel système et de déroutement

La charge horaire destinée à ce chapitre est départie comme suit :

1. Cours magistral : 5h
2. Travaux dirigés : 3h
3. Travail personnel de l'étudiant : 1h

2.2 Exécution de programmes

Avant de préciser ce que nous entendons par interruption, il est souhaitable de définir rapidement la notion d'exécution d'un programme sur une machine. En fait, notre objectif est de présenter un modèle général valable sur la plupart des machines.

Une machine est composée schématiquement d'un processeur (aussi appelée la C.P.U.), d'une mémoire principale et d'organes d'E/S. Le processeur est un circuit actif qui comporte des registres généraux et des registres spécialisés. L'ensemble des registres spécialisés forment le mot d'état du processeur (M.E.P. ou P.S.W. pour Processor Status Word). Parmi ces registres on trouve :

— Le compteur ordinal (CO) qui contient l'adresse de la prochaine instruction à exécuter ;
— Le mode d'exécution (MODE) qui peut être maître ou esclave ;
— Le masque d'interruptions que nous détaillerons plus tard.
— Un ou plusieurs pointeur(s) de pile.

— etc.

La notion de mode a été introduite essentiellement pour des raisons de protection, afin qu'un programme quelconque ne puisse accéder à des zones ou à des registres propres au S.E. Dans la pratique, la CPU distingue les instructions normales et les instructions privilégiées. Ces dernières ne sont utilisables qu'en mode maître. Elles permettent en général la modification des registres spécialisés et le dialogue avec les unités d'E/S.

Partons du principe que le S.E. s'exécute en mode maître et que les programmes utilisateur s'exécutent en mode esclave. La programmation directe des E/S est donc réservée au S.E. et les E/S des programmes utilisateurs devront passer par des requêtes au S.E.

Une exécution est une évolution discrète de l'état de la machine. Cet état est donné par le contenu de la mémoire et la valeur des registres de la CPU. Nous sommes donc capables d'observer l'évolution d'une machine mais seulement sur certains points que nous appellerons les points observables ou points interruptibles. Ces points sont situés dans le temps à la fin de l'exécution d'une instruction de la CPU. Le schéma 2.1 d´ecrit sommairement l'évolution des registres d'une C.P.U. simplifiée lors de l'exécution d'un programme

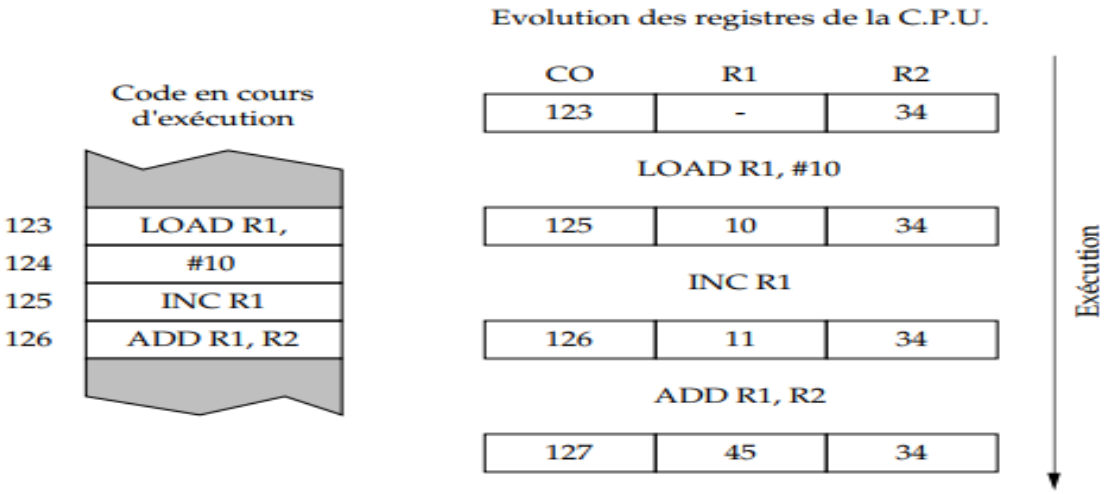


Figure 2.1 – *Un exemple d'exécution*

2.3 Le mécanisme des interruptions

Dans tous les types de système, il est toujours nécessaire de considérer un travail courant (le programme en cours d'exécution) et un travail exceptionnel dont le but est de traiter un événement. On peut citer les exemples suivants :

— Dans les systèmes de conduite de processus, certains événements importants (voir même graves) doivent être pris en compte dans les délais les plus brefs. En d'autres termes, il faut donc interrompre le travail courant (relevés des capteurs), pour exécuter un programme prioritaire.

— Il existe toujours un dialogue entre l'U.C. et les organes d'E/S. Notamment, une unité de disque ou une imprimante signalent à l'U.C. que l'E/S est terminée. Dans ce cas également, le travail courant doit être interrompu pour prendre en compte cette nouvelle situation de manière à optimiser l'utilisation des organes d'E/S.

Ces deux exemples ont un point en commun : les événements exceptionnels sont asynchrones c'est à dire qu'il n'est pas possible de prévoir leur arrivée. Pour contourner ce problème, les machines disposent d'un mécanisme général permettant de traiter ces événements asynchrones. C'est ce mécanisme complexe qui vous est présenté dans cette section.

Les interruptions permettent d'interrompre provisoirement le déroulement d'un programme en cours pour exécuter une routine considérée comme prioritaire. On associe à chaque cause d'interruption un numéro k qui l'identifie. On dispose également dans les adresses basses de la mémoire d'une table appelée le vecteur d'interruptions (vi). Les cases vi[k] de cette table contiennent l'adresse de la routine à exécuter lors d'une interruption de cause k. Cette routine est appelée le traitant d'interruption de cause k.

Plus précisément, lors d'une interruption de cause k, la CPU effectue dès la fin de l'instruction en cours les actions suivantes :

1. Sauvegarder la valeur du compteur ordinal et le mode d'exécution (dans une pile ou dans une case mémoire particulière suivant les C.P.U.) ;
2. Passer en mode maître ;
3. Forcer dans le compteur ordinal la valeur vi[k], c'est à dire l'adresse de la première instruction de la routine associée à l'interruption de cause k.

L'interruption est donc un mécanisme matériel puisque la sauvegarde et l'initialisation du compteur ordinal à partir du vecteur d'interruptions sont des opérations réalisées par la CPU. Le traitant représente la partie logicielle du mécanisme d'interruption. Il a (presque) toujours la structure suivante :

1. Sauvegarder la valeur des registres de la CPU (dans un emplacement particulier de la mémoire). Cette étape est couramment appelée la sauvegarde du contexte
2. Traiter la cause de l'interruption.
3. Restaurer la valeur des registres de la CPU et le mode du programme interrompu. C'est la restauration du contexte.
4. Forcer dans le compteur ordinal la valeur préalablement sauvegardée.

De cette description on tire deux conclusions : (1) les traitants d'interruption s'exécutent en mode maître (donc avec des droits étendus) ; (2) l'exécution du programme interrompu n'est pas perturbée par le traitement de l'interruption. L'étape 4 est souvent réalisée au moyen d'une instruction de la CPU qui provoque le retour au programme interrompu (RTI). Cette étape est appelée l'acquittement de l'interruption. Les principales utilisations du processus d'interruption sont les suivantes :

— Interruption logicielle (ou déroutement) provoquée par la CPU lors de la détection d'une situation anormale. Par exemple :
  — Appel explicite du S.E.

- Instruction incorrecte ou inconnue
- Violation de privilège
- Dépassement de capacité
- Division par zéro
- Tentative d'accès à une zone protégée de la mémoire.

— Interruption matérielle générée par une unité externe à la CPU afin de lui signaler l'apparition d'un événement extérieur. Par exemple :
- Fin d'une E/S
- Impulsion d'horloge
- Changement d'état d'un des périphériques
- Présence d'une valeur intéressante sur un capteur.

Certaines CPU n'ont qu'une seule cause d'interruption. Dans ce cas, un ou logique de toutes les causes possibles sera effectué et le traitant d'interruption qui est unique devra au préalable tester les indicateurs pour connaître la cause.

2.4 Les interruptions matérielles

Les interruptions matérielles se présentent comme un ensemble de fils numérotés reliant la CPU et les circuits externes de la machine. La présence d'un signal sur un de ces fils provoque une interruption du programme en cours d'exécution. Le numéro de cette interruption est directement lié au fil qui l'a déclenchée.

En résumé, un circuit extérieur génère une interruption sur la CPU afin de lui signaler un événement. Cette interruption stoppe le programme en cours pour lancer une routine du S.E. L'exécution de cette routine permet, dans les meilleurs délais, la prise en compte par le S.E. de l'événement extérieur

2.4.1 Système hiérarchisé d'interruptions

Les fils d'interruptions peuvent être hiérarchisés c'est-à-dire classés par ordre de priorités respectives. Un traitant d'interruption peut donc être lui-même interrompu par une demande d'interruption intervenant sur un fil de priorité supérieure. Il passe alors à l'état d'attente. La figure 2.2 représente l'activité des programmes dans le temps pour un système hiérarchisé à 8 niveaux où le niveau 0 est le plus prioritaire, le niveau 7 correspondant au programme d'arrière-plan.

Les systèmes d'interruption sont quelque fois plus élaborés et sont constitués d'un type d'organisation très modulaire ayant les caractéristiques suivantes :

— Les interruptions sont groupées en un certain nombre de niveaux hiérarchisés (décrits plus haut).

— Un niveau regroupe plusieurs sous-niveaux possédant chacun son fil d'interruption et sa priorité à l'intérieur du niveau ; les programmes associés aux sous-niveaux d'un même niveau ne peuvent s'interrompre les uns les autres, leur priorité respective n'intervenant que lors du choix si plusieurs d'entre eux sont en attente simultanément.

— Un sous-niveau regroupe lui-même plusieurs demandes d'interruptions, les causes d'interruption étant recherchées par test d'indicateurs.

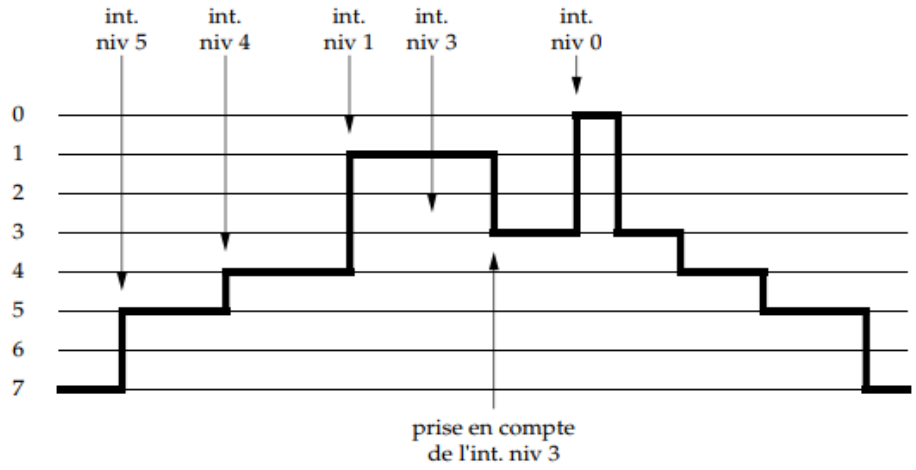


Figure 2.2 – *Effets de la hiérarchisation d'un système d'interruption*

2.4.2 Commande du système d'interruption

Chaque niveau d'interruption peut être dans l'un des états suivants :

- État désarmé : le niveau n'accepte aucune demande d'interruption.
- État armé : le niveau accepte et mémorise une demande d'interruption. On peut armer ou désarmer un niveau d'interruption par programme en utilisant des instructions privilégiées. Cette possibilité est donc réservée au S.E.
- État masqué : le niveau a été inhibé par programme de sorte que l'interruption a pu être mémorisée mais ne peut être prise en compte par la CPU.
- État d'attente : l'interruption peut être prise en compte immédiatement si deux conditions sont remplies :
  - Aucun niveau de priorité supérieure n'est en état d'attente ;
  - La CPU se trouve dans une phase interruptible (fin d'instruction). Le niveau passe alors à l'état actif.
- État actif : il implique la prise en compte de l'interruption par la CPU et dure pendant toute la durée du traitant d'interruption.

Des instructions privilegiées permettent d'armer (ou de désarmer), d'autoriser (ou de masquer), de déclencher un ou plusieurs niveaux d'interruption. Lorsque le nombre de niveaux d'interruption est limité, un registre spécialisé de la CPU contient ce que l'on appelle le masque d'interruption. A chaque niveau est associé un bit indiquant s'il est autorisé ou masqué.

2.5 Les appels systèmes

Nous avons vu précédemment que les programmes utilisateur s'exécutent en mode esclave. Les instructions privilégiées permettant la programmation des E/S leur sont donc interdites. Dans ces conditions, toute demande d'E/S et plus généralement toutes les actions demandant des droits étendus, passent par une requête en bonne et due forme au S.E.

Cette requête est réalisée par le truchement d'une instruction de la CPU qui provoque une interruption. Nous l'appellerons SVC pour SuperVisor Call mais on

utilise aussi le terme TRAP. Cette solution, bien que compliquée, a les avantages suivants :

— L'interruption provoque un branchement vers le traitant d'interruption mais aussi un changement de mode. Il y a donc un passage automatique du programme utilisateur en mode esclave au S.E. en mode maître.
— Il existe un et un seul point d'entrée vers le S.E. pour les processus utilisateur. Il est donc plus facile (du point de vue du concepteur du système) de sécuriser l'appel des primitives système.
— Si on part du principe que le vecteur d'interruptions se trouve dans une zone inaccessible au programme utilisateur, alors ce dernier n'a aucun moyen de passer en mode maître et l'instruction SVC est le seul point de passage.

Généralement, un appel système a la structure ci-dessous. Heureusement, les librairies standards disponibles dans tous les systèmes de développement offrent une interface plus agréable et se chargent de programmer en assembleur l'appel du système. Le choix entre les diverses routines se fait non pas par adressage (comme c'est le cas pour un sous programme) mais au moyen d'un paramètre supplémentaire passé soit dans un registre, soit dans la partie opérande de l'instruction SVC.

En fait, vu du programme utilisateur, et mise à part la forme de l'instruction d'appel elle même (SVC supervisor call), tout semble se passer comme un appel de procédure (empilement de l'adresse de retour, des paramètres...) mais en fait, comme nous venons de le voir, le mécanisme est beaucoup plus complexe.

2.6 Les déroutements

Un déroutement est une interruption qui intervient lorsqu'une anomalie a été détectée dans le déroulement d'une instruction, empêchant ainsi son exécution. On distingue trois types de causes :

1. Données incorrectes (division par zéro, débordement arithmétique, etc.) ;
2. Tentative de violation d'une protection et/ou d'une interdiction (violation de protection mémoire, utilisation d'une instruction privilégiée en mode esclave, etc.) ;
3. Impossibilité d'exécution d'une instruction (instruction inconnue ou instruction optionnelle absente de la configuration utilisée, etc.).

Selon la cause d'un déroutement, on peut éventuellement en supprimer l'effet. Ainsi, par exemple, on peut récupérer les erreurs arithmétiques ou encore les lectures au delà de la fin de la mémoire. Toutefois, le caractère strictement synchrone des déroutement interdit leur retard de prise en compte comme cela est possible pour les interruptions : en l'occurrence, la notion de masquage ne peut s'appliquer.

