COURS
SYSTEMES D'EXPLOITATION
(PARTIE II)

DR. CHAFIKA BENZAID

MAITRE DE CONFERENCES, CLASSE B

LSI-Département Informatique,
Faculté d'Electronique & Informatique, USTHB
Email: benzaid@hotmail.com

TABLE DE MATIERES

COURS SYSTEMES D'EXPLOITATION

OBJECTIF

Inculquer à l'étudiant le concept de machine virtuelle et de ses missions à savoir gestion des ressources de la machine réelle de manière transparente aux utilisateurs. L'approche retenue se base sur le modèle en couches.

A RETENIR

Je m'attends à votre présence dans la plupart des séances (Cours, TD, TP). Je ne prendrai pas les présences, mais si vous choisissez de ne pas assister à vos séances alors s'il vous plaît prenez la responsabilité de vos absences, surtout quand il s'agit de tests et des devoirs. Quand vous venez à la classe, vous devez changer votre téléphone portable en mode silencieux. Dans les tests, les téléphones portables doivent être complètement éteints.

RECOMMANDATIONS

- Montrer, à travers des exemples illustrant, comment les différents aspects présentés dans ce cours sont traités par UNIX.
- Prévoir des TPs illustrant les connaissances enseignées.
- Apprentissage et usage du langage C.

CHAPITRE III : GESTION DES E/S PHYSIQUES

1 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 se 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).

2 Matériel

2.1 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**)**.**

2.2 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.

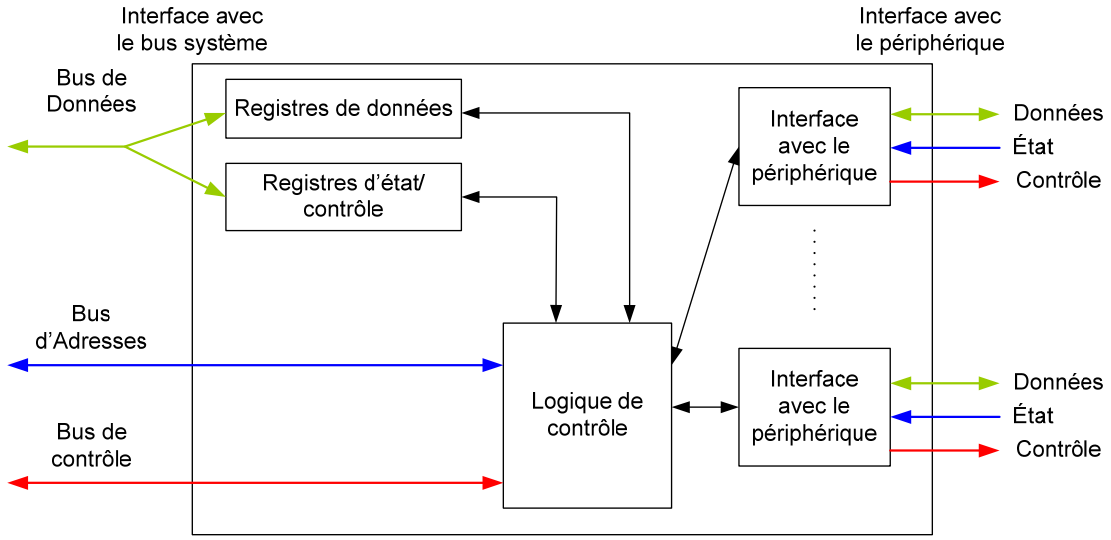


Figure 3.1 : Principaux composants d'un coupleur

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).

2.3 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.

2.4 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.

