CHAPITRE I : INTRODUCTION ÀUX SYSTÈMES D'EXPLOITATION

1 Système informatique = le matériel + le logiciel

L'objectif d'un système informatique est d'automatiser le traitement de l'information.

Un système informatique est constitué de deux entités : le matériel et le logiciel.

Côté matériel, un ordinateur est composé de : L'Unité Centrale (UC) pour les traitements, la Mémoire Centrale (MC) pour le stockage, et les Périphériques d'E/S: disque dur, clavier, souris, flash disque, carte réseau... accessibles via des pilotes de périphériques.

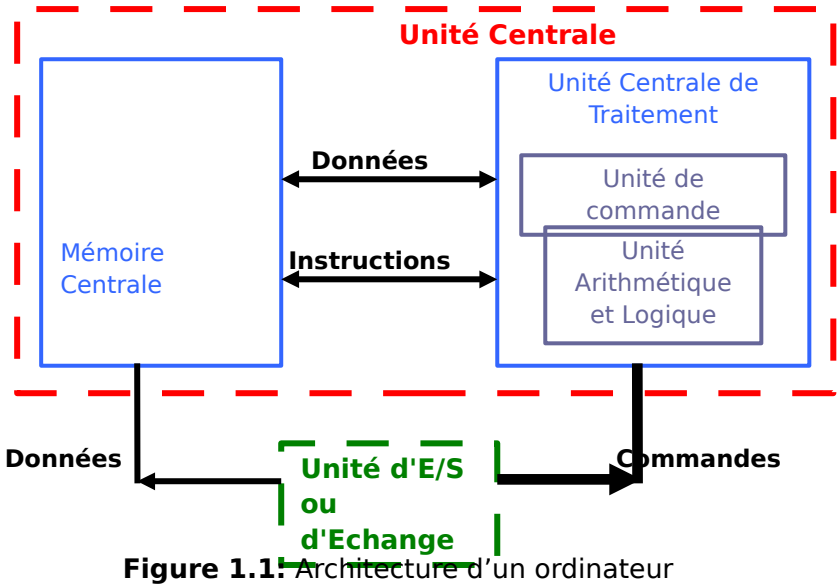


Figure 1.1: Architecture d'un ordinateur

Côté logiciel, un système informatique est composé de deux niveaux bien distincts : les Programmes d'application (achetés ou développés) et les logiciels de base. Dans les logiciels de base, on trouve le **système d'exploitation** (**S.E.**) et les utilitaires.

L'objectif du logiciel est d'offrir aux utilisateurs des fonctionnalités adaptées à leurs besoins. Le principe est de masquer les caractéristiques physiques du matériel.

2 Organisation d'un Système informatique

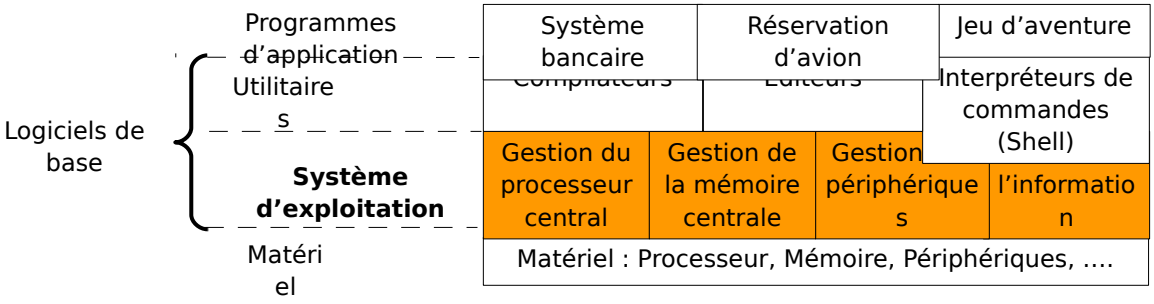


Figure 1.2 : Organisation d'un système informatique

3 L'ordinateur serait bien difficile à utiliser sans S.E.

En effet, le matériel seul ne sait pas faire grand chose :

- Il peut exécuter des programmes, **mais** comment entrer ces programmes dans la machine ? Comment les charger en mémoire ?
- Il peut sauvegarder des données sur disque **mais** comment accède-t-on à un disque ? Comment range-t-on les informations sur le disque ?
- Il peut lire ce qui est tapé au clavier **à condition** qu'on écrive un programme chargé de cette tâche.

Bref, nous avons besoin d'un ensemble de programmes (un logiciel) qui permette d'exploiter les ressources de la machine (périphériques, mémoire, processeur(s)). Ce logiciel, c'est le **système d'exploitation** (**S.E.**).

4 Qu'est ce qu'un Système d'Exploitation ?

Le système d'exploitation (**Operating System**, **O.S.**) est l'intermédiaire entre un ordinateur (ou en général un appareil muni d'un processeur) et les applications qui utilisent cet ordinateur ou cet appareil. Son rôle peut être vu sous deux aspects complémentaires :

1. **Machine étendue ou encore machine virtuelle (Virtual Machine)**

   Son rôle est de masquer des éléments fastidieux liés au matériel, comme les interruptions, les horloges, la gestion de la mémoire, la gestion des périphériques (déplacement du bras du lecteur de disquette) ...etc. Cela consiste à fournir des outils adaptés aux besoins des utilisateurs indépendamment des caractéristiques physiques.

   Exemple

   READ et WRITE = 13 paramètres sur 9 octets ; en retour le contrôleur renvoie 23 champs d'état et d'erreur regroupés sur 7 octets.

2. **Gestionnaire de ressources**

   Le système d'exploitation permet l'ordonnancement et le contrôle de l'allocation des processeurs, des mémoires et des périphériques d'E/S entre les différents programmes qui y font appel, avec pour objectifs : **efficacité** (utilisation maximale des ressources), **équité** (pas de programme en

attente indéfinie), **cohérence** (entre des accès consécutifs), et **protection** (contre des accès interdits).

Exemples

- 3 programmes essaient d'imprimer simultanément leurs résultats sur une même imprimante recours à un fichier tampon sur disque.
- L'accès concurrent à une donnée ; lecture et écriture concurrentes (par deux processus) sur un même compteur.

Ce rôle de gestionnaire de ressources est crucial pour les systèmes d'exploitation manipulant plusieurs tâches en même temps (**multi-tâches** (**Multitasking**)).

On peut trouver un S.E. sur les ordinateurs, les téléphones portables, les assistants personnels, les cartes à puce, …etc.

5 Fonctions d'un système d'exploitation général

Les rôles du système d'exploitation sont divers :

- **Gestion du processeur** : allocation du processeur aux différents programmes.
- **Gestion des objets externes** : principalement les fichiers.
- **Gestion des entrées-sorties** : accès aux périphériques, via les pilotes.
- **Gestion de la mémoire** : segmentation et pagination.
- **Gestion de la concurrence** : synchronisation pour l'accès à des ressources partagées.
- **Gestion de la protection** : respect des droits d'accès aux ressources.
- **Gestion des accès au réseau** : échange de données entre des machines distantes.

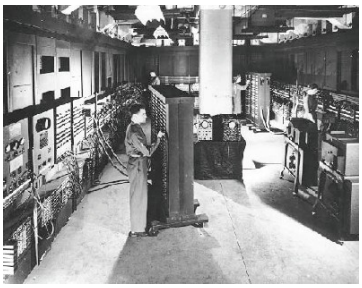
6 Historique

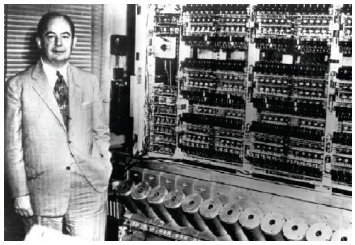
Les systèmes d'exploitation ont été historiquement liés à l'architecture des ordinateurs sur lesquels ils étaient implantés. Nous décrirons les générations successives des ordinateurs et observons à quoi ressemblait leur système d'exploitation.

6.1 Porte ouverte ou exploitation self service (1945-1955)

Les machines de la première génération (Figure 1.3), appelées **Machines à Tubes**, étaient dépourvues de tout logiciel. Les programmes utilisateurs étaient chargés en mémoire, exécutés et mis au point depuis un pupitre de commande.

Ces machines étaient énormes, remplissaient les salles avec des centaines de tubes à vide (**Vacuum Tubes**), coûteuses, très peu fiables et beaucoup moins rapides car le temps de cycle se mesurait en secondes. Les programmes étaient écrits directement en langage machine : ils étaient chargés en mémoire, exécutés et mis au point à partir d'un **pupitre de commande** (Figure 1.4). Au début de 1950, la procédure s'est améliorée grâce à l'introduction de **cartes perforées**.


ENIAC


Von Neumann

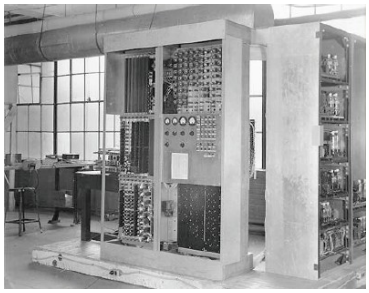

EDVAC

Figure 1.3 : Exemples de machines à tubes

Afin d'utiliser la machine, la procédure consistait à allouer des **tranches de temps** directement aux usagers, qui se réservent toutes les ressources de la machine à tour de rôle pendant leur durée de temps. Les périphériques d'entrée/sortie en ce temps étaient respectivement le lecteur de cartes perforées et l'imprimante. Un pupitre de commande était utilisé pour manipuler la machine et ses périphériques.

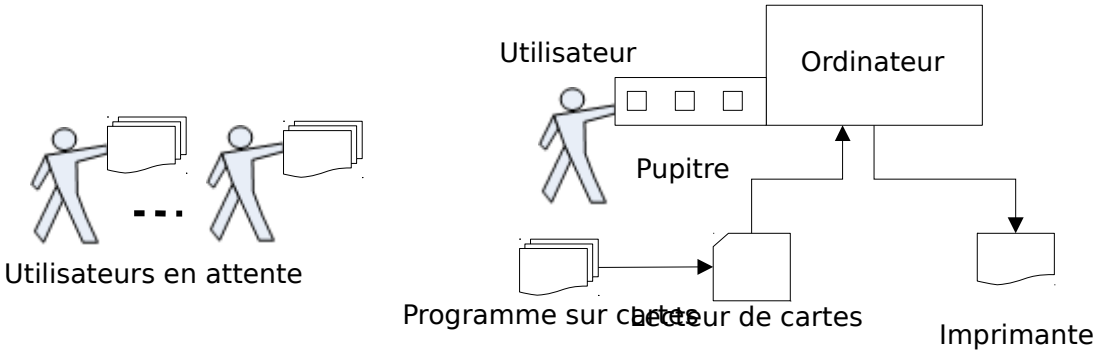


Figure 1. 4 : Utilisation de la machine en Porte Ouverte

Chaque utilisateur, assurant le rôle d'**opérateur**, devait lancer un ensemble d'opérations qui sont :

- Placer les cartes du programme dans le lecteur de cartes.
- Initialiser un programme de lecteur des cartes.
- Lancer la compilation du programme utilisateur.
- Placer les cartes données s'il y en a, dans le lecteur de cartes.
- Initialiser l'exécution du programme compilé.
- Détecter les erreurs au pupitre et imprimer les résultats.

Inconvénients

- Temps perdu dans l'attente pour lancer l'exécution d'un programme.
- Vitesse d'exécution de la machine limitée par la rapidité de l'opérateur qui appuie sur les boutons et alimente les périphériques.
- Pas de différences entre : concepteurs ; constructeurs ; programmeurs ; utilisateurs ; mainteneurs.

6.2 Traitement par lots (Batch Processing, 1955-1965)

Ce sont des systèmes réalisant le séquencement des jobs ou travaux selon l'ordre des cartes de contrôle à l'aide d'un **moniteur d'enchaînement**. L'objectif était de réduire les pertes de temps occasionnées par l'oisiveté du processeur entre l'exécution de deux jobs ou programmes (durant cette période, il y a eu apparition des machines à **transistor** avec unités de **bandes magnétiques**, donc évolution des ordinateurs).

L'idée directrice était de collecter un ensemble de travaux puis de les transférer sur une bande magnétique en utilisant un ordinateur auxiliaire (**Ex.** IBM 1401). Cette bande sera remontée par la suite sur le lecteur de bandes de l'ordinateur principal (**Ex.** IBM 7094) afin d'exécuter les travaux transcrits en utilisant un programme spécial (l'ancêtre des S.E. d'aujourd'hui. **Ex.** FMS : *Fortran Monitor System*, IBSYS). Les résultats seront récupérés sur une autre bande pour qu'ils soient imprimés par un ordinateur auxiliaire. Cette situation est illustrée à la Figure 1.5.

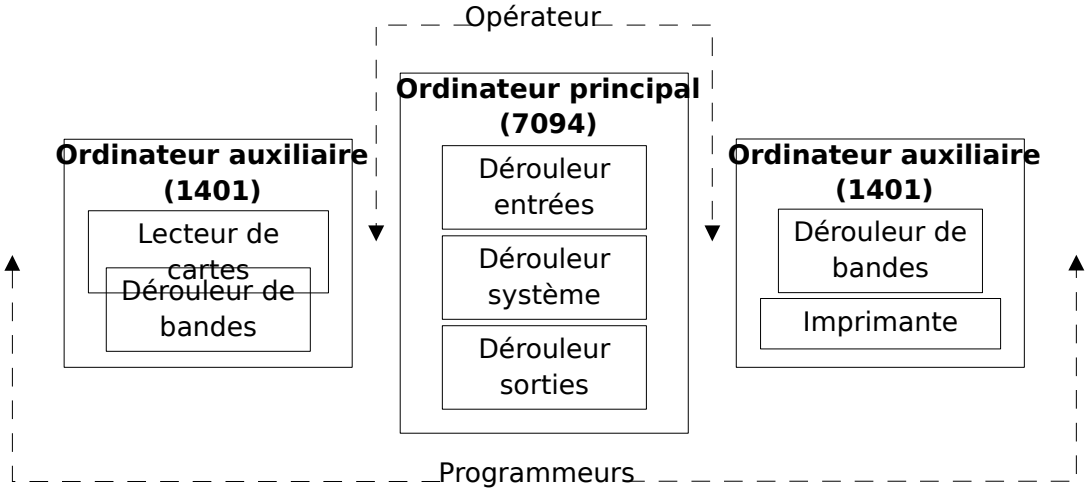


Figure 1.5 : Un système de traitement par lots

Quand le moniteur rencontre une carte de contrôle indiquant l'exécution d'un programme, il charge le programme et lui donne le contrôle. Une fois terminé, le programme redonne le contrôle au moniteur d'enchaînement. Celui-ci continue avec la prochaine carte de contrôle, ainsi de suite jusqu'à la terminaison de tous les jobs. La structure d'un travail soumis est montrée sur la Figure 1.6 :

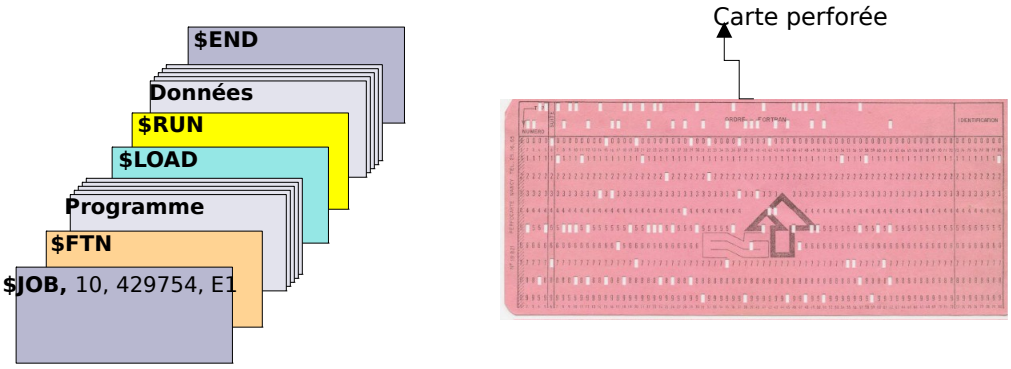


Figure 1.6 : Structure d'un travail FMS typique

Inconvénients

- Perte de temps dû à l'occupation du processeur durant les opérations d'E/S. (En effet, le processeur restait trop inactif, car la vitesse des périphériques mécaniques était plus lente que celle des dispositifs électroniques).
- Les tâches inachevées sont abandonnées.

6.3 Multiprogrammation (Multiprogramming, 1965-1970)

L'introduction des **circuits intégrés** dans la construction des machines a permis d'offrir un meilleur rapport coût/performance. L'introduction de la technologie des **disques** a permis au système d'exploitation de conserver tous les travaux sur un disque, plutôt que dans un lecteur de cartes (Arrivée des unités disques à stockage important et introduction de **canaux d'E/S**).

L'idée était alors, pour pallier aux inconvénients du traitement par lots, de maintenir en mémoire plusieurs travaux ou jobs prêts à s'exécuter, et partager efficacement les ressources de la machine entre ces jobs.

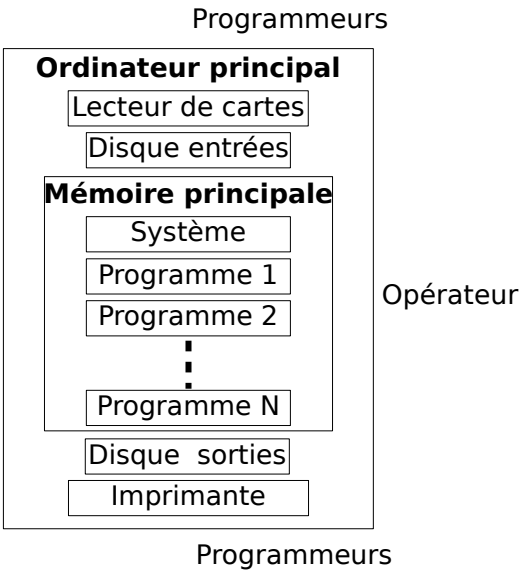
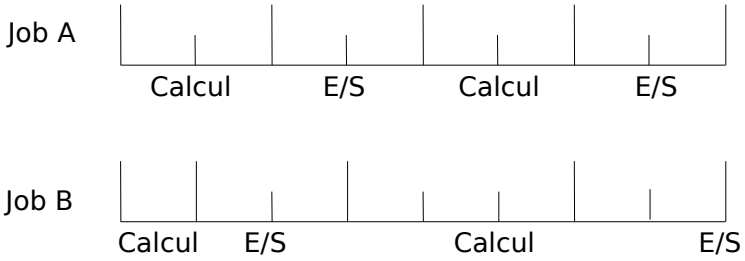


Figure 1.7 : Un système de multiprogrammation

En effet, le processeur est alloué à un job, et dès que celui-ci effectue une demande d'E/S, le processeur est alloué à un autre job, éliminant ainsi les temps d'attente de l'unité de traitement chargé des E/S, appelé canal d'E/S.

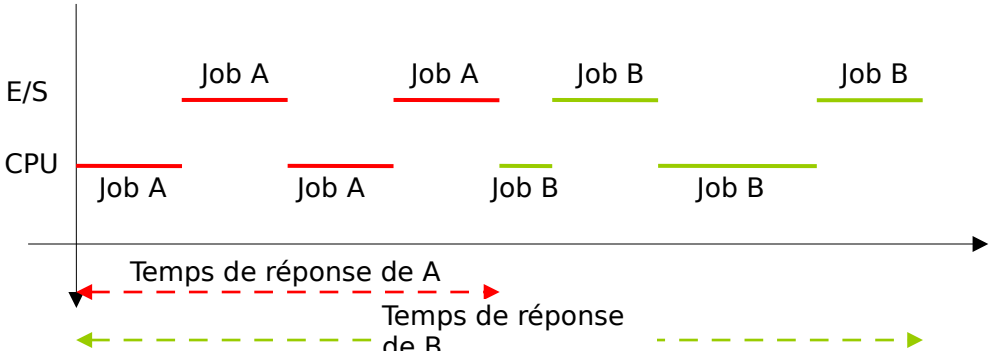
Exemple

Soient les deux programmes A et B suivants :

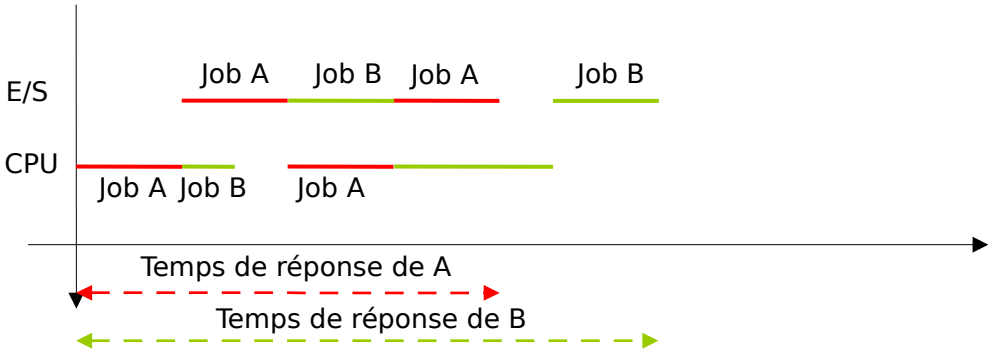


On suppose qu'on a un seul périphérique d'E/S.

- **Système mono-programmé**



- **Système multiprogrammé**



Comparaison de la Monoprogrammation et de la Multiprogrammation

Monoprogrammation	Multiprogrammation
▪ Mauvaise utilisation des ressources (processeur, mémoire, E/S, …etc.). ▪ Temps de réponse imposé par les jobs très longs. ▪ S.E. simple ; seule contrainte : protéger la partie résidente du système des utilisateurs.	▪ Possibilité de mieux équilibrer la charge des ressources. ▪ Mieux utiliser la mémoire (minimiser l'espace libre). ▪ Possibilité d'améliorer le temps de réponse pour les travaux courts. ▪ Protéger les programmes utilisateurs des actions des autres utilisateurs, et protéger aussi la partie résidente des usagers.

6.4 Temps partagé (Time Sharing, 1970-)

C'est une variante du mode multiprogrammé où le temps CPU est distribué en petites tranches appelées **quantum de temps**.

L'objectif est d'offrir aux usagers une interaction directe avec la machine par l'intermédiaire de terminaux de conversation, et de leur allouer le processeur successivement durant un quantum de temps, chaque utilisateur aura l'impression de disposer de la machine à lui tout seul. Il peut aussi contrôler le job qu'il a soumis directement à partir du terminal (corriger les erreurs, recompiler, resoumettre le job, …).

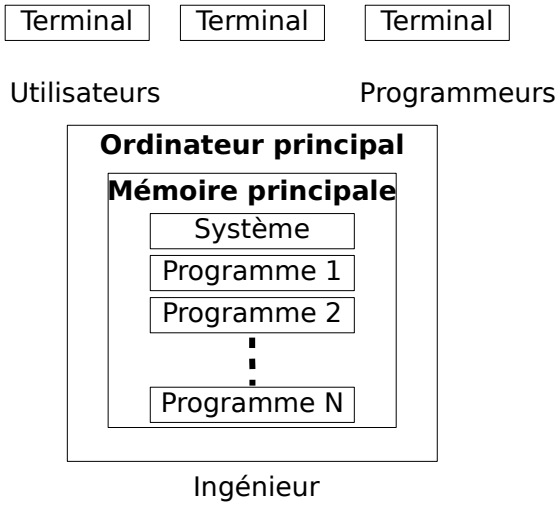


Figure 1.8 : Un système à temps partagé

Parmi les premiers systèmes à temps partagé, nous citons : **CTSS** (**C**ompatible **T**ime **S**haring **S**ystem), **MULTICS** (**MULT**iplexed **I**nformation and **C**omputing **S**ervice), **UNIX**, **MINIX**, **LINUX**. En fait, la plupart des systèmes d'aujourd'hui sont en temps partagé.

7 Exemple de systèmes d'exploitation : UNIX

7.1 Historique

L'histoire du système d'exploitation Unix commence en **1969** aux **laboratoires AT&T** de Bell, et ceci avec le développement d'une version simplifiée du système **MULTICS** par **Ken Thompson**. **Brian Kernighan** appela cette version **UNICS** (**Uniplexed Information and Computer Service**) qui devint ensuite **UNIX** et qui était entièrement écrite en **assembleur**.

Le système Unix a connu un véritable succès, lorsqu'il fut réécrit en **langage C** en **1973** par **Dennis Ritchie** et Thompson. Le langage de programmation C a d'ailleurs été conçu initialement par D. Ritchie en 1971 pour la refonte d'Unix et son portage sur de nombreuses architectures matérielles.

En **1975**, le système **Unix (V6)** est distribué aux universités et aux centres de recherches. La principale université qui va travailler sur Unix est l'**université de Berkeley**, qui va produire ses propres versions appelées **BSD** (**Berkeley Software Distribution**). A Berkeley, les efforts portent sur l'intégration des

protocoles réseaux TCP/IP, la gestion de la mémoire avec l'introduction de la pagination, la modification de certains paramètres du système (taille des blocs, nombre des signaux...) et l'ajout d'outils (l'éditeur **vi**, un interpréteur de commandes **csh**...).

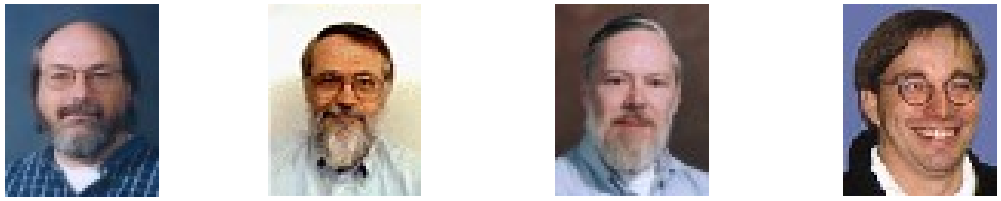
En **1979**, les Bell Labs sortent leur version appelée **UNIX V7**, avec en particulier, l'ajout de nouveaux utilitaires et un effort en matière de portabilité. Cette version est la première à être diffusée dans le monde industriel. On peut dire qu'elle est à l'origine du développement du marché Unix.

Les nombreuses modifications et améliorations apportées au système UNIX, par AT&T et Berkeley ont abouti aux versions **System V Release 4** d'AT&T et **4.4BSD** de Berkeley.

Le support de l'**environnement graphique** est apparu avec le système **XWindow** du **MIT** en **1984**.

La fin des années 80 est marquée par une croissance sans précédent du nombre de systèmes Unix dans le domaine des systèmes d'exploitation. Les principales versions actuelles sont System VR4, GNU/Linux, SUN Solaris, FreeBSD, IBM AIX, Microsoft Xenix...etc. Pour qu'un système d'exploitation puisse être un Unix, il faut qu'il respecte la norme **POSIX** (**Portable Operating System Interface**). Tout logiciel écrit en respectant la norme Posix devrait fonctionner sur tous les systèmes Unix conformes à cette norme.

Une version gratuite d'Unix porte le nom de **Linux** (code source disponible). Elle a été créée par **Linus Torvalds** en **1991**. Par la suite, un grand nombre de programmeurs ont contribué à son développement accéléré. Conçu d'abord pour tourner sur des machines avec le processeur 80x86, Linux a migré à plusieurs autres plate-formes.



Ken Thompson Brian Kernighan Dennis Ritchie Linus Torvalds

Figure 1.9 : Les Pionniers d'UNIX/LINUX

7.2 Architecture Générale d'UNIX

UNIX a été conçu autour d'une architecture en **couche** qui repose sur différents niveaux bien distincts (Voir Figure 1.11) ; à savoir :

- Le noyau
- Un interpréteur de commandes (le shell)
- Des bibliothèques
- Un nombre important d'utilitaires.

A. Le noyau

Le noyau (**Kernel**) est la partie centrale d'Unix. Il est **résident** ; il se charge en mémoire au démarrage. Il s'occupe de gérer les tâches de base du système :

- L'initialisation du système,
- La gestion des processus,
- La gestion des systèmes de fichiers,
- La gestion de la mémoire et du processeur,
- Etc.

Les programmes en **espace utilisateur** (**user-space**) appellent les services du noyau via des **appels systèmes** (**System Calls**). En effet, les appels systèmes font entrer l'exécution en **mode noyau**. Dans ce mode, le processus est assuré de garder le processeur jusqu'au retour au **mode utilisateur** lorsque l'appel système est terminé.

B. Bibliothèques

L'interface entre le noyau Unix et les applications est définit par une bibliothèque (**Ex.** libc.a pour le langage C). Elle contient les modules permettant d'utiliser les primitives du noyau mais aussi des fonctions plus évoluées combinant plusieurs primitives. D'autres bibliothèques sont utilisées pour des services spécialisés (fonctions graphiques,...).

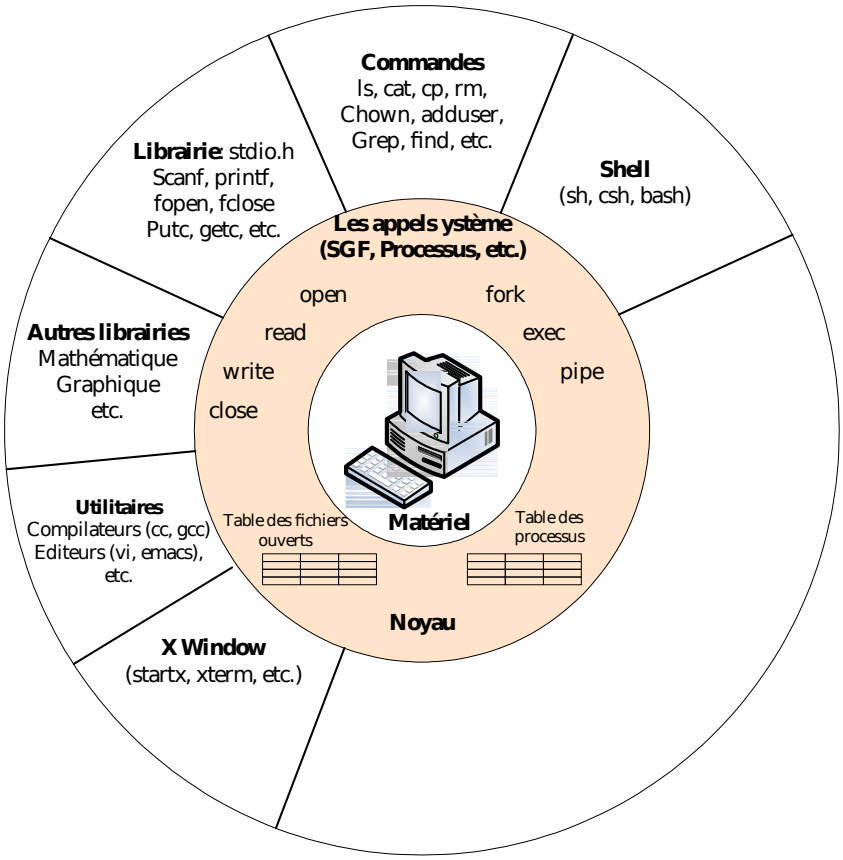


Figure 1.11 : Structure du système UNIX

C. Le Shell

Le shell désigne l'interface utilisateur sous UNIX. C'est un programme qui permet à l'utilisateur de dialoguer avec le noyau. Il joue un double rôle celui d'**interpréteur de commandes** et celui de **langage de programmation**. Il existe plusieurs shells différents mais les plus répondus sont:

- le Bourne Shell : sh
- le C-shell : csh
- le Korn-Shell : ksh
- Bash (Bourne Again Shell) : est un interpréteur (Shell) compatible sh qui exécute les commandes lues depuis l'entrée standard, ou depuis un fichier. C'est le shell par défaut sous Gnu/Linux.

D. Utilitaires

UNIX est livré avec un grand nombre de programmes utilitaires, parmi lesquels :

- Compilateurs : cc, gcc
- Gestionnaire d'applications : make
- Editeurs de texte : vi, emacs

7.3 Caractéristiques

Les principales caractéristiques, auxquelles est dû le succès d'UNIX, sont :

- **Portabilité** : Une des premières caractéristiques d'Unix est son écriture (à hauteur de 95%) en langage C, permettant ainsi une portabilité sur la plupart des architectures en allant des micro-ordinateurs jusqu'aux supercalculateurs.
- **Multi-utilisateurs** et **Multitâches** : Plusieurs utilisateurs peuvent accéder simultanément au système ; chaque utilisateur peut effectuer une ou plusieurs tâches en même temps.
- **Temps partagé** : c'est-à-dire que les ressources du processeur et du système sont réparties entre les utilisateurs.
- **Interface utilisateur interactive (shell)** : elle est constituée d'un programme séparé du noyau permettant à l'utilisateur de choisir son environnement de travail. Elle intègre un langage de commandes très sophistiqué (scripts).
- **Système de fichiers hiérarchique** : plusieurs systèmes de fichiers peuvent être rattachés au système de fichiers principal ; chaque système de fichiers possède ses propres répertoires.
- **Entrées-Sorties intégrées au système de fichiers** : les périphériques sont représentés par des fichiers, ce qui rend le système indépendant du matériel et en assure la portabilité ; l'accès aux périphériques est donc identique à l'accès aux fichiers ordinaires.
- **Gestion de la mémoire virtuelle** : un mécanisme d'échange entre la mémoire centrale (MC) et le disque dur permet de pallier un manque de MC et optimise le système.