En résumé, le mode esclave, les déroutements vers le S.E. en cas d'erreur et le mécanisme des appels système imposent un cadre strict pour l'exécution des programmes utilisateur. Les systèmes d'exploitation récents sont dits dirigés par les interruptions car ils ne s'exécutent que sur demande explicite. Cette demande provenant de l'extérieur (interruption matérielle) ou des programmes en cours d'exécution (déroutement et appel système)

2.7 Travaux dirigés

Exercice 1 *Question à réponses ouvertes*

1. *Parmi les instructions suivantes, lesquelles doivent être privilégiées ? ?*
   (a) *Changement des registres de gestion de mémoire*
   (b) *Ecriture du compteur de programme*
   (c) *Lecture de l'horloge*
   (d) *Réglage de l'horloge*
   (e) *Changement de priorités du processeur*
2. *Du point de vue de l'exécution d'un programmes, quels sont les constituants d'une machine ?*
3. *Quels sont les éléments constitutifs d'un processeur ?*
4. *Citez quelques registres qui forment le MEP.*
5. *De combien de mode d'exécution dispose le processeur ? citez-les en spécifiant les cas dans lesquels ils sont utilisés.*
6. *Qu'est-ce qu'une exécution ?*
7. *Quel est le rôle des interruptions dans un SE ?*
8. *Quelles sont les différentes actions exécutés par le SE en cas d'une Interruption de cause k ?*
9. *L'interruption est-elle une cause matérielle ou logicielle ? Justifiez votre réponse.*
10. *Quelle est la structure du traitant d'une interruption ?*
11. *Dans quel mode s'exécute le traitant d'une interruption ?*
12. *Quelles sont les principales utilisations du processus d'interruption ?*
13. *Un traitant d'interruption peut-il lui même être interrompu ? Si oui dans quelle circonstance ?*
14. *De combien de niveaux d'interruption dispose t-on ?*
15. *Présentez les différents états possibles d'un niveau d'interruption.*
16. *Qu'est-ce qu'un appel système ?*
17. *Qu'est-ce qu'un déroutement ?*
18. *Quelles sont les différentes causes possibles d'un déroutement ?*

Exercice 2 *On va travailler sur une machine très simple : la CPU ne comporte que trois registres spécialisés (le MODE, le compteur ordinal CO et le pointeur de pile SP) et deux registres généraux (R1 et R2). Pour simplifier, le mot d'état du processeur regroupe tous les registres.*

$$mep =< CO, MODE, SP, R1, R2 >$$

On vous rappelle que pour traiter les interruptions, il existe dans la machine un vecteur d'interruptions qui débute à l'adresse vi. Ce vecteur contient les adresses des traitants associés à chacune des causes d'interruption (2 causes possibles dans notre cas). On définit les constantes suivantes :

Numéro	*Signification*	*Constante*
0	*déroutement sur erreur d'adressage*	*INT_ERR_ADR*
1	*déroutement sur instruction inconnue*	*INT_ERR_INSTR*

Lorsqu'une interruption de numéro k se produit, le mécanisme (câblé) des interruptions déroule les actions de la figure 2.3 : Le gestionnaire d'interruptions offre

```
mem[SP] := ⟨CO,MODE⟩; // empiler le CO et le MODE
SP := SP + 1;
MODE := Maitre; // passer en mode maître
CO := vi[k]; // se brancher sur le traitant
```

FIGURE 2.3 – *mecanisme*

deux routines permettant à un traitant de sauvegarder et de restaurer facilement la valeur des registres de la CPU (voir fig 2.4) :

```
sauver_mep(var m:mep)
début
    SP := SP - 1;
    ⟨m.CO,m.MODE⟩ := mem[SP]
    m.SP := SP
    m.R1 := R1
    m.R2 := R2
fin
```

```
charger_mep(m:mep)
début
    SP := m.SP;
    mem[SP] := ⟨m.CO,m.MODE⟩
    SP := SP+1
    R1 := m.R1
    R2 := m.R2
    RTI
fin
```

FIGURE 2.4 – *Registres de la CPU*

L'instruction RTI (Return To Interrupted) se charge de revenir au code interrompu. Elle est définie de la manière suivante (fig 2.5) :

```
SP := SP - 1;
⟨CO,MODE⟩ := mem[SP]
```

FIGURE 2.5 – *Fig*

Attention : on ne revient pas de l'appel à la fonction charger_mep.

— *On cherche à écrire l'algorithme permettant de déterminer la taille réelle de la mémoire. Pour cela, nous allons utiliser le déroutement INT_ERR_ADR. La mémoire est organisée en barrettes, chaque barrette contient TAILLE_BANC mots de 32 bits, et nous ferons l'hypothèse importante que la mémoire est contiguë (c'est-à-dire qu'une fois que vous avez dépassé, vous êtes sûrs qu'il n'y a plus rien après). Ce programme fonctionne en mode maître (il est exécuté au démarrage du système).*

— *Sur certaines configurations (moins chères que les hauts de gamme), certaines instructions au lieu d'être câblées sont simulées de manière logicielle. Le mécanisme de déroutement est utilisé dans de tels cas de figure pour programmer cette simulation.*

1. *On vous demande d'écrire le traitant du déroutement INT_ERR_INSTR de manière à ce que les deux nouvelles instructions INC_R1 (incrémenter R1) et INC_R2 (idem) soient simulées correctement.*

Remarque 1. *En cas de déroutement, le compteur ordinal pointe sur le code opération de l'instruction qui a provoqué l'erreur.*

Remarque 2. *Le traitement des erreurs se résume à l'appel d'une fonction erreur().*

2. *En utilisant le mécanisme de la question précédente, on vous demande de simuler l'exécution d'une nouvelle instruction qui s'appelle SVC. Cette instruction possède un argument. Suivant la valeur de cet argument (1 ou 2), cette instruction incrémente le registre $R1$ ou $R2$. On peut (par exemple) l'utiliser de la manière suivante (fig 2.6) :*

```
LOAD R1, #10; // charger le registre R1 avec 10
SVC 1 // incrémenter le registre R1
```

Figure 2.6 – *Fig*

2.8 Conclusion

Nous avons débuté ce chapitre par la présentation du mode d'exécution des programmes dans un ordinateur. Puis nous avons présenté le mécanisme d'interruptions ainsi que les potentielles causes d'interruptions. Nous avons ensuite poursuivi avec la présentation des appels systèmes et des déroutements et avons achevé le chapitre par les travaux dirigés relatifs aux éléments élaborés tout au long du chapitre.

Nous considérons le présent chapitre comme le point d'entrée nécessaire à la bonne compréhension des différents autres composants du SE que sont la gestion de la mémoire, des fichiers, des processus, ... Ainsi, dans le capitre suivant, nous entrons dans le vif du sujet, en commençant par la gestion des processus et de leur ordonnancement.

Chapitre 3

Les processus et leur ordonnancement

3.1 Objectifs

Ce chapitre a pour principal objectif de présenter la façon dont les instructions sont exécutées en machine, en insistant sur l'ordonnacement des tâches telles qu'elles sont définies par le SE. A la fin du chapitre, l'apprenant devra être cappable :

— De définir la notion de processus ainsi que les différents états de ce dernier
— De comprendre les principes par lesquels le SE alloue le processeur aux processus
— Identifier les différents états d'un processus
— De créer les processus dans un SE
— D'identifer la différence entre un processus léger et un processus lourd

La charge horaire destinée à ce chapitre est répartie comme suit :

1. Cours magistral : 4h
2. Travaux dirigés : 2h
3. Travaux pratiques : 4h
4. Travail personnel de l'étudiant : 1h

3.2 Qu'est-ce qu'un processus ?

Un **processus** est un programme en cours d'exécution. Il faut d'emblée faire la différence entre un programme qui est un fichier inerte regroupant des instructions de la CPU et un processus qui un élément actif. Figeons un processus pour en observer ses composantes. Nous trouvons :

— Des **données** (variables globales, pile et tas) stockées dans une zone de la mémoire qui a été allouée au processus ;
— La **valeur des registres** (généraux et spécialisés) de la CPU lors de l'exécution ;
— Les **ressources** qui lui ont été allouées par le système d'exploitation (mémoire principale, fichiers ouverts, périphériques utilisés, etc.) ;

L'ensemble de ces composantes forme le **contexte d'exécution** d'un processus ou plus simplement le **contexte**

3.3 État d'un processus

Un processus n'est pas continuellement en train de s'exécuter. Si la machine comporte n processeurs identiques, à un instant donné il y a au maximum n processus actifs. En fait, parmi tous les processus qui sont susceptibles de s'exécuter, seulement un petit nombre s'exécutent réellement. L'allocation de la CPU aux processus qui la réclament est appelée **l'ordonnancement** de la CPU.

L'état opérationnel d'un processus est un moyen de représenter les différentes étapes de ce processus telles qu'elles sont gérées par le système d'exploitation. Le schéma 3.1 montre les divers états dans lesquels, dans une première approche intuitive, peut se trouver un processus :

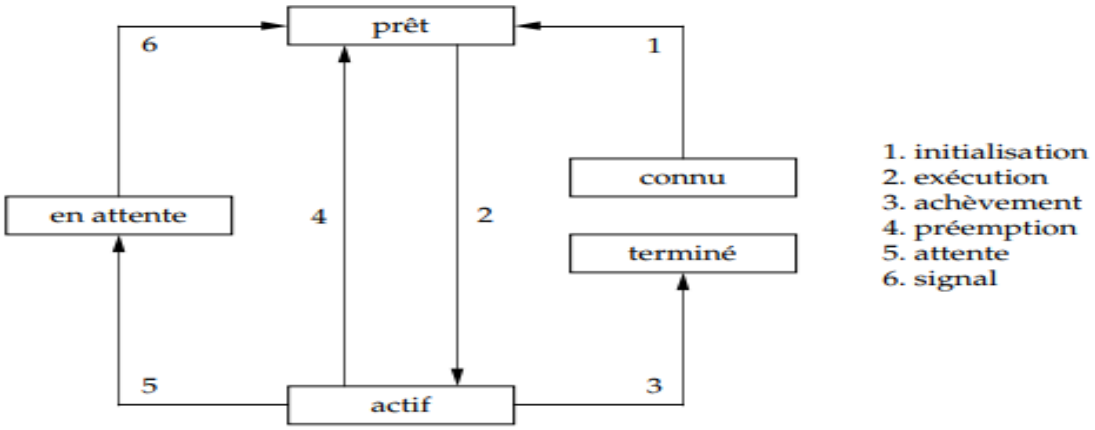


Figure 3.1 – *Schéma simplifié d'un CPU*

1. Initialement, un processus est connu du système mais l'exécution n'a pas débuté.
2. Lorsqu'il est initialisé, il devient prêt à être exécuté (1).
3. Lors de l'allocation de la CPU à ce processus il devient actif (2). Trois cas peuvent alors se présenter :
   (a) Le processus se termine (3)
   (b) Le processus est en attente (5) d'un événement et dès sa réception il redeviendra prêt (6)
   (c) Le processus est suspendu et se remet dans l'état prêt (4). Il y a réquisition ou préeemption de la CPU. Dans ce cas, le S.E. enlève la CPU au processus qui la détient.

La notion d'attente d'un événement mérite par son importance et sa complexité un petit exemple. Un éditeur de texte enchaîne continuellement la boucle de la figure 3.2 :

Lorsque le processus éditeur est actif il adresse une requête au S.E. pour lui demander une opération d'E/S (la lecture d'un caractère). Deux cas se présentent :

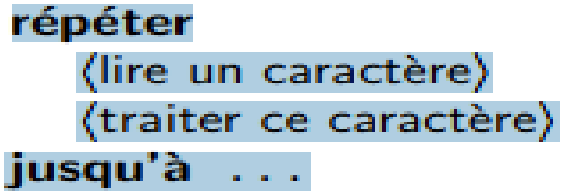


Figure 3.2 – *Un exemple d'exécution*

1. Si il existe un caractère dans le tampon d'entrée, ce dernier est renvoyé par le S.E. ;
2. Si le tampon d'entrée est vide, le S.E. va endormir le processus en changeant son état.

Lorsque l'utilisateur frappe une touche du clavier, le S.E. (qui avait préalablement sauvegardé la demande de l'éditeur) réveille l'éditeur qui pourra ainsi devenir actif et traiter ce fameux caractère.

Plus généralement, toutes les opérations lentes (en comparaison de la vitesse de la CPU) provoquent un arrêt momentané du processus demandeur et une reprise ultérieure lorsque l'opération est terminée. C'est notamment le cas pour les opérations d'E/S. Le but de ce mécanisme est de récupérer le temps d'attente pour exécuter un autre processus sur la CPU.

3.4 Gestion des processus

Les processus sont les principaux éléments actifs du système. Dans ce cadre, il est logique que la création d'un nouveau processus soit demandée par un processus. Il existe donc une filiation entre processus père et le(s) processus fils. Lors du démarrage de la machine, le S.E. lance un processus qui est orphelin puisqu'il n'a pas de père. Ce processus est souvent appelé init. Le premier rôle de ce processus est de lancer des fils qui auront chacun une fonction dans l'organisation générale de la machine. Par exemple

— Un processus pour gérer les E/S asynchrones avec les terminaux
— Un processus pour gérer les connexions au système avec demande et vérification d'un nom d'utilisateur et d'un mot de passe
— Un processus pour gérer l'allocation de la CPU aux processus
— etc.

Ces processus font partie du système d'exploitation. Ils s'exécutent donc avec des droits étendus. Nous les appellerons les processus système ou démons (daemons) par opposition aux processus utilisateur. Le système d'exploitation est donc composé d'un noyau résident qui ne s'exécute que sur demande explicite (interruptions et déroutements) et d'un ensemble de processus système qui ont chacun une fonction précise à assurer. Ce découpage présente deux avantages :

1. La partie résidente du système est réduite en taille ce qui permet d'éviter une trop grande consommation de mémoire par le système
2. Les processus systèmes ne sont pas forcément toujours prêts ou même toujours

présents en mémoire ce qui permet encore une fois de réduire la mémoire et le temps CPU consommé par le S.E. au détriment des processus utilisateur.

Si le système est organisé à base de plusieurs processus, des logiciels d'application peuvent également adopter cette structure. Si c'est le cas, il est nécessaire et même vital de fournir des outils permettant une communication et une synchronisation aisée entre les processus d'une même application. De plus, cette structuration à base de processus coopératifs est la seule capable d'utiliser facilement une structure matérielle multi-processeurs en associant un processus différent à chaque processeur. Nous avons parlé de la création d'un processus mais sa disparition est une étape importante ! Elle à lieu sur demande d'un processus étranger (système ou père) ou sur sa propre demande sous la forme d'un suicide. Ce dernier cas correspond à l'appel de la fonction standard exit() du langage C.

3.5 Poids lourds et poids légers

Nous avons évoqué plus haut les avantages liés à la structuration des applications sous la forme de processus coopératifs. Mais cette structure comporte également des inconvénients :

— Elle implique une communication massive entre les processus ce qui engendre un coût non négligeable de la part du système
— Elle augmente le nombre de commutations de contexte (c-à-d la sauvegarde et la restauration du contexte d'un processus interrompu) provoquant de ce fait une perte de temps de CPU.