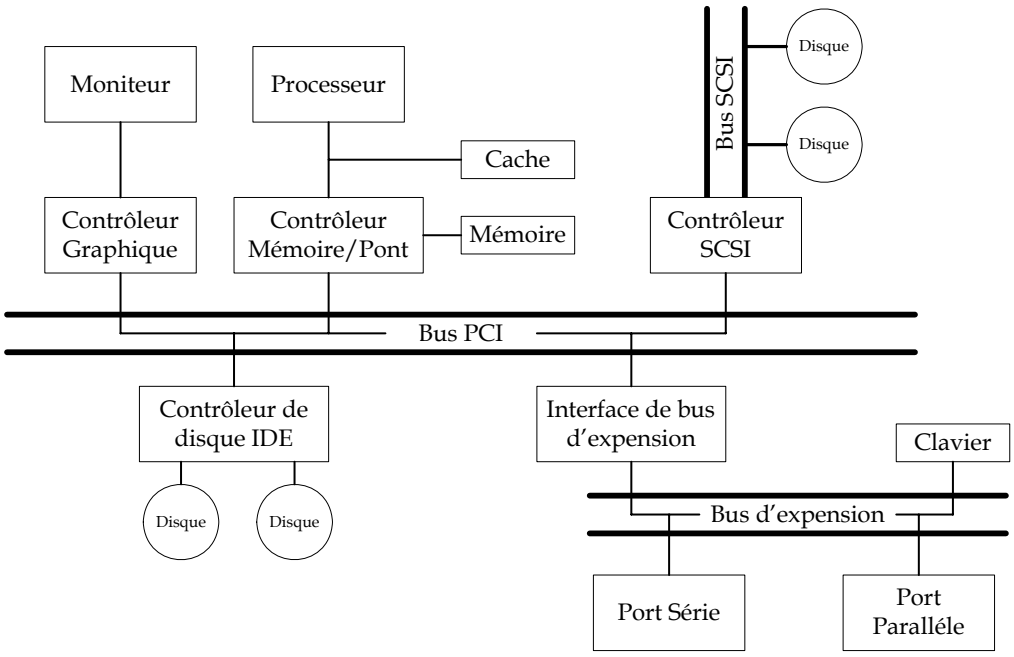


Figure 3.2 : Architecture typique des Bus d'un PC

3 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**.

4 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.

4.1 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 :

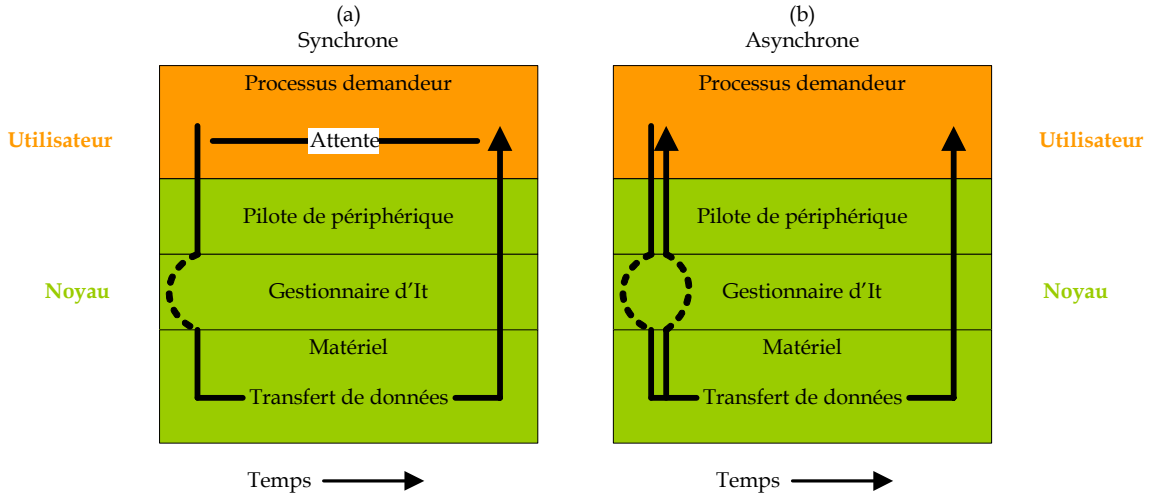


Figure 3.3 : Modes de pilotage des E/S directes

- **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.

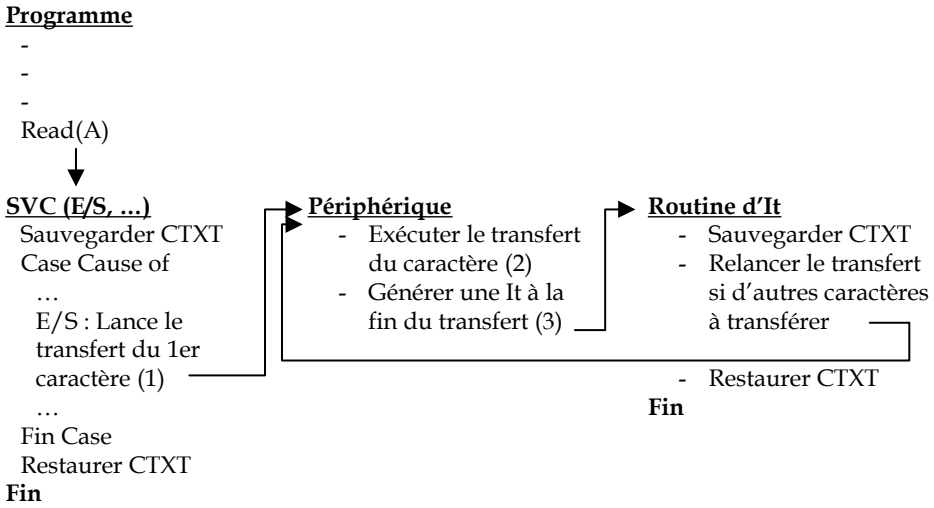


Figure 3.4 : Schéma général d'exécution d'une E/S asynchrone

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.