8 Fonctionnement d'un Système Informatique Moderne

Un système informatique moderne à usage général est constitué d'une mémoire, U.C., et d'un certain nombre de périphériques connectés par un bus commun fournissant l'accès à la mémoire, et régis par des cartes électroniques appelés contrôleurs.

Pour qu'un ordinateur commence à fonctionner (quand il est mis sous tension ou réinitialisé), il doit avoir un programme initial à exécuter. Ce programme initial, appelé **programme d'amorçage**, est simple : il initialise tous les aspects du système, depuis les registres de l'U.C. jusqu'aux contrôleurs de périphériques et contenu de la mémoire. Le programme d'amorçage doit savoir après comment charger le S.E. et comment commencer à l'exécuter.

Pour atteindre cet objectif, le programme d'amorçage cherche le noyau du S.E. à une adresse spécifiée (en général, au 1er secteur de l'unité disque, appelé secteur d'amorçage), puis, le charge en mémoire. Le S.E. démarre alors l'exécution du premier processus d'initialisation et attend qu'un événement se produise.

9 Interactions Utilisateur/Système

Pour un utilisateur, le système d'exploitation apparaît comme un ensemble de procédures, trop complexes pour qu'il les écrive lui-même. Les bibliothèques des **appels système** sont alors des procédures mises à la disposition des programmeurs. Ainsi un programme C/C++ peut utiliser des appels système d'Unix/Linux comme open(), write() et read() pour effectuer des Entrées/Sorties de bas niveau.

L'**interpréteur de commandes** constitue une interface utilisateur/système. Il est disponible dans tous les systèmes. Il est lancé dès la connexion au système et invite l'utilisateur à introduire une commande. L'interpréteur de commandes récupère puis exécute la commande par combinaison d'appels système et d'outils (compilateurs, éditeurs de lien, etc.). Il affiche les résultats ou les erreurs, puis se met en attente de la commande suivante. Par exemple, la commande de l'interpréteur (shell) d'Unix suivante permet d'afficher à l'écran le contenu du fichier appelé essai : cat essai.txt

L'introduction du graphisme dans les interfaces utilisateur a révolutionné le monde de l'informatique. L'interface graphique a été rendue populaire par le **Macintosh** de **Apple**. Elle est maintenant proposée pour la plupart des machines.

CHAPITRE II : MÉCANISMES DE BASE D'EXÉCUTION DES PROGRAMMES

1 Machine de VON-NEUMANN

9.1 Définition

Une machine de **Von-Neumann** est un calculateur électronique **à base de mémoire** dont les composants sont :

- Mémoire Centrale (MC).
- Processeur ou Unité Centrale (UC) ; pour effectuer les calculs et exécuter les instructions.
- Unités périphériques ou d'Entrée/Sortie (E/S).

9.2 Unité Centrale de Traitement (CPU, Central Process Unit)

Appelée aussi "**Processeur Central**" (PC), elle est composée de :

- L'Unité de Commande (UC).
- L'Unité Arithmétique et Logique (UAL).
- Les Registres (Données, Adresses, Contrôle, Instruction).
- Bus Interne.

Le processeur exécute des instructions ou actions d'un programme. C'est le cerveau de l'ordinateur. Une **action** fait passer, en un temps fini, le processeur et son environnement d'un **état initial** à un **état final**. Les actions sont constituées de suites d'instructions **élémentaires**.

1. Registres du processeur central

Les registres sont une sorte de mémoire interne à la CPU, à accès très rapide qui permettent de stocker des résultats temporaires ou des informations de contrôle. Le nombre et le type des registres implantés dans une unité centrale font partie de son architecture et ont une influence importante sur la programmation et les performances de la machine. Le processeur central dispose d'un certain nombre de registres physiques :

- **Le Compteur Ordinal (CO, Program Counter (PC), Instruction Pointer (IP)) :** Il contient l'adresse de la prochaine instruction à exécuter.
- **Le Registre d'Instruction (RI, Instruction Register (IR)) :** Il contient l'instruction en cours d'exécution.

  Exemple : Instruction **Add** ACC, A.
- **Les Registres Généraux :** Ils permettent de faire le calcul, la sauvegarde temporaire de résultats, …etc. Il s'agit du registre accumulateur, index (utilisé pour calculer l'adresse réelle d'un mot), …etc.

  Plus précisément, les registres de l'UAL se divisent en différents groupes :

- Les **Registres Arithmétiques** (ACC) qui servent aux opérations arithmétiques.
- Les **Registres d'Index** qui sont utilisés pour l'**adressage indexé**. Dans ce cas l'adresse effective d'un opérande est obtenue en ajoutant le contenu du registre d'index à l'adresse contenue dans l'instruction.
- Les **Registres de Base** qui permettent le calcul des adresses effectives. Un registre de base contient une adresse de référence, par exemple l'adresse physique correspondant à l'adresse virtuelle 0. L'adresse physique est obtenue en ajoutant au champ adresse de l'instruction le contenu du registre de base.
- Les **Registres Banalisés** qui sont des registres généraux pouvant servir à diverses opérations telles que stockage des résultats intermédiaires, sauvegarde des informations fréquemment utilisées, ... etc. Ils permettent de limiter les accès à la mémoire, ce qui accélère l'exécution d'un programme.

- **Le Registre Pile (Stack Pointer, SP) :** Il pointe vers la tête de la pile du processeur. Cette pile est une pile particulière (appelée **pile système**) est réservée à l'usage de l'unité centrale, en particulier pour sauvegarder les registres et l'adresse de retour en cas d'interruption ou lors de l'appel d'une procédure. Le pointeur de pile est accessible au programmeur, ce qui est souvent source d'erreur.

  Exemple : Utilisation dans le cas d'appels imbriqués de procédures A⇒B→C→D.

- **Le Registre Mot d'Etat (PSW, Program Statut Word) :** Il indique l'état du processeur donné par des **indicateurs** (**Flags**) qu'on verra ci-après.

2. Cycle d'exécution du processeur

Un programme séquentiel est composé d'une suite d'instructions. L'exécution du programme fait évoluer l'état de la machine d'un état à un autre. Cette évolution est **discrète** : l'état n'est observable qu'en des instants particuliers appelés "**points observables**", qui correspondent en général aux débuts et fins d'instructions.

Le processeur central exécute continuellement le cycle suivant :

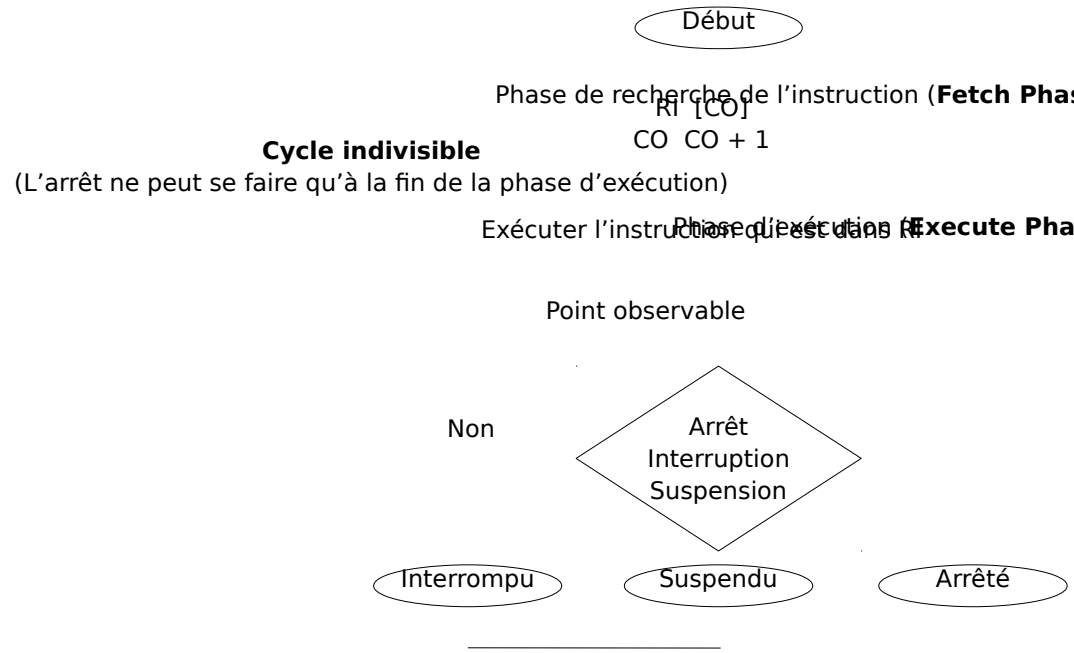


Figure 2.1 : Cycle d'exécution du processeur

Exemple

- Programme suspendu lors d'une demande d'E/S et le périphérique est en panne et ne répond pas, ou suspendu par le système pour une durée déterminée.
- Un programme est interrompu par l'exécution d'un programme plus prioritaire que lui.

Remarque

Le cycle de processeur est **indivisible** et ne peut être interrompu qu'à la fin de la phase **Execute** (i.e. on exécute au moins une instruction élémentaire d'un programme pour pouvoir interrompre).

3. Etat du processeur

Il est décrit par le contenu de son registre d'état ou mot d'état (PSW). Le PSW contient plusieurs types d'informations :

- **Etat d'exécution du programme**

  Le processeur peut être dans un état **actif** où il exécute des instructions, ou dans l'état d'**attente** où l'exécution est suspendue. L'état d'exécution en un point observable est donné par :

  1. L'adresse de la prochaine instruction à exécuter (CO).
  2. Les valeurs courantes des **codes conditions** qui sont des bits utilisés dans les opérations arithmétiques et comparaisons des opérandes.
  3. Le **mode d'exécution**. Pour des raisons de protection, l'exécution des instructions et l'accès aux ressources se fait suivant des modes d'exécution. Ceci est nécessaire pour pouvoir réserver aux seuls

programmes du S.E. l'exécution de certaines instructions. Deux modes d'exécution existent généralement :

- Mode **privilégié** ou **maître** (ou **superviseur**). Il permet :
  - L'exécution de tout type d'instruction. Les instructions réservées au mode maître sont dites **privilégiées** (Ex. instructions d'E/S, protection, …etc.).
  - L'accès illimité aux données.
- Mode **non privilégié** ou **esclave** (ou **usager**). Il permet :
  - Exécution d'un répertoire limité des instructions. C'est un sous ensemble du répertoire correspondant au mode maître.
  - Accès limité aux données d'usager.

4. masques d'interruption (seront détaillés dans la suite).

- Informations sur le contexte accessible en mémoire et les droits d'accès associés : Tables de segments de données, indicateurs de protection de mémoire, …etc.

Remarque

L'état d'un processeur n'est observable qu'entre deux cycles du processeur.

10 Cheminement d'un Programme dans un Système

Le passage d'un programme de la forme externe à la forme interne se fait en plusieurs étapes, selon le cheminement suivant :

Un programme est généralement écrit dans un langage évolué (Pascal, C, VB, Java, etc.). Pour faire exécuter un programme par une machine, on passe généralement par les étapes suivantes (Voir Figure 2.2) :

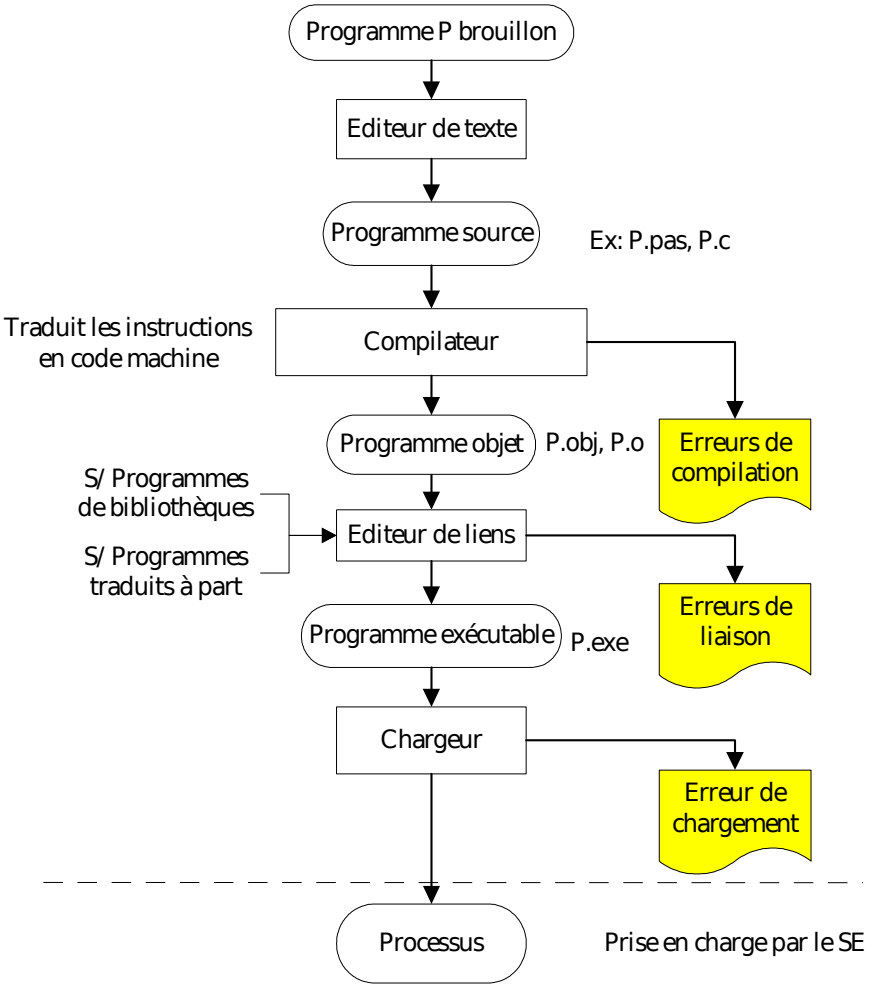


Figure 2.2 : Cheminement d'un programme dans un système

10.1 Editeur de texte (Text Editor)

C'est un logiciel interactif qui permet de saisir du texte à partir d'un clavier, et de l'enregistrer dans un fichier.

Exemple : Bloc Notes, vi, emacs, ...etc.

10.2 Compilateur (Compiler)

Un compilateur est un programme qui traduit des programmes écrits dans des langages évolués (Pascal, C, Ada, Java, ...etc.) en programme binaires ou en langage machine, appelés aussi **objets**.

Le **langage machine** est un ensemble de codes binaires directement décodables et exécutables par la machine.

10.3 Editeur de liens (Linker)

Un programme source peut être constitué :

1. des instructions,
2. des données localement définies,
3. des données externes, et

4. des procédures ou sous-programmes externes (bibliothèques de fonctions).

Avant d'exécuter le programme objet, il est nécessaire de rassembler les parties non locales et les procédures externes avec le programme.

L'éditeur de liens est donc un logiciel permettant de combiner plusieurs programmes objets en un seul programme.

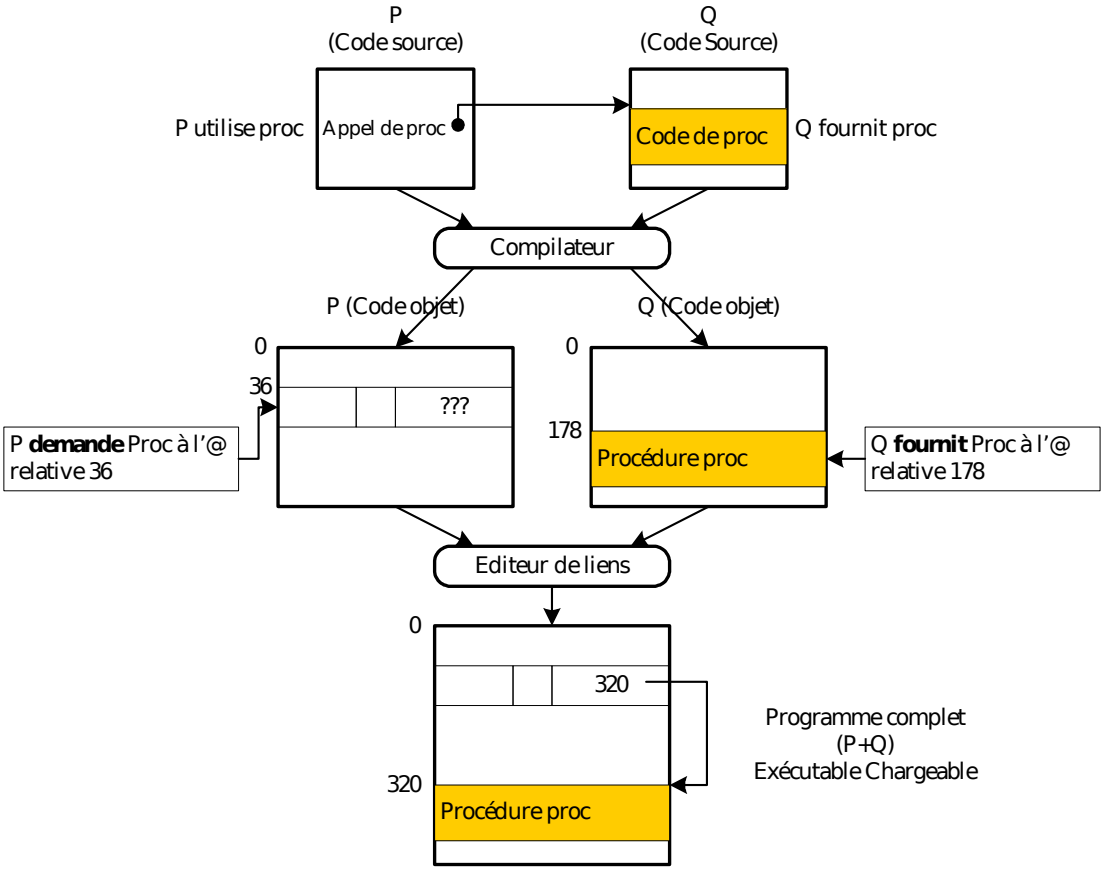


Figure 2.3 : Principe d'édition de liens

Pour pouvoir développer de gros programmes, on structure ceux-ci en modules que l'on traduit indépendamment. Ainsi, un programme peut être constitué de plusieurs fichiers dont l'un est le programme principal. Ce dernier fait appel aux sous-programmes, ce qui donne lieu à des références extérieures.

Lors de la compilation, le compilateur ne peut pas remplacer les références extérieures par les adresses correspondantes puisqu'il ne les connaît pas. Le programme objet généré contient donc le code machine des instructions et la liste des références extérieures, ainsi que la liste des adresses potentiellement référençables depuis l'extérieur du module. Après la compilation, chaque sous-programme commence à **l'adresse logique 0**.

L'éditeur de liens ordonne les sous-programmes et le programme appelant, modifie les adresses de chacun d'eux et produit un programme final dont l'origine est à l'adresse 0.

10.4 Le chargeur (Loader)

Après l'édition de liens, le programme exécutable doit être en MC pour être exécuté. Le chargeur est un programme qui installe ou charge un exécutable en MC à partir d'une adresse déterminée par le S.E.

11 Concepts de Processus et de Multiprogrammation

11.1 Définition

Un **processus** (**Process**) est un **programme en cours d'exécution.** Tout processus possède des **caractéristiques** propres (**Ex.** un numéro d'identification), des **ressources** qu'il utilise (comme des fichiers ouverts) et se trouve à tout moment dans un **état** (en exécution ou en attente ...).

Un processus est constitué d' :

- Un **code exécutable** du programme en exécution.
- Un **contexte** qui est une image décrivant l'environnement du processus.

11.2 Contexte d'un processus

Le contexte d'un processus est l'ensemble des données qui permettent de reprendre l'exécution d'un processus qui a été interrompu. Il est formé des contenus de :

- Compteur Ordinal (CO)
- Mot d'état PSW
- Registres généraux
- Pile

Le CO et le PSW représentent le **petit contexte**, et les registres généraux et la pile représentent le **grand contexte**.

11.3 Image mémoire d'un processus

L'espace mémoire alloué à un processus, dit **image mémoire** (**Memory Map**) du processus, est divisé en un ensemble de parties :

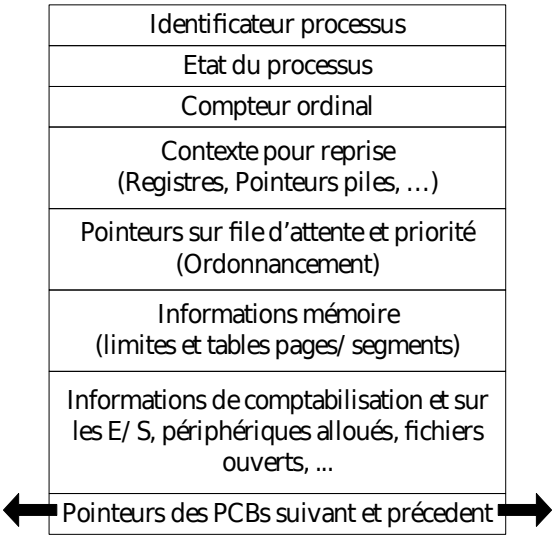
1. **Code (Text) ;** qui correspond au code des instructions du programme à exécuter. L'accès à cette zone se fait en **lecture seulement** (**Read-Only**).
2. **Données (Data) ;** qui contient l'ensemble des constantes et variables déclarées**.**
3. **Pile (Stack) ;** qui permet de stocker
   - Les valeurs des registres,
   - Variables locales et paramètres de fonctions,
   - Adresse de retour de fonctions.

4. **Tas (Heap) ;** une zone à partir de laquelle l'espace peut être alloué dynamiquement **en cours d'exécution (Runtime)**, en utilisant par exemple les fonctions **new** et **malloc**.

11.4 Descripteur de Processus (PCB)

Chaque processus est représenté dans le SE par un **bloc de contrôle de processus** (**Process Control Bloc, PCB**). Le contenu du PCB (Figure 2.4) varie d'un système à un autre suivant sa complexité. Il peut contenir :

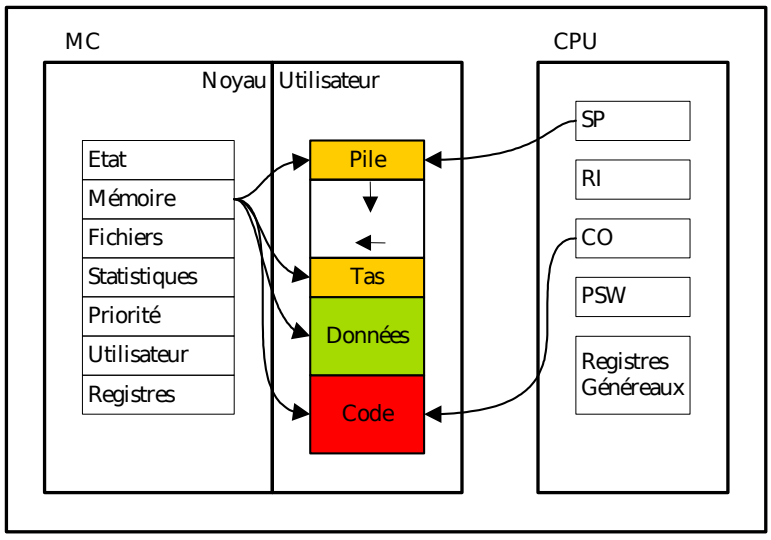
1. **Identité du processus** ; chaque processus possède deux noms pour son identification :
   - Un **nom externe** sous forme de chaîne de caractères fourni par l'usager (c'est le nom du fichier exécutable).
   - Un **nom interne** sous forme d'un entier fourni par le système. Toute référence au processus à l'intérieur du système se fait par le nom interne pour des raisons de facilité de manipulation.
2. **Etat du processus** ; l'état peut être nouveau, prêt, en exécution, en attente, arrêté, …etc.
3. **Contexte du processus** ; compteur ordinal, mot d'état, registres ;
4. **Informations sur le scheduling de l'UC ;** ces informations comprennent la priorité du processus, des pointeurs sur les files d'attente de scheduling, …etc.
5. **Informations sur la gestion mémoire** ; ces informations peuvent inclure les valeurs des registres base et limite, les tables de pages ou les tables de segments.
6. **Information sur l'état des E/S** ; l'information englobe la liste de périphériques d'E/S alloués à ce processus, une liste de fichiers ouverts, … etc.
7. **Informations de Comptabilisation ;** ces informations concernent l'utilisation des ressources par le processus pour facturation du travail effectué par la machine (chaque chose a un coût).



Figue 2.4 : Bloc de contrôle de processus (PCB)

Le système d'exploitation maintient dans une table appelée «**table des processus**» les informations sur tous les processus créés (une entrée par processus : Bloc de Contrôle de Processus PCB). Cette table permet au SE de localiser et gérer tous les processus.

Donc, un processus en cours d'exécution peut être schématisé comme suit (Figure 2.5) :



Figue 2.5 : Schéma général d'un processus en cours exécution

11.5 Etat d'un processus

Un processus prend un certain nombre d'états durant son exécution, déterminant sa situation dans le système vis-à-vis de ses ressources. Les trois principaux états d'un processus sont :

- **Prêt (Ready)** : le processus attend la libération du processeur pour s'exécuter.
- **Actif (Running)** : le processus est en exécution.
- **Bloqué (Waiting)** : le processus attend une ressource physique ou logique autre que le processeur pour s'exécuter (mémoire, fin d'E/S, … etc.).

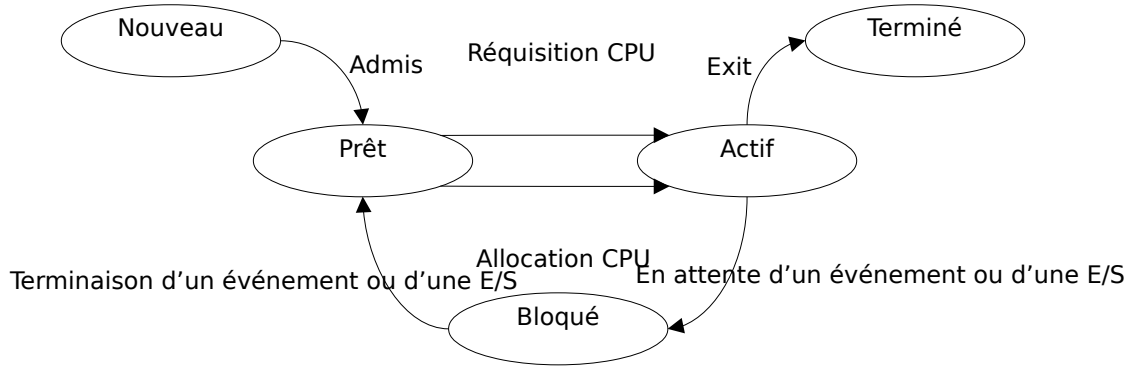


Figure 2.6 : Diagramme de transition des états d'un processus

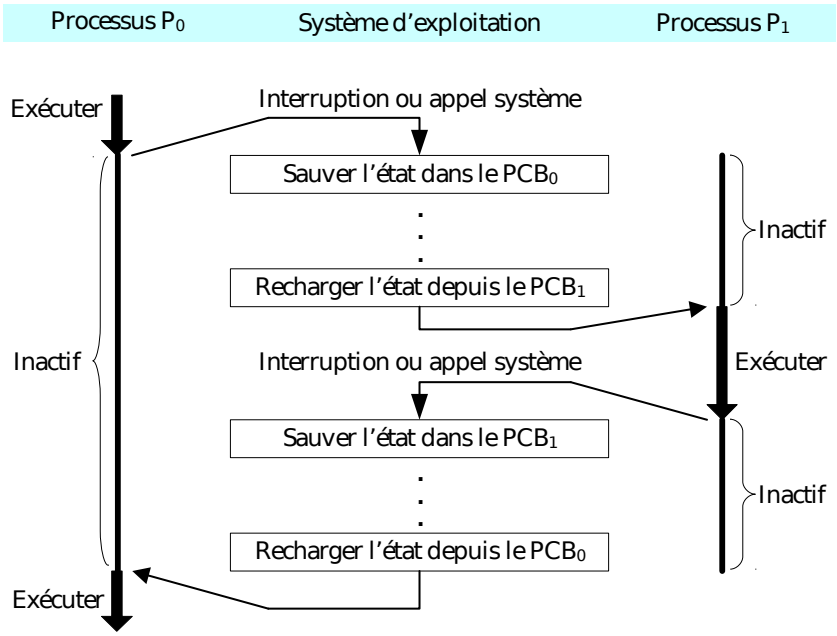
11.6 Multiprogrammation

Dans un système multiprogrammé, le processeur assure l'exécution de plusieurs processus en parallèle (pseudo-parallélisme).

Le passage dans l'exécution d'un processus à un autre nécessite une opération de sauvegarde du contexte du processus arrêté, et le chargement de celui du nouveau processus. Ceci s'appelle la **commutation du contexte** (**Context Switch**).

A. Mécanisme de commutation de contexte

La commutation de contexte consiste à changer les contenus des registres du processeur central par les informations de contexte du nouveau processus à exécuter.



La commutation du contexte se fait en deux phases :

- La **première phase** consiste à commuter le petit contexte (CO, PSW) par une instruction **indivisible**.
- La **deuxième phase** consiste quant à elle à commuter le grand contexte par celui du nouveau processus.

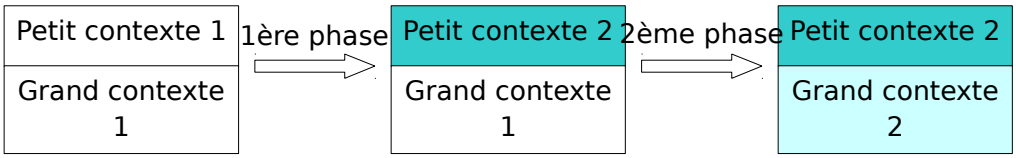


Figure 2.8 : Les phases d'une commutation de contexte

Remarque

La commutation du contexte est déclenchée suivant l'état d'un indicateur qui est consulté par le processeur à chaque point observable.

12 Les Systèmes d'Interruption

12.1 Problématique

Dans un ordinateur, ils coexistent deux types de programmes :

- Les **programmes usagers** qui font un calcul utile.
- Les **programmes du S.E.** qui font un travail de **superviseur** de tous les événements qui arrivent à la machine.

Ces deux types se partagent les ressources communes de la machine et plus particulièrement le processeur.

Lorsqu'un programme est en cours d'exécution, plusieurs événements peuvent arriver :

a. **Les événements synchrones** qui sont liés à l'exécution des programmes, comme :
   1. Division par zéro.
   2. Exécution d'une instruction inexistante ou interdite.
   3. Tentative d'accès à une zone protégée.
   4. Appel à une fonction du S.E.

b. **Les événements asynchrones** qui ne sont pas liés à l'exécution du programme en cours :
   1. Fin d'opération d'E/S.
   2. Signal d'horloge.

La supervision de ces deux types d'événements se fait par des contrôles continus sur l'arrivée de ceux-ci.

Question

Par quel mécanisme peut-on réduire le temps de supervision du S.E. ?

Solution

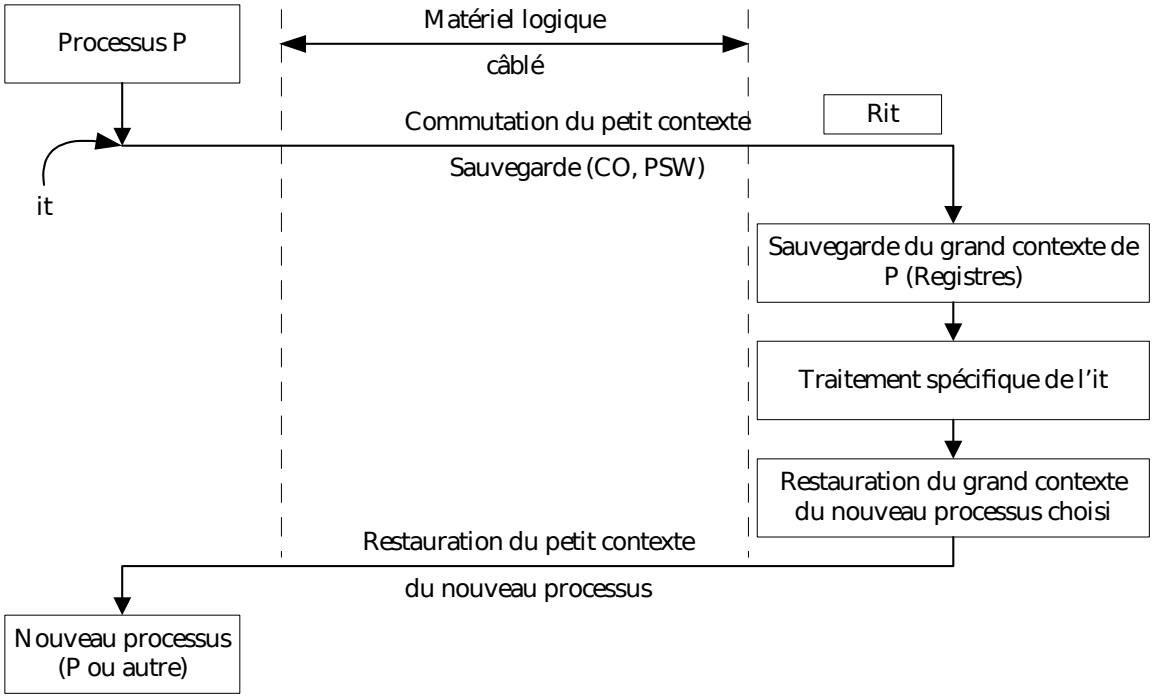
Au lieu que ça soit le processeur qui contrôle continuellement l'état d'une ressource, c'est plutôt à la ressource d'informer le processeur central sur son état au moment significatif. C'est le **principe des interruptions** de programmes.