La notion de thread et de systèmes multi-threads vise à régler ce type de problème. Dans les systèmes multi-threads un processus est défini comme un ensemble de threads. Un thread (aussi appelé processus de poids léger ou lightweight process LWP) est un programme en cours d'exécution qui partage son code et ses données avec les autres threads d'un même processus. Bien entendu, les piles sont propres à chaque thread pour éviter que les appels de fonctions et les variables locales ne se mélangent. Cette solution présente plusieurs avantages :

— Si un processus ne comporte qu'un seul thread nous revenons au modèle classique ; les systèmes multi-threads sont donc plus généraux
— Il n'y a plus à mettre en place une communication entre les threads d'un même processus puisqu'ils agissent tous sur les mêmes données
— Le temps de commutation entre les threads d'un même processus est réduit car le contexte est le même, et seuls les registres de la CPU doivent être sauvegardés
— En associant un (ou plusieurs) thread(s) à chaque processeur on peut facilement exploiter une structure multi-processeurs.

3.6 Ordonnancement des processus

Le systeme d'exploitation doit donc gérer l'ensemble des processus, dans le but de maintenir le taux d'utilisation du processeur le plus élevé possible. Pour cela, le

systeme d'exploitation met en oeuvre une politique d'ordonnancement, algorithme régissant la commutation des tâches

3.6.1 Critères d'ordonnancement

L'algortihme d'ordonnancement doit identifier le processus qui sera actif, de manière à assurer la "**meilleure**" performance du système. Selon les algorithmes, et selon les objectifs recherchés, différents critères vont être optimisés, parmi lesquels :

- Utilisation de l'UC : il s'agit de prendre en compte le temps pendant lequel l'unité centrale exécute un processus
- Utilisation répartie : c'est le pourcentage de temps pendant lequel est utilisé l'ensemble des ressources. On évalue l'utilisation de la mémoire, des E/S, .. plutôt que le temps processeur
- Débit : c'est le nombre de processus pouvant être exécutés par le système durant une période de temps donnée. Le calcul du débit doit prendre en compte la longueur moyenne d'un processus. Sur des systèmes aux processus longs, le débit est inférieur à celui des systèmes aux processus courts
- Temps de rotation : durée moyenne de l'exécution d'un processus. Il est inversement proportionnel au débit
- Temps d'attente : durée moyenne d'attente des processus du proceseur
- Temps de réponse : le temps moyen pour répondre aux entrées de l'utilisateurs (systèmes interactifs)
- Equité : degré auquel tous les processeurs reçoivent une chance égale de s'exécuter. On évalue entre autre le fait de ne pas permettre à un processus de souffrir de famine, à savoir rester bloqué indéfiniment.

3.6.2 Principes

Un processus est donc successivement en phases de calcul, et en phase d'entrées-sorties. L'idée est donc de tenter de se faire recouvrir une phase d'E/S d'un processus avec des phases de calcul d'un autre processus.

3.6.3 Algorithmes d'ordonnancement

Au début de l'informatique, l'ordonnancement était souvent non préemptif, ou coopératif : un processus conservait le contrôle de l'unité centrale jusqu'à ce qu'il se bloque ou qu'il se termine. Pour des travaux par lots, le système convenait, le temps de réponse n'ayant pas grande importance. Sur les sytèmes interactifs actuels, c'est l'ordonnancement préemptif qui est utilisé : le systeme d'exploitation peut préempter un processus avant qu'il ne se bloque ou ne se termine, afin d'attribuer le processeur à un autre processus. On recense six algorithmes d'ordonnancement répandus

FIFO - First In First Out

Le FIFO est le plus simple des alogrithmes d'ordonnancement. En effet, l'ordonnanceur (non préemptif), utilise une pile dans laquelle sont stockés dans l'ordre d'arrivée les processus en attente du processeur.

Exemple 1 *Soit trois processus P1,P2,P3 faisant uniquement du calcul. Les temps d'exécution des trois processus sont respectivement 15, 4 et 7 millisecondes. Le chronogramme d'exécution des processus peut être représenté comme sur la figure 3.3 :*

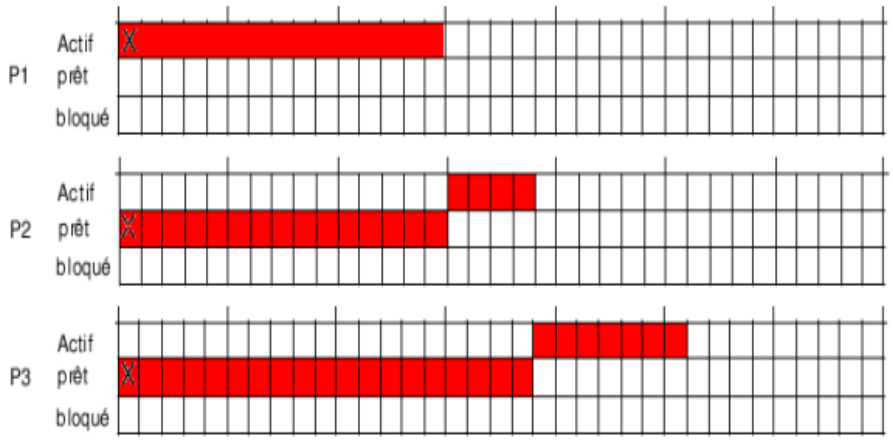
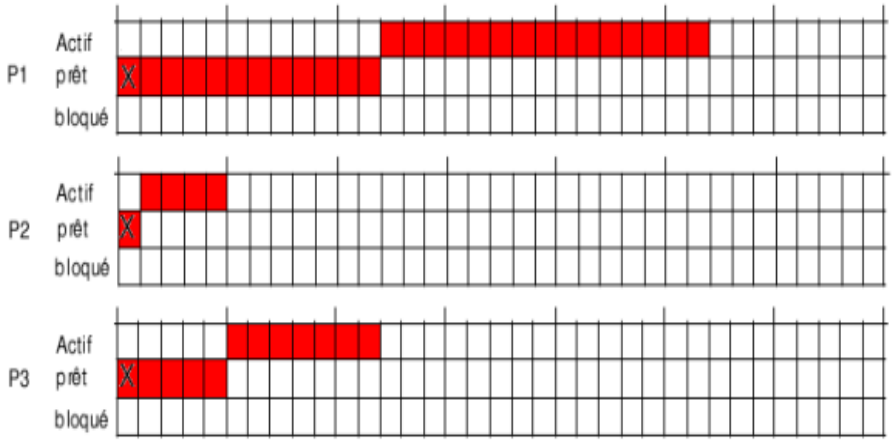


Figure 3.3 – *Exécution de l'algorithme d'ordonnancement FIFO*

L'ordonnancement FIFO a donc tendance a privilégier les processus longs ou tributaires de l'unité centrale

SJF - Shortest First Job

L'ordonnancement basé sur l'algorithme du travail le plus court d'abord est également non préemptif.A partir d'une estimation du temps necessaire à l'exécution des processus, le prochain élu est donc le plus court. Ce qui nous donne pour l'exemple précédent (en supposant qu'au départ les trois processus soient bloqués) la figure 3.4.



Exemple 2

Figure 3.4 – *Exécution de l'algorithme d'ordonnancement SJF*

Cet algorithme privilégie les programmes courts.

SR - Shortest Remaining Time

L'ordonnancement du temps restant le plus court est la version préemptif de l'algorithme SJF : chaque fois qu'un processus passe dans l'état prêt (à sa création ou à la fin d'une E/S), on compare la valeur estimée du temps de traitement restant à celle du processus en cours. On préempte le processus en cours si son temps restant est plus long. L'algorithme privilégie les programmes courts, mais il peut y avoir un risque de famine pour les programmes longs.

RR - Round Robin

L'ordonnancement par tourniquet (Round Robin) est préemptif, et sélectionne le processus attendant depuis le plus longtemps. Un quantum de temps est spécifié, et à chacun de ces intervalles, on choisit une nouvelle sélection. Plus le quantum est faible, plus les temps de réponse sont courts, mais plus les commutations sont fréquentes, d'où une dégradation possible des performances.

Exemple 3 *Nous reprenons l'exemple précédent, en supposant que P1,P2 et P3 ont été soumis dans cet ordre. Le quantum est fixé à 3ms. l'illustration est présentée à la figure 3.5*

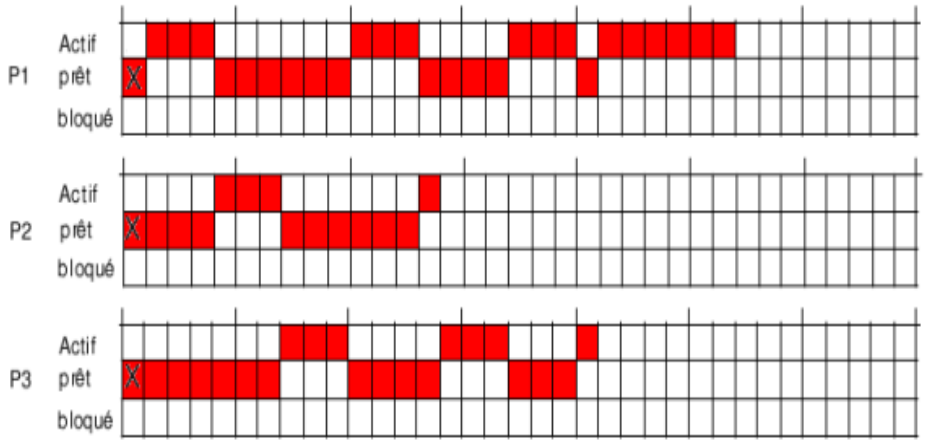


Figure 3.5 – *Exécution de l'algorithme d'ordonnancement RR*

Avec priorité

Pour l'ordonnancement préemptif à priorité, on affecte une valeur à chaque processus. Le processus actif est celui qui à la priorité la plus élevée (valeur la plus faible), et en cas d'égalité, on utilise FIFO. Les priorités peuvent être fixées en fonction des caractéristiques du processus (beaucoup d'E/S, utilisation de la mémoire,...), de l'utilisateur, ou même d'une priorité fixée par l'administrateur ou l'utilisateur (commande nice sous Unix). Pour éviter le risque de famine des processus de trop faible priorité, on peut augmenter la priorité en fonction du temps d'attente écoulé.

Exemple 4 *Soit P1,P2,P3 et P4 des processus de temps d'exécution respectifs 10,1,2,3, et de priorité respectives 3,1,3,2. La figure 3.6 présente l'exécution de l'algorithme avec priorité*

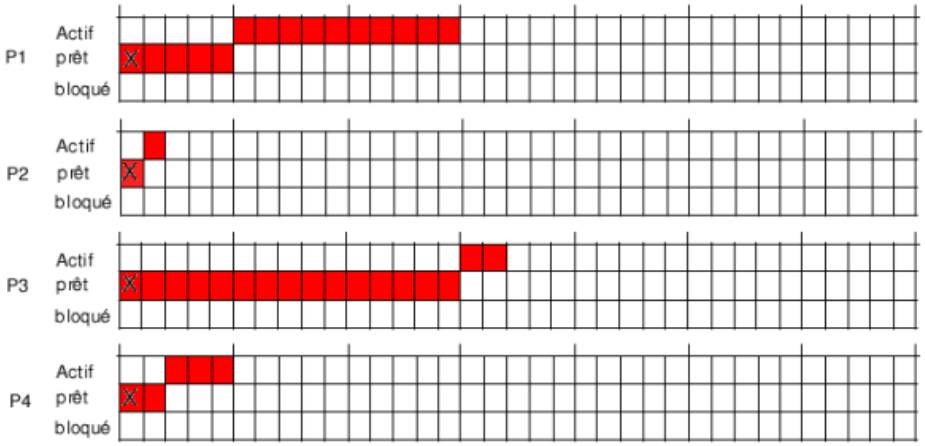
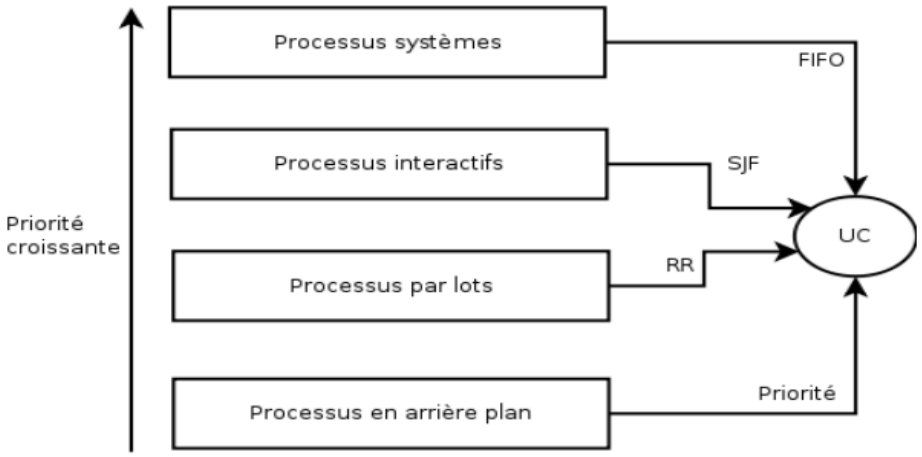


Figure 3.6 – *Exécution de l'algorithme d'ordonnancement avec priorité*

MQS - Multilevel Queue Scheduling

L'ordonnancement par files multi-niveaux est une extension de l'ordonnancement avec priorité, mais en plaçant les processus de même prioité dans des files d'attente distinctes. Par exemple, les systèmes en temps partagé supportent généralement la notion de processus en premier (foreground) et en arrière plan (background). Les premiers sont souvent des processus interactifs, alors que les seconds n'ont besoin de s'exécuter que si le processeur est libre. Donc les deux types de processus nécessitent deux ordonnancements différents



Exemple 5

Figure 3.7 – *Exécution de l'algorithme d'ordonnancement MQS*

Dans cet exemple, chaque file de processus est ordonnancée par un algorithme différent.

Les processus des files de plus basse priorité ne peuvent s'exécuter que lorsque les processus de la file supérieure sont tous bloqués. Il y a donc risque de famine.

3.7 Travaux dirigés

Exercice 1 *Question à réponse ouverte*

1. *Qu'est-ce qu'un processus ?*
2. *Quelles sont les composantes d'un processus ?*
3. *Qu'est-ce qui détermine le nombre d'instructions qui s'exécutent simultanément sur un processeur à un instant donné ?*
4. *Faire le schéma simplifié des différents états d'un processus.*
5. *Qu'est-ce qu'un processus orphelin ? Combien en existe-t-il dans le système d'exploitation ? Nommez-lesL.*
6. *Qu'est-ce qu'un processus démons ?*
7. *Qu'est-ce qu'un processus zombie ?*
8. *Quelle différence établissez-cous entre un processus de poids lourd et un processus de poid léger ?*
9. *Qu'est-ce qu'un algorithme d'ordonnancement ?*

Exercice 2 *Les algorithmes d'ordonnancement*
Soient cinq processus d´ecrits par la table ci-dessous. Déterminez l'ordre d'exécution de ces cinq processus pour chacun des algorithmes d'ordonnancement suivants : FIFO (premier arrivé premier servi), SJF (le plus court d'abord), SRT (le temps restant le plus court) et RR (tourniquet avec un quanta fixé à 10 unités de temps).