12.2 Définition (Interruption)

Une **interruption** est une réponse à un événement qui interrompt l'exécution du programme en cours à un **point observable** (interruptible) du processeur central. Physiquement, l'interruption se traduit par un signal envoyé au processeur. Elle permet de forcer le processeur à **suspendre** l'exécution du programme en cours, et à déclencher l'exécution d'un programme prédéfini, spécifique à l'événement, appelé **routine d'interruption** (Rit).

12.3 Mécanismes de gestion des interruptions

A. Organigramme général



Remarque

Le programme repris après le traitement de l'interruption peut être le programme interrompu ou autre.

La mise en œuvre d'un système d'interruption nécessite la réponse aux problèmes techniques suivants :

1. Sous quelles conditions une interruption pourra-t-elle arriver ?
2. Comment le PC reconnaîtra-t-il l'arrivée d'une interruption, et par quoi répondra-t-il ?
3. Comment le PC arrivera-t-il à identifier la cause d'interruption ?
4. Comment le PC traitera-t-il les requêtes simulées d'interruption ?
5. Comment le PC réagira-t-il devant l'interruption d'une interruption ?

B. Conditions d'arrivée d'une interruption

Une interruption ne peut arriver au processeur que dans les conditions suivantes :

1. Le système d'interruption est **actif**.
2. Le PC est à un point observable (interruptible).
3. L'interruption est **armée**.
4. L'interruption est **démasquée**.
5. L'interruption est plus **prioritaire** que le programme en cours.

- **Système d'interruption actif**

  Dans certains cas, le processeur a besoin d'interdire toute interruption possible. Pour cela, il dispose d'un **mécanisme d'activation/désactivation globale des interruptions**.

Dans ces conditions, aucune interruption ne peut interrompre le PC, et toute it est retardée à la prochaine activation du système d'interruption.

- **L'interruption est armée**

  Une interruption désarmée ne peut interrompre le PC. Ceci se passe comme si la cause de l'it était supprimée. Toute demande d'interruption faite durant son désarmement est perdue.

  On utilise ce procédé quand on désire qu'un élément ne doive plus interrompre.

- **L'interruption est démasquée**

  Parfois, il est utile de protéger, contre certaines interruptions, l'exécution de certaines instructions (par exemple, les programmes d'it eux-mêmes). Une interruption masquée ne peut alors interrompre le PC, mais toute demande d'interruption faite durant le masquage est retardée (mémorisée) pour être traitée à la levée du masquage.

  Les informations concernant l'état masqué des interruptions figurent dans le mot d'état du processeur.

  On utilise le procédé de masquage pour définir des règles de priorité entre différentes causes d'interruption. Ainsi, les interruptions de même niveau de priorité ou d'un niveau plus bas peuvent être masquées, alors qu'une interruption de priorité supérieure est en cours d'exécution.

Remarques

1. Le masquage porte sur un niveau ou une cause d'interruption, contrairement à l'activation qui porte sur l'ensemble du système d'interruption.
2. Suivant les systèmes, ces procédés (masquage, aucunement, activation) peuvent exister totalement ou partiellement.

C. Types d'interruption

Les interruptions sont classées en deux grandes classes :

- Les interruptions **externes** ou **matérielles**.
- Les interruptions **internes** ou **logicielles**.

1. Interruptions externes

Ce sont les interruptions causées par des organes externes au processeur central, comme les horloges de temps, les périphériques d'E/S, …etc.

Ces interruptions **asynchrones** (c'est-à-dire, peuvent arriver à tout moment indépendamment de l'exécution du programme en cours) sont dues à :

- Périphérique prêt.
- Erreur durant l'E/S.
- Fin d'E/S.
- Ecoulement d'un délai de garde (horloge).
- Réinitialisation du système.
- …etc.

2. Interruptions internes

Ce sont des interruptions causées par l'exécution du programme à l'intérieur du PC. Ces interruptions sont synchrones et se divisent en deux sous-classes :

- Les **déroutements** (**trap** ou **exception**) qui sont dus à des erreurs lors de l'exécution d'un programme et en empêchent la poursuite de son exécution. Ces erreurs peuvent avoir diverses causes :
  - Tentative d'exécution d'une opération interdite ou invalide.
  - Violation d'accès à une zone mémoire protégée ou inexistante.
  - Division par zéro.
  - Débordement arithmétique.
  - …etc.

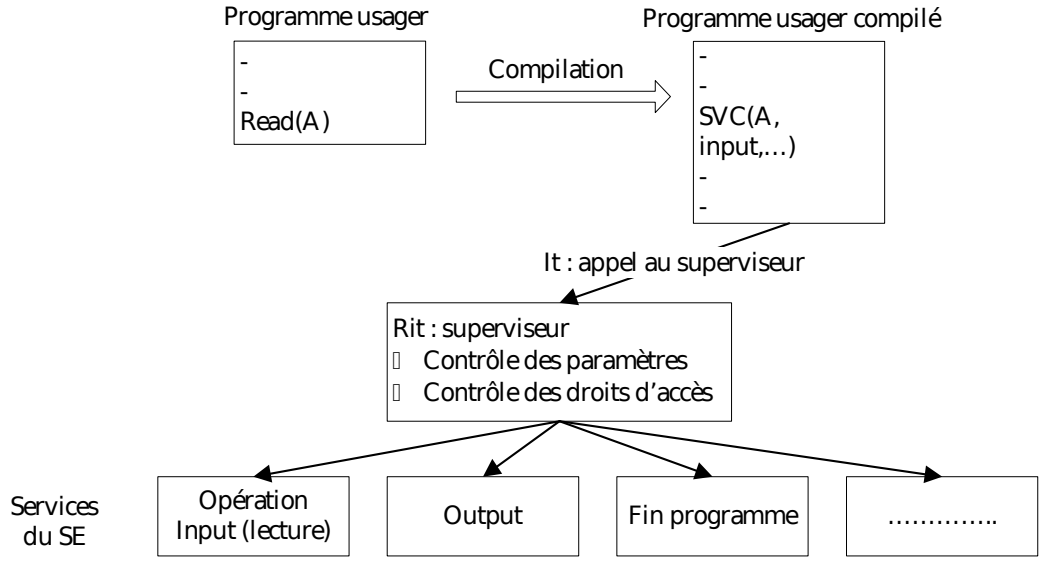
  Remarque

  Un déroutement ne peut être masqué ou retardé, mais il peut être supprimé (comme pour les déroutements liés aux opérations arithmétiques).

- Les **appels au superviseur** (**SuperVisor Call**, **SVC**) qui est une instruction permettant, à partir d'un programme utilisateur d'accéder à un service du S.E. (**Ex.** demande d'E/S, allocation dynamique de la mémoire, fin de programme, accès à un fichier, …etc.).

  Cette façon de procéder permet au système de :
  - Se protéger des usagers.
  - Vérifier les droits d'accès au service demandé.

  Exemple



L'appel au superviseur est en fait un appel de procédure système avec un passage de paramètres. Le service demandé par l'utilisateur sera choisi au moyen d'un paramètre supplémentaire désignant la cause de l'appel.

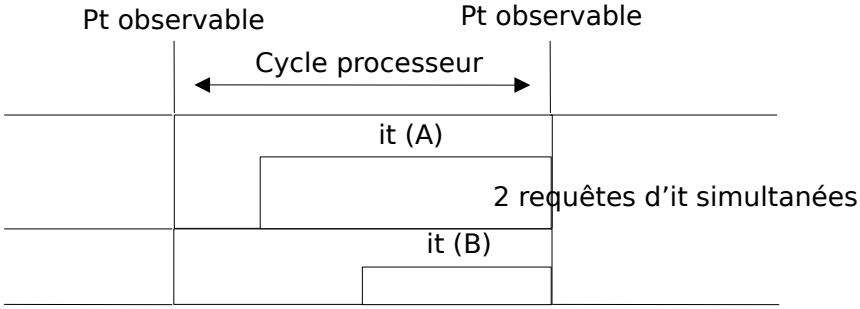
Toutefois, l'appel au superviseur d'un usager est traité comme interruption pour permettre une commutation du contexte usager par le contexte superviseur, car l'utilisation d'un service de superviseur nécessite un changement d'état (droits d'accès, mode d'exécution, priorité, …etc.).

Remarque

La distinction entre interruption, déroutement et appel au superviseur se base sur la fonction, mais le mécanisme est commun.

D. Requêtes simultanées d'interruption

Durant un cycle processeur, plusieurs demandes d'interruption peuvent arriver simultanément :



Le traitement des requêtes simultanées se fait suivant un certain ordre de priorité établi entre les niveaux d'interruption. La priorité peut être fixe ou modifiable.

E. Requêtes imbriquées

Pour des raisons de temps critiques, certaines demandes d'interruption doivent être prises en compte, même quand le processeur est en train de traiter une interruption, on parle alors d'interruptions imbriquées.

Du point de vue fonctionnement, l'interruption d'une interruption est traitée comme une interruption classique de programme. En réalité, le processeur ignore même qu'il s'agit d'interruptions imbriquées.

CHAPITRE III : GESTION DES E/S PHYSIQUES

13 Introduction

Au cours de son exécution, un programme interagit avec l'environnement extérieur. Cette interaction permet à l'utilisateur, d'une part, d'alimenter le programme avec les données à traiter et d'autre part, de récupérer le résultat du traitement. Pour ce faire, des **organes d'entrées/sorties** (appelés aussi **périphériques**) sont utilisés comme interface entre l'utilisateur et le système.

On appelle **Entrée/Sortie** (**Input/Output**) toute opération de transfert d'informations (données et programmes) entre l'ordinateur (processeur et

mémoire centrale) et les organes externes (périphériques de stockage et de communication).

14 Matériel

14.1 Les périphériques

Un périphérique (**Device**) est un appareil qui interagit avec l'UC et la mémoire. Certains périphériques sont branchés à l'intérieur de l'ordinateur (disques durs, …etc.) alors que d'autres sont branchés sur des interfaces externes de l'ordinateur (clavier, écrans, souris, imprimantes, …etc.).

Il existe deux grandes catégories de périphériques, les **périphériques blocs** (**Block Devices**) et les **périphériques caractères** (**Character Devices**).

- **Périphériques caractères :** Ils envoient ou reçoivent les données octets par octets. Parmi les périphériques caractères, on peut citer : le clavier, la souris, les imprimantes, les terminaux, …etc. Les données sont transmises les unes derrière les autres, on parle alors d'**accès séquentiel** (**Sequential Access**).
- **Périphériques blocs :** Ils acceptent les données par blocs de taille fixe, chaque bloc ayant une adresse propre. Parmi les périphériques blocs, on peut citer : les disques, la carte vidéo, …etc. Le grand avantage par rapport aux périphériques caractères est qu'il est possible d'aller lire ou écrire un bloc à tout moment, on parle alors d'**accès aléatoire** (**Random Access**)**.**

14.2 Les contrôleurs

Un contrôleur (**Controller**) est une unité spéciale, appelée aussi **module d'E/S** (**I/O module**) ou **coupleur**, qui sert d'**interface** entre le périphérique et l'UC. Par exemple, le module d'E/S servant d'interface entre l'UC et un disque dur sera appelé contrôleur de disque.

Les contrôleurs d'E/S ont plusieurs fonctions. En voici les principales:

- Lire ou écrire des données du périphérique.
- Lire ou écrire des données de l'UC/Mémoire. Cela implique du décodage d'adresses, de données et de lignes de contrôle. Certains modules d'E/S doivent générer des interruptions ou accéder directement à la mémoire.
- Contrôler le périphérique et lui faire exécuter des séquences de tâches.
- Tester le périphérique et détecter des erreurs.
- Mettre certaines données du périphérique ou de l'UC en mémoire tampon afin d'ajuster les vitesses de communication.

Un contrôleur dispose, pour chaque périphérique qu'il gère, de trois (03) types de registres :

- **Registres de données** (**Data Registers**) : destinés à contenir les informations échangées avec le périphérique. Ils peuvent être lus (entrée) ou écrits (sortie).
- **Registre d'état** (**State Register**) : qui permet de décrire l'état courant du coupleur (libre, en cours de transfert, erreur détectée,…).

- **Registre de contrôle** (**Control Register**) : qui sert à préciser au coupleur ce qu'il doit faire, et dans quelles conditions (vitesse, format des échanges,...).

Le contrôleur ou coupleur a la responsabilité de déplacer les données entre le(s) unité(s) périphérique(s) qu'il contrôle et sa mémoire tampon ou buffer. La taille de ce buffer varie d'un contrôleur à un autre, selon le périphérique contrôlé et son unité de transfert.

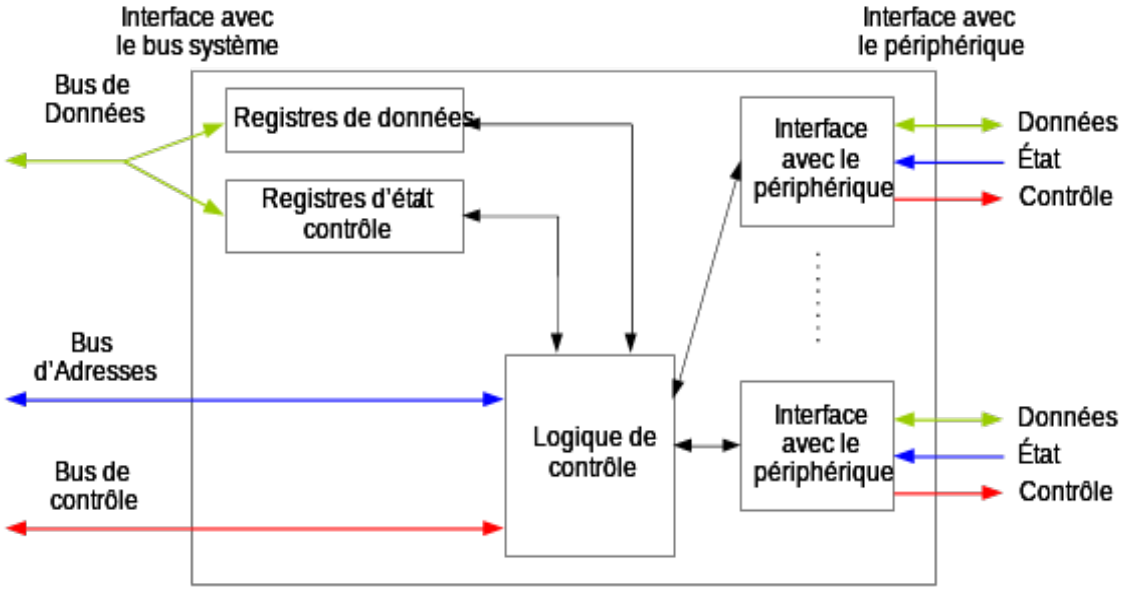


Figure 3.1 : Principaux composants d'un coupleur

Le coupleur possède aussi une logique de contrôle pour décoder l'adresse et les lignes de contrôle (ou pour faire du DMA), et une ou plusieurs interfaces avec un ou plusieurs périphériques (Voir Figure 3.1).

14.3 Les canaux

Sur les gros ordinateurs, des canaux d'E/S (**I/O Channels**) allègent le travail du processeur principal pour sa communication avec les contrôleurs (contrôle et synchronisation). Un canal d'E/S est un processeur spécialisé qui gère un ou plusieurs périphériques.

14.4 Les bus

Les contrôleurs d'E/S sont connectés sur des bus, reliés à d'autres bus par des contrôleurs de bus (souvent appelés interfaces ou ponts). Le processeur et la mémoire sont eux-mêmes sur des bus.

Chaque bus a ses propres caractéristiques. Les bus peuvent être fort différents. Néanmoins, tous les bus ont une largeur comprenant un nombre de lignes de données et d'adresses, une vitesse de communication, un type de connecteur et un protocole qui décrit la façon dont sont échangées les données sur le bus.

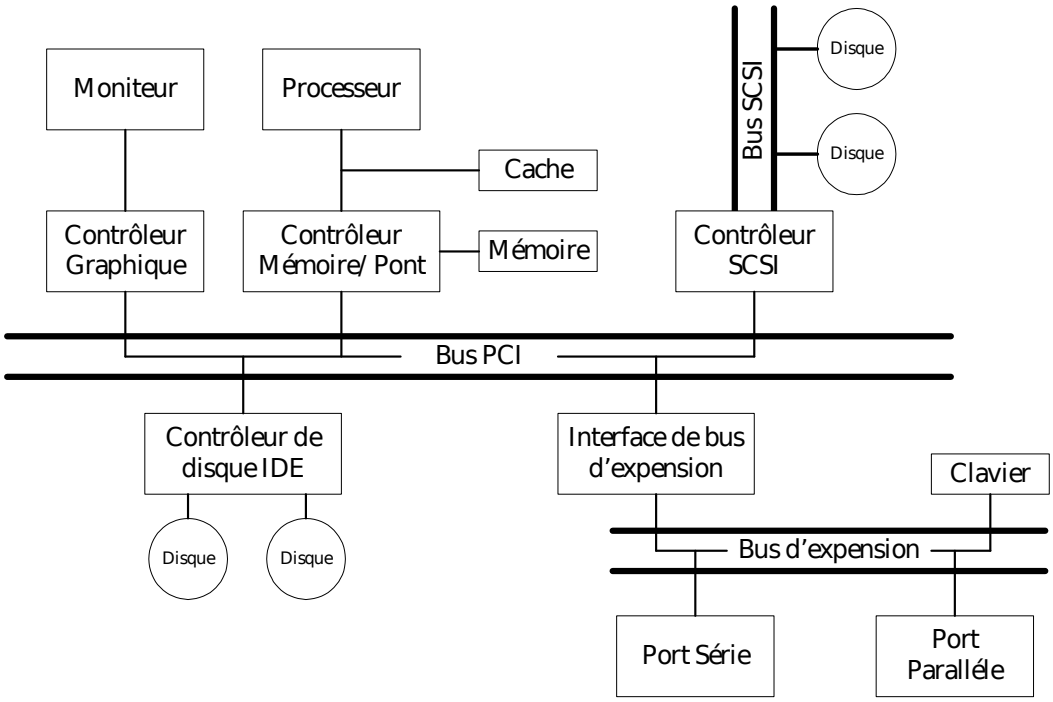


Figure 3.2 : Architecture typique des Bus d'un PC

15 Projection des E/S

La communication entre le processeur et le coupleur se fait par l'intermédiaire des **registres du coupleur**. La **désignation** de ces registres fait appel à l'une des deux techniques suivantes :

1. **Mappage sur les ports** (**Port-Mapped I/O, PMIO**) : Dans le cas où les périphériques et la mémoire centrale ont chacun leur propre espace d'adressage. L'accès s'effectue via des **instructions spécialisées** (**IN** et **OUT** en assembleur) qui permettent d'accéder au périphérique.
2. **Mappage en mémoire** (**Memory-Mapped I/O, MMIO**) : Dans le cas où les périphériques et la mémoire centrale partagent le même espace d'adressage. Les registres peuvent alors être lus ou modifiés par des instructions ordinaires. Dans ce cas, les E/S sont dites **couplées** ou **mappées en mémoire**.

16 Modes de pilotage d'une E/S physique

Le pilotage d'une unité périphérique par le processeur central nécessite une synchronisation entre les actions du processeur (où s'exécute le pilote) et l'unité périphérique pilotée.

Cette synchronisation est nécessaire au processeur (pilote) pour connaître si le périphérique est prêt, si l'opération d'E/S est terminée, …etc.

16.1 E/S physique directe

C'est une E/S physique **contrôlée par le processeur** central d'où l'appellation directe. Dans ce cas, deux modes peuvent être utilisés :

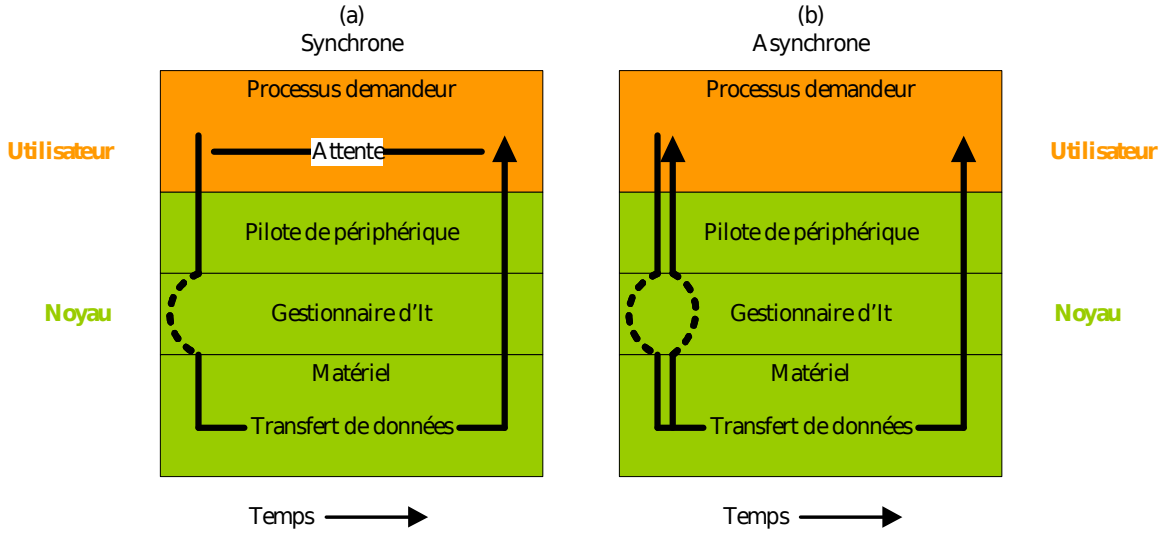


Figure 3.3 : Modes de pilotage des E/S directes

- **Mode synchrone**

  Dans ce mode, le processeur (pilote) est mobilisé à suivre l'opération d'E/S pendant toute la durée du transfert (**polling**). La fin du transfert est détectée par la consultation d'un indicateur spécifique au coupleur ou au périphérique.

  Pour démarrer une opération d'E/S, le processeur (pilote) charge les informations nécessaires dans les registres appropriés du contrôleur de périphérique. Celui-ci examine à son tour le contenu de ces registres pour déterminer l'action à effectuer (lecture, écriture, ...etc.) et lance le transfert.

  La forme générale du pilote est la suivante :

  Pilote synchrone d'E/S

  - Initialise le transfert (sens du transfert : lecture, écriture, adresse du périphérique, ...etc.).
  - Vérifie la disponibilité du périphérique.
  - Lance le transfert.
  - Reste en attente (active) jusqu'à la fin du transfert.

  Fin.

  Inconvénients

  Le CPU exécute une boucle d'**attente active** (**busy loop**), à la place de laquelle il pourrait faire des calculs pour le compte d'autres programmes → Perte de temps CPU.

- **Mode asynchrone**

  Dans ce mode de pilotage, le processeur est libéré du contrôle de fin de transfert. Une **interruption** est générée par le coupleur du périphérique avertissant ainsi le processeur (pilote) de la fin du transfert.

Durant l'opération du transfert, le processeur peut exécuter un autre programme.

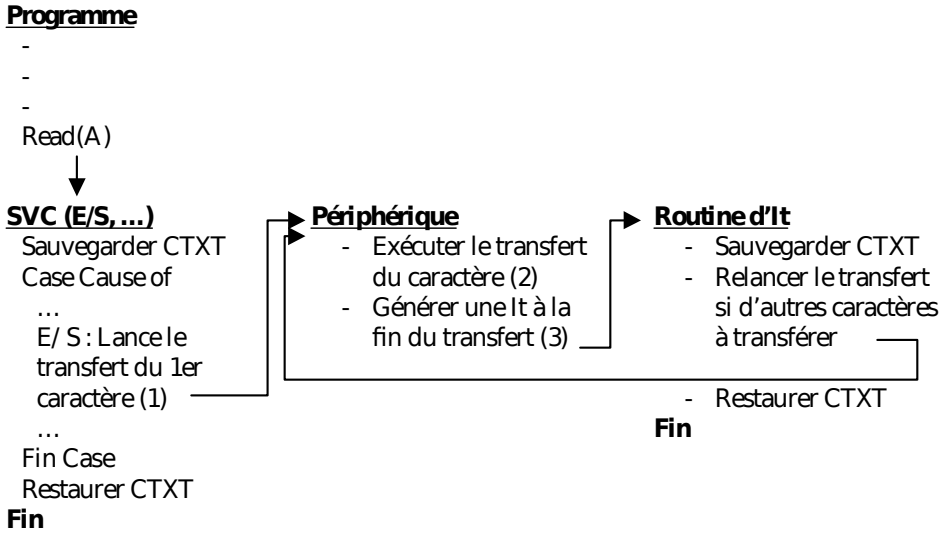


Figure 3.4 : Schéma général d'exécution d'une E/S asynchrone

Le rôle du programme usager est restreint à lancer le transfert du 1er caractère, le transfert des caractères suivants est lancé par la routine d'interruption, elle-même appelée à la fin du transfert de chaque caractère.

Dans ce mode, le pilote gère l'interface coupleur du périphérique, traite les interruptions émises, détecte et traite les cas d'erreurs.

Avantage

Utilisation plus rationnelle du CPU. En effet, durant le transfert des caractères, le processeur peut exécuter d'autres traitements (programmes).

Inconvénient

Perte de temps occasionnée par l'exécution des routines d'interruption, et des changements de contextes et programmes.

16.2 E/S physique indirecte

L'E/S physique est indirecte lorsque ce n'est plus le processeur qui se charge du suivi du déroulement de cette E/S. Ceci se fait soit en utilisant un **contrôleur d'accès direct à la mémoire** (DMA) ou en utilisant des processeurs spécialisés dits **canaux**.

A. Le contrôleur DMA (Direct Memory Access)

Pour éviter à l'unité centrale d'intervenir à chaque transfert de caractère, les ordinateurs utilisent un composant supplémentaire appelé **contrôleur à accès direct à la mémoire**, ou **DMA** (**D**irect **M**emory **A**ccess). Selon les architectures, le DMA est propre à un périphérique, au disque par exemple, ou partagé par plusieurs périphériques. Il peut être connecté entre un contrôleur de périphériques et le bus mémoire (Voir Figure 3.5), permettant ainsi aux périphériques d'accéder à la mémoire sans passer par le CPU.

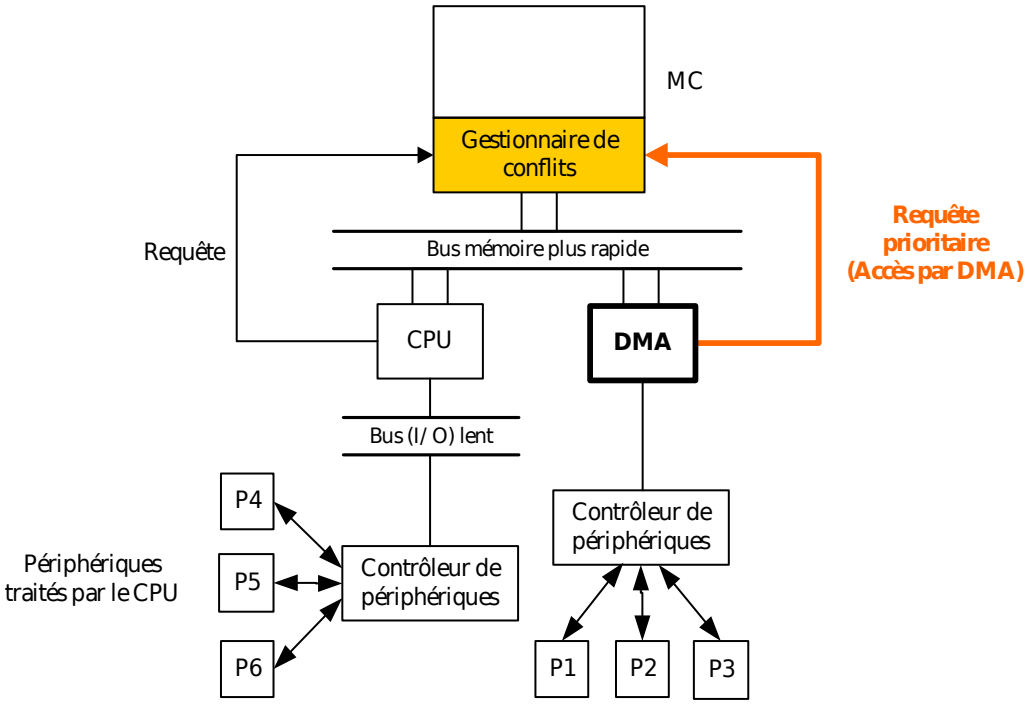


Figure 3.5 : Architecture typique avec DMA

Donc, l'utilisation du DMA décharge l'UC d'une partie importante du travail de contrôle et d'exécution des E/S. Le DMA se charge entièrement du transfert d'un bloc de données. L'UC initialise l'échange en communiquant au contrôleur DMA :

- L'identification du périphérique concerné.
- Le sens du transfert.
- L'adresse en MC du 1er mot à transférer.
- Le nombre de mots à transférer.

La programmation d'un coupleur DMA se fait, comme pour une interface par l'écriture de données dans des "registres". Pour cela, le DMA dispose (Voir Figure 3.6) d' :

- Un **registre d'adresse** (**Adress Register**) qui va contenir l'adresse où les données doivent être placées ou lues en mémoire.
- Un **registre de données** (**Data Register**).
- Un **compteur** (**Data Count**) qui compte le nombre de données échangées.
- Un dispositif de commande capable d'exécuter le transfert.

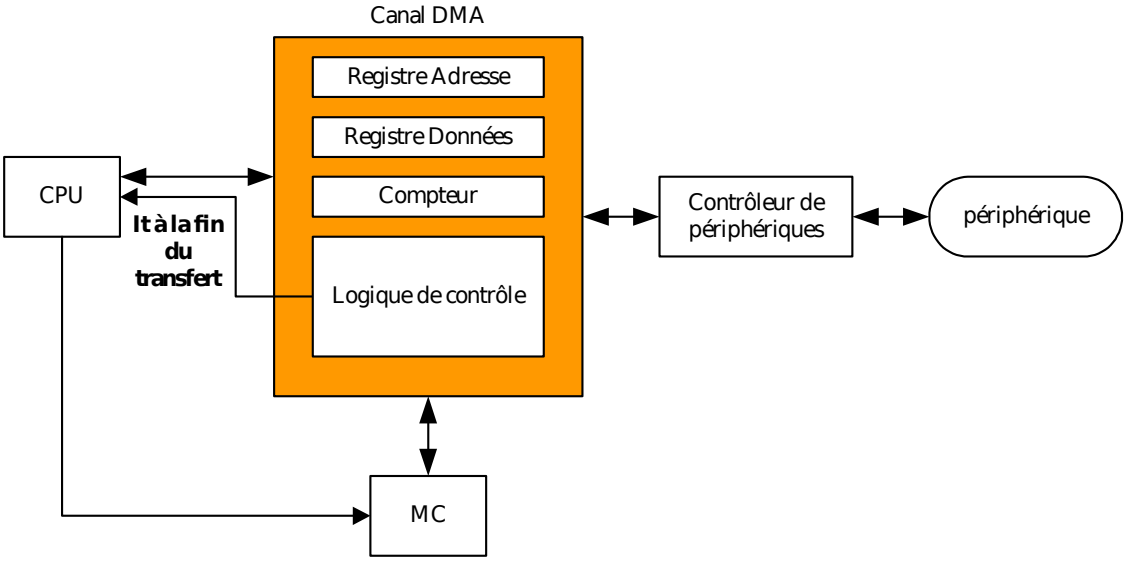


Figure 3.6 : Registres d'un contrôleur DMA

1. Processus de réalisation d'un transfert DMA

Le principe de fonctionnement est simple. L'unité centrale transmet au DMA les paramètres d'une commande à exécuter (en général, il s'agit du transfert d'une suite de caractères entre un périphérique et un tampon en mémoire) ;

Le DMA commande alors en totale autonomie l'ensemble du transfert des caractères ;

En fin de commande, le DMA envoie une interruption à l'unité centrale pour lui indiquer que le transfert est terminé et que le DMA est à nouveau disponible pour recevoir une commande.

Pratiquement, le DMA contient, pour chaque commande en cours d'exécution, une série de registres contenant l'ordre à exécuter, une adresse en mémoire centrale et un compteur d'octets. A chaque transfert de caractère, l'adresse est augmentée de 1 et le compteur diminué de 1. L'interruption de fin de transfert est envoyée au processeur lorsque le compte d'octets atteint la valeur 0.

La figure ci-dessous (Figure 3.7), résume les principales étapes d'un transfert DMA :

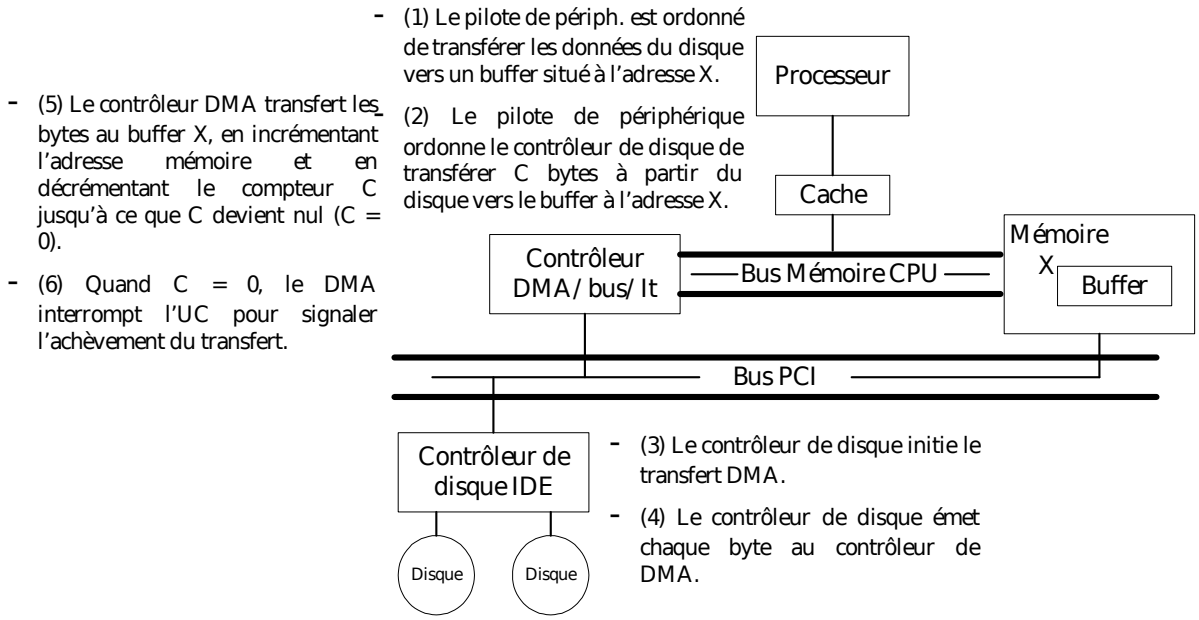


Figure 3.7 : Processus d'un transfert DMA

2. DMA et vol de cycles mémoire

Un mécanisme DMA nécessite :

- La mise en place d'un chemin de passage entre le canal et la MC, matérialisé par un bus.
- Un gestionnaire des **conflits d'accès** à la MC entre le canal et le CPU.

Pour cela, une technique de **vols de cycles** (**Cycle Stealing**) est utilisée (Voir Figure 3.8).

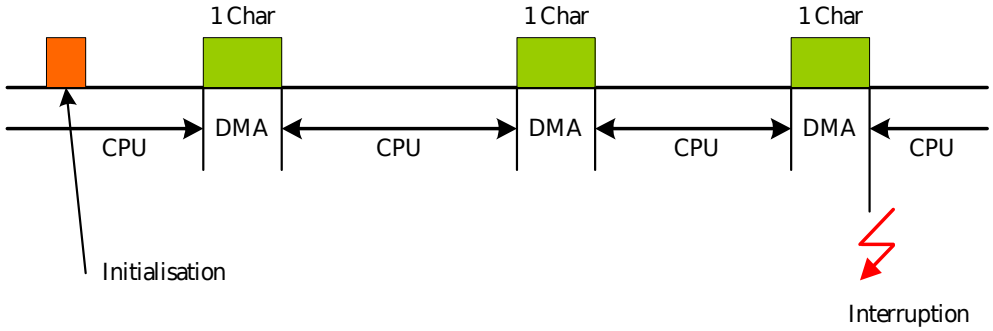


Figure 3.8 : Principe de vol de cycles

Elle consiste à freiner le fonctionnement du processeur en prenant des cycles mémoires pendant lesquels le contrôleur DMA transférera de l'information. En effet, les contrôleurs DMA sont prioritaires sur le processeur central pour l'accès à la mémoire, car ils doivent réagir rapidement à des événements externes.

17 Traitement d'E/S simultanées

Lorsqu'une requête d'E/S arrive en mode synchrone, une requête est en cours d'exécution à la fois, puisque le CPU attend la fin d'E/S.

Néanmoins, en mode asynchrone et mode DMA, l'E/S demandée est lancée. Puis, le contrôle est rendu immédiatement à un programme utilisateur, qui peut formuler de nouvelles requêtes d'E/S.

A cet effet, le S.E maintient une table contenant une entrée pour chaque périphérique d'E/S ; c'est la **table d'état des périphériques** (**Device-Status Table**).

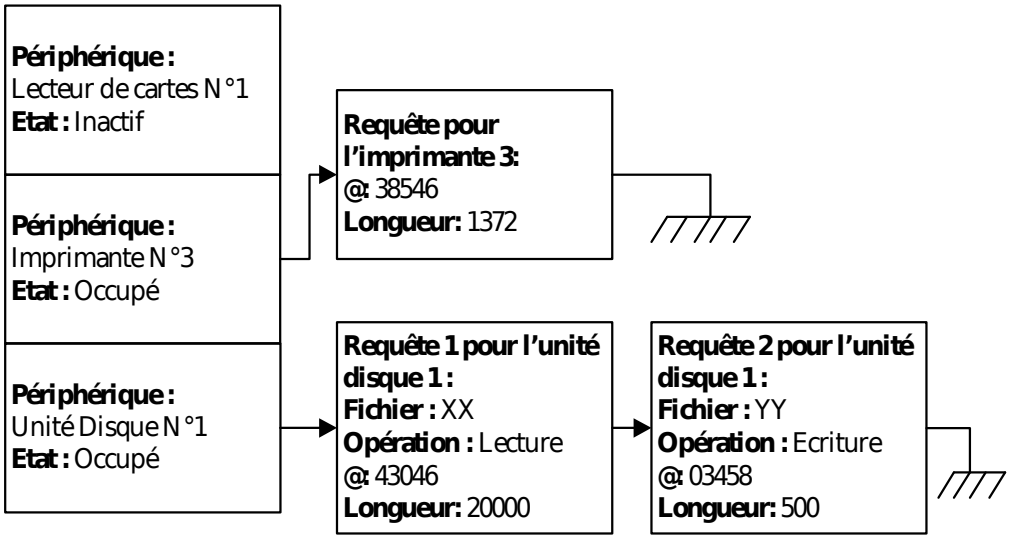


Figure 3.9 : Table d'état des périphériques

Chaque entrée de la table (Voir Figure 3.9) indique le type du périphérique, son adresse et son état (inactif ou occupé), ainsi qu'une liste des requêtes formulées d'E/S. Cette liste contient, pour chaque requête, le type d'opération, l'adresse de données et la longueur des données, …etc.

18 E/S Tamponnées

Bien que le rôle final d'une E/S soit l'échange de caractères entre un périphérique et une zone de mémoire d'un utilisateur, les logiciels d'E/S utilisent souvent une zone intermédiaire, appelée **tampon d'E/S** (**I/O Buffer**), dans la **mémoire du système**. L'utilisation des tampons est justifiée par l'**amélioration des performances** d'un périphérique.

Exemples

- Dans un échange avec un disque magnétique, l'essentiel du temps pris par l'échange vient du positionnement du bras et du délai rotationnel. Il est alors classique de ranger dans un tampon le contenu de toute une piste.
- La lecture de caractères au clavier peut se faire par anticipation ; c'est à dire que l'utilisateur peut frapper des caractères avant que le programme ait envoyé un ordre d'entrée. Les caractères frappés sont stockés dans le tampon et seront plus tard transférés dans une zone utilisateur.
- Dans un système en temps partagé, l'espace mémoire d'un utilisateur peut être vidé de la mémoire principale. C'est en particulier le cas lorsqu'un processus est bloqué en attente de la fin d'une E/S lente. Le recours à un tampon système est alors obligatoire.

19 Couches Logicielles d'E/S

Le système d'exploitation s'occupe de gérer les E/S à l'aide de quatre (04) niveaux de logiciel (Voir Figure 3.10), soit :

- **Logiciel** faisant partie de l'**espace utilisateur** (**User-Space Software**). Il représente les **procédures standards** (**programmes de bibliothèque**), appelées à partir des programmes pour formuler une requête d'E/S au superviseur et mettant en œuvre des fonctions supplémentaires (**Ex.** formatage des E/S, gestion des **spools**, ...etc.).
- **Logiciel indépendant du matériel** (**Device-independent Software**). Le rôle principal de cette couche est de donner une interface uniforme (commune à toutes les E/S) au logiciel des utilisateurs. Les principales fonctionnalités offertes par cette couche sont :
  - Adressage des périphériques par leur nom,
  - Protection des périphériques,
  - Mise en mémoire tampon de données,
  - Signalisation d'erreurs (erreurs de programmation comme écrire sur un disque inexistant, erreurs qui n'ont pu être résolues par les pilotes, ...),
  - Allocation et libération des périphériques dédiés (i.e. non partageables, en gérant une file d'attente par exemple...),
  - Fourniture d'une taille de bloc indépendante du périphérique.
- Un programme d'E/S appelé **pilote** (**Driver** ou **Handler**) commandant le fonctionnement élémentaire de chaque unité périphérique. Le pilote de périphériques est la seule partie logicielle à connaître toutes les spécificités du matériel. Le handler gère directement l'interface du coupleur du périphérique, traite les interruptions émises par celui-ci, détecte et traite les cas d'erreurs.

  Il est généralement **invisible** aux utilisateurs du système. Le handler contient donc les primitives permettant de commander le périphérique. Ce driver est constitué de deux procédures, quasiment indépendantes : une **procédure traitant l'initialisation d'un transfert** et une **procédure de traitement de l'interruption** associée à une fin de transfert.
- Le **gestionnaire d'interruptions** (**Interrupt Handler**) dont le rôle est d'informer le pilote associé au périphérique de la fin de l'E/S.

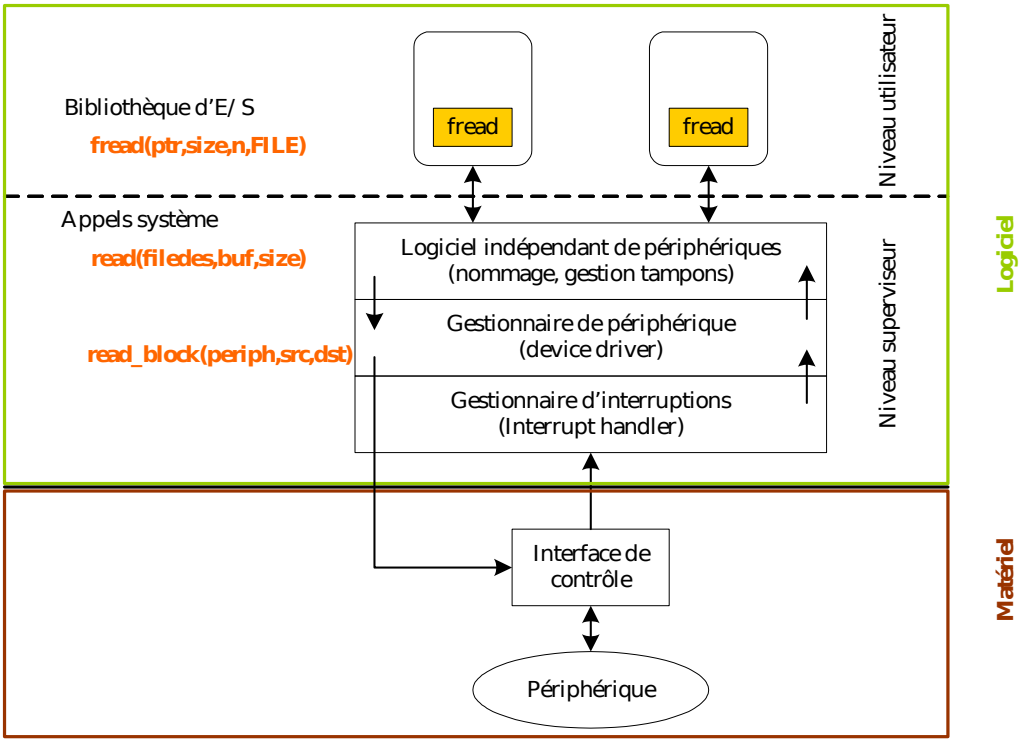


Figure 3.10 : Structure en couches d'un système d'E/S

CHAPITRE VI : INTERFACE DE SYSTÈME DE FICHIERS (UNIX)

1 Définitions

Un **fichier** (**File**) désigne un ensemble d'informations stockées sur un support de stockage permanant, en vue de leur conservation et de leur utilisation dans un système informatique. Chaque fichier est identifié par un nom auquel on associe un **emplacement** sur le disque (une référence) et possède un ensemble de propriétés : ses **attributs**. Il représente la plus petite entité logique de stockage sur un disque.

Le **système de fichiers** (**File System**) est la partie du système d'exploitation qui se charge de gérer les fichiers. La gestion consiste en la création (identification, allocation d'espace sur disque), la suppression, les accès en lecture et en écriture, le partage de fichiers et leur protection en contrôlant les accès.

2 Types de Fichiers

On distingue différents types de fichiers :

19.1 Fichiers ordinaires (Regular Files)

Les fichiers ordinaires (appelés aussi fichiers **réguliers**) peuvent contenir du texte, du code exécutable ou tout type d'information (binaire, c'est à dire non texte). Unix n'impose **aucune structure** interne au contenu des fichiers ; ils sont considérés comme une **suite d'octets**.

19.2 Fichiers répertoires (Directory Files)

Un répertoire (**directory)** peut être vu comme un fichier spécial contenant des liens vers d'autres répertoires ou fichiers. Cela permet de regrouper dans un même répertoire, les fichiers ayant des caractéristiques communes (même propriétaire, les programmes, . . .) et de hiérarchiser le système de fichier.

19.3 Fichiers spéciaux (Special Files)

Les fichiers spéciaux correspondent à des ressources ; ils sont associés à des dispositifs d'entrées/sorties physiques. Ils n'ont pas de taille. Ils sont traités par le système comme des fichiers ordinaires, mais les opérations de lecture/écriture sur ces fichiers activent les dispositifs physiques associés. On distingue :

- les **fichiers spéciaux mode caractère** (**Character Devices**) qui permettent de modéliser les périphériques d'E/S série tels que les terminaux, les imprimantes et les modems.
- les **fichiers spéciaux mode bloc** (**Block Devices**) qui permettent de modéliser les disques.

19.4 Autres types de fichiers

- Les **liens symboliques** (**Symbolic Links**) qui représentent des pointeurs indirects ou des raccourcis vers des fichiers. Un lien symbolique est implémenté comme un fichier contenant un chemin d'accès.

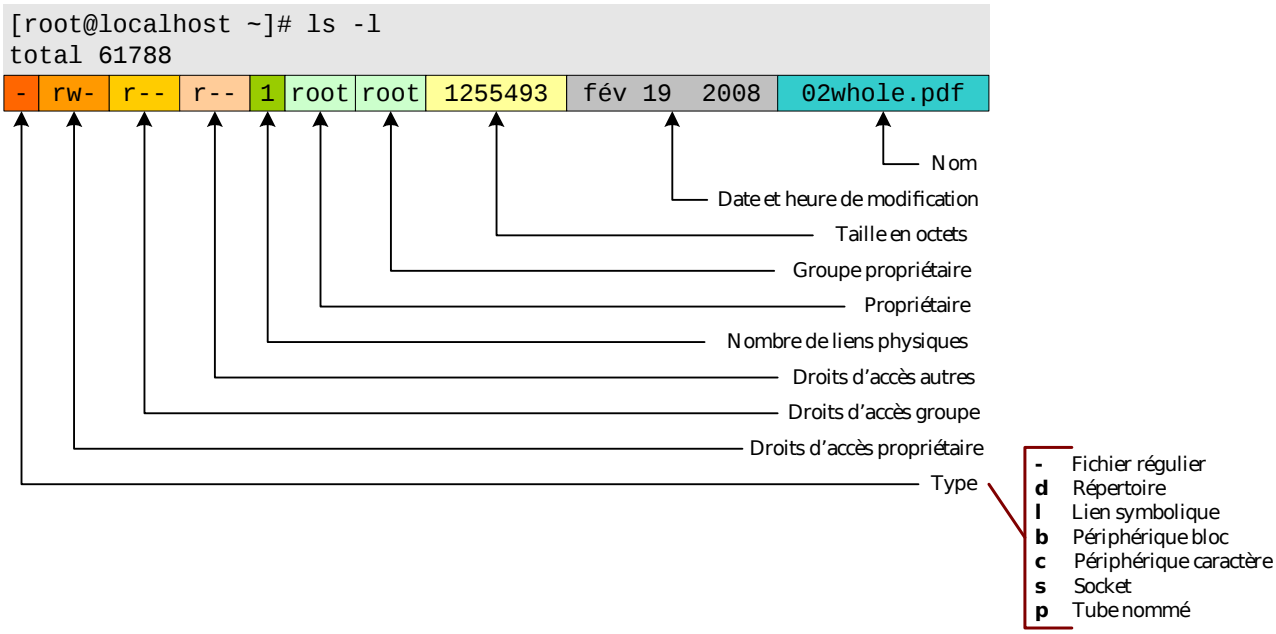
- Les **tubes nommés** (**Named Pipe**, **FIFO**) permettant la communication entre processus.
- Les **sockets** internes qui sont destinées à la communication bidirectionnelle entre processus.

3 Attributs d'un Fichier

Chaque fichier possède un ensemble d'attributs (appelés aussi **méta-données**), parmi lesquels :

- Un propriétaire (**Owner**).
- Un groupe (**Group**).
- Une série de droits d'accès (**Access Permissions**) : lecture (**R, Read**), écriture (**W, Write**), exécution (**X, Execute**). Ces droits sont définis pour l'utilisateur, le groupe, et le reste du monde (**Other**).
- Type de fichier : fichier régulier, répertoire, périphérique caractère, périphérique bloc, tube, etc.
- Dates de création/modification/dernier accès.

L'option **–l** de la commande **ls** (**l**i**s**t) permet de lister les attributs des fichiers et répertoires.



Ces méta-données sont traditionnellement stockées dans une structure de données appelée **i-noeud** (**inode ; i**ndex **node**). Chaque inode possède un **numéro** (**unique**). La commande **stat** permet d'afficher l'intégralité du contenu de l'inode et l'option **–i** de la commande **ls** permet d'afficher le numéro d'inode.

4 Lien Physique vs. Lien Symbolique

Chaque fichier (ordinaire ou répertoire) est référencé par son nom. Un lien est une entrée d'un répertoire qui référence un fichier qui se trouve dans un autre répertoire. Il existe deux sortes de liens : **physiques** et **symboliques**.

Un **lien physique** (**Hard Link**) correspond à l'ajout d'un nouveau nom pour le même fichier. Les liens physiques ne sont autorisés que pour les fichiers de données (et donc interdits pour les répertoires ou les fichiers spéciaux) et ne peuvent exister que dans le **même** système de fichiers que le fichier lié.

Un **lien symbolique** (**Soft Link**) est un fichier spécial contenant le chemin du fichier ou du répertoire lié. Il peut se trouver n'importe où dans la hiérarchie.

La création des liens physiques ou symboliques se fait à l'aide de la commande **ln**. La commande :

```
$ ln nomfich nomlien
```

crée un lien **physique** appelé **nomlien** sur le fichier **nomfich**. Par contre la commande :

```
$ ln -s nomfich nomlien
```

crée un lien **symbolique** appelé **nomlien** sur le fichier **nomfich**.

5 Organisation du Système de Fichiers

Le système Unix organise son information sous la forme d'une **arborescence** ; les **feuilles** étant les **fichiers** (**Files**) et les **nœuds** des **répertoires** (**Directories**). Cette organisation en cascade de répertoires et sous-répertoires commence par un **répertoire racine**, noté **/**. Chaque répertoire peut contenir des fichiers et des **sous-répertoires**.

Le tableau ci-dessous illustre quelques répertoires types d'un système de fichiers UNIX :

Répertoire	**Sous-répertoires**	**Contenu**
/bin		Binaires de base. (**BIN**aries)
/sbin		Binaires système. (**S**ystem **BIN**aries)
/dev		Fichiers spéciaux de périphériques. (**DEV**ices)
/etc	**/etc/rc.d** **/etc/init.d**	Fichiers et répertoires de configuration du système. Sous-répertoire de démarrage des services sous UNIX.
/lib		Bibliothèques standards. (**LIB**raries)
/mnt		Répertoires de périphériques amovibles.
/proc		Répertoire dédié aux processus.
/tmp		Fichiers temporaires. Ce répertoire doit être nettoyé régulièrement.
/usr	**/usr/include** **/** **usr/share/man**	Sous-répertoire des fichiers d'entête. Sous-répertoire des manuels UNIX.
/var		Fichiers de taille variable, tels que les fichiers journaux (logs), les fichiers du spouleur d'impression ou bien les mails en attente.

6 Déplacement dans le Système de Fichiers

Pour accéder à un fichier, on doit préciser son **chemin d'accès** (**Pathname**) en indiquant les noms des différents sous-répertoires qui mènent à ce fichier, séparés par des **/**. On distingue deux types de chemins :

- **Chemins absolus** (**Absolute Paths**) qui spécifient la suite des répertoires à traverser en partant de la **racine**. Un chemin absolu **doit** forcément **commencer** par un **slash** (i.e. **/**).

Exemple

/home/warda/exo.c : chemin d'accès absolu au fichier exo.c se trouvant dans le répertoire warda, lui-même dans le répertoire home de la racine.

- **Chemins relatifs** (**Relative Paths**) qui partent d'un certain **répertoire de référence** qui représente généralement le **répertoire de travail** (**Working Directory**) de l'utilisateur. Le répertoire de travail est le répertoire dans lequel l'utilisateur se trouve actuellement.

  Exemple

  Si on se trouve dans le répertoire /home/warda, le fichier de nom absolu /home/warda/exo.c peut être désigné simplement par exo.c.

  En ce qui concerne les chemins relatifs, un certain nombre de raccourcis sont utilisés:

  - **.** qui désigne le répertoire courant,
  - **..** qui désigne le répertoire parent du répertoire courant,
  - **~/** qui désigne le **répertoire personnel** (**Home Directory**) de l'utilisateur. Ce répertoire représente le répertoire de travail par défaut dans lequel l'utilisateur sera positionné à l'ouverture de sa session.

La recherche des fichiers correspondant aux **exécutables** est simplifiée par l'utilisation de la **variable d'environnement PATH**. La variable PATH stocke la liste des chemins (séparés par le caractère **:**) qui doivent être cherchés afin d'exécuter un programme (fichier exécutable) sans faire référence à un chemin absolu ou relatif.

Le tableau ci-après liste quelques commandes communes pour le déplacement dans le système de fichiers :

Utiliser …	**Pour …**
ls **ls** path	Afficher la liste des fichiers (et donc des répertoires) non cachés d'un répertoire. (**l**i**s**t)
ls –a	Afficher tous les fichiers, y compris les fichiers cachés.
ls –aR	Afficher tous les fichiers, y compris les fichiers cachés et tous les fichiers dans tous les sous-répertoires.
cd /path	Changer le répertoire de travail courant par le répertoire indiqué par le chemin absolu /path. (**c**hange **d**irectory)
cd path	Descendre dans la hiérarchie, à partir du répertoire courant, vers le répertoire indiqué par le chemin relatif path.
cd ..	Monter d'un niveau dans la hiérarchie à partir du répertoire courant.
cd **cd ~**	Retourner au répertoire de connexion (home directory) de l'utilisateur.
pwd	Afficher le nom du répertoire courant. (**p**rint **w**orking **d**irectory)

7 Descripteur de Fichier

Les fichiers ouverts par un processus sont manipulés à travers des descripteurs de fichiers. Un descripteur de fichier (**File Descriptor**) est un **entier** indexant une entrée de la **table des descripteurs de fichier** (**File Descriptors Table**). Il existe une table des descripteurs de fichiers rattachée à chaque processus. Le nombre d'entrées de cette table, correspondant au nombre

maximum de fichiers que peut ouvrir simultanément un processus, est donné par la pseudo-constante **NOFILE** définie dans le fichier en-tête **<sys/param.h>**.

Un descripteur de fichier représente un pointeur vers la **table des fichiers ouverts** (**Open Files Table**) dans le système.

Les trois (03) premières entrées de la table des descripteurs de fichier dans chaque processus sont automatiquement allouées et réservées dès la création, pour les fichiers suivants :

- **0** : Entrée standard (**St**andar**d In**put) /dev/stdin,
- **1** : Sortie standard (**St**andard **Out**put) /dev/stdout,
- **2** : Sortie standard pour les messages d'erreur (**St**andard **Err**or) /dev/stderr.

Chaque entrée correspond à une structure de donnée contenant :

- des informations sur le fichier associé,
- le mode d'ouverture (lecture, écriture),
- la position courante dans le fichier (offset).

Les droits d'accès sont vérifiés lors de la création du descripteur.

8 Opérations de Base sur les Répertoires (Voir Manuel TP)

Création d'un répertoire

Suppression d'un répertoire (vide)

Lister le contenu d'un répertoire

Changement de nom / emplacement

Rechercher un fichier

Parcourir les sous-répertoires

9 Opérations de Base sur les Fichiers (Voir Manuel TP)

Manipulation sur les fichiers

• Copier ou supprimer un fichier

• Déplacer ou renommer un fichier

• Afficher le contenu d'un fichier texte

10 Fichiers et Permissions

19.5 Principe

Sous UNIX, pour un fichier ou répertoire, on distingue trois (03) catégories d'utilisateurs : le **propriétaire** (**u** : user), les **membres du groupe** (**g** : group), et les **autres utilisateurs** (**o** : others). Pour chaque catégorie, il est possible d'attribuer des droits de **lecture** (**r** : read), d'**écriture** (**w** : write) ou d'**exécution** (**x** : execute). L'interprétation de ces droits varie selon qu'il s'agisse d'un fichier ou d'un répertoire.

- Pour les fichiers, l'interprétation est la suivante :
  - **r** : l'utilisateur peut lire le contenu du fichier.

- **w** : l'utilisateur peut modifier le contenu du fichier.
- **x** : l'utilisateur peut exécuter le fichier.

- Pour les répertoires, l'interprétation est comme suit :
  - **r** : l'utilisateur peut lister le contenu du répertoire.
  - **w** : l'utilisateur peut créer, renommer ou supprimer des fichiers dans le répertoire.
  - **x** : l'utilisateur peut accéder au répertoire et travailler avec son contenu.

19.6 Changer les permissions

Les permissions sur fichiers et répertoires ne peuvent être modifiées que par leurs **propriétaires**, ou par le **super-utilisateur** (**root**). La commande **chmod** (**ch**ange file **mod**es) permet de modifier ces permissions (voir manuel Linux).

On peut utiliser soit la représentation **symbolique** (r, w, x), ou **octale** pour référencer une permission. La figure ci-dessous (voir Figure 4.1) illustre comment les permissions sont affichées et comment elles peuvent être référencées.

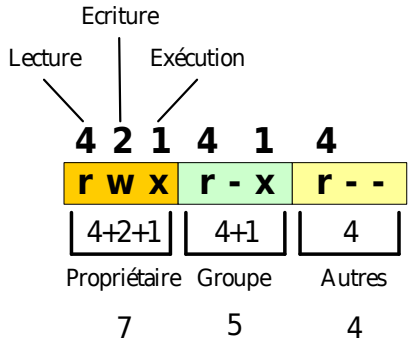


Figure 4.1 : Représentation des droits d'accès

La figure ci-dessous (Figure 4.2) résume la représentation symbolique des permissions en utilisant la commande chmod :

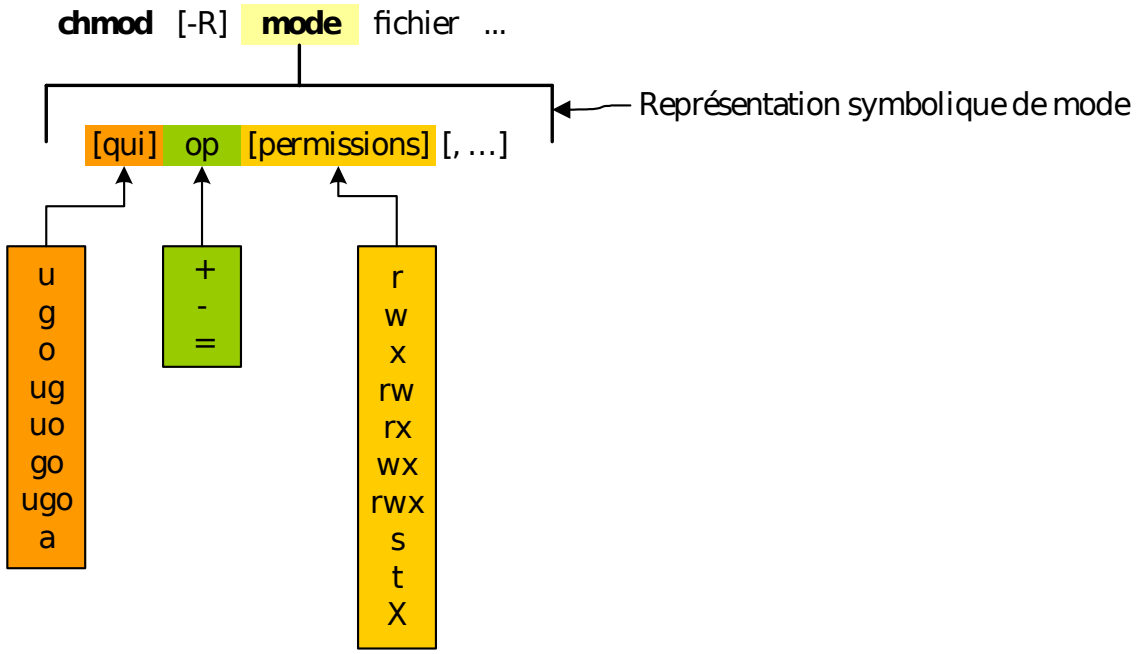


Figure 4.2 : Représentation symbolique de la commande chmod

Exemples

- **chmod u+x,g+x,o+x** monfichier

  Ajouter la permission d'exécution au fichier monfichier, pour le propriétaire, le groupe, et les autres.

- **chmod ugo+x** monfichier

  Identique à la précédente.

- **chmod a+x** monfichier

  Identique à la précédente.

- **chmod g-w,o-w** monfichier

  Enlever le droit d'écriture sur le fichier monfichier, pour le groupe et les autres.

- **chmod u=rwx** monfichier

  Attribuer les droits de lecture, écriture et exécution sur le fichier mon fichier, pour le propriétaire.

- **chmod a=** monfichier

  Enlever tous les droits à tout le monde (propriétaire, groupe et autres).

- **chmod 711** monfichier

  Attribuer les droits de lecture, écriture et exécution sur le fichier monfichier, pour le propriétaire (7) et le droit d'exécution sur ce fichier pour le groupe et les autres (1).

19.7 Droits par défaut

Le système attribue des droits par défaut aux fichiers et répertoires lors de leur création. Par défaut, le système positionne les droits de lecture et d'écriture pour toutes les catégories d'utilisateur (i.e. **rw-rw-rw** (**666**)). Les répertoires reçoivent en plus les droits d'exécution (i.e. **rwxrwxrwx** (**777**)).

Ces droits peuvent être modifiés à l'aide de la commande **umask**. Cette commande indique les droits qu'ils ne doivent pas être accordés aux fichiers et répertoires lors de leur création, autrement dit les droits à soustraire à partir des droits par défaut quand les fichiers et répertoires sont créés.