Pour chaque exécution et chaque processus, calculez le temps d'attente afin de comparer les stratégies d'ordonnancement.

processus	date d'arrivée	durée
P1	0	10
P2	5	29
P3	35	3
P4	29	7
P5	24	12

Exercice 3 ***Ordonnancement des processus***

1. *On considère trois tâches T1, T2 et T3 qui nécessitent respectivement 2UT, 4UT et 2UT pour leur exécution. On suppose que le temps de commutation (swap in et swap out) entre deux processus est de 1UT. Répondre aux deux questions suivantes en tenant compte du temps des swap in et swap out. UT signifie Unité de Temps.*
   (a) *Quelle est la durée totale d'exécution des trois tâches dans un traitement par lots (batch processing en anglais) ? Quelle est la durée des commutations (swap in et swap out) ?*
   (b) *Quelle est la durée totale d'exécution des trois tâches dans le cas où l'ordonnancement se fait avec l'algorithme du Tourniquet (Round Robin) ? Quelle est la durée des swap in et swap out ? On suppose que la valeur du quantum est de 2UT*
   (c) *Quel est le principal défaut de l'algorithme d'ordonnancement Highest Job Priority First (HJPF) ? HJPF choisit à chaque commutation le processus le plus prioritaire. Comment corriger cet inconvénient ?*

3.8 Conclusion

Chapitre 4

La gestion de la mémoire

4.1 Objectifs

Dans ce chapitre il est question de présenter le processus par lequel le système d'exploitation gère la mémoire centrale de l'ordinateur. Ainsi, à la fin du chapitre, l'étudiant devra être cappable :

— De comprendre le principe d'allocation de la mémoire aux processus
— D'établir la différence entre les différents types de partition de la mémoire
— De maitriser les principes de segmentation et de pagination de la mémoire
— De maitriser les techniques utilisées par le SE pour le partage de la mémoire

La charge horaire destinée à ce chapitre est départie comme suit :

1. Cours magistral : 5h
2. Travaux dirigés : 4h
3. Travail personnel de l'étudiant : 1h

4.2 Concepts de base

4.2.1 Mémoire logique

La notion de ressource logique conduit à séparer les problèmes d'utilisation d'une ressource particulière des problèmes d'allocation de cette ressource. Pour un processus, la mémoire logique est le support de l'ensemble des informations potentiellement accessibles, c'est à dire, l'ensemble des emplacements dont l'adresse peut être engendréee par le processeur lors de l'exécution de ce processus.

L'allocation de mémoire consiste à concrétiser cette mémoire logique par des supports physiques d'information tels que mémoire principale, disques magnétiques, etc. En bout de chaîne , l'accès d'un processus à une information se traduit par l'accès d'un processeur physique à un emplacement de mémoire principale adressable par ce processeur. L'information accessible à un processus est définie par :

— L'ensemble des informations désignables dans son programme (objets) ;
— L'ensemble des informations de désignation (noms) ;
— La mise en correspondance noms/objets.

Dans un programme écrit en langage évolué, noms et objets sont définis par ce langage. Ils sont différents de ceux que manipule le processeur physique. Le programme doit donc subir une série de transformations appelée liaison. Celle-ci comporte une étape de traduction (mise en correspondance des objets avec les emplacements mémoire et des noms avec les adresses relatives correspondantes), une étape d'édition de lien (liaison entre programmes traduits séparément), et enfin une étape de chargement (fixation définitive des adresses, jusque là définies à une translation près). La séparation conceptuelle des problèmes de désignation et liaison, d'une part, et des problèmes d'allocation mémoire, d'autre part, peut être schématisée par la figure 4.1.

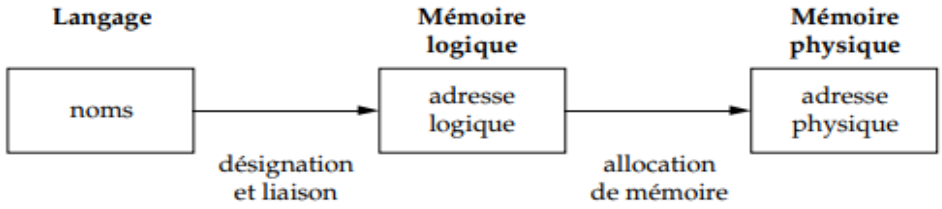


Figure 4.1 – *Transformation des adresses*

Le fait que la notion de mémoire logique ne soit pas restée un outil conceptuel, mais ait été mise en œuvre sur certaines machines par des dispositifs physiques de transformation d'adresse a conduit à ce que la séparation des fonctions soit plus ou moins bien respectée.

- Mémoire logique contiguë : Elle est constituée d'une suite d'emplacements identiques (mots) organisés de manière séquentielle et désignés par des entiers consécutifs appelés adresses logiques. Un objet est une information occupant un mot ou plusieurs mots consécutifs ; il est désigné par l'adresse logique du premier mot. Cette organisation est donc identique à celle des emplacements d'une mémoire physique.
- Mémoire logique non contiguë ou segmentée : Elle est constituée d'un ensemble de segments. Un segment est une suite de mots et regroupe généralement des informations de même nature. Il peut avoir une taille variable. Les mots contenus dans un segment sont désignés par des entiers consécutifs appelés déplacements. L'adresse logique d'un mot est donc un couple (numéro de segment, déplacement dans le segment) appelé adresse segmentée. Un objet, qui peut occuper un segment entier ou une suite de mots consécutifs dans un segment, est désigné par l'adresse segmentée de son premier mot.
- Mémoire physique non contiguë. Le placement des mémoires logiques en mémoire physique peut être contiguë ou pas. Dans ce dernier cas, les pages qui composent la mémoire logique sont disséminées dans différentes pages physiques. C'est une organisation en mémoire paginée ou segmentée et pagine.

4.2.2 Allocation de la mémoire

On distingue différentes manières de réaliser la mise en correspondance entre organisation de la mémoire logique et implantation de cette mémoire logique en mémoire physique :

— Réimplantation dynamique : la correspondance logique/physique est variable dans le temps. De ce fait, l'allocation de la mémoire physique se fait par zones (de taille variable) et/ou par pages (de taille fixe).
— Correspondance fixe (aussi appelée implantation statique). La correspondance est établie une fois pour toutes au moment de la compilation. C'est le cas dans les systèmes à partition unique ou fixes.
— Correspondance dynamique (aussi appelée réimplantation dynamique). La correspondance logique/physique peut aussi être dynamique et ce de deux manières

1. Elle peut être fixée au moment du chargement du processus et donc varier entre deux exécutions. La mémoire est allouée sous forme de zone contiguës appelés des partitions, c'est le système des partitions variables.
2. Elle peut également varier durant l'exécution du processus. Les objets sont donc déplacés à l'intérieur la mémoire centrale. Bien entendu, ces déplacements, opérés par le système, doivent être transparents pour le processus. C'est le cas dans les systèmes paginés ou segmentés qui allouent la mémoire par pages (de taille fixe) ou segments (de taille variable).

L'allocation de mémoire doit permettre à un processus l'accès à un objet défini en mémoire logique, en amenant en temps voulu cet objet en mémoire principale (la seule directement adressable). Une politique d'allocation mémoire doit donc apporter une solution aux deux problèmes suivants :

1. Réealiser la correspondance entre adresses logiques et adresses physiques
2. Réaliser la gestion de la mémoire physique (allocation des emplacements, transfert de l'information).

   Lorsque les informations appartiennent à plusieurs utilisateurs, deux contraintes supplémentaires apparaissent :

   (a) Réaliser le partage d'information entre ces utilisateurs ;
   (b) Assurer la protection mutuelle d'informations appartenant à des usagers distincts

Une politique d'allocation de mémoire idéale aurait pour effet d'assurer qu'à tout instant l'information nécessaire à l'exécution de l'instruction en cours soit immédiatement accessible au processeur, donc se trouve en mémoire principale. Cet objectif n'est en général pas atteint : on cherche alors à réduire la probabilité que l'information soit absente de la mémoire lorsqu'elle est nécessaire (défaut de page). Le problème se résume alors à deux questions :

1. Quand charger un objet en mémoire principale ?
   — Lorsqu'on en a besoin (chargement à la demande),
   — Avant d'en avoir besoin (préchargement).
2. Où charger cet objet ?
   — S'il y a assez de place libre, dans quels emplacements le charger (placement)
   — Sinon, quel(s) objet(s) renvoyer en mémoire secondaire afin de libérer de la place en mémoire principale (remplacement).

Plusieurs critères seront utilisés pour imaginer, évaluer et comparer les algorithmes d'allocation de mémoire :

— Critères liés à l'utilisation de la ressource mémoire, mesurée par exemple par le taux de place perdue (ou inutilisable).
— Critères liés à l'accès à l'information, comme le temps moyen d'accès ou le taux de défauts de page.
— Critères plus globaux caractérisant des performances induites par l'allocation de la mémoire : taux d'utilisation de la CPU, temps de réponse d'un système interactif, etc.

4.3 Partage de la mémoire sans réimplantation

4.3.1 Système à partition unique (va-et-vient simple)

Dans les systèmes à partition unique (aussi appelé va-et-vient simple ou swapping), une zone fixe de mémoire est réservée aux processus des usagers (voir figure 4.2)). Les programmes sont conservés sur disque sous forme absolue. Pour être exécuté, un programme est d'abord amené en mémoire principale, dans sa totalité. L'allocation de processeur aux programmes détermine donc les transferts. En cas de réquisition du processeur, le programme en cours doit être sauvegardé sur disque avant le chargement de son successeur.

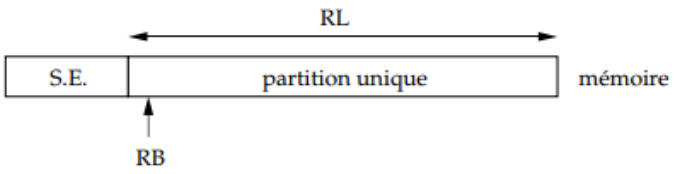


Figure 4.2 – *Système à partition unique*

Afin d'éviter que des erreurs d'adressage du processus utilisateur ne viennent altérer le S.E. résident, la partition unique peut être délimitée par des registres de la CPU (registre de base RB pour le début et registre limite RL pour la taille). A chaque accès à une case d'adresse α, la CPU vérifie que ($RB \leq \alpha < RB + RL$). Si ce test échoue, un déroutement pour erreur d'adressage est généré.

Ce schéma a l'avantage de la simplicité. Son principal inconvénient est de laisser la CPU inutilisée pendant la durée des transferts. Il est employé sur des installations de petite taille lorsque les contraintes de temps de réponse sont compatibles avec la durée et la fréquence des transferts. Des améliorations permettent de réduire le volume d'information transférée et donc la perte de temps pour la CPU :

— Lorsqu'un programme est sauvegardé sur disque, on ne range que la partie modifiée (en pratique, la zone des données)
— L'algorithme de la peau d'oignon permet d'épargner des transferts : lorsqu'un programme est recouvert par un autre de taille plus petite, il suffit pour restaurer le plus gros de recharger la partie recouverte.

Ces améliorations n'apportent néanmoins qu'un gain limité, la taille des transferts n'intervenant que pour une faible part dans le temps requis. Il serait préférable de pouvoir exécuter un programme pendant la durée de transfert d'un autre. Pour ce faire, il faut donc conserver simultanément en mémoire plusieurs programmes, en partie ou en totalité. Ce mode de partage est appelé multi-programmation. Une multi-programmation sans réimplantation dynamique est possible par la partition de la mémoire.

4.3.2 Partition fixe de la mémoire

Dans un système à partitions fixes, la mémoire est partagée de façon statique en un nombre fixe de partitions, les tailles et limites de ces partitions étant définies lors de la génération du système. Chaque programme est affecté de façon fixe à une partition au moment de la construction de son image mémoire par l'étape d'édition de liens. Les programmes (sous leur forme exécutable) sont conservés sur disque sous forme absolue, et les adresses qui y figurent sont les adresses physiques correspondant à l'implantation de chacun d'eux dans la partition qui lui a été attribuée.
Pendant qu'un programme est transféré (en entrée ou sortie), un autre programme peut être exécuté dans une autre partition ; il faut bien entendu disposer d'un processeur d'entrée/sortie autonome (canal ou ADM). La figure 4.3 schématise l'implantation des programmes et le chronogramme d'activité dans un système à partitions fixes.

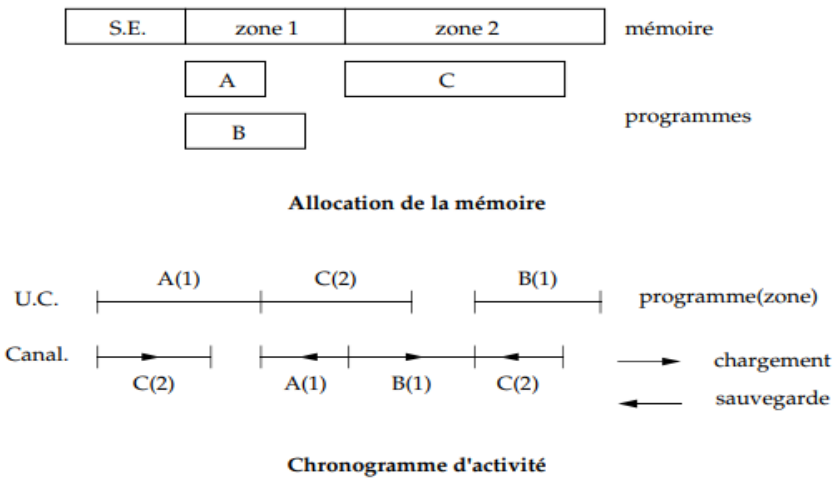


Figure 4.3 – *Système à partition fixe*

En réalité, le chronogramme peut être plus complexe, chaque programme pouvant lui-même exécuter des entrées-sorties. Dans ce cas, le processeur est également affecté à un autre programme.
Les systèmes à partitions fixes sont couramment utilisés sur des petites et moyennes installations où un petit nombre d'usagers interactifs coexistent avec un travail de fond. Il est alors possible de définir au moment de la génération du système, des tailles de partitions adaptées aux différentes classes de programmes. Le temps de réponse moyen des processus interactifs dépend

du rapport des temps d'exécution aux temps de transferts, lui-même fonction du degré de multiplexage des partitions.

4.4 Systèmes à partitions variables

Dans un système à partitions variables, le découpage en partitions n'est pas fixé une fois pour toutes, mais il est redéfini à chaque début d'exécution d'un processus. En conséquence, le chargement d'un programme (fixation des adresses) ne peut être fait qu'au dernier moment, lorsqu'une place lui est attribuée.
L'allocation de la mémoire par partitions de tailles variables suppose l'existence d'un mécanisme de réimplantation dynamique. L'utilité de celui-ci apparaîtra dans la désignation d'objets appartenant à des partitions qui auront du être déplacées en mémoire centrale.

4.4.1 Réimplantation dynamique par registre de base

Le principe que nous allons décrire est simple. Disposant d'un registre particulier ou registre de base, son contenu est systématiquement ajouté à toute adresse engendrée par un processus, le résultat constituant une adresse physique de l'information désignée. Si les adresses d'un programme sont relatives à son début (i.e. si le programme est implanté à l'adresse logique 0), il suffit que le registre de base soit affecté à son adresse d'implantation en mémoire physique (voir figure 4.4).
Dans ces conditions, le programme pourra être chargé en n'importe quel endroit de la mémoire. En particulier, déplacer globalement un programme dont l'exécution est commencée, peut s'opérer très facilement, à condition de modifier en conséquence la valeur contenue dans le registre de base. De plus, si programme et données sont atteints par l'intermédiaire de registres distincts, leur déplacement pourra être effectué indépendamment.