La valeur par défaut du masque sur un système UNIX est **022**, ce qui fait que les fichiers seront créés avec les permissions **644** (rw-r--r--) et les répertoires avec les permissions **755** (rwxr-xr-x).

	Fichiers			**Répertoires**		
Permissions par défaut	**666**			**777**		
	rw-	rw-	rw-	rwx	rwx	rwx
umask	**022**			**022**		
	000	010	010	000	010	010
Résultat	**644**			**755**		
	rw-	r--	r--	rwx	r-x	r-x

11 Montage d'un système de fichiers

19.8 Types de systèmes de fichiers

Unix supporte différents types de systèmes de fichiers.

Système de fichiers	**Description**
ext2	Système de fichiers standard sous UNIX.
ext3	version journalisée de ext2.
reiserFS	Système de fichiers journalisé. Il offre de meilleures performances pour les petits fichiers.
swap	Utilisé comme une mémoire virtuelle par le système d'exploitation. Ce système de fichier n'est autre qu'une partie du disque dur utilisée pour stocker temporairement des parties de la mémoire centrale.
iso9660	Système de fichiers utilisé par les CD-ROM.
ntfs	Système de fichiers de Windows NT/2000/XP.
vfat	
xfs	Système de fichiers journalisé. L'objectif de ce système de fichiers était de supporter de très gros fichiers (i.e. avec une taille de l'ordre de **tera-octet = 10^{12} octets**) et des taux de transferts élevés pour jouer et enregistrer en temps réel de la vidéo.

Un système de fichiers journalisé (Journaled File System) contient un journal des écritures, ce qui le rend moins sensible à un arrêt brutal de l'ordinateur. Au démarrage suivant, un test vérifie le système de fichiers (filesystem check) et corrige les erreurs si ce dernier n'a pas été démonté correctement.

19.9 Volumes et partitions

Un **volume** est une quantité fixe d'espace de stockage sur un ou plusieurs disques. Un disque peut contenir plusieurs volumes et un volume peut s'étendre sur plusieurs disques.

Un volume peut être créé en subdivisant un disque dur en **partitions**. Une partition, dite aussi **disque logique** (**Logical Disk**), est une portion de l'espace de stockage sur un disque physique, qui va contenir **un et un seul** système de fichiers. On distingue différents types de partitions :

- Une **partition primaire** (**Primary Partition**) est une partition physique qui est prête à recevoir un système de fichier quand elle est créée. Un disque dur est limité à **quatre** (04) partitions primaires.
- Une **partition étendue** (**Extended Partition**) ne peut pas être formatée, par conséquent elle ne peut pas supportée un système de fichier. Une partition étendue peut contenir des **partitions logiques** (**Logical Partitions**) qui sont des divisions logiques des secteurs de stockage qui peuvent être formatées afin de supporter un système de fichier.

La création d'un volume se fait en suivant les étapes suivantes :

1. Partitionner le disque dur,
2. Formater la partition, en sélectionnant un système de fichier approprié,
3. Monter le volume (le rendre disponible à l'utilisateur).

19.10 Partitionnement d'un disque : commande fdisk

Pour créer des partitions sur un disque dur, on utilise la commande **fdisk**. L'utilisation de l'option **-l** de la commande fdisk permet de lister les partitions existantes.

Utiliser …	**pour …**
fdisk /dev/nom_periph	créer des partitions sur le périphérique nom_periph.
d (à l'intérieur de fdisk)	supprimer une partition.
l (à l'intérieur de fdisk)	lister les types de partitions supportés.
m (à l'intérieur de fdisk)	afficher le fichier d'aide.
n (à l'intérieur de fdisk)	créer une nouvelle partition.
p (à l'intérieur de fdisk)	afficher la table de partitions pour ce périphérique.
q (à l'intérieur de fdisk)	quitter fdisk sans sauvegarder les changements.
w (à l'intérieur de fdisk)	sauvegarder les changements et quitter fdisk.

19.11 Règles de nommage des partitions

La règle de notations des périphériques (disques durs, lecteurs de CDRom, DVD, …etc.) est la suivante :

- **Périphériques IDE**

Les périphériques IDE sont accessibles via des fichiers spéciaux nommés avec des noms de la forme **hdx**, où **x** est une lettre identifiant le disque sur le bus IDE.

- Le périphérique IDE maître sur le premier port est accessible via **/dev/hda**.
- Le périphérique IDE esclave sur le premier port est accessible via **/dev/hdb**.
- Le périphérique IDE maître sur le deuxième port est accessible via **/dev/hdc**.
- Le périphérique IDE esclave sur le deuxième port est accessible via **/dev/hdd**.

- **Périphériques SCSI**

  Les périphériques SCSI sont accessibles via des fichiers spéciaux dont le nom est de la forme **sdx**.

  - Le premier disque dur SCSI détecté est **/dev/sda**.
  - Le deuxième disque dur SCSI détecté est **/dev/sdb**.
  - …etc.

- **Lecteurs de disquette**

  Les lecteurs de disquette sont accessibles via des fichiers spéciaux dont le nom est de la forme **fdn**, où **n** est le numéro du lecteur.

  - Le premier lecteur de disquette est **/dev/fd0**.
  - Un deuxième lecteur de disquette s'appellera **/dev/fd1**.
  - …etc.

- **Disques SATA, USB**

  Ces périphériques sont nommés de la même façon que les périphériques SCSI.

La règle de numérotation des partitions d'un disque dur est la suivante :

- Les **partitions primaires** et la **partition étendue** sont **numérotées de 1 à 4**.
- Les **partitions logiques** sont numérotées obligatoirement **à partir de 5**. Par exemple, /dev/hda5, /dev/hda6, …etc.

19.12 Formatage d'une partition : commande mkfs

La commande mkfs (**m**a**k**e **f**ile **s**ystem) permet d'installer un système de fichier dans un volume.

Formater une partition avec …	**Utiliser …**	**Exemple**
ext2	**mkfs –t ext2 /dev/**device **mke2fs /dev/**device	**mkfs –t ext2 /dev/fd0** **mke2fs /dev/fd0** les deux commandes formatent la disquette avec ext2.

ext3	**mkfs –t ext3 /dev/**device **mke2fs –j /dev/**device	**mkfs -t ext3 /dev/hdb2** **mke2fs -j /dev/hdb2** Les deux commandes formatent la deuxième partition sur le deuxième disque dur IDE avec ext3.
ReiserFS	**mkfs -t reiserfs /dev/**device **mkreiserfs /dev/**device	**mkfs -t reiserfs /dev/sda2** **mkreiserfs /dev/sda2** Les deux commandes formatent la deuxième partition sur le premier disque dur SCSI avec le système de fichier Reiser.
swap	**mkswap /dev/**device	**mkswap /dev/hda1** formate la première partition sur le premier disque IDE comme étant une partition de swap.
NTFS	**mkfs -t ntfs /dev/**device **mkntfs /dev/**device	**mkfs -t ntfs /dev/hda1** **mkntfs /dev/hda1** Les deux commandes formatent la première partition sur le premier disque IDE avec le système de fichier NTFS.

19.13 Montage de volume

Le mécanisme de **montage** permet de raccorder une partition à un répertoire de l'arborescence principale. Ainsi, le fait de monter une partition dans le répertoire /mnt/rep rendra l'ensemble des fichiers de la partition accessible à partir de ce répertoire, appelé **point de montage** (**Mount Point**).

Pour cela, on utilise la commande **mount**. Elle a besoin de plusieurs arguments, dont, le périphérique à monter (i.e. le volume), le point de montage (l'endroit où vous souhaiter monter le périphérique), le système de fichier (ext3, vfat, ntfs, ...), et certaines options.

```
$ mount –options <nomvolume> <pointmontage>
```

Exemple

La commande

```
$ mount /dev/hda5 /home/essai
```

permet de monter la partition /dev/hda5 sur le point de montage /home/essai.

La commande

```
$ mount
```

permet d’afficher la liste des volumes **actuellement** montés. Ceci revient à afficher le contenu du fichier **/etc/mtab**.

Inversement, pour démonter un volume, c’est la commande **umount** qu’il faut utiliser.

```
$ umount <pointmontage>
```

Exemple

La commande

```
$ mount /home/essai
```

permet de démonter la partition /dev/hda5 du répertoire /home/essai.

Dans certains cas, on ne peut pas démonter une partition :

1. si une commande s'exécute dans la partition,
2. si des fichiers sont ouverts dans la partition,
3. si l'on a un répertoire courant dans un répertoire de la partition.

Exemple

```
$ cd /mnt
$ umount /mnt
umount: /mnt: Device busy
```

Les commandes **fuser** (**f**ile **user**) et **lsof** (**l**i**s**t of **o**pen **f**iles) permettent d'identifier les fichiers ouverts et quels processus sont en train d'utiliser la partition et qui empêchent le démontage.

19.14 Montage automatique

Pour pouvoir monter automatiquement un volume au démarrage du système, il suffit de rajouter une entrée correspondant à ce volume dans le fichier **/etc/fstab**. Ce fichier est organisé en six (06) colonnes :

Nom de partition	Point de montage	Système de fichiers	Options	Dump	Check
LABEL=/	/	ext3	defaults	1	1
none	/proc	proc	defaults	0	0
/dev/hda1	/win	ntfs	ro,defaults	0	0
/dev/sda1	/mnt/usb	vfat	rw,user,noauto	0	0

Figure 4.3 : Format du fichier /etc/fstab

1. **Colonne 1 :** Nom de la partition à monter (**Ex.** /dev/hda2, none, …etc.).
2. **Colonne 2 :** Point de montage (**Ex.** /var, …etc.).
3. **Colonne 3 :** Type de système de fichiers (**Ex.** ext2, ext3, iso9660, ntfs, … etc.).
4. **Colonne 4 :** Options de montage de la partition (nouser, rw, ro, auto, exec, defaults, …etc.).
5. **Colonne 5 :** précise si le système de fichiers doit être sauvegardé par l'utilitaire mémoire **dump** ou non (**=1 si oui** et **=0 si non**).
6. **Colonne 6 :** utilisée par l'utilitaire **fsck** (**f**ile **s**ystem **c**hec**k**) pour déterminer dans quel ordre vérifier les partitions (**=0 ⇒ ne pas tester** et **⇒0 ≠ l'ordre de test**).

Exemple

La partition /dev/hda5 doit être montée sur le répertoire /home/essai. Le système de fichier est de type ext3 et les paramètres de montage sont defaults (i.e. rw, suid, dev, exec, auto, nouser, async). Ce système de fichiers peut être sauvegardé par dump et sa priorité de vérification (avec fsck) est 2.

L'entrée correspondante dans le fichier /etc/fstab est :

```
/dev/hda5     /home/essai      ext3  defaults   1     2
```

CHAPITRE VI : GESTION DU PROCESSEUR

Nous avons vu précédemment que, pour améliorer la rentabilité des machines, il est nécessaire de pouvoir exécuter des activités parallèles ou tâches parallèles, comme exécuter un transfert d'E/S en même temps qu'un calcul interne au processeur.

A cet effet, sont apparus les systèmes multitâches (exécutant plusieurs tâches à la fois), et les systèmes multiutilisateurs (plusieurs utilisateurs travaillent en même temps). Dans ce cas le système doit gérer plusieurs programmes (processus), et les exécuter dans les meilleurs délais possibles en partageant le temps processeur. De plus, il doit donner à chaque utilisateur l'impression de disposer du processeur à lui seul.

Pour cela, un programme du SE s'occupe de gérer l'utilisation ou plutôt **l'allocation du processeur** aux différents programmes : c'est le **Scheduler**.

La gestion du processeur central se base principalement sur **l'organisation** ou **stratégie** qui permet l'allocation du processeur aux différents programmes qui existent dans la machine.

2 Files d'attente de scheduling

Afin d'implanter les différents états d'un processus, une file d'attente correspondant à chaque état (sauf l'état actif) est utilisée. Ces files consistent en un ensemble de descripteurs de processus ou PCB chaînés l'un à l'autre.

En effet, chaque processus est représenté dans le système par un PCB, qui contient toutes les informations nécessaires pour sa gestion.

Le processus passe alors, durant sa vie, par les files suivantes :

1. Quand le processus rentre dans le système, il est initialement inséré dans une **file d'attente de travaux** (**job queue**). Cette file d'attente contient tous les processus du système.
2. Quand le processus devient résidant sur MC et qui est prêt et attend pour s'exécuter, il est maintenu dans une liste appelée la **file d'attente des processus prêts** (**Ready queue**). La file **Prêt** contient donc l'ensemble des processus résidents en MC et prêts à l'exécution.
3. Le processus attend dans la file **Prêt** jusqu'à ce qu'il soit sélectionné pour son exécution, et qu'on lui alloue le CPU. Une fois alloué le CPU, le processus devient **Actif**. Actif est concrètement donné par un pointeur (non une file) vers le PCB associé au processus en cours d'exécution.
4. Etant actif, le processus pourrait émettre une requête d'E/S vers un périphérique particulier. il est alors placé (son PCB est placé) dans la **file d'attente du périphérique** (**Device queues**). Chaque périphérique possède sa propre file d'attente.
5. Enfin, le processus actif pourrait créer lui-même un nouveau sous-processus (appelé **processus fils**) et attendre sa fin.

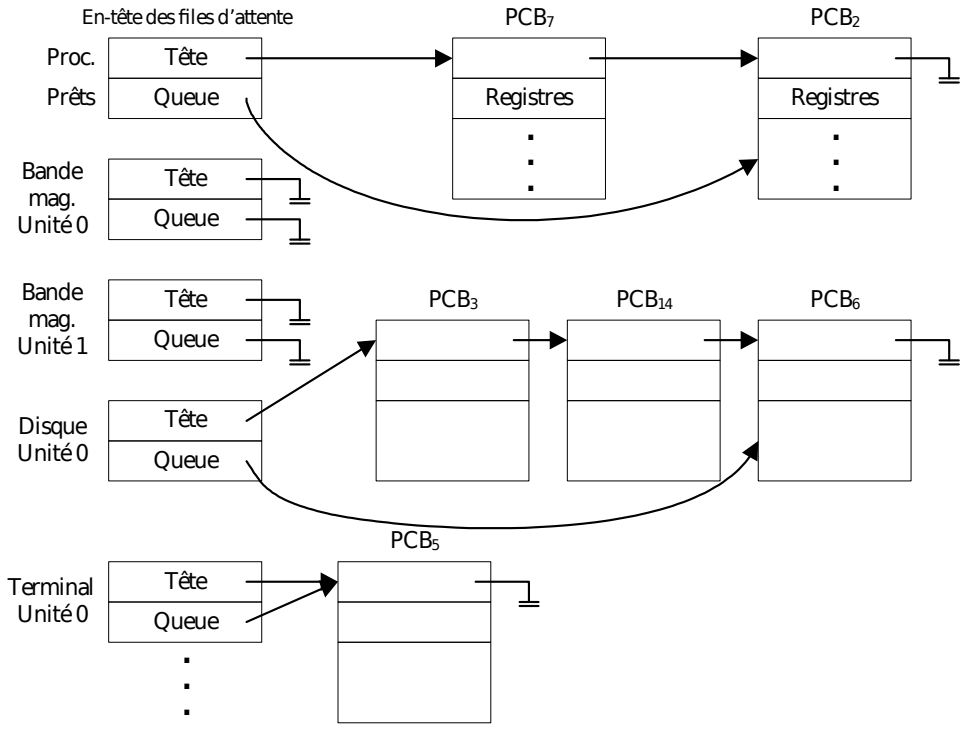


Figure : La file d'attente des proc. Prêts et plusieurs files d'attente des périph. d'E/S

Le diagramme des files d'attente de la figure ci-dessous est une représentation communément utilisée afin d'étudier le scheduling des processus.

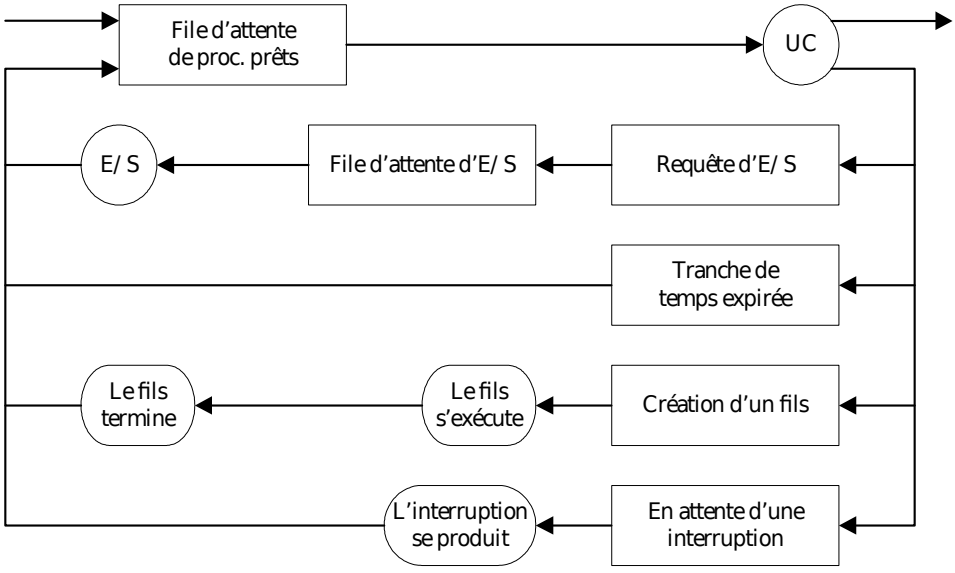


Figure : Représentation du diagramme des files d'attente du scheduling des processus

3 Concept de Scheduling

19.15 Définition

On appelle « **Scheduling** » du processeur l'organisation qui sous-tend à l'allocation du processeur central aux programmes.

On appelle «**Scheduler** » la partie du SE qui s'occupe de cette organisation et répartit le temps processeur entre ces programmes.

19.16 Critères de performance de scheduling

Le choix d'une politique de scheduling doit tenir compte des points suivants :

- **Utilisation de l'UC (CPU utilization) –** utiliser la CPU le maximum possible.
- **Débit (Throughput) –** nombre de processus qui terminent leur exécution par unité de temps.
- **Temps de rotation (Turnaround time) –** le temps depuis le lancement du processus jusqu'à sa terminaison (les attentes incluses).
- **Temps de réponse (Response time)**
- **Temps d'attente (Waiting time) –** temps d'un processus dans la file d'attente des processus prêts.
- **Equité (Fairness) –** le Scheduler doit servir les programmes de même priorité d'une manière juste et équitable.

Il est souhaitable de **maximiser** l'utilisation de l'UC et la capacité de traitement (i.e. Débit) et de **minimiser** le temps de rotation, le temps d'attente et le temps de réponse.

19.17 Scheduling avec ou sans préemption

Une politique de Scheduling est dite **non préemptive** si, une fois le processeur central est alloué à un programme, celui-ci le gardera jusqu'à la fin de son exécution, ou jusqu'à se bloquer sur une ressource non disponible ou en attente de la satisfaction d'une demande de ressource.

Exemple

- Systèmes à temps partagé : préemptif.
- Systèmes à temps réel : non préemptif (traitement critique non interruptible).

19.18 Priorité de scheduling

L'allocation du processeur central aux programmes peut se faire suivant certaines valeurs de priorité, qui son affectées aux programmes automatiquement par le système de Scheduling, ou bien de manière externe par les usagers ou le responsable d'exploitation de la machine (administrateur système de la machine).

Les priorités peuvent être allouées statiquement ou dynamiquement.

- **Priorité statique :** une fois qu'une priorité est allouée à un programme, celle-ci ne changera pas jusqu'à la fin de son exécution.
- **Priorité dynamique :** la priorité affectée à un programme change en fonction de l'environnement d'exécution du programme. Cet environnement est donné en général par la charge du système (nombre de processus en cours d'exécution), l'ancienneté (à chaque fois que le programme prend plus de temps, on diminue sa priorité).

4 Différents Niveaux de Scheduling

Il existe trois niveaux de Scheduling :

- Le **scheduling de haut niveau** (**Long-Term scheduling**) qui décide du degré de la multiprogrammation (nombre maximal de processus dans le système).
- Le **scheduling de niveau intermédiaire** (**Medium-Term scheduling**) gère le déplacement des processus d'une file d'attente à une autre. Dans la majorité des systèmes actuels, les niveaux intermédiaire et haut sont combinés -> haut niveau.
- Le **scheduling de bas niveau** (**Short-Term scheduling**) (dispatcher ou répartiteur) est sollicité plusieurs fois par seconde et doit constamment résider en mémoire. Il permet de déterminer le processus prêt à utiliser le processeur central.

5 Politiques de Scheduling

19.19 Politique « Premier Arrivé Premier Servi » (FCFS, First Come First Served- FIFO)

Consiste à servir le 1er processus arrivé dans le système.

C'est une politique **non préemptive** qui désavantage les processus courts.

Exemple

Considérons cinq processus A, B, C, D et E dont les durées et leurs arrivages respectifs sont donnés dans la table ci-après. Faire un schéma qui illustre son exécution et calculer le **temps de réponse** de chaque processus, le **temps moyen de réponse**, le **temps d'attente** et le **temps moyen d'attente** en utilisant la politique FCFS.

Le **temps de réponse** pour chaque processus est obtenu en soustrayant le temps d'arrivée du processus du temps de terminaison :

Temps de réponse = Date de fin d'Exe. – Temps d'arrivée.

Le **temps d'attente** est calculé en soustrayant le temps d'exécution du temps de réponse :

Temps d'attente = Temps de réponse – Durée d'Exe.

Le schéma d'exécution est : AAABBBBBBCCCCDDE

Processus	Durée	Temps d'arriv	Date Début	Date fin	Temps de	Temps d'atte

	d'Exe.	ée	d'Exe.	d'Exe.	réponse	nte
A	3	0	0	3	3	0
B	6	1	3	9	8	2
C	4	4	9	13	9	5
D	2	6	13	15	9	7
E	1	7	15	16	9	8
Temps de réponse moyen			(3+8+9+9+9)/5= 38/5 = 7,6 UT			
Temps d'attente moyen			(0+2+5+7+8)/5 = 23/5 = 4,6 UT			

Remarque

Temps moyen d'attente élevé si de longs processus sont exécutés en premier.

19.20 Politique du Job le Plus Court d'Abord (SJF, Shortest Job First)

Consiste à servir les processus courts avant les processus longs.

C'est une politique **non préemptive**, dont la mise en œuvre nécessite la connaissance préalable du temps d'exécution des processus.

Exemple

Reprenez l'exemple précédent avec la politique SJF.

Le schéma d'exécution : AAABBBBBBEDDCCCC

Processus	Durée d'Exe.	Temps d'arrivée	Date Début d'Exe.	Date fin d'Exe.	Temps de réponse	Temps d'attente
A	3	0	0	3	3	0
B	6	1	3	9	8	2
C	4	4	12	16	12	8
D	2	6	10	12	6	4
E	1	7	9	10	3	2
Temps de réponse moyen			(3+8+12+6+3)/5= 32/5 = 6,4 UT			
Temps d'attente moyen			(0+2+8+4+2)/5 = 16/5 = 3,2 UT			

19.21 Politique du Job ayant le Plus Court Temps Restant (SRTF, Shortest Remaining Time First)

lorsqu'un processus est en cours d'exécution, et qu'un nouveau processus ayant un temps d'exécution plus court que celui qui reste pour terminer l'exécution du processus en cours, ce processus est arrêté (préempté), et le nouveau processus est exécuté.

Cette méthode équivaut à la méthode SJF mais **préemptive**.

Exemple

Reprenez l'exemple précédent avec la politique SRTF.

Le schéma d'exécution : AAABCCCCEDDBBBBB

Processus	Durée d'Exe.	Temps d'arrivée	Date Début d'Exe.	Préemptée A	Reprise A	Date fin d'Exe.	Temps de réponse	Temps d'attente

A	3	0	0			3	3	0
B	6	1	3	4	11	16	15	9
C	4	4	4			8	4	0
D	2	6	9			11	5	3
E	1	7	8			9	2	1
Temps de réponse moyen			(3+15+4+5+2)/5= 29/5 = 5,8 UT					
Temps d'attente moyen			(0+9+0+3+1)/5 = 13/5 = 2,6 UT					
Nbr de changements de CTXT			6					

19.22 Politique à base de priorité

Des priorités sont affectées aux différents processus et ils sont activés en fonction de cette priorité. Le processus élu par le scheduler est celui qui a la plus haute priorité parmi les processus éligibles.

La priorité affectée à un processeur peut dépendre de l'utilisateur qui a lancé l'exécution du processus, de la quantité de ressources demandée par le processus, etc. Ce type de politiques est particulièrement utile dans les systèmes temps réel où il s'agit souvent de répondre à des événements à caractères urgents. Dans ce contexte, la priorité est affectée en fonction des contraintes liées au processus lui-même.

L'UCT est donnée au processus prêt avec la plus haute priorité avec ou sans préemption. Dans le cas d'une politique **à base de priorité préemptive**, l'arrivée d'un processus plus prioritaire entraîne l'arrêt du processus en cours d'exécution et l'attribution du processeur au nouveau processus.

Cette politique peut poser le problème de **famine**, où les processus moins prioritaires risquent de ne jamais être exécutés. Une solution à ce problème est l'utilisation du principe de **vieillissement** qui permet de modifier la priorité d'un processus en fonction de son âge et de son historique d'exécution.

Exemple

Considérons quatre processus A, B, C et D dont leurs durées d'exécution, leurs dates d'arrivée et leurs priorités respectives sont données dans la table ci-après. Faire un schéma qui illustre son exécution et calculer le **temps de réponse** de chaque processus, le **temps moyen de réponse**, le **temps d'attente** et le **temps moyen d'attente** ainsi que le **nombre de changements de contexte** effectués en utilisant la politique à base de priorité avec préemption.

Le schéma d'exécution : AAABBBDDBBBCCCC

Processus	**Durée d'Exe.**	**Temps d'arrivée**	**Prio.**	**Date Début d'Exe.**	**Préempté A**	**Repris A**	**Date fin d'Exe.**	**Temps de réponse**	**Temps d'attente**
A	3	0	3	0			3	3	0
B	6	3	1	3	6	8	11	8	2
C	4	5	2	11			15	10	6
D	2	6	0	6			8	2	0
Temps de réponse moyen				(3+8+10+2)/4= 23/4 = 5,75 UT					
Temps d'attente moyen				(0+2+6+0)/4 = 8/4 = 2 UT					

Remarque

La méthode SJF est un cas particulier de la politique de Scheduling par priorité ***p*** où ***p*=1/*t*** (***t*** est le temps CPU estimé).

Lorsque ce temps ***t*** est important, sa priorité *p* diminue ⇒ le processus est moins prioritaire.

19.23 Politique du Tourniquet (Round Robin, RR)

Cette politique consiste à allouer le processeur aux processus suivant une durée d'exécution limitée appelée « **Quantum** ». C'est une politique **préemptive**, qui s'adapte bien aux systèmes à temps partagé.

Implantation de l'algorithme Round Robin

Elle est réalisée à l'aide d'une file d'attente circulaire des processus prêts, organisée en FIFO. Un processus qui arrive est mis en queue de la file. Le processus élu par le Scheduler est celui en tête de file.

Pour signaler l'écoulement du Quantum, on utilise une horloge, initialisée au lancement du processus élu. Celui-ci se terminera de deux manières :

- Soit à l'écoulement du Quantum, une interruption horloge se déclenche et le processus est remis alors en queue de la file, après avoir sauvegardé le contexte du processus dans son PCB
- Soit par libération du processeur à la suite d'une demande de ressource ou de fin totale d'exécution.

Les performances de cette politique dépendent de :

- **La valeur du Quantum**

  Si le Quantum est très grand, la politique Round Robin se confond avec la politique FCFS. Si le Quantum est très petit, les processus auront l'impression de disposer du processeur à lui seul. En effet, le processeur tourne à la vitesse de 1/n de sa vitesse réelle (n étant le nombre de processus)

- **La durée de commutation**

  Cette commutation se fait à chaque allocation du processeur et nécessite un certain temps d'exécution (10 à 100 ⇒s).

Exemple 1

Reprenez l'exemple précédent avec la politique RR. On suppose que Quantum = 1 et que la durée de commutation = 0.

Le schéma d'exécution : ABABACBDCEBDCBCB

Processus	Durée d'Exe.	Temps d'arrivée	Date Début d'Exe.	Préempté A	Repris à	Date fin d'Exe.	Temps de réponse	Temps d'attente
A	3	0	0	1	2	5	5	2
				3	4			
B	6	1	1	2	3	16	15	9
				4	6			
				7	10			

				11	13			
				14	15			
C	4	4	5	6	8	15	11	7
				9	12			
				13	14			
D	2	6	7	8	11	12	6	4
E	1	7	9			10	3	2
Temps de réponse moyen			(5+15+11+6+3)/5= 40/5 = 8 UT					
Temps d'attente moyen			(2+9+7+4+2)/5 = 24/5 = 4,8 UT					
Nbr de changements de CTXT			16					

Exemple 2

Considérons trois processus A, B, et C dont les durées et leurs arrivages respectifs sont donnés dans la table ci-après. Faire un schéma qui illustre son exécution et calculer le **temps de réponse** de chaque processus, le **temps moyen de réponse**, le **temps d'attente** et le **temps moyen d'attente** ainsi que le **nombre de changements de contexte** effectués en utilisant la politique RR.

On suppose que le Quantum = 3 et la durée de commutation = 1.

Processus	**Durée d'Exe.**	**Temps d'arrivée**	**Date Début d'Exe.**	**Préempté A**	**Repris à**	**Date fin d'Exe.**	**Temps de réponse**	**Temps d'attente**
A	8	0	1	4	9	22	22	14
				12	20			
B	5(2)3	3	5	8	17	28	25	15
				19	25			
C	4	7	13	16	23	24	17	13
Temps de réponse moyen			(22+25+17)/3= 64/3 = 21,33 UT					
Temps d'attente moyen			(14+15+13)/3 = 42/3 = 14 UT					
Nbr de changements de CTXT			8					

19.24 Politique à Plusieurs Niveaux de Queues (Multi-Level Queues)

Dans cette stratégie, la file des processus prêts est subdivisée en plusieurs files suivant la classe des processus (batch, interactifs, temps réel, …).

Chaque file est gérée suivant une politique de Scheduling propre à elle, et s'adaptant mieux à la classe des processus qu'elle contient.

Exemple

- Processus batch μ FCFS.
- Processus interactifs $\rightarrow$ Round Robin.

Le Scheduling entre les files elles-mêmes peut se faire :

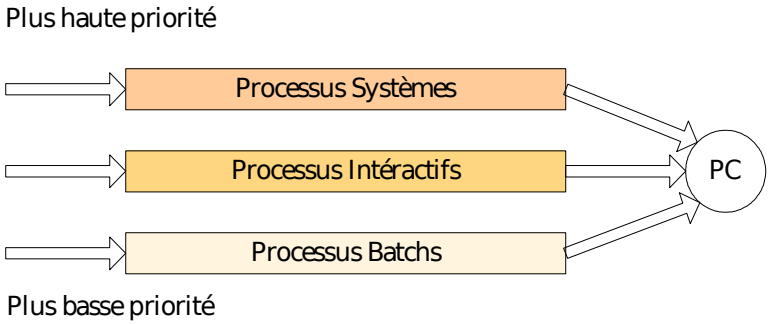
- Soit par priorité.

  Ex. la file INTERACTIF est plus prioritaire que la file Batch.

- Soit par partage du temps du processeur.

  Ex. 80% $\rightarrow$ Interactifs, 20% $\rightarrow$ Batch.

Exemple



19.25 Politique à Plusieurs Niveaux de Queues Dépendantes (Multilevel Feedback Queues)

Dans la politique précédente, un processus ne change pas de files. Par contre, dans la politique à plusieurs niveaux dépendants, un processus peut changer de file suivant les changements de comportement qu'il peut avoir durant son exécution. Ceci est réalisé afin d'isoler les processus consommateurs de temps CPU, et pouvoir lancer des processus de faible priorité en les plaçant dans des files à plus haute priorité. En effet :

- Un processus fait parfois beaucoup de calcul et parfois beaucoup d'E/S.
- Un processus qui a séjourné un grand temps dans une file à haute priorité est placé dans une file de plus basse priorité.

Cette politique est définie par les paramètres suivants :

- Nombre de files.
- L'algorithme de Scheduling de chaque file.
- Une méthode déterminant dans quelle file placer un nouveau processus.
- Une méthode de transition d'un processus d'une file à une autre.