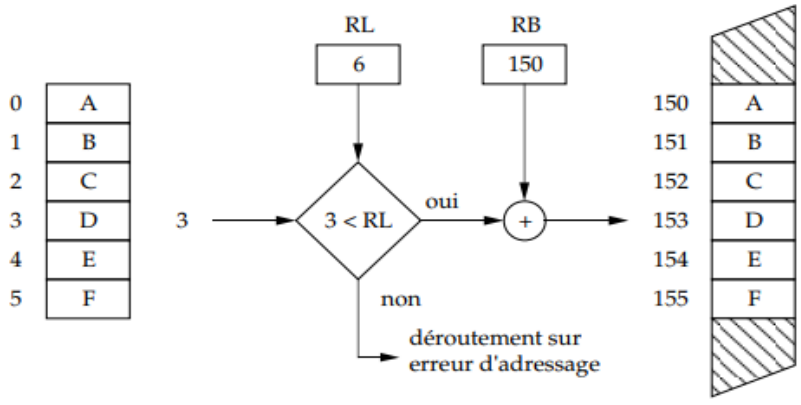


Figure 4.4 – *Passage logique/physique par registre de base et registre limite*

4.4.2 Algorithme de Gestion de la mémoire par zone

Disposant d'une file constituée par les programmes en attente de traitement, un choix doit être opéré afin de déterminer leur ordre de lancement. Cet ordre pourra être tout simplement celui de la file d'attente ou dicté par des contraintes de priorités calculées par le système en fonction des demandes de ressources (place mémoire, nombre de périphériques, etc.) ou du temps d'exécution présumé. En tout état de cause, cet ordre sera aussi fonction de la taille des différentes partitions libres. Il faut auparavant résoudre les problèmes suivants :

— Choix d'une représentation des partitions
— Définition des critères de sélection d'une partition libre
— Politique de libération d'une partition occupée
— Décision à prendre lorsqu'aucune partition ne convient

Représentation des partitions

Une partition est définie par sa taille et son adresse de début, contenues dans un descripteur. En supposant que les tailles demandées sont variables, le nombre de partitions le sera aussi. En conséquence, il est préférable, plutôt que de regrouper les descripteurs dans une table, de les situer dans les partitions elles-mêmes et de les chaîner entre eux.
L'ordre du chaînage a une influence sur l'efficacité des algorithmes. On peut choisir l'ordre de libération des partitions, mais le plus souvent, on utilise l'un des deux classements suivants :

1. Classement par adresses croissantes ou décroissantes
2. Classement par tailles croissantes ou décroissantes

Algorithmes de sélection

Une demande étant émise, on connaît la taille requise pour charger le programme du processus demandeur. Le plus souvent, cette demande sera satisfaite grâce à une partition de taille supérieure ; la différence, ou résidu est rattachée à la liste des partitions libres, pour autant que cette différence ne soit pas trop petite. Deux possibilités peuvent être envisagées quant au choix de la partition libre pour satisfaire une demande :

— Prendre la première possible, c'est à dire, parcourir la liste jusqu'à ce que l'on en trouve une dont la taille est supérieure ou égale à la demande (first-fit) ;
— Prendre la partition la plus petite possible, celle donnant le plus petit résidu (best-fit).

L'allocation d'une partition à un processus peut se décomposer en deux phases, recherche de la partition selon l'algorithme choisi puis placement du résidu dans la liste. Le classement par tailles croissantes évite de parcourir toute la liste pour trouver la plus petite partition possible (permettant ainsi une implémentation aisée du best-fit). Par contre le placement du résidu impose une modification du chaînage.

A l'opposé, le classement par adresses croissantes autorise une gestion rapide des résidus (seule la taille doit être modifiée, le chaînage demeurant inchangé) pour peu que le chargement s'opère en bas de partition. Cette technique est mieux adaptée à l'algorithme du first-fit.
On peut constater que certaines tailles sont demandées plus fréquemment que les autres. Dans ces conditions, on améliore l'efficacité de l'allocation en réservant un certain nombre de partitions possédant ces tailles privilégiées. En cas d'épuisement de cette réserve, le mécanisme classique est utilisé.

Libération d'une partition

Trois cas sont à considérer lors de la libération d'une partition :
— La partition libérée est entourée de deux partitions libres
— La partition libérée est entourée d'une partition libre et d'une partition occupée
— La partition libérée est entourée de deux partitions allouées.
Chaque fois que cela est possible (deux premiers cas), il est utile de regrouper les partitions libres contiguües afin de réduire la fragmentation de la mémoire. Il est évident que le classement par adresses croissantes est alors le plus efficace.

4.4.3 Fragmentation et compactage

Le phénomène le plus gênant dans le type d'allocation étudié ici est celui de la fragmentation de la mémoire, qui apparaît au bout d'un certain temps de fonctionnement et qui est dû à la multiplication des résidus de petite taille On peut aboutir à une situation où aucune partition de taille suffisante n'est disponible pour satisfaire une demande, alors que la somme des tailles de partitions libre est largement supérieure. Une solution consiste à compacter les partitions allouées en les déplaçant vers une extrémité de la mémoire, laissant apparaître ainsi à l'autre extrémité une partition libre de taille égale à la somme des tailles des partitions libres primitives.
Le compactage peut s'effectuer de deux façons possibles :
— Par recopie à l'intérieur de la mémoire physique en utilisant une instruction de type MOVE. (voir ci-dessous), opération monopolisant le processeur central,

```
MOVE <adresse d´epart>, <adresse arrivée>, <longueur>
```

— Par recopies successives des partitions sur disque puis du disque en mémoire, à la place voulue, en utilisant un processeur d'entrée-sortie. L'opération est alors plus longue, mais a le mérite de libérer le processeur central pour poursuivre l'exécution des autres programmes
La figure 4.5 donne un exemple simple de compactage. Il montre les différentes stratégies de recopie des partitions occupées de manière à limiter la quantité de données à déplacer. On passe dans cet exemple de 200 ko à 50 ko.
Il est bien certain que seule une possibilité de réimplantation dynamique permet d'opérer un tel compactage. D'autre part, Knuth a montré que lorsque l'algorithme d'allocation ne peut satisfaire une demande, cela intervient alors

que le taux de remplissage de la mémoire est tel qu'après compactage, la même situation va à nouveau apparaître très rapidement, obligeant le système à consacrer une grande partie de son temps à effectuer des compactages successifs.
En conclusion, une telle forme d'allocation n'est guère adaptée à un système interactif, mais convient mieux lorsque le nombre de partitions allouées est faible, et leur temps d'allocation grand (traitement par trains de travaux).

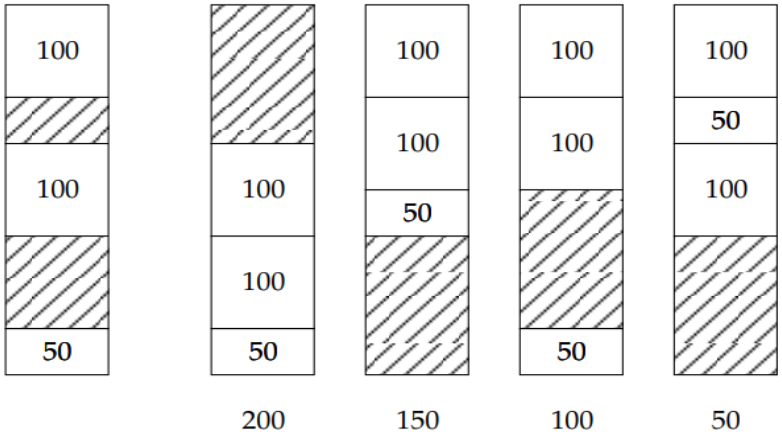


Figure 4.5 – *Différentes possibilités de compactage de la mémoire*

4.5 Mémoire paginée

Une mémoire paginée est divisée en blocs de taille fixe, ou pages logiques, qui servent d'unités d'allocation. La mémoire physique est elle-même divisée en blocs de même taille appelé pages physiques. Nous présentons successivement les mécanismes de pagination d'une mémoire contiguë paginée et d'une mémoire paginée segmentée.

4.5.1 Pagination d'une mémoire contiguë

La figure 4.6 représente le schéma général d'une mémoire contiguë paginée. Le rôle de la boîte marquée "Fonction de pagination" est d'établir une correspondance entre les adresses de pages logiques et les adresses de pages physiques de manière à se qu'une page logique puisse être rangée dans une page physique quelconque. Les pages physiques deviennent ainsi des ressources banalisées dont la gestion est plus simple que celle de partitions de taille variable.
Le nombre d'emplacements d'une page (physique ou logique) est toujours une puissance de 2. Notons 2^l la taille (nombre d'emplacements) d'une page (logique ou physique) et 2^n le nombre de pages. Il y a pour l'instant autant de pages logiques que de pages physiques.
Une adresse logique paginée est alors construite par concaténation d'un numéro de page logique (n bits) et d'un déplacement dans la page (l bits). De

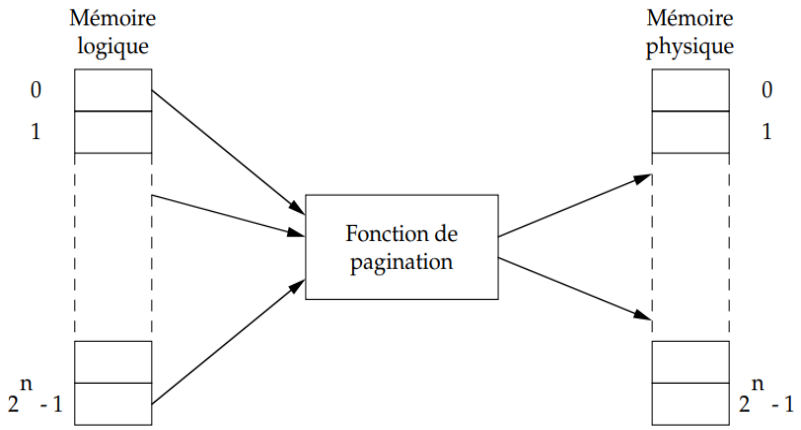


Figure 4.6 – *Mémoire linéaire paginée*

même, une adresse physique est la concaténation d'un numéro de page physique (n bits) et d'un déplacement (l bits). Les tailles de page usuelles vont de 0,5ko à 32ko.
Étant donne un numéro de page logique (npl), la fonction de pagination permet de trouver le numéro de la page physique (npp) qui la contient. Dans un souci d'efficacité, cette fonction est réalisée par un mécanisme matriel.
La réalisation la plus courante de la fonction de pagination utilise une table de pages en mémoire, indexée par un numéro de page logique (table desc de la figure 4.7). Lors d'un accès à la mémoire, la correspondance adresse logique/adresse physique (qui est une opération matérielle), est mise en œuvre comme suit :

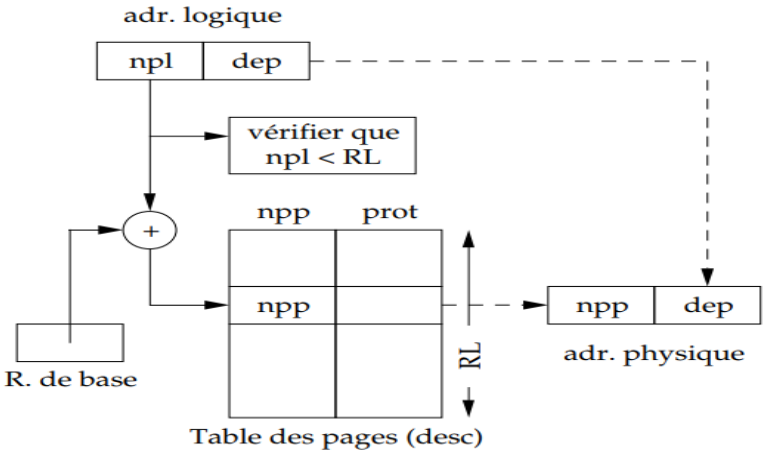


Figure 4.7 – *Organisation d'une table de pages*

```
(npl, d´eplacement) = <adresse logique>
si (npl < RL) alors
        si <les protections sont respectées> alors
                <adr. physique> = (desc[npl].npp, déplacement)
        sinon
                <déroutement sur violation de protection
```

```
        fin si
sinon
        <déroutement sur erreur d'adressage>
fin si
```

Le champ desc[npl].prot indique le mode d'accès autorisé à la page logique npl. Cette information est utilisée par les mécanismes de protection et un accès non autorisé provoque un déroutement pour violation de protection.
Notons qu'une table de pages représente le contenu d'une mémoire logique particulière. Si le système d'exploitation permet à chaque processus, ou à chaque usager du système, de définir une mémoire logique distincte, il doit gérer une table de pages distincte par processus ou par usager.
Le pointeur vers l'origine de cette table (RB) fait alors partie du contexte du processus ou de l'usager. Les tables des pages se trouvent en mémoire physique, dans la partition réservée au système d'exploitation.
La mémoire logique d'un processus n'est plus représentée d'une manière contiguë en mémoire centrale (voir figure 4.8). En effet, l'indirection des accès par la table de pages permet de loger les pages logiques dans n'importe quelle page physique. De ce fait, la gestion de la mémoire physique revient simplement à gérer une liste des pages physiques libres sans idée de regroupement. Les problèmes liés à la fragmentation externe disparaissent mais la fragmentation interne se fait plus présente puisque la page devient l'unité élémentaire d'allocation et de libération (en pratique plusieurs kilo-octets).

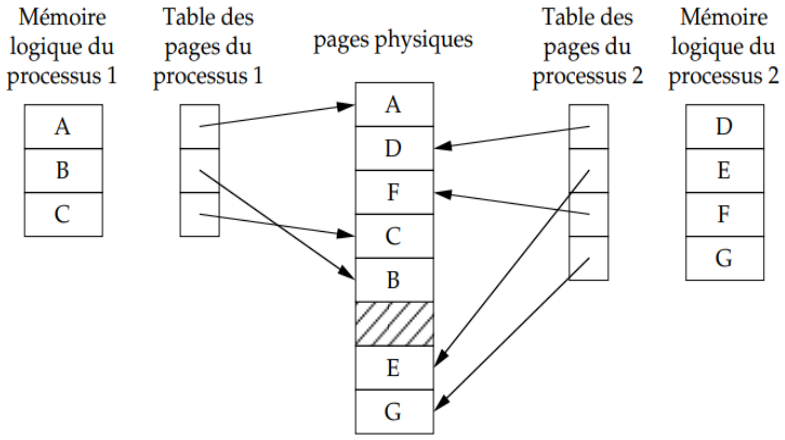


Figure 4.8 – *Un exemple sur deux mémoires logiques paginées*

L'accès à une page logique nécessite maintenant deux références à la mémoire en raison de la consultation de la table des pages. Cette augmentation du temps d'accès moyen est bien sur intolérable. La réduction de ce coût passe par deux points :
— Observer le comportement des processus (vis à vis de la mémoire)
— Optimiser la transformation des adresses au moyen d'un circuit particulier : les mémoire associatives.
Pour éviter l'accès à cette table (et donc réduire le temps d'accès moyen), on passe par un circuit particulier : une mémoire associative. Mais avant de présenter cette mémoire il faut discuter de l'utilisation de la mémoire par les

processus.