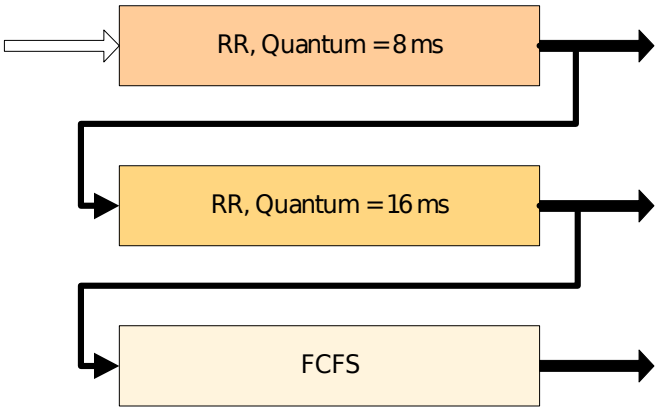
Exemple

Soient Trois files:

- Q0 – gérer par la politique RR avec un quantum de 8 millisecondes.
- Q1 – gérer par la politique RR avec un quantum de 16 millisecondes.
- Q2 – gérer par la politique FCFS.

La politique de scheduling appliquée est décrite comme suit :

- Un nouveau processus est placé dans Q0 au début; à sa première exécution, il reçoit 8 millisecondes. S'il ne termine pas son exécution, il est replacé dans Q1.
- Si un processus de la file Q1 est servi (16 msec) et ne se termine pas, il est replacé dans Q2.



6 Scheduling sur un Multiprocesseurs

Si plusieurs UCs sont disponibles, le problème de scheduling est plus complexe. Les processeurs dans un système multiprocesseurs sont **identiques** (**homogènes**) en ce qui concerne leur fonctionnalité, ce qui permet d'avoir une **répartition de la charge**.

Dans un tel schéma, deux approches de scheduling peuvent être utilisées :

- **Approche symétrique** où chaque UC peut exécuter le scheduling et la répartition. Une seule liste prêt pour toutes les UCs (division travail = load sharing).
- **Approche asymétrique** où certaines fonctions sont réservées à une seule UC. Files d'attentes séparées pour chaque UC.

7 Scheduling Temps Réel

Systèmes temps réel rigides (hard): les échéances sont critiques (**Ex.** contrôle d'une chaîne d'assemblage, animation graphique). Il est essentiel de connaître la durée des fonctions critiques. Il doit être possible de garantir que ces fonctions sont effectivement exécutées dans ce temps (réservation de ressources). Ceci demande une structure de système très particulière.

Systèmes temps réel souples (soft): les échéances sont importantes, mais ne sont pas critiques (**Ex.** systèmes téléphoniques). Les processus critiques reçoivent la priorité.

CHAPITRE V : GESTION DE LA MÉMOIRE

La mémoire est le point central dans un système d'exploitation, c'est à travers elle que l'unité centrale communique avec l'extérieur.

La mémoire est une ressource importante de stockage de données et de programmes qui doit être gérée avec prudence. Un programme ne peut s'exécuter que si ses instructions et ses données (au moins partiellement) sont en mémoire centrale. L'utilisation d'un ordinateur en multiprogrammation, pose comme condition obligatoire que la mémoire centrale soit utilisée et/ou partagée entre les différents processus.

La gestion de la mémoire est du ressort du **gestionnaire de la mémoire**. Le rôle du gestionnaire de la mémoire est de connaître les parties libres et occupées, d'allouer de la mémoire aux processus qui en ont besoin, de récupérer de la mémoire à la fin de l'exécution d'un processus et de traiter le **recouvrement** (**va-et-vient** (**Swapping**)) entre le disque et la mémoire centrale, lorsqu'elle ne peut pas contenir tous les processus actifs.

8 Hiérarchie

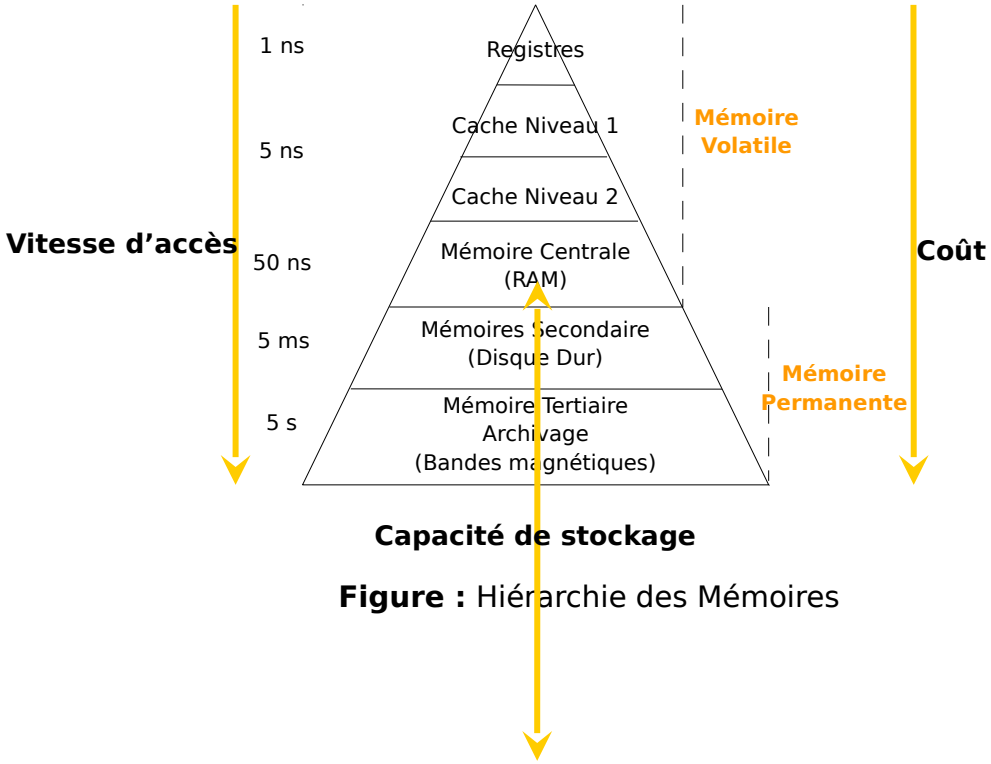


Figure : Hiérarchie des Mémoires

- **Registres :** mémoire très rapide, directement accessible au processeur en un cycle avec une capacité de quelques dizaines à quelques centaines d'octets.
- **Mémoire cache interne (L1) :** mémoire rapide, intégrée au processeur, accessible en quelques cycles, et de capacité de quelques dizaines de Koctets (**Ex.** 128 ko pour les premiers ATHLON, 32 ko pour les pentiums 2/3). Généralement, divisée en deux (02) pour les données et pour les programmes.
- **Mémoire cache externe (L2) :** mémoire moins rapide que le cache L1 mais plus volumineuse (256ko à 2Mo). Habituellement à l'extérieur du CPU. Elle ne fait aucune différence entre les données et les programmes.
- **Mémoire centrale (Dynamic Random Access Memory) :** d'une capacité de plusieurs Moctets mais pourrait monter jusqu'à plusieurs Goctets, et d'un temps d'accès de l'ordre de plusieurs dizaines de cycles du processeur (50 - 100ns).

- **Disque magnétique :** avec une capacité de plusieurs dizaines de Goctets, et d'un temps d'accès de plusieurs dizaines de ms.

9 Mémoire Centrale

Le terme mémoire désigne un composant destiné à contenir une certaine quantité de données, et à en permettre la consultation. La mémoire centrale, dite aussi la **mémoire vive** (**Random Access Memory, RAM**), est un ensemble de mots mémoires où chaque mot a sa propre **adresse**. L'octet représente la plus petite quantité de mémoire adressable. Plusieurs types de mémoires sont utilisés, différentiables par leur technologie (DRAM, SRAM, ...etc.), et leur forme (SIMM, DIMM, ...etc.).

Il s'agit d'une mémoire **volatile ;** ce qui sous−entend que son contenu est perdu lorsqu'elle n'est plus alimentée électriquement.

La mémoire vive sert à mémoriser des programmes, des données à traiter, les résultats de ces traitements, ainsi que des données temporaires.

Deux types d'opérations peuvent s'effectuer sur les mots mémoires ; à savoir la **lecture** (**Read**) et l'**écriture** (**Write**) :

- **Pour lire la mémoire**: une adresse doit être choisie à partir du bus d'adresse et le bus de contrôle doit déclencher l'opération. Les données à l'adresse choisie se retrouvent sur le bus de données.
- **Pour écrire la mémoire:** une adresse doit être choisie à partir du bus d'adresse et le bus de contrôle doit déclencher l'opération. Les données à l'adresse choisie sont remplacées par celles sur le bus de données.

10 Objectifs

Le gestionnaire de la mémoire est un module du système d'exploitation dont le rôle est la **gestion de la mémoire principale (RAM)**. Le but d'une bonne gestion de la mémoire est d'augmenter le rendement global du système. Pour cela, Le plus grand nombre possible de processus en exécution doit y être gardé en MC, de façon à optimiser le fonctionnement du système en multiprogrammation en assurant une bonne répartition entre les E/S et la demande CPU.

Le gestionnaire de la mémoire doit répondre aux questions suivantes :

- Comment organiser la mémoire ? (une ou plusieurs partitions, le nombre et la taille des partitions fixes ou variables au cours du temps).
- Faut-il allouer une zone contiguë à chaque processus à charger en mémoire ? Faut-il allouer tout l'espace nécessaire à l'exécution du processus entier ? (**politique d'allocation**).
- Comment mémoriser l'état de la mémoire? Parmi les parties libres en mémoire, lesquelles allouées au processus? (**politique de placement**)
- S'il n'y a pas assez d'espace en mémoire, doit-on libérer de l'espace en retirant des parties ou des processus entiers? Si oui lesquels ? (**politique de remplacement**).
- Les adresses figurant dans les instructions sont-elles relatives? Si oui, comment les convertir en adresses physiques.

- Si plusieurs processus peuvent être résidents en mémoire, comment assurer la protection des processus (éviter qu'un processus soit altéré par un autre?)

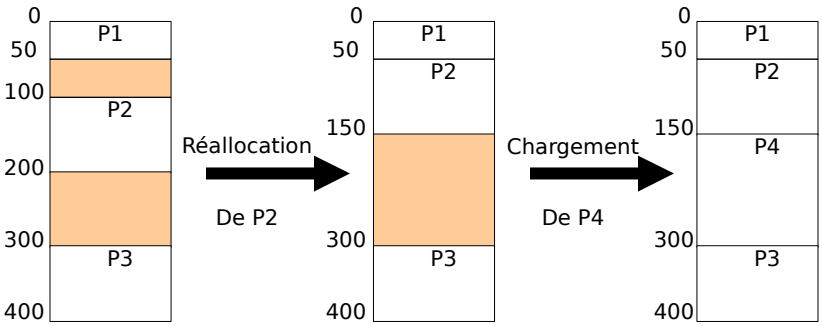
Et Il doit viser les objectifs suivants :

19.26 La réallocation (Dynamic Relocation)

Le système d'exploitation alloue à chaque programme une zone mémoire. Mais tous les programmes n'ont pas la même taille. De plus, les programmes sont lancés par les utilisateurs, puis terminent à des moments que le système ne connaît pas à l'avance. Chaque fois qu'un utilisateur demande le lancement d'un programme, le système doit trouver une place dans la mémoire pour le charger : il y a réorganisation des programmes en mémoires.

Exemple

Soient les programmes P1, P2, P3 en mémoire selon la figure suivante. Soit un quatrième programme P4 de taille 150 demandait à être exécuté. Malgré que la place totale disponible dans la mémoire soit de 150, mais ce sera impossible de chargé ce programme, car les 150 unités disponibles ne forment pas un espace contigu dans lequel on peut charger le programme P4. Un réarrangement de l'espace mémoire permettra de charger P4 comme suit :



19.27 Le partage (Sharing)

Parfois, il est utile de partager un espace mémoire entre plusieurs processus. Ainsi, le sous-système de la gestion de la mémoire doit autoriser des accès contrôlés sans compromettre la protection.

Exemple

Un éditeur de texte partagé entre plusieurs utilisateurs d'un système à temps partagé.

19.28 La protection (Protection)

La coexistence de plusieurs processus en mémoire centrale nécessite la protection de chaque espace mémoire vis-à-vis des autres. Par conséquent, un processus P1 ne peut accéder à l'espace d'un processus P2 que s'il est autorisé.

19.29 Organisation logique

Le gestionnaire de la mémoire divise l'espace mémoire en **zones logiques** appelées **partitions** ou **segments** ou **pages**. Cette organisation ne reflète pas nécessairement l'organisation physique de la mémoire.

11 Caractéristiques liées au chargement d'un programme

19.30 Espace d'adressage (utilisateur/noyau)

En réalité, l'espace d'adressage est divisé en deux (02) parties : une partie visible dans **l'espace utilisateur**, et une partie visible par **l'espace noyau**.

Dans l'espace réservé au noyau, on y trouve toutes les structures de données gérées par celui-ci (PCB, liste de pages libres, caches, etc.), y compris le noyau lui même.

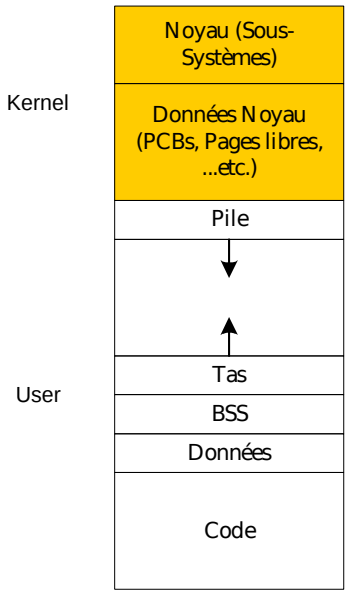


Figure : Espace d'adressage (User/Kernel)

19.31 Rappels sur la compilation

La production d'un programme passe par plusieurs phases, comme illustré à la figure ci-après :

- Écriture en langage source (C/C++, Pascal,...).
- Compilation en module objet (langage machine).
- Peuvent inclure des bibliothèques.
- Appel aux bibliothèques : point de branchement.
- Assemblage en image binaire.
- Édition de liens statiques.
- Édition de liens dynamiques.

- Chargement en mémoire pour exécution.

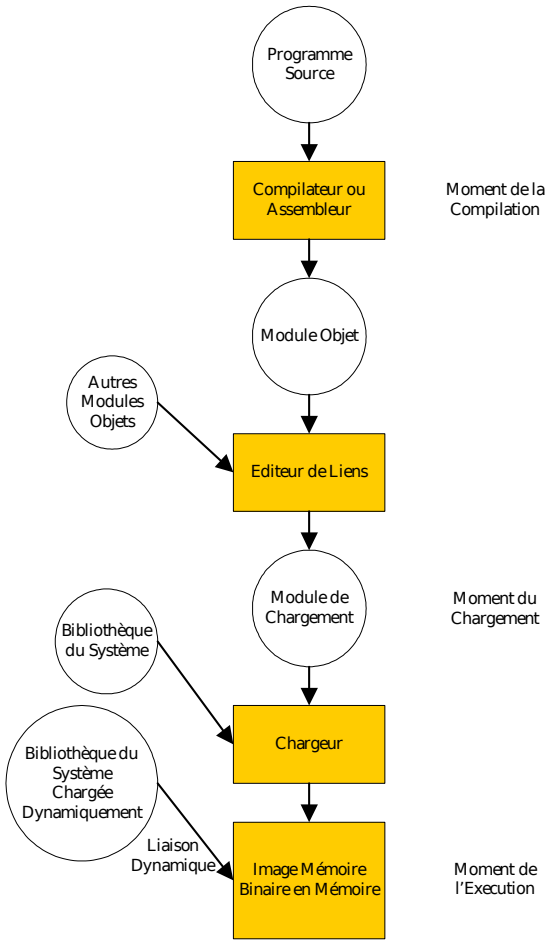


Figure : Traitement en plusieurs Phases d'un Programme Utilisateur

19.32 Représentation des adresses d'un objet

Il existe plusieurs types d'adressages :

- **Adresses symboliques** qui représentent les noms des objets (Données et Instructions) contenus dans le code source. Par exemple, int compteur.
- **Adresses logiques** qui correspondent à la traduction des adresses symboliques lors de la phase de compilation (Module objet). Le système qui permet d'établir cette traduction est appelé **système de nommage** (**Naming System**). Par exemple, le 50ème mot depuis le début d'espace mémoire.
- **Adresses physiques** qui représentent l'emplacement physique des adresses logiques d'un programme lors de son exécution. La traduction des adresses logiques en adresses physiques est assurée par **l'unité de traduction d'adresses** (**MMU, Memory Management Unit**). Par exemple, l'emplacement mémoire situé à l'adresse FFF7.

19.33 Liaison des instructions et données à des adresses mémoire

Le programme réside sur disque comme Fichier exécutable. Il est ensuite chargé en mémoire afin d'être exécuté. Il peut résider, le plus souvent, dans une partie quelconque de la mémoire. Quand le processus est exécuté, il accède aux instructions et aux données de la mémoire.

D'une manière classique, on peut effectuer la **liaison** (**binding**) d'instructions et de données à des adresses mémoire à n'importe quelle étape tout au long du chemin :

- **Pendant la compilation (Compilation Time Binding)**

  Si l'emplacement du processus en mémoire est connu à priori, le compilateur génère des **adresses absolues**. Si plus tard, l'emplacement du début change, il sera donc nécessaire de compiler à nouveau le code.

- **Pendant le chargement (Load Time Binding)**

  Si l'emplacement du processus en mémoire n'est pas connu, le compilateur génère un **code translatable**. Dans ce cas, la liaison finale est reportée jusqu'au moment du chargement. Si, plus tard, l'adresse de début change, il suffit seulement de recharger le code utilisateur pour incorporer cette valeur changée.

- **Pendant l'exécution (Execution Time Binding)**

  Si le déplacement dynamique du processus est possible, la liaison doit être retardée jusqu'au moment de l'exécution. On doit disposer d'un support matériel spécial pour effectuer cette liaison (**Ex.** les registres base et limite).

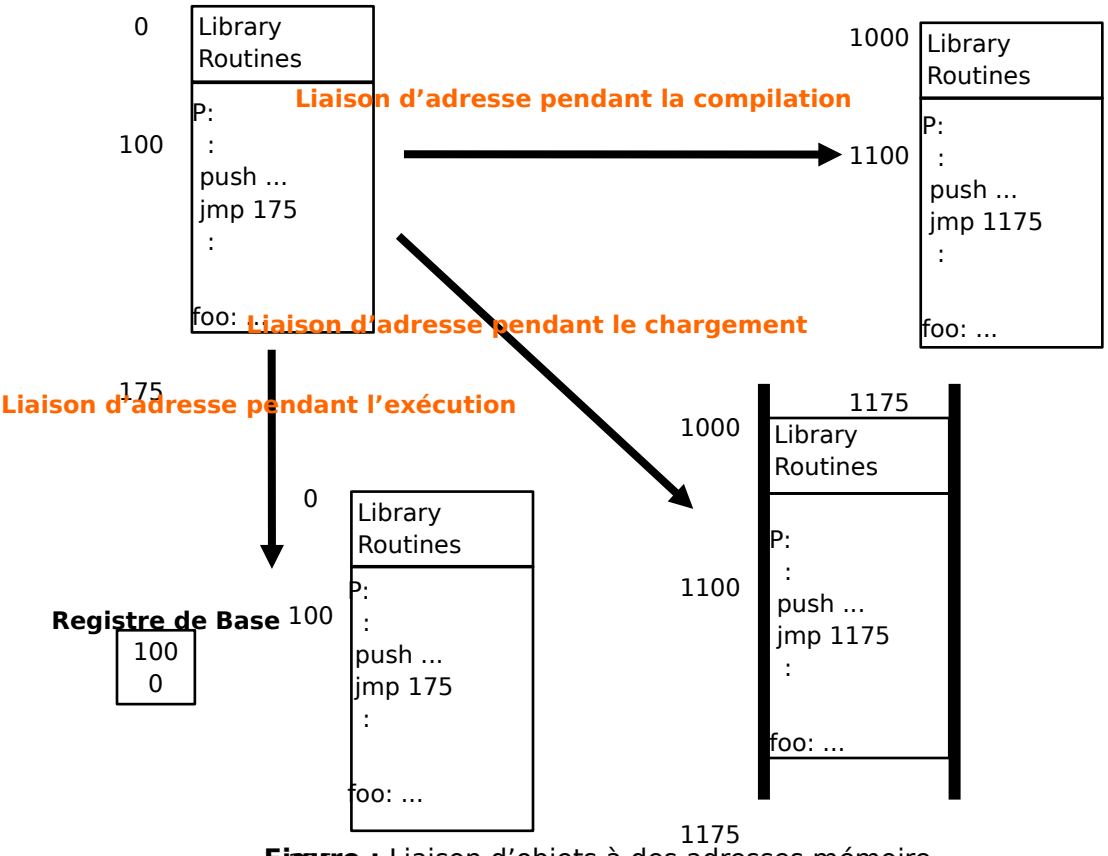


Figure : Liaison d'objets à des adresses mémoire

L'exécution d'un programme est liée à deux concepts de bases ; à savoir le **chargement** (**Loading**) et l'**édition de liens** (**Linking**).

- Le **chargement** qui consiste à charger en mémoire physique, le programme, ou une de ses parties, prêt à s'exécuter. On distingue deux types de chargement :
  - **Statique (avant l'exécution)**
  - **Dynamique (pendant l'exécution) (Dynamic Loading)**

    La routine n'est chargée que quand elle est appelée. Ce qui permet une meilleure utilisation de l'espace mémoire; une routine non utilisée n'est jamais chargée. Le chargement dynamique est utile quand une quantité importante de code est nécessaire pour traiter des cas qui se produisent rarement.
- L'**édition de liens** qui consiste à lier les différentes parties d'un programme pour en faire une entité exécutable. La traduction des références entre les différents modules se réalise de deux manières :
  - **Statique (avant l'exécution)**
  - **Dynamique (sur demande pendant l'exécution) (Dynamic Linking)**

L’édition de liens est reportée jusqu’au moment de l’exécution. Une portion de code, **stub**, est utilisée pour localiser la routine de la bibliothèque appropriée. Le stub se remplace par l’adresse de la routine, et exécute la routine.

19.34 Espace d'adressage logique versus physique

L'unité centrale manipule des **adresses logiques** (emplacement relatif). Les programmes ne connaissent que des adresses logiques, ou virtuelles. **L'espace d'adressage logique** (virtuel) est donc un ensemble d'adresses pouvant être générées par un programme.

L'unité mémoire manipule des **adresses physiques** (emplacement mémoire). Elles ne sont jamais vues par les programmes utilisateurs. **L'espace d'adressage physique** est un ensemble d'adresses physiques correspondant à un espace d'adresses logiques.

La conversion, au moment de l’exécution, des adresses logiques à des adresses physiques est effectuée par l’**unité de gestion mémoire** (**MMU**) qui est un dispositif matériel. Dans le schéma MMU, la valeur du **registre de translation** est additionnée à chaque adresse générée par un processus utilisateur au moment où elle est envoyée à la mémoire.

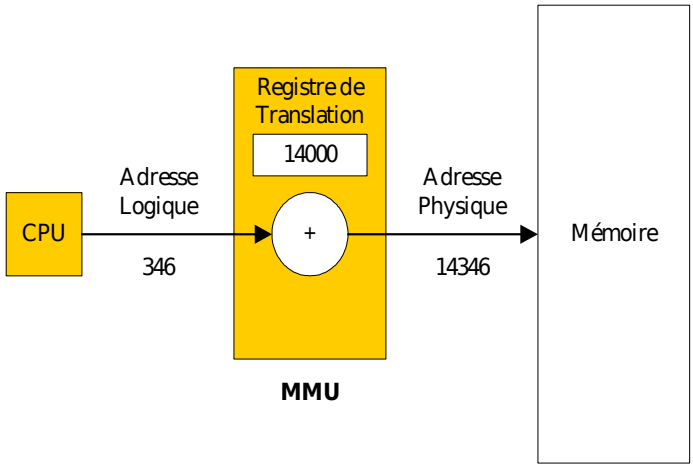


Figure : Translation Dynamique en Utilisant un Registre de Translation

Le concept d’un espace mémoire logique relié à un espace mémoire physique est crucial pour une bonne gestion mémoire.

12 Stratégie d’allocation

19.35 Allocation contiguë (Contiguous Allocation)

A. Monobloc (Single Contiguous Store Allocation)

Dans ce cas, la mémoire est subdivisée en **deux partitions** contiguës, une pour le système d’exploitation résident souvent placer en mémoire basse avec le vecteur d’interruptions et l’autre pour le processus utilisateur. Elle n'autorise qu'un seul processus actif en mémoire à un instant donné dont tout l’espace mémoire usager lui est alloué.

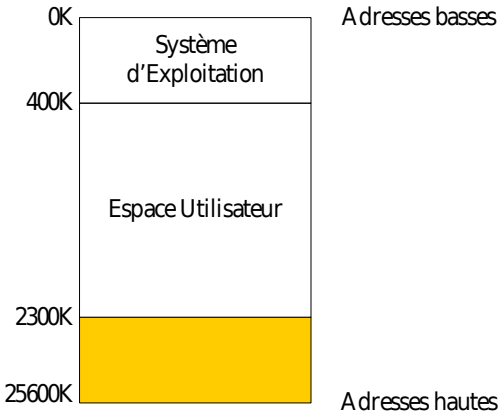


Figure : Organisation de la mémoire en Monobloc

Cette technique en voie de disparition est limitée à quelques micro-ordinateurs (**Ex.** CP/M et PC/DOS).

La gestion de la mémoire s'avère simple ; le système d'exploitation doit garder trace de deux zones mémoires. L'algorithme d'allocation peut être décrit par l'organigramme ci-contre :

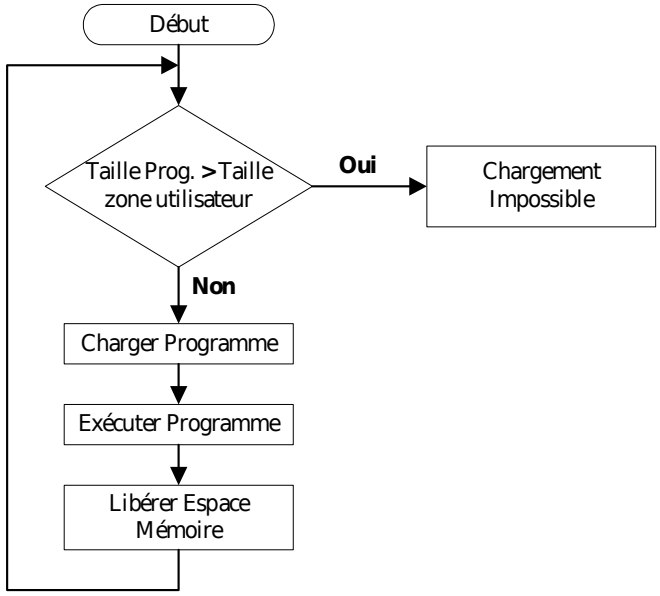


Figure : Algorithme d'Allocation mémoire d'un programme utilisateur

L'inconvénient majeur de cette stratégie est la **sous utilisation** (i.e. mauvaise utilisation) de la mémoire ; dans le sens où les programmes occupent (en général) partiellement l'espace usager. Par ailleurs, la taille des programmes usagers est limitée par la taille de l'espace usager.

B. Partitions multiples (Multiple-Partition Allocation)

Cette stratégie constitue une technique simple pour la mise en œuvre de la multiprogrammation. La mémoire principale est divisée en régions séparées ou

partitions mémoires ; chaque partition dispose de son espace d'adressage. Le partitionnement de la mémoire peut être **statique (fixe)** ou **dynamique (variable)**.

Chaque processus est chargé entièrement en mémoire. L'exécutable contient des adresses relatives et les adresses réelles sont déterminées au moment du chargement.

1. **Partitions fixes (Fixed Partition Store Allocation)**

   Cette solution consiste à diviser la mémoire en partitions fixes, de tailles pas nécessairement égales, à **l'initialisation du système**. Ce schéma a été utilisé à l'origine par le système OS/360 d'IBM (appelé **MFT**, **Multiprogramming with a Fixed number of Tasks**).

   Le système d'exploitation maintient une **table de description des partitions** (**PDT, Partition Description Table**) indiquant les parties de mémoire disponibles (**hole**) et celles qui sont occupées.

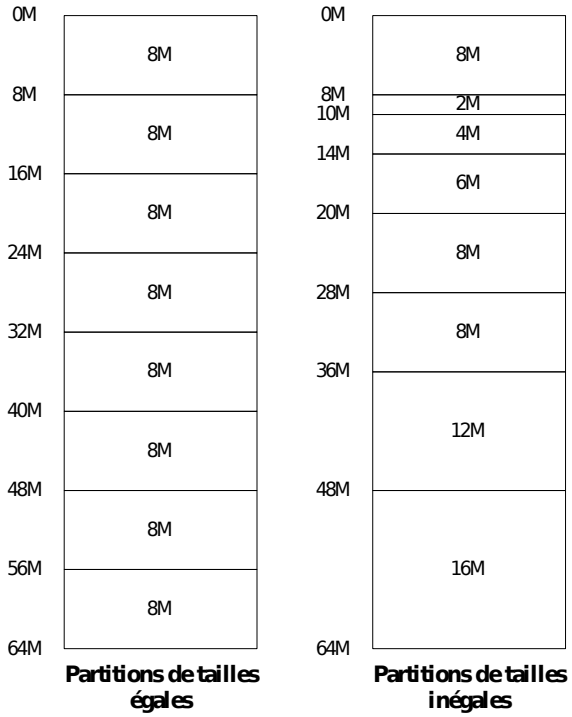


Table de description des partitions

Figure : Schéma d'allocation contiguë à partitions multiples fixes

- **Algorithme de placement**

  Les programmes n'ayant pu se loger en mémoire sont placés dans une **file d'attente** (une file unique ou une file par partition)

  a. **Utilisation de files multiples**

     Chaque nouveau processus est placé dans la file d'attente de la plus petite partition qui peut le contenir.

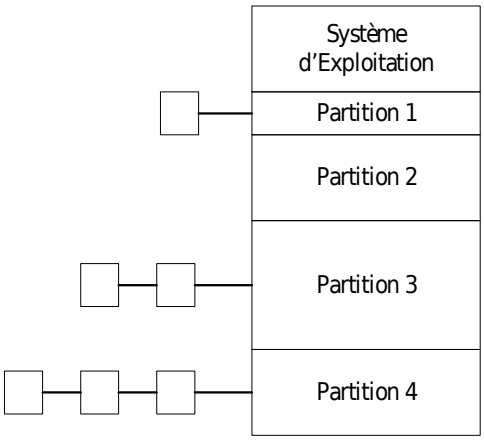


Figure : Partitions fixes avec files multiples

Cette technique est efficace seulement si les "tailles" ont un **nombre similaire** de programmes.

Les inconvénients de cette stratégie est qu'on perd en général de la place au sein de chaque partition et aussi, il peut y avoir des partitions inutilisées (leur file d'attente est vide).

b. Utilisation d'une seule file

Cette technique utilise une seule file d'attente globale.

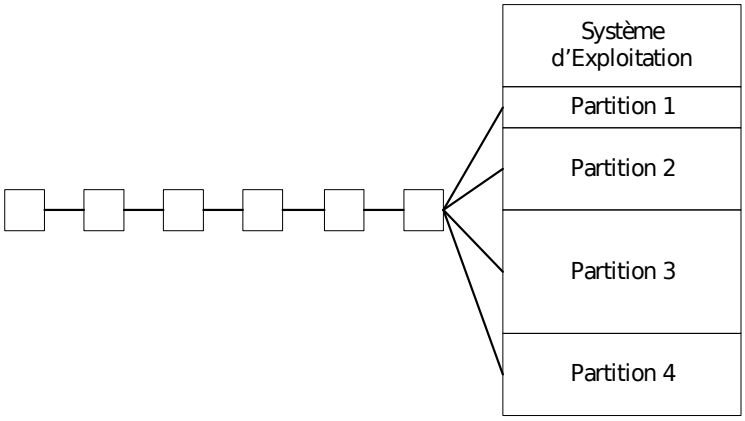


Figure : Partitions fixes avec une seule file

Il existe différentes stratégies d'attribution d'une partition libre à un processus en attente :

- Dès qu'une partition se libère, on lui affecte le **premier processus** de la file qui peut y tenir. L'inconvénient est qu'on peut ainsi affecter une partition de grande taille à un petit processus et perdre beaucoup de place.
- Dès qu'une partition se libère, on lui affecte le **plus grand processus** de la file qui peut y tenir. L'inconvénient est qu'on pénalise les processus de petite taille.