4.5.2 Comportement des processus en mémoire paginée

Le comportement d'un processus dans son espace logique détermine ses demandes de mémoire physique. Il est donc utile de connaître les caractéristiques de ce comportement pour améliorer l'efficacité des algorithmes de gestion dynamique de la mémoire. Donnons d'abord quelques d´efinitions :

— – L'écoulement du temps est repéré par l'exécution des instructions successives : l'exécution d'une instruction définit une unité de temps. Ce temps est dit virtuel car il suppose que le programme dispose de toutes les ressources nécessaires (mémoire et processeur). En cas de partage de ressources, on peut ainsi raisonner sur un programme donné en faisant abstraction des autres

— La mémoire logique paginée est découpée en page contiguës de taille fixe. L'accès à un emplacement d'une page est appelé référence à cette page. Le numérotage des pages permet d'étiqueter les références

— Le comportement du processus est défini par la série des numéros de pages référencées au cours de l'exécution. Cette séquence s'appelle chaîne de référence pour le processus considéré

— L'exécution d'une instruction peut donner lieu à plusieurs références distinctes : pages contenant l'instruction, le ou les opérandes

L'expérience montre que les chaînes de références des processus possèdent des caractéristiques communes que nous définirons d'abord de manière qualitative.

— Non-uniformité. Soit n_i le nombre total de références à une page pi. La r´epartition des ni n'est pas uniforme : un faible pourcentage des pages cumule généralement un taux très important du nombre total des références. Il est courant de constater que plus des 75% des références intéressent moins de 20% des pages

— Propriété de Localité. Sur un temps d'observation assez court, la répartition des références présente une certaine stabilité : les références observées dans un passé récent sont en général une bonne estimation des prochaines références.

A partir de cette constatation de localité, on peut créer un modèle de comportement des programmes.

Dans ce modèle, le déroulement d'un programme est défini comme une succession de phases séparées par des transitions. Une phase i est caractérisée par un ensemble de pages S_i et un intervalle de temps virtuel T_i. Lorsque le programme entre en phase i, il y reste un temps T_i en effectuant principalement des références à des pages de S_i. Ensuite, il subit une transition durant laquelle les références aux pages sont dispersées, avant d'entrer dans la phase $i + 1$.

Les phases constituent donc des périodes de comportement stable et relativement prévisible, alors que les transitions correspondent à un comportement plus erratique. L'expérience montre que les périodes de transition ne repré-

sentent qu'une faible partie du temps virtuel total, la majeure partie du temps virtuel étant occupé par des phases de longues durée (quelques centaines de milliers d'instructions).
Qualitativement, ce type de comportement s'explique par le fait que les programmes sont souvent organisés en procédures possédant chacune un contexte spécifique, que les accès aux données sont souvent concentrés (parcours de tableau), que les programmes comportent des boucles concentrant aussi les références.
La notion d'ensemble de travail ("working set") est également utilisée pour caractériser le comportement des programmes et prévoir d'après l'observation. Soit $W(t,T)$ l'ensemble des pages ayant été référencées entre les temps $t-T$ et t. D'après la propriété de localité, ces pages ont une probabilité plus élevée que les autres de faire l'objet d'une référence au temps t à condition toutefois que la taille de la fenêtre d'observation T soit convenablement choisie. En admettant un comportement suivant le modèle phase/transition, T devra être inférieur à T_i en phase i.

4.5.3 Mémoire associative et pagination

Une mémoire associative est un ensemble de couple <entrée, sortie>. La présence d'une valeur sur le bus d'entrée provoque soit l'apparition d'une valeur de sortie soit un signal d'échec indiquant que cette entrée n'existe pas dans la mémoire associative (figure 4.9). Ces mémoires ont quelques dizaines à quelques centaines d'entrées et leur coût très élevé a empêché leur extension à des mémoires de plus grande taille.

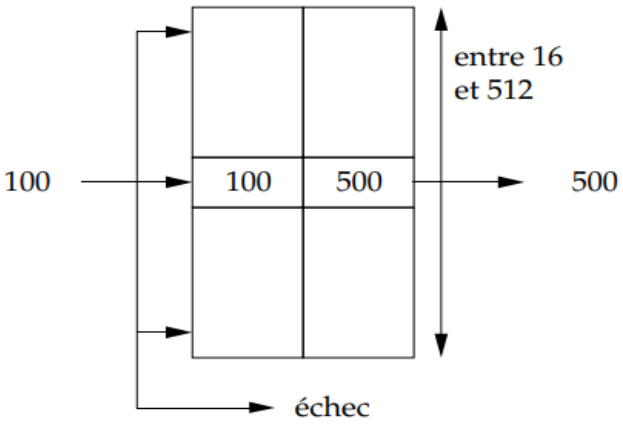


Figure 4.9 – *Schéma d'une mémoire associative*

Dans cette mémoire associative on conserve les couples <npl, npp> relevés lors des accès les plus récents. En raison de la propriété de localité des programmes, on a une probabilité élevée (80% à 95% avec les tailles usuelles) de trouver dans la mémoire associative le numéro de la page logique adressée et donc de déterminer sa page physique. Ce n'est qu'en cas d'échec que l'on passe par la table des pages ; la mémoire associative est alors mise à jour, le

couple <npl, npp> courant remplaçant le plus anciennement utilisé (figure 4.10).

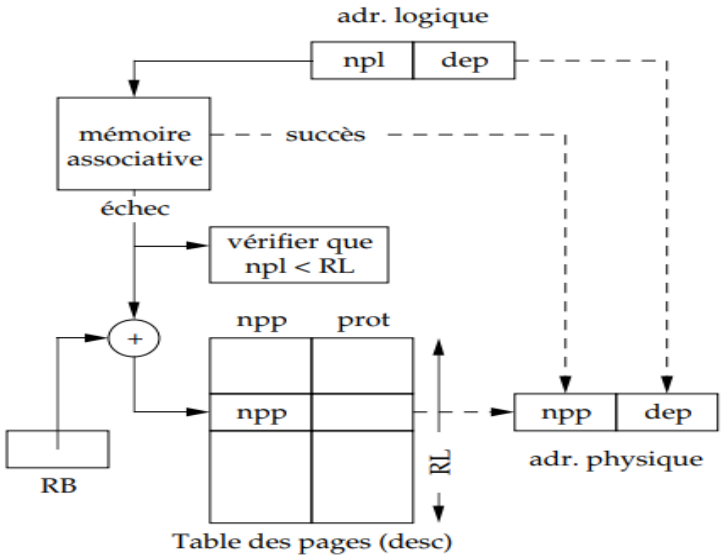


Figure 4.10 – *Pagination avec mémoire associative*

En partant du principe que l'accès à la mémoire physique prend 100 ns et que le temps de recherche de la mémoire associative est de 20 ns, le temps moyen d'accès est compris entre

$$0,8 \times (100 + 20) + 0,2 \times (100 + 20 + 100) = 140ns$$

$$0,95 \times (100 + 20) + 0,05 \times (100 + 20 + 100) = 125ns$$

suivant la probabilité de réussite et donc la taille de la mémoire associative. Finalement, le temps d'accès moyen n'a augmenté que de 25%, mais la gestion de la mémoire est beaucoup plus souple et les problèmes de fragmentation externe n'existent plus.

4.5.4 Partage et protection de l'information

L'utilisation d'informations partagées entre plusieurs mémoires logiques soulève trois problèmes :

— La désignation : comment adresser de manière uniforme les informations partagées

— Le partage physique : comment assurer que les informations partagées existent en exemplaire unique

— La protection : comment garantir le respect des règles d'accès (éventuellement sélectives) aux informations partagées.

Dans un système paginé, l'unité élémentaire de partage est la page. Pour être partagés, les informations doivent se trouver sur une (ou plusieurs) page logique partagée. Cette page peut être chargée dans une page physique quelconque ; les tables de pages des mémoires logiques où figure cette page logique contiennent alors, à l'entrée correspondante, le même numéro de page

physique. Dans l'exemple présenté par la figure 4.11 les pages contenant le programme (Pa et Pb) sont partagées mais les pages de données (D1, ..., D7) ne le sont pas.

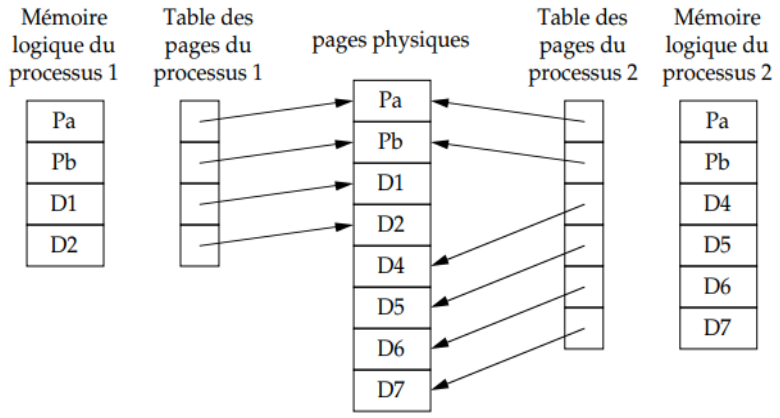


Figure 4.11 – *Partage de pages entre mémoires logiques paginées*

Si l'unité de partage est la page, une page physique partagée peut recevoir des droits d'accès distincts dans chaque mémoire logique où elle figure. Ces droits sont spécifiés à l'entrée correspondante de la table de pages.

4.6 Mémoire segmentée

4.6.1 Principe de la segmentation

Dans les systèmes de mémoire paginée, la division en pages est arbitraire et ne tient pas compte du mode d'organisation des données d'un processus. Notamment, il est fort probable que certaines structures de données vont se trouver à cheval sur plusieurs pages ce qui peut être la cause de temps d'accès plus importants.
L'objectif des mémoires segmentées est de mettre en rapport la structure logique de la mémoire (vue des processus) et l'implantation physique de cette mémoire. Pour ce faire, la mémoire logique segmentée d'un processus est définie comme un ensemble de segments numérotés à partir de zéro (figure 4.12). Un segment est une zone contiguë de taille variable
Une adresse logique dans un système segmenté (aussi appelée adresse segmentée) est un couple <no de segment, déplacement>
Comme dans les mémoires paginées, le S.E. maintient une table des segments pour chaque processus (figure 4.13). La correspondance proprement dite est établie par le matériel de la manière suivante :

```
<seg,dépl> := <adresse logique segmentée>
si (seg < RL) et (d´epl < desc[seg].taille) alors
        si <les protections sont respectées> alors
```

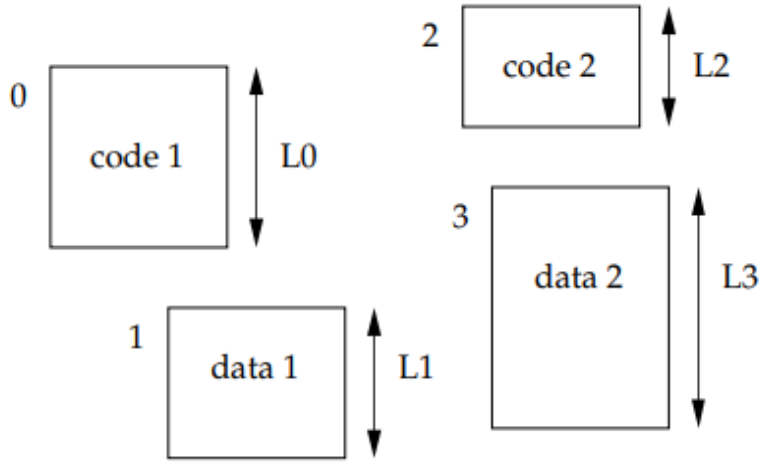


Figure 4.12 – *Une mémoire segmentée*

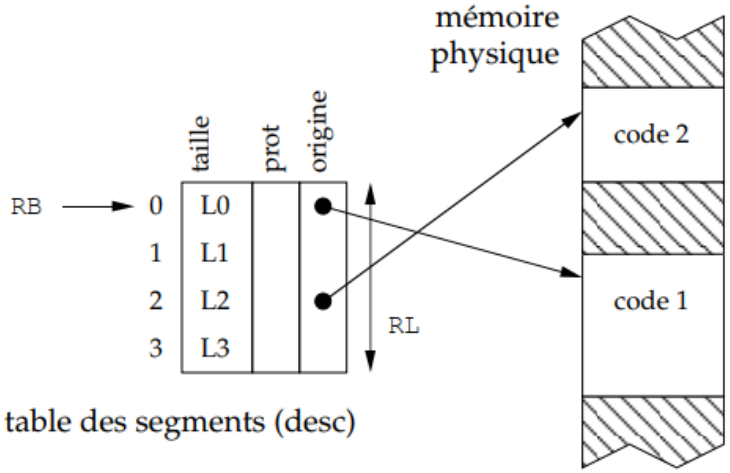


Figure 4.13 – *Table des segments d'un processus*

```
                <adresse physique> = desc[seg].origine + dépl
        sinon
                <déroutement sur violation de protection>
        fin si
sinon
        <déroutement sur erreur d'adressage>
fin si
```

Ce mécanisme doit être couplé à une mémoire associative pour améliorer le temps d'accès moyen en évitant l'utilisation de la table des segments.
Les avantages de cette organisation sont doubles : d'une part, la notion de segment est directement utilisable dans un processus et de ce fait on peut espérer une réduction des temps d'accès moyens (les accès fréquents étant regroupés sur un petit groupe de segments, voire même sur un segment unique) ; d'autre part, la notion de protection est plus facilement utilisable puisqu'elle

porte directement sur des segments, c'est à dire des objets logiques.
Les segments sont des zones contiguës (du moins pour l'instant). On retrouve donc les problèmes d'allocation/libération de zones et l'apparition d'une fragmentation externe éventuellement corrigée par des compactages de la mémoire. Dans le cas de la mémoire segmentée, ces compactages impliquent une remise à jour des pointeurs origine des tables de segments.

4.6.2 Pagination d'une mémoire segmentée

La pagination d'une mémoire segmentée vise à rendre plus souple l'allocation de mémoire aux segments en levant la restriction de contiguïté pour le placement d'un segment. Une entrée de la table des segments contient, outre les informations propres au segment (taille, protection, type), un pointeur vers la table des pages de ce segment. Comme dans les techniques précédentes, une mémoire associative conserve les dernières références.

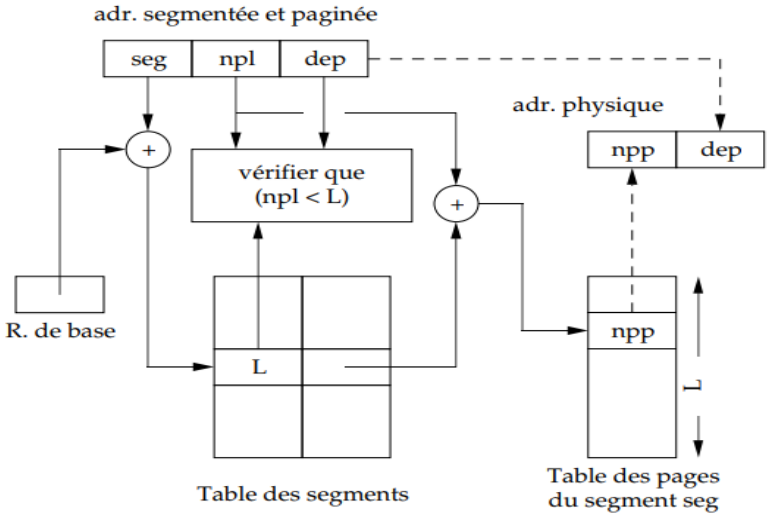


Figure 4.14 – *Pagination d'une mémoire segmentée*

Exemple : Le système Multics utilise une allocation par pages pour sa mémoire segmentée. En raison du nombre élevé de segments, les tables de pages des segments et les tables de segments elles-mêmes peuvent dépasser la taille d'une page et être elles-mêmes paginées. La table des pages d'un segment est maintenue en mémoire principale tant que ce segment est actif (c'est à dire que le fichier correspondant est ouvert pour au moins un processus)

4.6.3 Partage de segments

Dans une mémoire logique segmentée, le partage s'applique aux segments et les tables de pages des segments partagés sont elles-mêmes partagées (dans le cas d'un système segmenté paginé). Tous les descripteurs d'un segment contiennent alors, non pas l'adresse de ce segment, mais celle de sa table de pages qui est unique.

Si l'unité de partage est le segment, la protection sélective s'applique globalement au segment. Les droits d'accès à un segment pour un processus figurent dans la table des segments de ce processus. Si des droits d'accès individuels aux pages du segment sont spécifiés, ils figurent dans la table de pages partagée par les processus utilisateurs et sont donc les mêmes pour tous. Ils doivent alors être compatibles avec les droits globaux associés au segment.

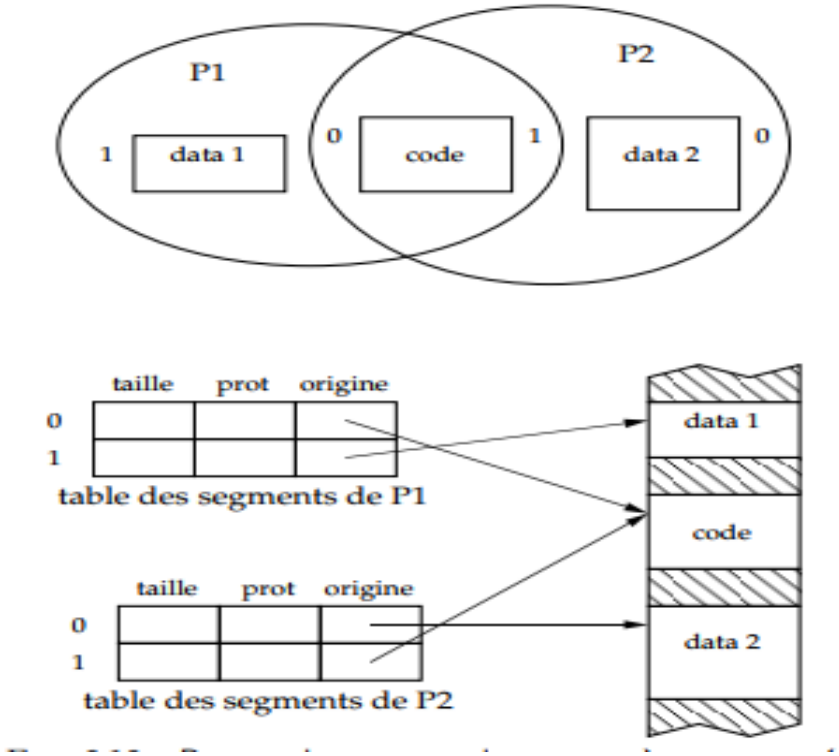


Figure 4.15 – *Partage de segments dans un système segmenté*

4.7 Travaux pratiques

4.8 Conclusion

Chapitre 5

Le système de gestion des fichiers

5.1 Objectifs

Ce chapitre a pour but de présenter le système de gestion de fichiers tel que perçu par le système d'exploitation. A la fin du chapitre, l'étudiant devra pouvoir :

1. Connaitre le concept de système de gestion de fichiers et donner son rôle
2. Présenter le rôle du formatage et du partitionnement d'un disque
3. Présenter les différents type de partitionnement d'un disque
4. Établir la différence entre un fichier et un répertoire
5. Établir la différence entre un chemin d'accès absolu et un chemin d'accès relatif

La charge horaire destinée à ce chapitre est départie comme suit :

1. Cours magistral : 3h
2. Travaux dirigés : 2h
3. Travaux pratiques : 4h
4. Travail personnel de l'étudiant : 1h

5.2 Introduction

Le système de gestion de fichiers (SGF) est la partie la plus visible d'un système d'exploitation qui se charge de gérer le stockage et la manipulation de fichiers (sur une unité de stockage : partition, disque, CD, disquette). Un SGF a pour principal rôle de gérer les fichiers et d'offrir les primitives pour manipuler ces fichiers.

5.3 Formatage et partition

5.3.1 Le partitionnement

Le partitionnement consiste à «cloisonner» le disque. Il permet la cohabitation de plusieurs systèmes d'exploitation sur le même disque (il permet

d'isoler certaines parties du système). L'information sur le partitionnement d'un disque est stockée dans son premier secteur (secteur zéro), le **MBR** (Master Boot Record). Deux types de partitionnement :

1. Primaire : On peut créer jusqu'à 4 partitions primaires sur un même disque
2. Etendue : est un moyen de diviser une partition primaire en sous-partitions (une ou plusieurs partitions logiques qui se comportent comme les partitions primaires, mais sont créées différemment (pas de secteurs de démarrage))

Dans un même disque, on peut avoir un ensemble de partitions (multi-partition), contenant chacune un système de fichier (par exemple DOS et UNIX)

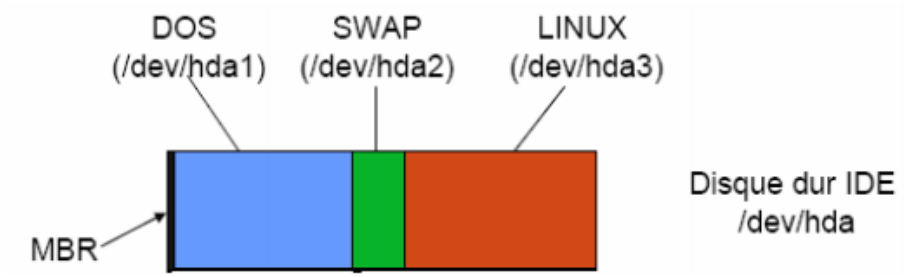


Figure 5.1 – *Multipartition d'un disque*

5.3.2 Le formatage

Avant qu'un système de fichiers puisse créer et gérer des fichiers sur une unité de stockage, son unité doit être formatée selon les spécificités du système de fichiers. Le formatage inspecte les secteurs, efface les données et crée le répertoire racine du système de fichiers. Il crée également un superbloc pour stocker les informations nécessaires à assurer l'intégrité du système de fichiers.

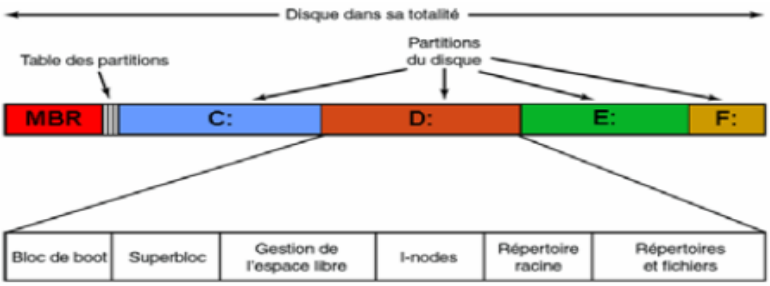


Figure 5.2 – *Organisation du système de fichiers*

Un superbloc contient notamment : L'identifiant du système de fichiers (C :, D : ..), Le nombre de blocs dans le système de fichiers, La liste des blocs

libres, l'emplacement du répertoire racine, la date et l'heure de la dernière modification du système de fichiers, une information indiquant s'il faut tester l'intégrité du système de fichiers.

5.4 Les notions de fichier et de répertoire

5.4.1 Les fichiers

Un fichier est l'unité de stockage logique mise à la disposition des utilisateurs pour l'enregistrement de leurs données : c'est l'unité d'allocation. Le SE 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 les systèmes ou certains outils tels que 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 de ces attributs sont indiqués par l'utilisateur, d'autres sont complétés par le système d'exploitation.

5.4.2 Les répertoires

Un répertoire est une entité crée pour l'organisation des fichiers. En effet on peut enregistrer des milliers, voir des millions de fichiers sur un disque dur et il devient alors impossible de s'y retrouver. Avec la multitude de 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. Du point de vue SGF, un répertoire est un fichier qui dispose d'une structure logique : il est considéré comme un tableau qui contient une entrée par fichier. L'entrée du répertoire permet d'associer au nom du fichier (nom externe au SGF) les informations stockées en interne par le SGF. Chaque entrée peut contenir des informations sur le fichier (attributs du fichier) ou faire référence à (pointer sur) des structures qui contiennent ces informations. La figure 5.3 présente la structure d'un répertoire pour le système MS-DOS, tandis que la figure 5.4 présente celle d'un système UNIX.
Dans le cas de UNIX, chaque fichier à un i-noeud
On distingue plusieurs structures pour les répertoires :

— La structure plate à un niveau : organisée en plusieurs répertoires mais chacun d'eux ne peut contenir que des fichiers. Aujourd'hui absurde, cette approche existait à l'époque des premiers systèmes d'exploitation car le nombre de fichiers était limité

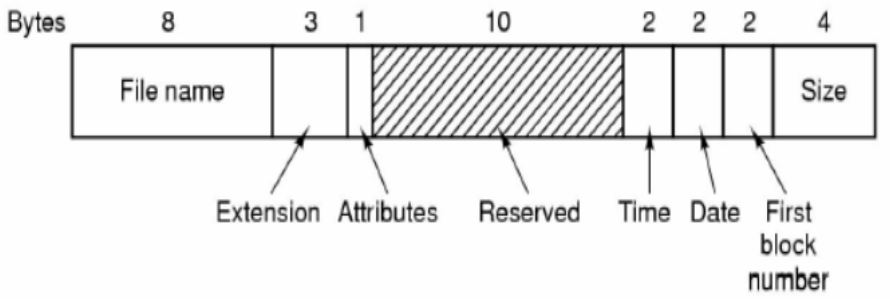


Figure 5.3 – *Structure d'un répertoire : cas de MS-DOS (32 Octets)*

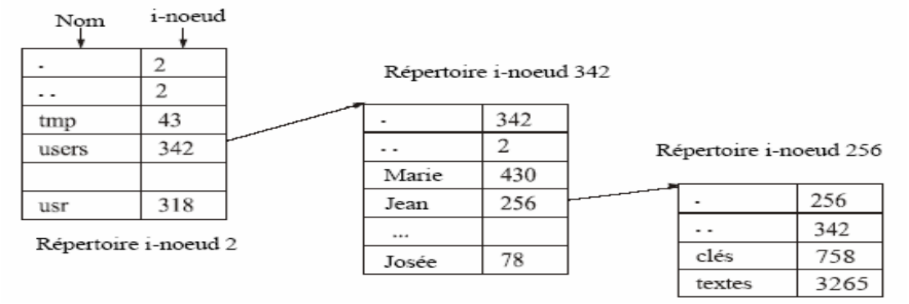


Figure 5.4 – *Structure d'un répertoire : cas d'UNIX (14 Octets)*

— La structure à deux niveaux : chaque utilisateur dispose de son propre répertoire dans lequel il peut conserver des fichiers et des répertoires
— La structure arborescente : contient un nombre arbitraire de niveaux et chaque répertoire peut contenir des fichiers et des sous répertoires.

Le nom complet d'un fichier est formé d'une liste des répertoires qu'il faut traverser à partir du haut de la hiérarchie (le répertoire racine (root directory)) plus le nom_du_fichier. Les répertoires sont séparés par un caractère qui dépend du système d'exploitation :
`">" pour Multics, "/" pour UNIX, "\" pour Dos et Winxx et ": " pour MacOS.`
Un tel chemin (exprimé à partir de la racine) est appelé chemin absolu. Voici un exemple de chemin absolu sous MS-DOS
`c:\cours\chapitre4.txt et sous Unix /home/user1/rapport.txt`
Par contre, un chemin qui ne commence pas par la racine est un chemin relatif.
Ces deux concepts de fichier et de répertoire sont considérés par le système d'exploitation comme une seule entité différentiable par un bit à rajouter aux attributs.
Sous Unix, le répertoire racine (le répertoire /) contient les sous répertoires de la figure 5.5

5.5 Rôle d'un système de gestion de fichiers

Un SGF a pour principal rôle de gérer les fichiers et d'offrir les primitives pour manipuler ces fichiers. Il effectue généralement les tâches suivantes :
— Fournit une interface conviviale pour manipuler les fichiers (vue fournie à

/bin	commandes binaires utilisateur essentielles (pour tous les utilisateurs)
/boot	fichiers statiques du chargeur de lancement
/dev	fichiers de périphériques
/etc	configuration système spécifique à la machine
/home	répertoires personnels des utilisateurs
/lib	bibliothèques partagées essentielles et modules du noyau
/mnt	point de montage pour les systèmes de fichiers montés temporairement
/proc	système de fichiers virtuel d'information du noyau et des processus
/root	répertoire personnel de root (optionnel)
/sbin	binaires système (binaires auparavant mis dans /etc)
/sys	*état des périphériques (model device) et sous-systèmes (subsystems)*
/tmp	fichiers temporaires

Figure 5.5 – *Les différents sous-répertoires du répertoire recine sous UNIX*

l'utilisateur). Il s'agit de simplifier la gestion des fichiers pour l'utilisateur (généralement, l'utilisateur fournis seulement les attributs nom et extension du fichier, les autres attributs sont gérés implicitement par le SGF). Cette interface fournit la possibilité d'effectuer plusieurs opérations sur les fichiers. Ces opérations permettent généralement d'ouvrir, de fermer, de copier, de renommer des fichiers et des répertoires.

— La gestion de l'organisation des fichiers sur le disque (allocation de l'espace disque aux fichiers)

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

— La gestion des fichiers dans un environnement Multi-Utilisateurs, la donnée d'utilitaires pour le diagnostic, la récupération en cas d'erreurs, l'organisation des fichiers

5.5.1 La gestion de l'organisation de l'espace disque

Sur le disque, un fichier est sauvegardé sur un ensemble de clusters, appelés également blocs. Le SGF manipule alors des blocs numérotés de 0 à N-1 (N = taille du disque/taille d'un bloc). Chaque fichier (ordinaire ou répertoire) d'un système de fichiers est stocké sur l'unité de stockage du système de fichiers. Ses données sont dans des blocs de taille fixe (512, 1024, ou 2048 octets, ...) et à chaque fichier est alloué un nombre de blocs.
La lecture ou l'écriture d'un élément d'un fichier impliquera le transfert vers la mémoire du bloc entier qui contient cet élément.