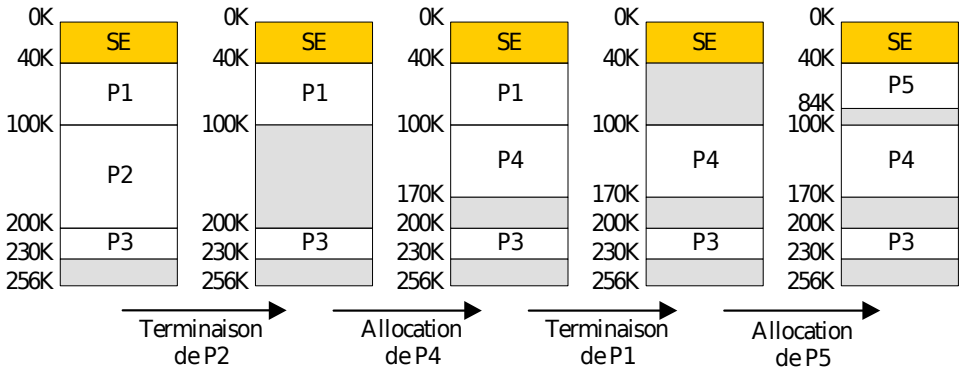
2. Partitions variables (Dynamic Partition Store Allocation)

Le gaspillage de mémoire et la fragmentation des systèmes à partitions fixes conduisirent à la conception de **partitions variables**. Dans ce cas, la mémoire est découpée **dynamiquement**, suivant la demande. Ainsi, à chaque programme est allouée une partition exactement égale à sa taille. Quand un programme termine son exécution, sa partition est récupérée par le système pour être allouée à un autre programme complètement ou partiellement selon la demande.

On n'est plus limité par des partitions trop grandes ou trop petites comme avec les partitions fixes. Cette amélioration de l'usage de la MC nécessite un mécanisme plus complexe d'allocation et de libération ; par exemple, il faut maintenir une liste des espaces mémoires disponibles.

Exemple

Soit une mémoire de 256K dont 40K sont occupés par le système d'exploitation. Ayant cinq jobs P1, P2, P3, P4 et P5 avec les besoins suivants en mémoire 60K, 100K, 30K, 70K et 44K respectivement. Cet exemple illustre la stratégie d'allocation en utilisant des partitions variables :



- **Etat de la mémoire**

  Pour gérer l'allocation et la libération de l'espace mémoire, le gestionnaire doit connaître l'état de la mémoire.

  - **Tables de bits (Bit Maps)**

    La mémoire est un ensemble d'unités d'allocation. Un bit est associé à chaque unité. Lorsqu'on doit ramener un processus de k unités, le gestionnaire de la mémoire doit alors parcourir la table des bits à la recherche de k zéros consécutifs.

    Cette méthode est rarement utilisée car la méthode de recherche est lente (k zéros consécutifs), et elle présente aussi l'inconvénient d'être relativement gourmande en espace mémoire, si on veut contrôler assez finement l'utilisation de la mémoire.

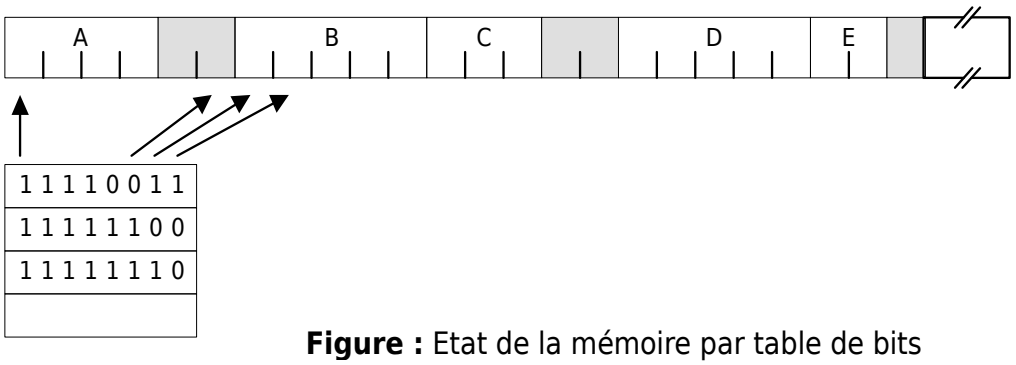


Figure : Etat de la mémoire par table de bits

– **Listes chaînées (Linked List)**

La mémoire est représentée sous forme d'une liste chaînée. Chaque entrée de la liste spécifie une zone libre (**Hole, H**) ou un processus (**Process, P**), son adresse de départ, sa longueur et un pointeur sur l'entrée suivante.

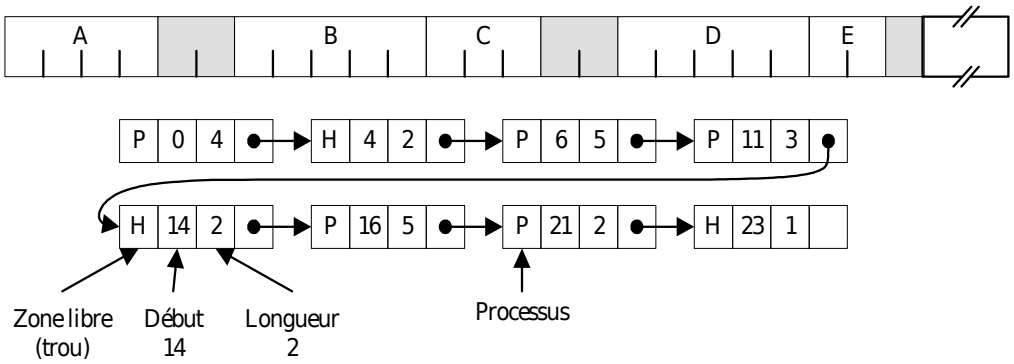


Figure : Etat de la mémoire par liste chaînée

▪ **Algorithme de placement**

Il existe plusieurs algorithmes afin de déterminer l'emplacement d'un programme en mémoire. Le but de tous ces algorithmes est de maximiser l'espace mémoire occupée, autrement dit, diminuer la probabilité de situations où un processus ne peut pas être servi, même s'il y a assez de mémoire.

– **First-fit (le premier trouvé) :** Le programme est mis dans le premier bloc de mémoire suffisamment grand à partir du début de la mémoire.

– **Next-fit (le prochain trouvé) :** Cet algorithme est une variante du précédent où le programme est mis dans le premier bloc de mémoire suffisamment grand à partir du dernier bloc alloué.

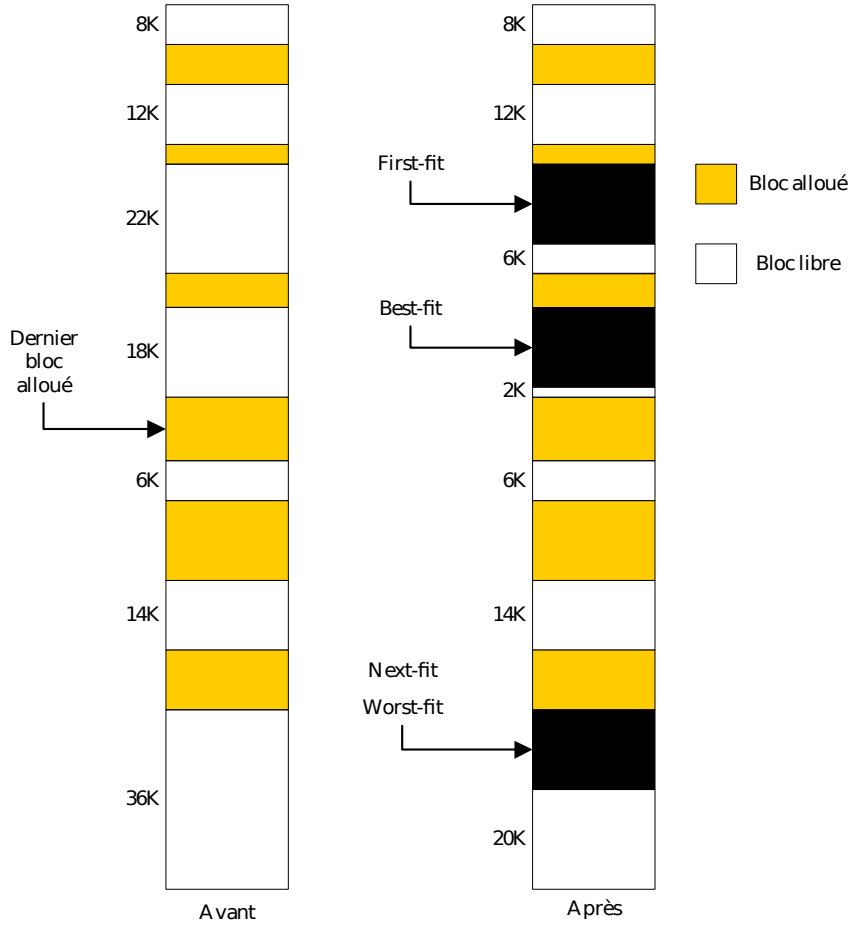
– **Best-fit (le meilleur choix) :** Le programme est mis dans le bloc de mémoire le plus petit dont la taille est suffisamment grande pour l'espace requis.

- **Worst-fit (le plus mauvais choix) :** Le programme est mis dans le bloc de mémoire le plus grand pour que la zone libre restante soit la plus grande possible.
- **Quick-fit (placement rapide) :** On utilise des listes de trous de même taille. On peut, par exemple, utiliser une table de n entrées, où la première entrée pointe sur la tête d'une liste chaînée des zones de 4K, la deuxième sur la tête d'une liste chaînée des zones de 8K, ...etc.

Même si Best-fit semble le meilleur, ce n'est pas l'algorithme retenu en pratique: il demande trop de temps de calcul. Par ailleurs, Next-fit crée plus de fragmentation que First-fit. Les simulations désignent First-fit, malgré sa simplicité apparente, comme la **meilleure stratégie**.

Exemple

Cet exemple illustre la configuration de la mémoire avant et après l'allocation d'un bloc de 16Kbyte en utilisant les différents algorithmes de placement.



3. Fragmentation mémoire

Les partitions multiples entraînent une fragmentation de la mémoire. Il y aurait suffisamment de mémoire libre pour charger un processus, mais

aucune partition n'est de taille suffisante. La fragmentation peut être de deux types :

- **Fragmentation interne (Internal Fragmentation)**

  La mémoire allouée peut être légèrement plus grande que la mémoire requise. Cette différence est appelée **fragmentation interne** – de la mémoire qui est interne à une partition mais n'est pas utilisée.

- **Fragmentation externe (External Fragmentation)**

  La fragmentation externe se présente quand il existe un espace mémoire total suffisant pour satisfaire une requête, mais il n'est pas contigu ; la mémoire est fragmentée en un grand nombre de petits trous (i.e. blocs libres) où un programme ne peut être chargé dans aucun de ces trous.

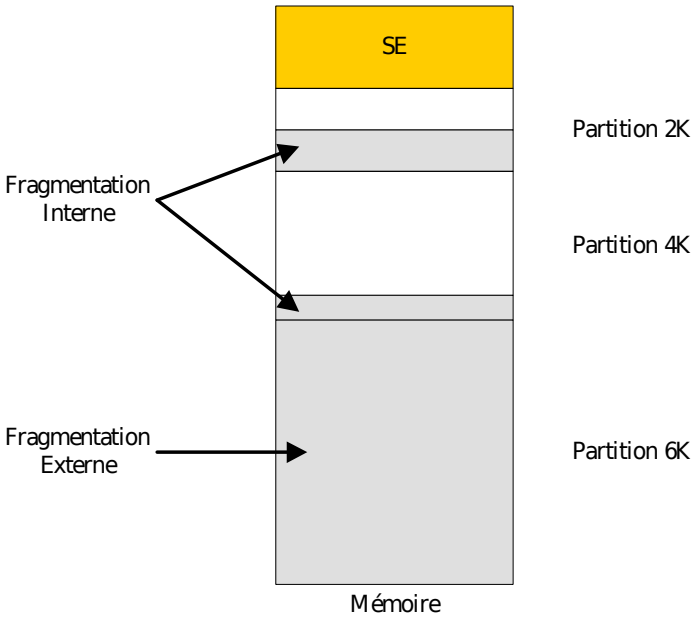


Figure : Fragmentation interne et externe

4. Compactage (Compaction)

Le compactage est une solution pour la fragmentation externe qui permet de regrouper les espaces inutilisés (i.e. les blocs libres) dans une partie de la mémoire (**Ex.** début, milieu). Le compactage entraîne un changement d'adresse. Par conséquent, les adresses du programme doivent être mise à jour. **Très coûteuse** en temps CPU, cette opération est effectuée le moins souvent possible.

Le compactage est possible seulement si la **translation des adresses est dynamique** et si elle est effectuée au moment de l'exécution.

Les programmes sont déplacés en mémoire de façon à réduire à 1 seul grand trou plusieurs petits trous disponibles.

L'opération de compactage est effectuée quand un programme qui demande d'être exécuté ne trouve pas une partition assez grande, mais sa taille est plus petite que la fragmentation externe existante

La compaction n'est effectuée que si l'on utilise les registres base et étendue et elle nécessite l'arrêt de l'exécution des processus.

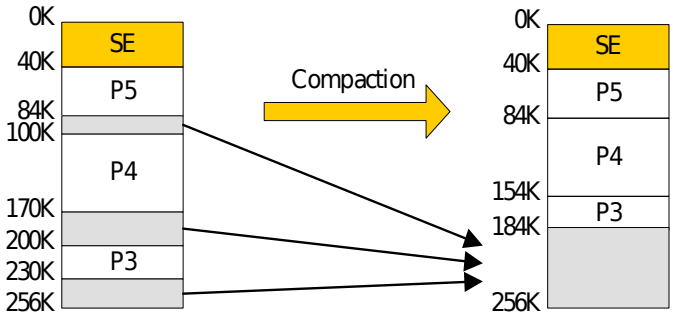


Figure : Compactage

5. Va-et-vient (Swapping)

Puisque la mémoire ne peut pas contenir les processus de tous les utilisateurs, il faut placer quelques uns sur le disque, en utilisant le **système de va-et-vient**. Il consiste en le **transfert** de blocs mémoire de la mémoire secondaire à la mémoire principale ou vice-versa (**Swapping**).

Le va-et-vient est mis en œuvre lorsque tous les processus ne peuvent pas tenir simultanément en mémoire. Un processus qui est inactif (soit bloqué, soit préempté) peut donc être déplacé temporairement sur une partie réservée du disque, appelée **mémoire de réserve** (**Swap Area** ou **Backing Store**), pour permettre le chargement et donc l'exécution d'autres processus. Cette opération est connue sous le nom de **Swap-Out**. Le processus déplacé sur le disque sera ultérieurement rechargé en mémoire pour lui permettre de poursuivre son exécution ; on parle dans ce cas d'une opération **Swap-In**.

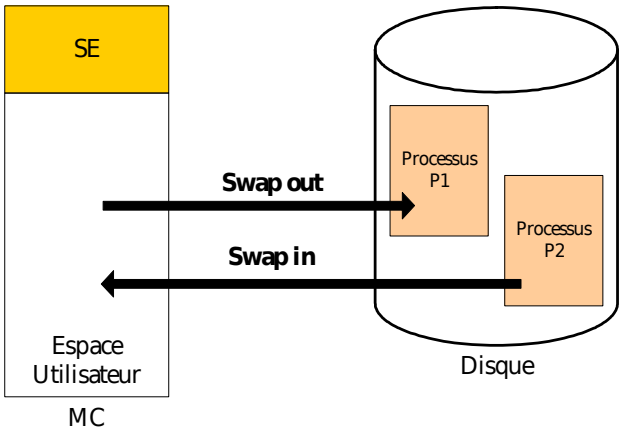


Figure : Système de swapping

Les stratégies de choix des processus à écrire sur disque sont sensiblement identiques à celles mises en œuvre pour l'ordonnancement des processus.

Sur le disque, la zone de va-et-vient d'un processus peut être allouée à la demande. Quand un processus est déchargé de la mémoire centrale, on lui recherche une place. Les places de va-et-vient sont gérées de la même manière que la mémoire centrale.

La zone de va-et-vient d'un processus peut aussi être allouée une fois pour toute au début de l'exécution. Lors du déchargement, le processus est sûr d'avoir une zone d'attente libre sur le disque.

19.36 Allocation non contiguë (Non Contiguous Allocation)

L'objectif principal de l'allocation non contiguë est de pouvoir charger un processus en exploitant au mieux l'ensemble des trous mémoire.

Un programme est divisé en morceaux dans le but de permettre l`allocation séparée de chaque morceau. Les morceaux sont beaucoup plus petits que le programme entier et donc permettent une utilisation plus efficace de la mémoire. Par conséquent, les petits trous peuvent être utilisés plus facilement.

Les gestionnaires mémoire modernes utilisent deux mécanismes fondamentaux basés sur la réimplantation dynamique d'adresse ; à savoir : la **pagination** et la **segmentation**. La segmentation utilise des parties de programme qui ont une valeur logique (des modules). Par contre, la pagination utilise des parties de programme arbitraires (division du programme en pages de longueur fixe). Ces deux techniques peuvent être combinées, on parle dans ce cas de la technique de **segmentation paginée**.

A. Pagination (Paging)

1. Principe de la pagination

Dans le mécanisme de pagination, l'espace d'adressage du programme est découpé en petits blocs de même taille appelés **pages**. L'espace de la mémoire physique est lui-même découpé en blocs de **taille fixe** appelés **cases** ou **cadres de page (Frame Page) ;** ce qui facilitera la correspondance d'une page à un frame**.**

La taille d'une case est égale à la taille d'une page. Cette taille est définie par le matériel, comme étant une **puissance de 2**, variant entre 512 octets et 8192 octets, selon l'architecture de l'ordinateur. Le choix d'une puissance de 2 comme taille de page **facilite** particulièrement **la traduction** d'une adresse logique en un numéro de page et un déplacement dans la page.

Les pages logiques d'un processus peuvent donc être assignées aux cadres disponibles n'importe où en mémoire principale. Dans ce cas, un processus peut se retrouver éparpillé n'importe où dans la mémoire physique.

Un schéma de pagination permet d'éliminer la fragmentation externe car toutes les pages sont de même taille. Cependant, une fragmentation interne peut se produire si la dernière page de l'espace d'adressage logique n'est pas pleine.

2. Schéma de translation d'adresses

L'association d'une page logique avec une page physique est décrite dans une table appelée **table de pages (Page Table).**

La translation des adresses logiques en adresses physiques est à la charge de la **MMU**. Le support matériel pour la pagination est montré dans la figure ci-après :

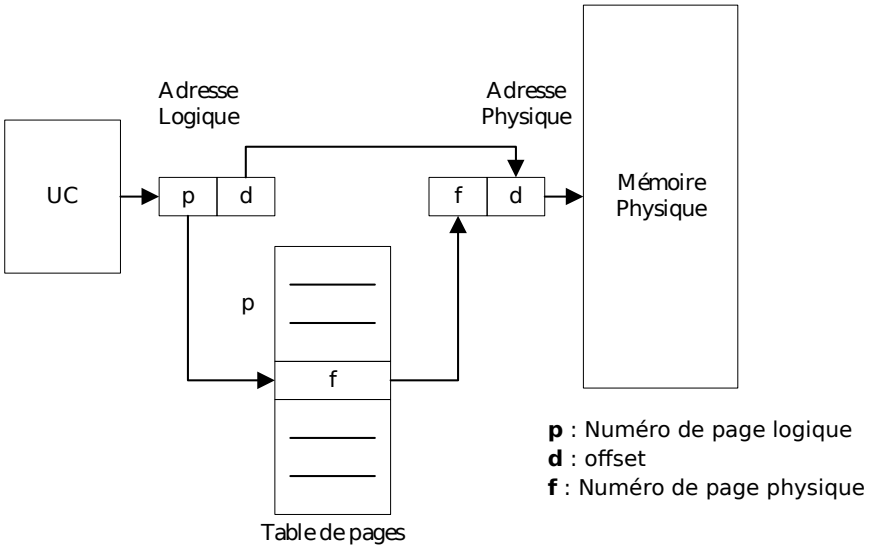


Figure : Matériel pour la pagination

On divise chaque adresse (logique) générée par l'UC en deux parties :

- Un **numéro de page** (**p, Page Number**) qui est utilisé comme un index dans la table des pages. La table des pages contient l'adresse de base de chaque page dans la mémoire physique.
- Un **déplacement dans la page** (**d, Page Offset**) qui est combinée à l'adresse de base de la page pour définir l'adresse mémoire physique qui sera envoyée à l'unité mémoire.

Pour un espace d'adressage logique de $\mathbf{2^m}$ **octets**, en considérant des pages de $\mathbf{2^n}$ **octets**, les **m - n premiers bits** d'une adresse logique correspondent au numéro de page **p** et les **n bits** restants au déplacement **d** dans la page.

Numéro de page	Déplacement dans la page
p	d
m - n	n

Donc, si **T** est la taille d'une page et **U** une adresse logique, alors l'adresse paginée **(p,d)** est déduite à partir des formules suivantes :

p = U Div T (où, Div est la division entière)

d = U Mod T (où, Mod est le reste de la division)

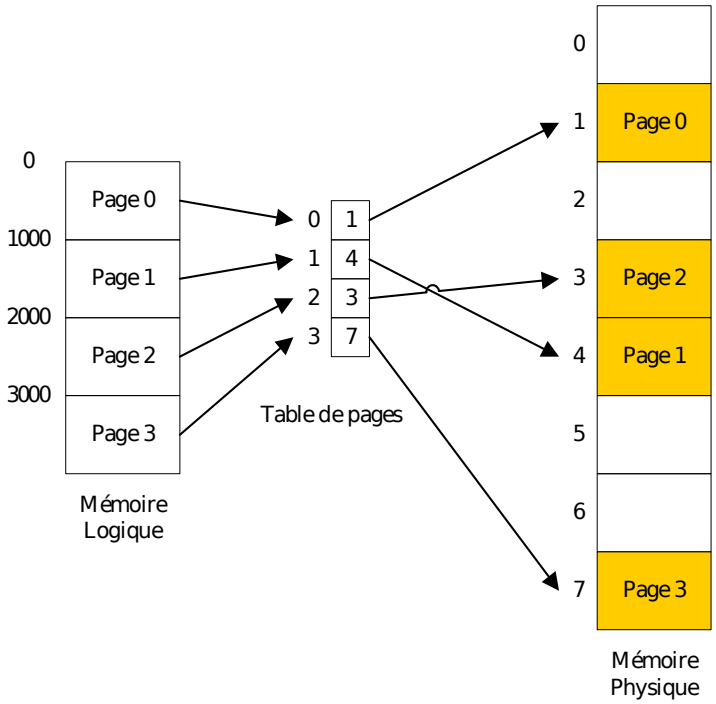


Figure : Modèle de pagination de la mémoire logique et physique

3. Table des pages

La traduction d'une adresse logique en une adresse physique nécessite une table de pages contenant l'équivalence **<Numéro de page logique, Numéro de page physique>.**

Chaque processus dispose de sa propre table de pages. Par conséquent, chaque opération de commutation de contexte se traduit également par un changement de table de pages, de manière à ce que la table active corresponde à celle du processus actif.

La table des pages est généralement maintenue en mémoire centrale. Il existe un registre machine spécial de la MMU pointant vers la table de pages du processus actif, appelé **PTBR** (**Page-Table Base Register**). Chaque processus sauvegarde dans son **PCB** la valeur de PTBR correspondant à sa table.

Dans un système paginé, il faut donc deux accès mémoire pour chaque référence mémoire (p,d) :

- Un premier accès permet de lire l'entrée de la table des pages correspondant à la page **p** cherchée et délivre une adresse physique **c** de case dans la mémoire centrale.
- Un second accès est nécessaire à la lecture ou l'écriture de l'octet recherché à l'adresse **c+d**.

Pour accélérer les accès à la mémoire centrale et compenser le coût lié à la pagination, un **cache associatif** (**TLB, Translation Look-aside Buffers**) est utilisé. Le TLB est une **table de registres** très rapides à l'intérieur du

CPU (i.e. MMU) qui contient quelques entrées de la table de pages les plus **récemment accédés**.

Lors de la traduction d'une adresse, on essayera d'abord de la traduire par l'intermédiaire du TLB. Si la page était présente on ne sollicite pas la table de pages, et on fait la traduction immédiatement : un **seul accès mémoire** est alors nécessaire pour accéder à l'octet recherché. Sinon, on fait une recherche dans la table de pages, et on met à jour le TLB pour d'éventuelles références futures. Cette mise à jour suppose que l'on éjecte une page qui était déjà présente dans le cache si ce dernier est plein.

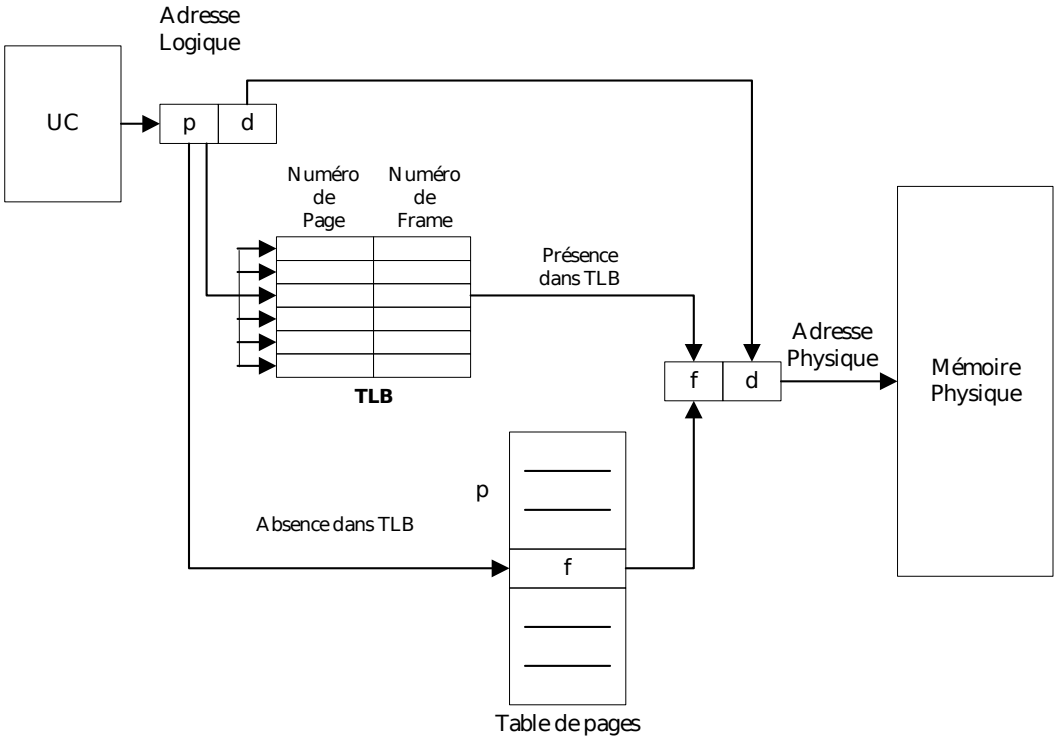


Figure : Matériel pour la pagination avec TLB

Exemple

On suppose disposer des paramètres suivants :

$\mathbf{T_{MC}}$ qui représente le temps d'accès à la mémoire centrale

$\mathbf{T_{TLB}}$ qui représente le temps d'accès à la mémoire associative

P qui représente la probabilité pour que l'entrée de la page référencée soit dans la mémoire associative.

Donnez, en fonction de ces trois paramètres, le temps d'accès effectif (moyen) à la mémoire paginée : **Tamp** = $P*(T_{MC}+T_{TLB}) + (1-P)*(2*T_{MC}+T_{TLB})$

B. Segmentation

1. Principe de la segmentation

Un aspect fondamental de la mémoire paginée est la séparation de la vue de l'utilisateur de la mémoire et la mémoire physique réelle. Or en général, un processus est composé d'un ensemble d'unités logiques :

- Les différents codes : le programme principal, les procédures, les fonctions bibliothèques.
- Les données initialisées.
- Les données non initialisées.
- La pile d'exécution.

Cette vue utilisateur n'est pas reflétée dans une mémoire paginée. La segmentation est une stratégie de gestion mémoire qui reproduit le découpage mémoire tel qu'il est décrit logiquement par l'usager.

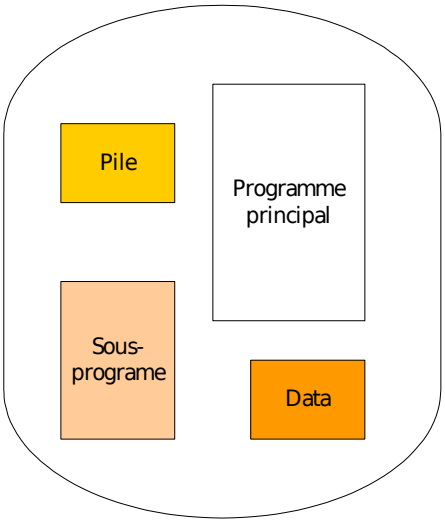


Figure : Vue de l'utilisateur d'un programme

Dans une mémoire segmentée, chaque unité logique d'un programme usager est stockée dans un bloc mémoire, appelé « segment » à l'intérieur duquel les adresses sont relatives au début du segment. Ces segments sont de **tailles différentes**. Un programme sera donc constitué d'un ensemble de segments de code et de données, pouvant être dispersés en MC.

La segmentation facilite l'édition de liens, ainsi que le partage entre processus de segments de données ou de codes.

Contrairement au schéma de pagination, la segmentation permet d'éliminer la fragmentation interne car l'espace mémoire alloué à un segment est de taille égale exactement à la taille du segment. Cependant, une fragmentation externe peut se produire due au fait qu'on fait une allocation dynamique de l'espace mémoire.

2. Schéma de translation d'adresses

Chaque segment est repéré par son numéro **S** et sa longueur variable **L**. Un segment est un ensemble d'adresses logiques contiguës.

Une adresse logique est donnée par un couple **(S, d)**, où **S** est le numéro du segment et **d** le déplacement dans le segment.

L'association d'une adresse logique à une adresse physique est décrite dans une table appelée **table de segments (Segment Table).**

Chaque entrée de la table de segments possède un **segment de base** et un **segment limite**. Le segment de base contient l'adresse physique de début où le segment réside en mémoire, tandis que le registre limite spécifie la longueur du segment.

La translation des adresses logiques en adresses physiques est à la charge de la **MMU**. Le support matériel pour la segmentation est montré dans la figure ci-après :

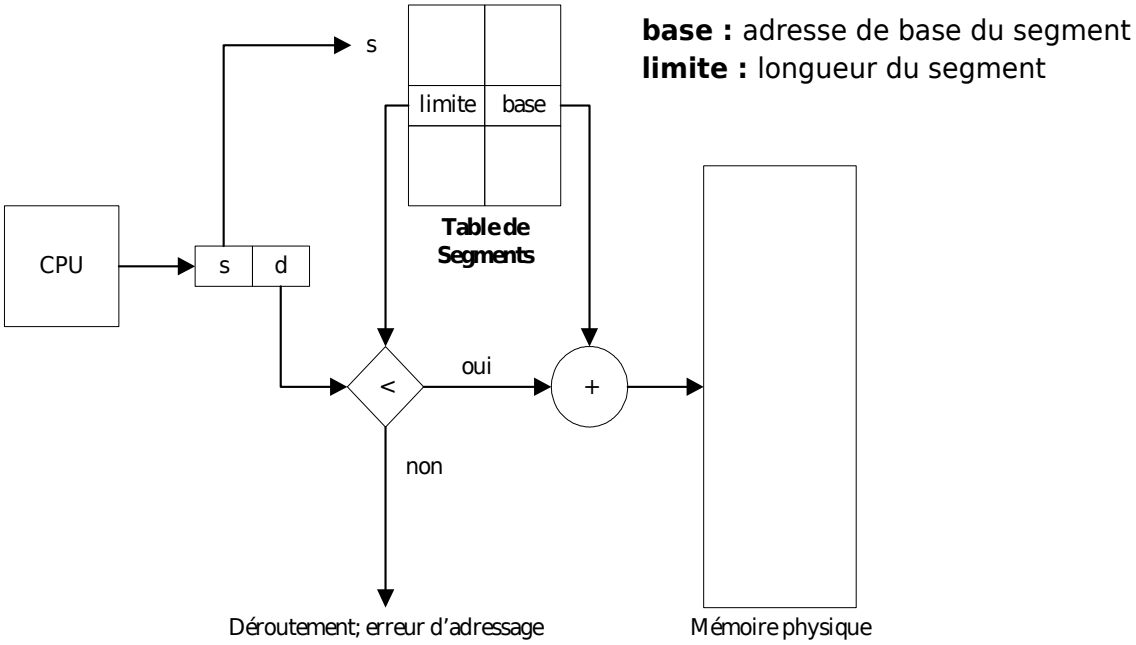


Figure : Matériel pour la segmentation

Le déplacement **d** dans le segment doit être compris entre 0 et la limite du segment (**i.e. d → [0, L [**) et dans ce cas **l'adresse physique est calculée** en additionnant la valeur de **d** à la base du segment. Dans le cas contraire, une **erreur d'adressage** est générée.