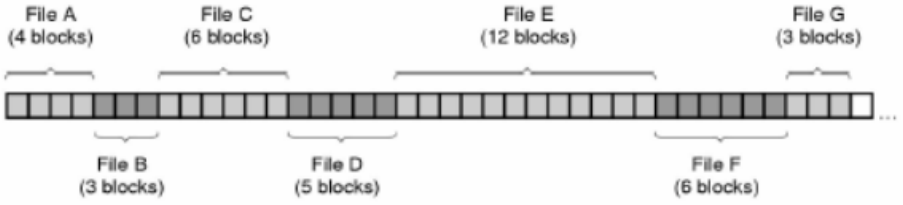
5.6 Les techniques d'allocation des blocs sur le disque

On distingue trois manières d'organiser les blocs d'un fichier : contiguë, chaînée et indexée.

5.6.1 Allocation contiguë

Pour chaque fichier à enregistrer, le système recherche une zone suffisamment grande pour accueillir le fichier. Le fichier sera alors constitué de plusieurs blocs contigus. Cette méthode présente l'avantage de la rapidité de l'accès (les blocs étant contigus, on limite les déplacements de la tête le lecture/écriture, coûteux en temps). Cependant, elle présente un grand nombre d'inconvénients :

— Le dernier bloc a toutes chances d'être sous-utilisé et ainsi, on gaspille de la place. Le pourcentage de place perdue est d'autant plus grand que la taille moyenne des fichiers est faible, ce qui est la réalité
— Il est difficile de prévoir la taille qu'il faut réserver au fichier : un fichier est amené à augmenter de taille, par conséquent il faut prévoir de l'espace libre après le dernier secteur alloué. Si le fichier est agrandi, il faudra le déplacer pour trouver un nouvel ensemble de blocs consécutifs de taille suffisante
— La perte d'espace sur le disque : si on prévoit trop d'espace libre, le fichier risque de ne pas l'utiliser en entier. En revanche, si on prévoit trop peu d'espace libre, le fichier risque de ne pas pouvoir être étendu
— Problème de fragmentation externe : c'est l'espace perdu en dehors des fichiers. On peut effacer des données ou supprimer des fichiers ce qui libère des blocs 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 a un espace assez volumineux (voir exemple figure 5.6)



Table

Fichier	Début	Fin
A	0	3
B	4	6
C	7	12
D	13	17
E	18	29
F	30	35
G	36	38

Figure 5.6 – *Allocation gontiguë d'espace disque pour 7 fichiers*

5.6.2 Allocation chainée (non contiguë)

Le principe est d'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. Lorsque le fichier change de taille, la gestion des blocs occupés est simple. Il n'y a donc aucune limitation de taille, si ce n'est l'espace disque lui même (voir figure 5.7).
Cette méthode présente l'avantage de l'élimination du problème de fragmentation externe. Aussi le faite de ne pas nécessiter une structure spéciale pour sa mise en place, constitue un autre avantage. En revanche, les inconvénients ici aussi sont multiples :

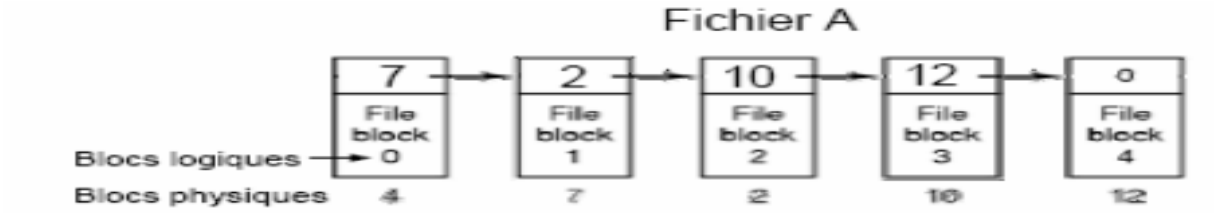


Figure 5.7 – *Allocation chainée*

— L'accès au fichier est totalement séquentiel, on doit toujours commencer le parcours du fichier à partir du début
— La perte d'un chaînage entraîne la perte de tout 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.

5.6.3 Allocation Contiguë indexé

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, les informations sur les numéros de blocs peuvent être obtenue à tout moment (voir figure 5.8.

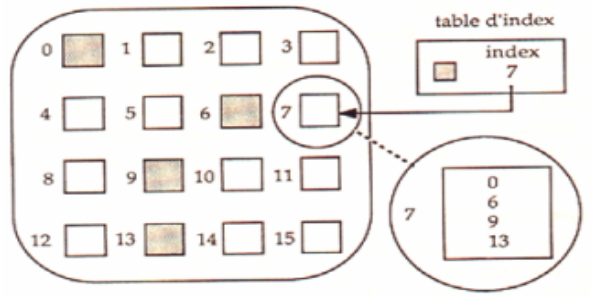


Figure 5.8 – *Allocation indexée*

La plus part des systèmes actuels appliquent ce mode. MS-DOS utilise la **FAT** (File Allocation Table) pour y conserver les chaînages entre les blocs. Windows NT utilise la **MFT** (Master File Table) associé au système **NTFS** (New Technology File System) .UNIX, GNU/Linux utilisent le **I-Node** (Index node).

1. FAT : On parle généralement de système de fichiers FAT16 et FAT32.
   — Le FAT16 est utilisé par MS-DOS. En FAT16, les numéros de blocs sont écrits sur 16 bits. Si on suppose que la taille d'un bloc est 32Ko, la taille maximale adressables est alors 2Go (2^16x32 Ko = 2097152 Ko = 2Go)
   — Le FAT32 est pris en charge par Windows 95 et les versions qui ont suivis. Les numéros de blocs sont écrits sur 32 bits (en réalité, sur 28bits, 4 bits étant réservés). Si on suppose que la taille d'un bloc est de 32 ko, la taille maximale adressable théoriquement est de 8 To (228

x 32 Ko = 8 To). Toutefois, Microsoft la limite volontairement à 32 Go sur les systèmes Windows 9x afin de favoriser NTFS.

2. NTFS : Le système de fichiers NTFS (New Technology File System) est utilisé par Windows2000, WindowsNT, Windows XP et Windows Vista. Il utilise un système basé sur une structure appelée MFT (Master File Table), permettant de contenir des informations détaillées sur les fichiers. Ce système permet ainsi l'utilisation de noms longs, mais, contrairement au système FAT32, il est sensible à la casse, c'est-à-dire qu'il est capable de différencier des noms en majuscules de noms en minuscules.

   Coté performances, l'accès aux fichiers sur une partition NTFS est plus rapide que sur une partition de type FAT car il utilise un arbre binaire performant pour localiser les fichiers. La limite théorique de la taille d'une partition est de 16 hexa octets (17 milliards de To), mais la limite physique d'un disque est de 2To (va encoder en 64 bits ==> = $2^64 = 18446744073709551616$ = 16 EiB (1 exbibyte = $1EiB = 2^60$ bytes).

   C'est au niveau de la sécurité que NTFS prend toute son importance, car il permet de définir des attributs pour chaque fichier.

3. Structure d'un I-Node
   La structure d'I-Node est utilisée par le système de gestion de fichier ext3fs d'Unix ou GNU/Linux (ext3fs pour third extented file system). Un nœud d'index est constitué d'attributs décrivant le fichier ou le répertoire et d'adresses de blocs contenant des données. Cette structure possède plusieurs entrées, elle permet au système de disposer d'un certain nombre de données sur le fichier :
   - La taille
   - L'identité du propriétaire et du groupe : un fichier en Unix est crée par un propriétaire, qui appartient à un groupe
   - Les droits d'accès : pour chaque fichier, Unix définit trois droits d'accès (lecture (r), écriture (w) et exécution (x)) pour chaque classe d'utilisateurs (trois types d'utilisateur propriétaire, membre du même groupe que le propriétaire, autres). Donc à chaque fichier, Unix associe neuf droits
   - Les dates de création, de dernière consultation et de dernière modification
   - Le nombre de références existant pour ce fichier dans le système
   - Les dix premiers blocs de données
   - D'autres entrées contiennent l'adresse d'autres blocs (on parle alors de bloc d'indirection) :
     - Une entrée pointe sur un bloc d'index qui contient 128 ou 256 pointeurs sur bloc de données (simple indirection)
     - Une entrée pointe sur un bloc d'index qui contient 128 ou 256 pointeurs sur bloc d'index dont chacun contient 128 ou 256 pointeurs sur bloc de données (double indirection)
     - Une entrée pointe sur un bloc d'index qui contient 128 ou 256 pointeurs sur bloc d'index dont chacun contient 128 ou 256 pointeurs sur bloc d'index dont chacun contient 128 ou 256 pointeurs sur

bloc de données (triple indirection)

La structure d'I-Node est conçue afin d'alléger le répertoire et d'en éliminer les attributs du fichier ainsi que les informations sur l'emplacement des données.

Une entrée dans un I-Node d'un répertoire contiendra donc un nom d'un fichier ou sous-répertoire et l'INode associé.

Exemple 1 *Si on suppose que la taille d'un bloc est de 1Ko, un fichier sous Unix peut avoir la taille maximale suivante :* $10x1Ko + 256x1Ko + 256x256x1Ko + 256x256x256x1Ko$*, ce qui donne en théorie plus de 16Go. En réalité, la taille réelle maximale d'un fichier est inférieure à cette valeur à cause de l'utilisation de pointeurs signés pour le déplacement au sein d'un fichier.*
Pour les fichiers les plus longs, trois accès au disque suffisent pour connaître l'adresse de tout octet du fichier. La figure 5.9 présente un exemple d'utilisation d'I-Node

5.6.4 La création de fichiers par le SE

Un fichier ou un répertoire sont tout deux créé e suivant les mêmes étapes qui sont les suivantes :

— La création d'une structure de données pour décrire le fichier. Les attributs du fichier sont sauvegardés dans cette structure de données

— La création du fichier proprement dit. Il s'agit d'allouer au fichier un certain nombre de blocs sur le disque selon sa taille. Les blocs d'un fichier vont contenir des données sans aucune structure particulière. Les blocs d'un répertoire ont par contre une structure bien particulière, en effet ils contiennent des noms et des attributs de fichiers et de sous répertoires

5.7 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.
Bitmap : Approche statique utilise une table de bits (vecteur de bits n blocs) comportant autant de bits que de blocs sur le disque. A chaque bloc du disque, correspond un bit dans la table, positionné à 1 si le bloc est occupé, à 0 si le bloc est libre (ou vice versa) (voir figure 5.10.
Si les blocs 3, 4, 5, 9, 10, 15, 16 sont libres : 11100011100111100...
Cette solution est utilisée pour trouver n blocs contigus, elle est utilisée dans les systèmes : NTFS, ext2fs
Exemple 2 *Par exemple, un disque de 300 Mo, organisé en blocs de 1 Ko. Supposons que chaque bloc soit adressé par 4 octets. Chaque bloc de la liste pourra contenir 255 (1024/4) adresses de blocs libres. La liste comprendra donc au plus 307.200/255 = 1205 blocs. Cette solution mobilise beaucoup plus de place que la précédente.*

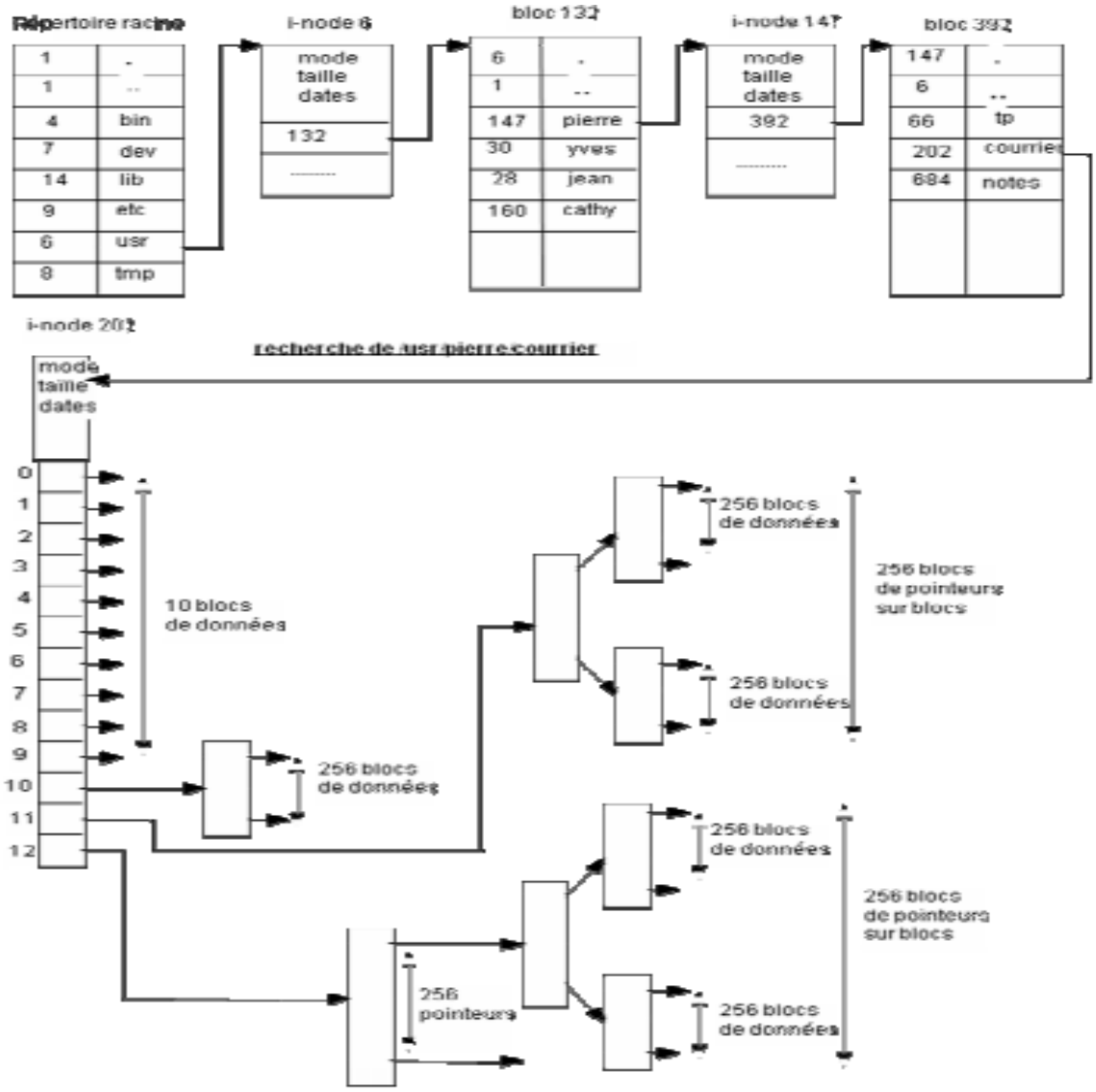


Figure 5.9 – *Exemple d'utilisation d'I-Node*

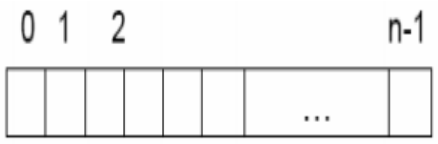


Figure 5.10 – *Vecteur de bits à n bloc*

5.8 Travaux Pratiques

5.9 Conclusion

Bibliographie

[1] Saïd Abdedaïm et Pascal Caron, *Système*, Département d'Informatique de Rouen Universite de Rouen, 76821 Mont-Saint Aignan, Cédex

[2] J. Gispert, J. Guizol, J.L. Massat, *Support de cours système d'exploitation*, Département d'informatique, Faculté de Luminy, 163, Avenue de Luminy, Case 901, 13288 Marseille, cedex 9, 23 février 2012