Exemple

Sur un système utilisant la segmentation simple, calculez l'adresse physique de chacune des adresses logiques, à partir de la table des segments ci-après.

On suppose que les adresses soient décimales au lieu de binaires.

Segment	Base	Limite
0	1100	500

1	2500	1000
2	200	600
3	4000	1200

Adresse logique	Adresse physique
<0, 300>	300<500, donc @=1100+300=1400
<2, 800>	800>600, donc Erreur d'adressage
<1, 600>	600<1000, donc @=2500+600=3100
<3, 1100>	1100<1200, donc @=4000+1100=5100
<1, 1111>	1111>1000, donc Erreur d'adressage

3. Table des segments

D'une manière similaire à la table des pages, la table des segments peut être implantée par des registres rapides, le PDP11/45 utilise cette approche ; il possède huit registres de 16 bits dont 3 bits pour le numéro de segment et 13 bits pour le déplacement segment. Ce qui permet de référencer 8 segments et chaque segment peut adresser jusqu'à 8k octets.

Quand le nombre de segments est élevé, cette méthode n'est pas utilisable. La table des segments est alors maintenue en mémoire. Un **registre de base de la table de segments** (**Segment Table Base Register, STBR**) pointe vers la table de segments. De même, comme le nombre de segments utilisés par un programme peut varier largement, un **registre de la longueur de la table des segments** (**Segment Table Limit Register : STLR**) est utilisé.

Chaque processus sauvegarde dans son **PCB** la valeur de STBR correspondant à sa table de segments. Il sauvegarde aussi le nombre de segments dans le processus. Au moment de la commutation de contexte, ces informations seront chargées dans les registres appropriés d'UC.

Pour une adresse logique **(S, d)**, on vérifie d'abord que le numéro de segment **S** est valide avec la condition **S<STLR**. Si la condition est vérifiée, on additionne le numéro du segment au **STBR**, en obtenant l'adresse **(STBR+S)** en mémoire de l'entrée de la table de segments. Ensuite, on vérifie la validité du déplacement dans le segment avec la condition **d < [(STBR+S).limite]**. Si cette condition est vérifie, on calcule l'adresse physique de l'octet désiré comme **(STBR+S).base+d**.

Il faut donc deux accès mémoire par adresse logique.

C. Segmentation paginée

La taille d'un segment peut être importante, d'où un temps de chargement long qui peut en résulter. La pagination des segments peut être une solution. Cette technique a été inventée pour le système Multics.

Les programmes sont divisés en segments et chaque segment est paginé dans un espace d'adressage linéaire.

A chaque processus, on associe une table de segments et une table de pages. Donc chaque adresse de segment n`est pas une adresse de mémoire, mais une adresse au tableau de pages du segment

Une adresse logique **(S, D)**, avec **S** numéro de segment et **D** déplacement dans le segment, est transformée en **(S, p, d)**, où **p** est un numéro de page et **d** un déplacement dans la page **p**. chaque adresse devient alors un triplet **(S, p, d)**.

La traduction d'une adresse logique en adresse physique se fait selon le schéma suivant :

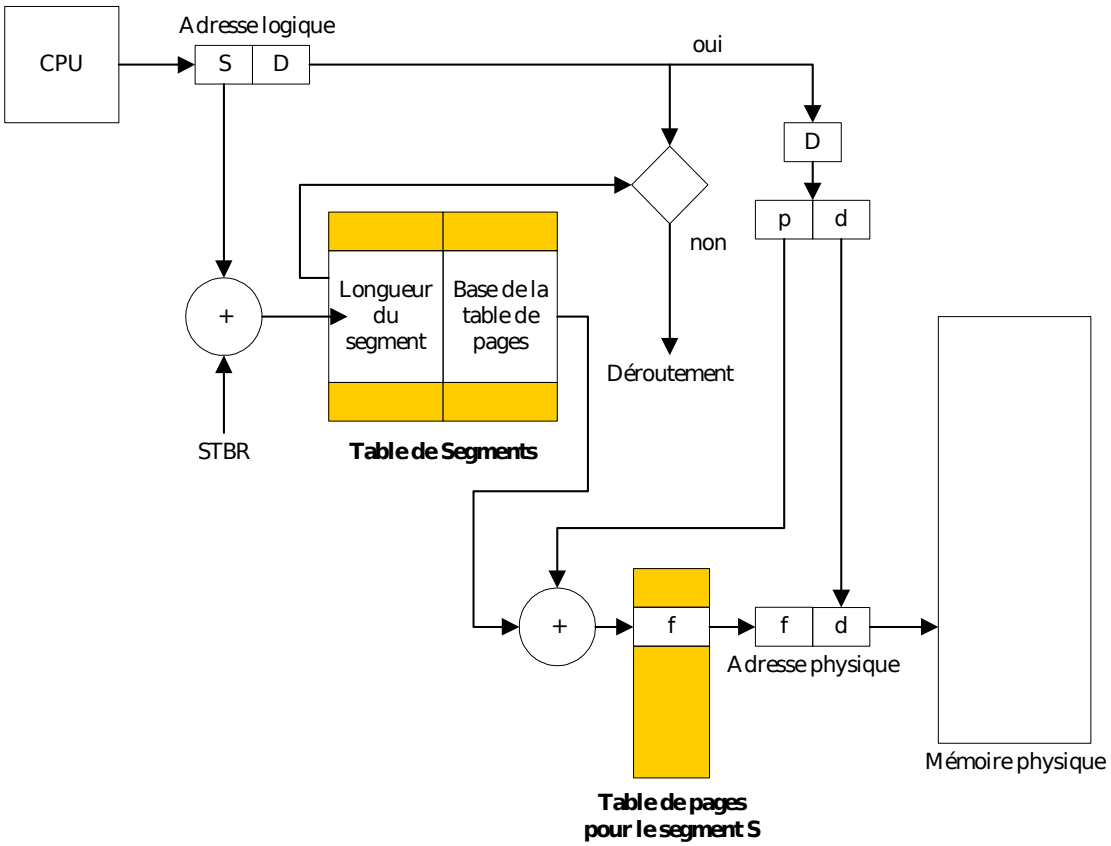


Figure : Matériel pour la segmentation paginée

L'utilisation d'une mémoire associative, stockant des triplets **(S, p, adresse du frame)**, gérée comme cache contenant les adresses des pages les plus récemment désignées, peut accélérer la recherche des frames.

Bien que cette technique élimine la fragmentation externe, elle introduit de nouveau la fragmentation interne.

13 Mémoire virtuelle

La **mémoire virtuelle** (**Virtual Memory**) est une technique qui permet l'exécution de processus pouvant ne pas être dans sa totalité en MC.

La mémoire virtuelle est la **séparation** entre la mémoire logique de l'utilisateur et la mémoire physique. Ceci rend possible l'exécution de processus dont la taille est beaucoup plus grande que la taille de la mémoire physique.

Un processus est donc constitué de morceaux (pages ou segments) ne nécessitant pas d'être tous en MC durant l'exécution. À fin qu'un programme soit exécuté, seulement les morceaux qui sont en exécution ont besoin d'être en MC. Les autres morceaux peuvent être sur mémoire secondaire (**Ex.** disque en général), prêts à être amenés en mémoire centrale sur demande.

La capacité d'exécuter un processus se trouvant partiellement en mémoire présenterait **plusieurs avantages** :

- La taille d'un programme n'est plus limitée par la taille de la mémoire physique.
- Comme chaque programme utilisateur pourrait occuper moins de mémoire physique, il serait possible d'exécuter plus de programmes en même temps.
- On a besoin de moins d'E/S pour effectuer des chargements ou des swappings en mémoire d'un programme utilisateur.

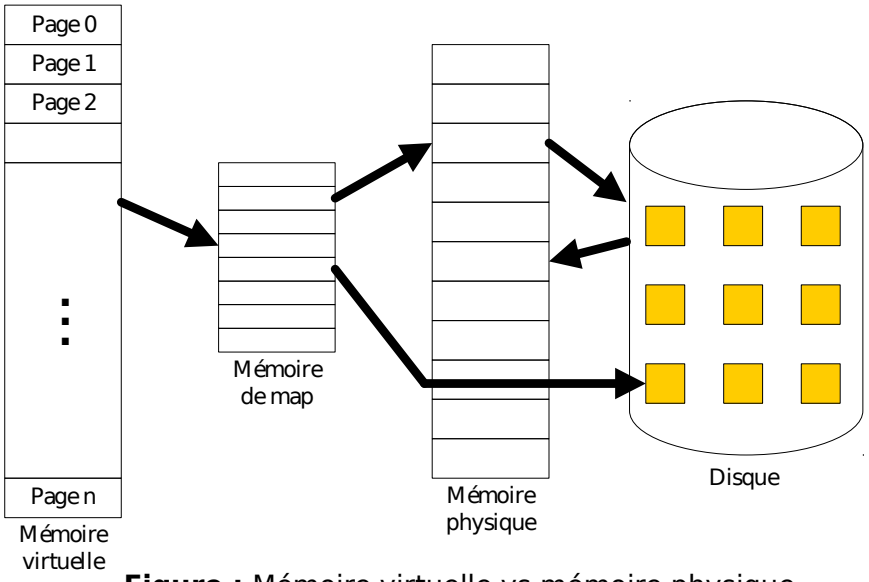


Figure : Mémoire virtuelle vs mémoire physique

19.37 Recouvrement

Le **recouvrement** (**Overlay**) est une technique qui permet de remplacer une partie de la MC par une autre.

À un instant donné, un programme n'utilise qu'une petite partie du code et des données qui le constituent. Les parties qui ne sont pas utiles en même temps

peuvent donc se **“recouvrir”** ; c'est-à-dire, occuper le **même emplacement** en mémoire physique.

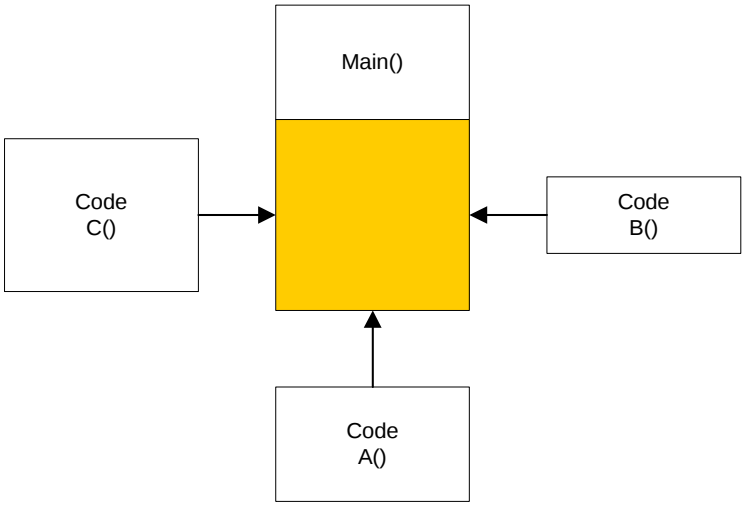
Cette technique consiste à permettre à l'utilisateur de diviser son programme en segments, appelés **segments de recouvrement** (**Overlays**) et à charger un segment en mémoire. Le segment 0 s'exécute en premier. Lorsqu'il se termine, il appelle un autre segment de recouvrement qui sera chargé dans le même emplacement mémoire.

Exemple

Soit un programme **P** constitué de la procédure principale **Main()** et de trois (03) autres procédures **A()**, **B()** et **C()**. Les procédures **A()**, **B()** et **C()** ne s’appellent pas entre elles, ce qui fait qu’elles ne sont pas nécessaires en même temps en MC. Donc, à tout moment, on a besoin de charger en MC que **Main() & A()** ou **Main() & B()** ou sinon **Main() & C()** afin d’optimiser l’utilisation de l’espace mémoire.

La solution consiste à sauvegarder le code des ces procédures sur disque et ensuite le chargé dans le même espace mémoire quand ceci est nécessaire ; c'est-à-dire à l’appel d’une procédure son code sera chargé en mémoire centrale en écrasant le code de la procédure déjà appelée.

Les procédures A(), B() et C() peuvent se recouvrir.



L’inconvénient majeur de cette technique est la lenteur d’exécution due au temps d’E/S (Ex. la lecture de B() se fait après l’exécution de A() si on appelle B() après avoir appelé A()). Par ailleurs, le programmeur doit se charger du découpage de son programme en segments de recouvrement. Ce qui nécessite une connaissance parfaite de la structure de son programme et de ses données. Quand la taille des programmes est grande, cette connaissance devient difficile.

19.38 Pagination à la demande

Le principe de la mémoire virtuelle est couramment implémenté avec la pagination à la demande (**On-Demand Paging**) ; c'est-à-dire que les pages des processus ne sont chargées en MC que lorsque le processeur demande à y accéder.

La pagination à la demande est semblable à un système de pagination avec va-et-vient (swapping). La figure ci-dessous illustre l'action de va-et-vient :

Figure : Pagination avec swapping

Cependant, plutôt que de transférer le processus en entier dans la mémoire, on utilise un **swapping paresseux** (**Lazy Swapping**) qui ne transfère une page en mémoire que si l'on a besoin.

Comment le processeur puisse distinguer entre les pages qui sont en mémoire, et celles qui sont sur disque afin de détecter leur éventuelle absence ?

Chaque entrée de la table des pages comporte un champ supplémentaire, le **bit validation V** (**Valid-Invalid Bit**), qui est à **Valide** (i.e. **1**) si la page est effectivement présente en MC, **Invalide** (i.e. **0**) sinon. Initialement, ce bit est **initialisé à Invalide** pour toutes les entrées de la table.

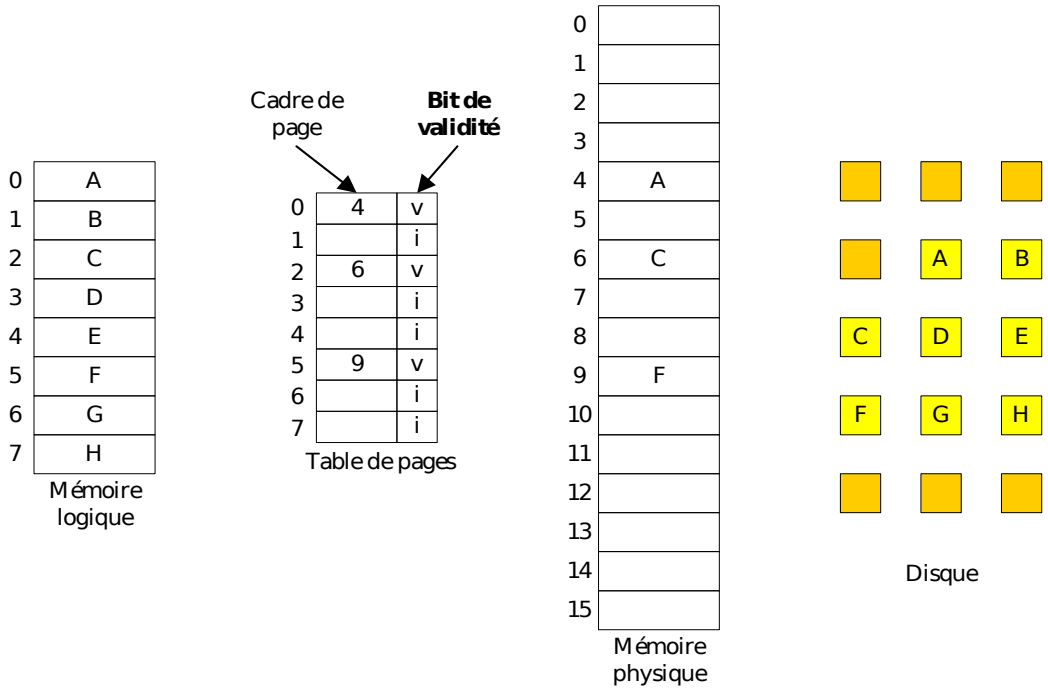


Figure : Tables de pages ave pages absentes

Mais que se passe-t-il si le processus essaye d'utiliser une page qui n'est pas chargée en mémoire ?

L'accès à une page marquée invalide provoque un déroutement de **défaut de page** (**Fault Page**) vers le système d'exploitation. La procédure permettant de manipuler ce défaut de page est la suivante :

1. Déterminer si la référence mémoire est valide. Dans le cas négatif ; c'est-à-dire l'adresse ne se trouve pas dans l'espace d'adressage, il s'agit d'une erreur d'adressage.
2. S'il s'agit d'une référence valide mais la page n'est pas encore chargée en mémoire, on doit la charger.
3. Trouver un frame libre pour charger la page manquante.
4. Lancer une opération d'E/S pour lire la page manquante à partir de l'unité de swapping.
5. Mise-à-jour de la table de pages.
6. Redémarrer l'instruction interrompue, le processus peut alors accéder à la page.

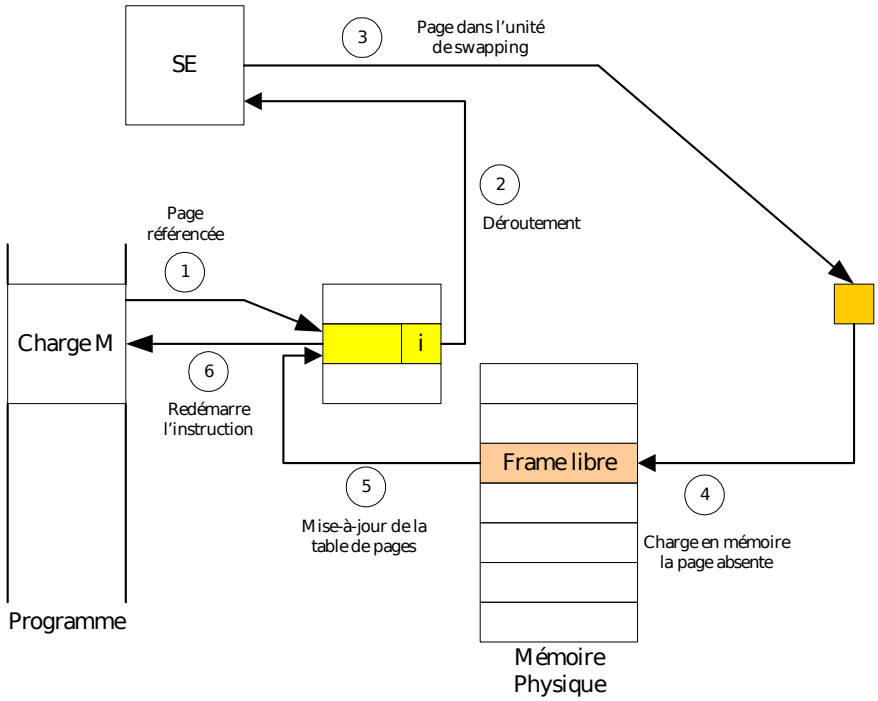


Figure : Les étapes de prise en charge d'un défaut de page

19.39 Performance de la pagination à la demande

La pagination à la demande influe sur le temps d'accès effectif et par conséquent sur les performances du système.

Si on désigne par **ma** le temps d'accès à la mémoire, par **p** la probabilité d'un défaut de page ($0 \in p \leq 1$), et par **Tdp** le temps de traitement d'un défaut de page, alors le temps d'accès effectif (**Tae**) est égal :

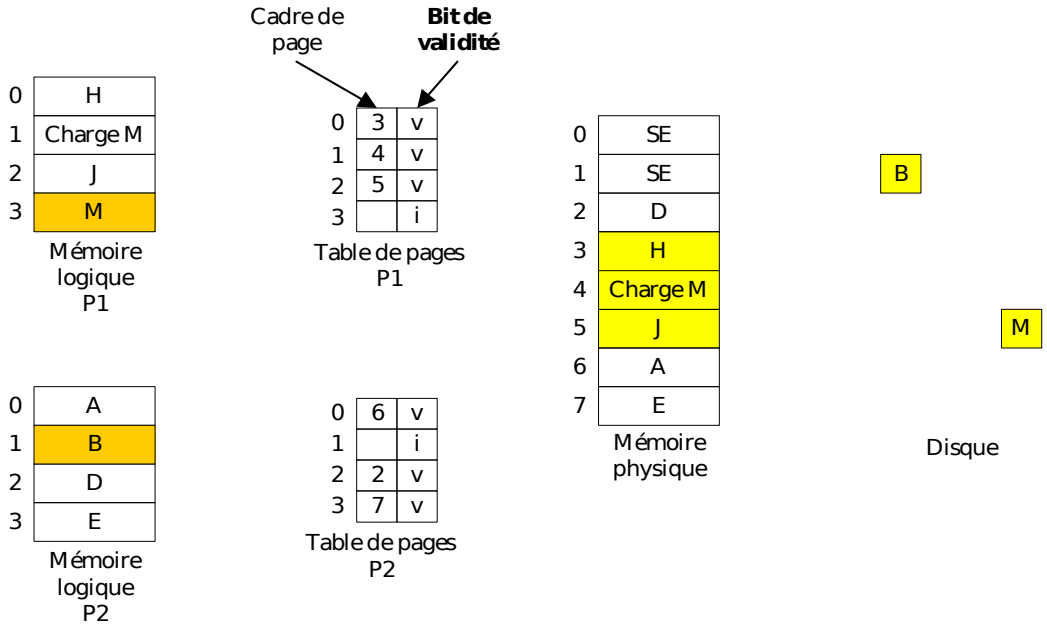
$$\mathbf{Tae = (1-p) * ma + p * Tdp}$$

19.40 Remplacement de page

L'algorithme précédent suppose qu'il existe toujours une case libre en mémoire centrale. Et s'il n'existe pas de cases (frames) libres?

Exemple

Sur cet exemple, si le système désire charger la page 3 de P1 ou la page 1 de P2, il va se rendre compte de l'absence d'un frame libre en MC.



S'il n'y a pas de cadres libres en mémoire, on doit retirer une page de la mémoire pour la remplacer par celle demandée. On dit qu'il y a **remplacement de page** (**Page Replacement**). La page retirer de la mémoire est dite **page victime**.

La procédure de traitement de défaut de page, avec prise en compte de remplacement de page, se présente maintenant comme suit :

1. Trouver l'emplacement de la page désirée sur le disque (unité de swapping).
2. Trouver un frame libre :
    i. S'il existe, l'utiliser.
    ii. Sinon, utiliser un algorithme de remplacement de pages pour sélectionner un frame victime.
    iii. Recopier la page victime sur le disque et mettre à jour la table de pages.
3. Charger la page désirée dans le frame récemment libéré et mettre à jour la table de pages.
4. Redémarrer le processus interrompu.

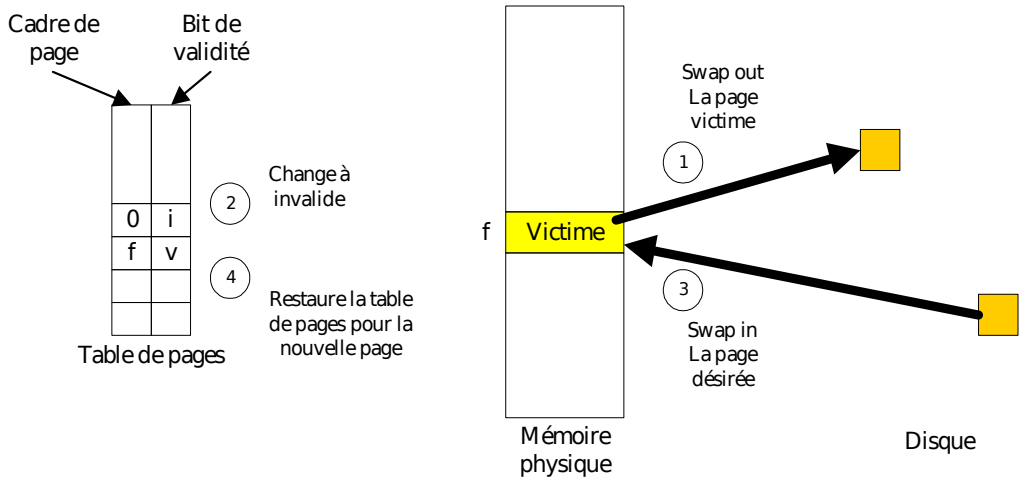


Figure : Remplacement de pages

On remarque qu'on a besoin de **deux** (**02**) **transferts** de pages pour effectuer un remplacement de pages (swap-in et swap-out). Ceci fait augmenter le temps de traitement du défaut de pages, et par conséquent le temps d'accès effectif.

Afin de réduire ce temps, une solution consiste à **ne réécrire la page victime** sur le disque que si elle a été **modifiée** depuis son chargement en mémoire. Cette solution est matérialisée par l'ajout d'un **bit de modification** (**Modify Bit** ou **Dirty Bit**) à chaque entrée de la table de pages. Ce bit est mis à **1** si la page a été modifiée, sinon il est mis à **0**.

Le choix de la page victime peut se faire de deux manières : remplacement global ou remplacement local.

- **Remplacement local**

  La page victime doit forcément appartenir à l'espace d'adressage du processus en défaut de page.

- **Remplacement global**

  La page victime peut appartenir à l'espace d'adressage de n'importe lequel des processus présents en MC.

  Cette politique est la plus souvent mise en œuvre car elle donne de meilleurs résultats quant à l'utilisation de la MC.

- **Pré-chargement**

  Le pré-chargement (**Prepaging**) ou l'**allocation par anticipation** tente de prédire les pages que le processus utilisera et les charge ont leurs références avant qu'elles soient réellement référencées, évitant ainsi les défauts de page, et un blocage du processus demandeur.

  Cette prédiction permet donc de réduire les défauts de pages et par conséquent améliorer les temps d'exécution. Cependant, le problème réside dans la difficulté de prédire efficacement les pages qui vont être référencées.

- **Algorithmes de remplacement de pages**

Il existe plusieurs algorithmes de remplacement de pages. Ces algorithmes sont conçus dans l'objectif de **minimiser le taux de défaut de pages** à long terme.

L'évaluation de ces algorithmes s'effectue en comptant sur une même séquence de références mémoires, le nombre de défauts de pages provoqués. Cette séquence est appelée **chaîne de références** (**Reference String**) et elle est définie par la séquence des numéros de pages référencées successivement au cours de l'exécution d'un processus.

Exemple

Soit une mémoire paginée sont la taille d'une page est de 100 octets. L'exécution d'un processus fait référence aux adresses suivantes :

100, 432, 101, 612, 102, 103, 101, 611, 102, 103, 104, 610, 102, 103, 104, 101, 609, 102, 105

Cette séquence est réduite à la chaîne de références suivante :

1, 4, 1, 6, 1, 6, 1, 6, 1, 6, 1

Afin de déterminer le nombre de défauts de pages pour une chaîne de références et un algorithme de remplacement, on a besoin de connaître le **nombre de frames disponibles**. Bien entendu, si le nombre de frames augmente, le nombre de défauts de pages diminue.

Exemple

Soit la chaîne de références : 1, 4, 1, 6, 1, 6, 1, 6, 1, 6, 1.

Avec trois (03) frames disponibles : on aura trois (03) défauts de pages.

Avec un frame disponible : on aura onze (11) défauts de pages.

Pour illustrer les algorithmes de remplacement, on utilisera une mémoire avec trois (03) frames et la chaîne de références suivante :

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

1. Algorithme FIFO (FIFO Algorithm)

Avec cet algorithme, c'est la page la plus

Cet algorithme mémorise dans une file FIFO les pages présentes en mémoire, de la page la plus ancienne à la page la plus récente. Lorsqu'un défaut de page se produit, il retire la plus ancienne, c'est à dire celle qui se trouve en tête de file. Une nouvelle page est place en queue de file.

Exemple

Appliquer cet algorithme sur la chaîne de références définie ci-dessus

	7	**0**	**1**	**2**	**0**	**3**	**0**	**4**	**2**	**3**	**0**	**3**	**2**	**1**	**2**	**0**	**1**	**7**	**0**	**1**
Frame 1	7	7	7	2	2	2	2	4	4	4	0	0	0	0	0	0	0	7	7	7
Frame 2		0	0	0	0	3	3	3	2	2	2	2	2	1	1	1	1	1	0	0
Frame 3			1	1	1	1	0	0	0	3	3	3	3	3	2	2	2	2	2	1
Défaut de page	D	D	D	D		D	D	D	D	D	D			D	D			D	D	D
Taux de défaut de page							(15/20) * 100 = 75%													

L'algorithme FIFO est simple à mettre en œuvre. Cependant, cet algorithme ne tient pas compte de l'utilisation de chaque page. Par exemple, à la dixième référence la page 0 est retirée pour être remplacée par la page 3 puis tout de suite après, la page retirée est rechargée. L'algorithme est rarement utilisé car il y a beaucoup de défauts de page.

Anomalie de Belady (Belady's Anomaly)

Intuitivement, on peut penser que plus il y a de frames disponibles, moins il y a de défauts de pages. Ceci n'est pas toujours le cas. Belady a montré par un contre-exemple que l'algorithme FIFO donne plus de défauts de pages avec l'accroissement du nombre de frames. Ceci est connu sous le nom de l'**anomalie de Belady**.

Exemple

Appliquer l'algorithme FIFO sur la chaîne de références

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

En considérant les deux cas suivants :

- **1er cas :** Trois (03) frames disponibles.

	1	**2**	**3**	**4**	**1**	**2**	**5**	**1**	**2**	**3**	**4**	**5**
Frame 1	1	1	1	4	4	4	5	5	5	5	5	5
Frame 2		2	2	2	1	1	1	1	1	3	3	3
Frame 3			3	3	3	2	2	2	2	2	4	4
Défaut de page	D	D	D	D	D	D	D			D	D	
Taux de défaut de page	(9/12) * 100 = 75%											

- **2ème cas :** Quatre (04) frames disponibles.

	1	**2**	**3**	**4**	**1**	**2**	**5**	**1**	**2**	**3**	**4**	**5**
Frame 1	1	1	1	1	1	1	5	5	5	5	4	4
Frame 2		2	2	2	2	2	2	1	1	1	1	5
Frame 3			3	3	3	3	3	3	2	2	2	2
Frame 4				4	4	4	4	4	4	3	3	3
Défaut de page	D	D	D	D			D	D	D	D	D	D
Taux de défaut de page	(10/12) * 100 = 83,33%											

2. Algorithme de remplacement optimal (Belady ou **Optimal Algorithm)**

L'algorithme optimal de Belady consiste à retirer la page qui sera référencée le plus tard possible dans le futur ; c'est-à-dire, la page pour laquelle la prochaine référence est la plus éloignée dans le temps.

Cette stratégie est impossible à mettre en œuvre car il est difficile de prévoir les références futures d'un programme.

Le remplacement de page optimal a été cependant utilisé comme base de référence pour les autres stratégies, car il minimise le nombre de défauts de page.

Exemple

Appliquer l'algorithme de Belady sur la chaîne de références ci-après, avec trois (03) blocs disponibles.

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

Frame 1	7	7	7	2	2	2	2	2	2	2	2	2	2	2	2	2	2	7	7	7
Frame 2		0	0	0	0	0	0	4	4	4	0	0	0	0	0	0	0	0	0	0
Frame 3			1	1	1	3	3	3	3	3	3	3	3	1	1	1	1	1	1	1
Défaut de page	D	D	D	D		D		D			D			D				D		
Taux de défaut de page							(9/20) * 100 = 45%													

BIBLIOGRAPHIE

- N. SALMI, "Principes des Systèmes d'Exploitation", Pages Bleues, les Manuels de l'Etudiant, 2007.
- B. LAMIROY, L. NAJMAN, H. TALBOT, "Systèmes d'exploitation", Collection Synthex, Pearson Education, 2006.
- A. BELKHIR, "Système d'Exploitation, Mécanismes de Base", OPU, 2005.
- A. Silberschatz, P.B. Galvin, G. Gagne, "Operating System Concepts", 7th Edition, John Wiley & Sons Editions, 2005, 921 p.
- A. TANENBAUM, "*Systèmes d'Exploitation : Systèmes Centralisés – Systèmes Distribués*", 3ème édition, Editons DUNOD, Prentice Hall, 2001.
- A. Silberschatz, P. B. Galvin, "Principes des Systèmes d'Exploitation", traduit par M. Gatumel, 4ème édition, Editions Addison-Wesly France, SA, 1994.
- M. GRIFFITHS, M. VAYSSADE, "*Architecture des Systèmes d'Exploitation*", Edition Hermès, 1990.
- S. KRAKOWIAK, "*Principes des Systèmes d'Exploitation des Ordinateurs*", Editions DUNOD, 1987.
- A. TANENBAUM, "Architecture de l'Ordinateur", Editions Pearson Education, 2006.
- J. M. LERY, "Linux", Collection Synthex, Pearson Education, 2006.
- J. Delacroix, "LINUX, Programmation Système et Réseau, Cours et Exercices Corrigés", Editions DUNOD, 2003.
- M. J. BACH, "*Conception du système UNIX*", Editions Masson, 1989.