4.2 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.

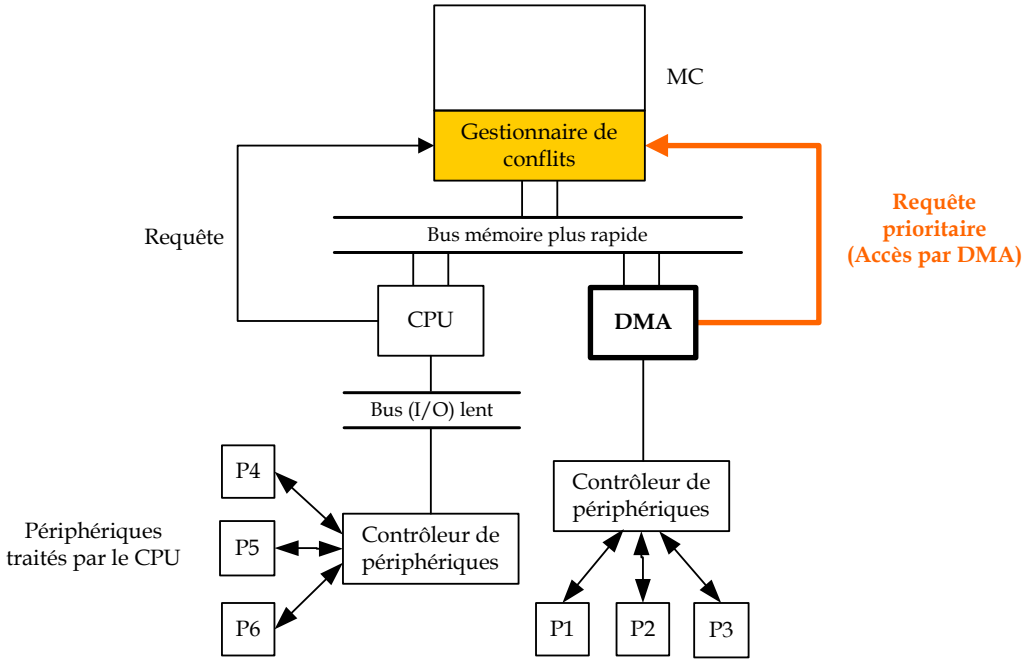


Figure 3.5 : Architecture typique avec DMA

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.

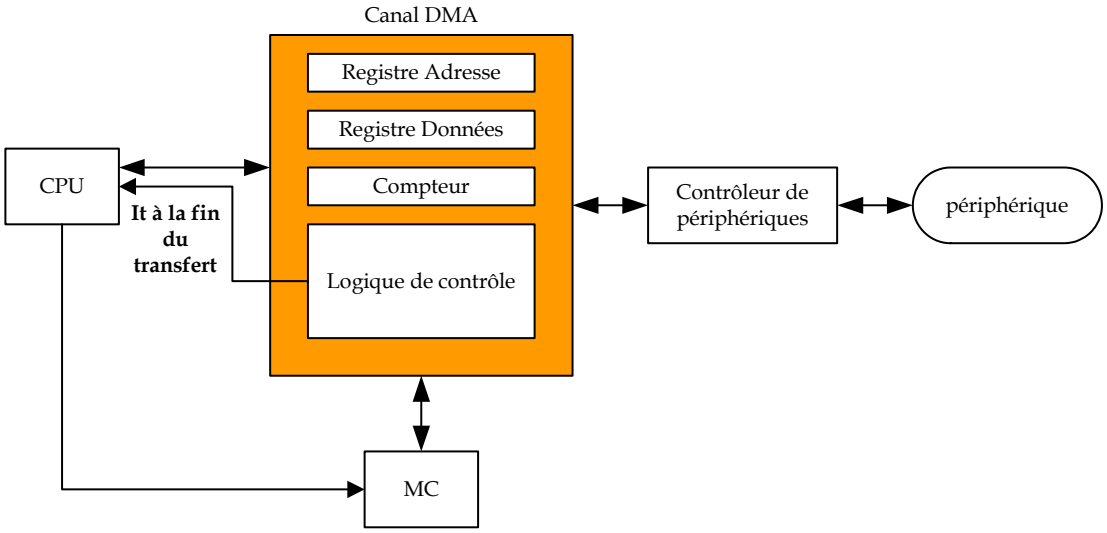


Figure 3.6 : Registres d'un contrôleur DMA

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 :

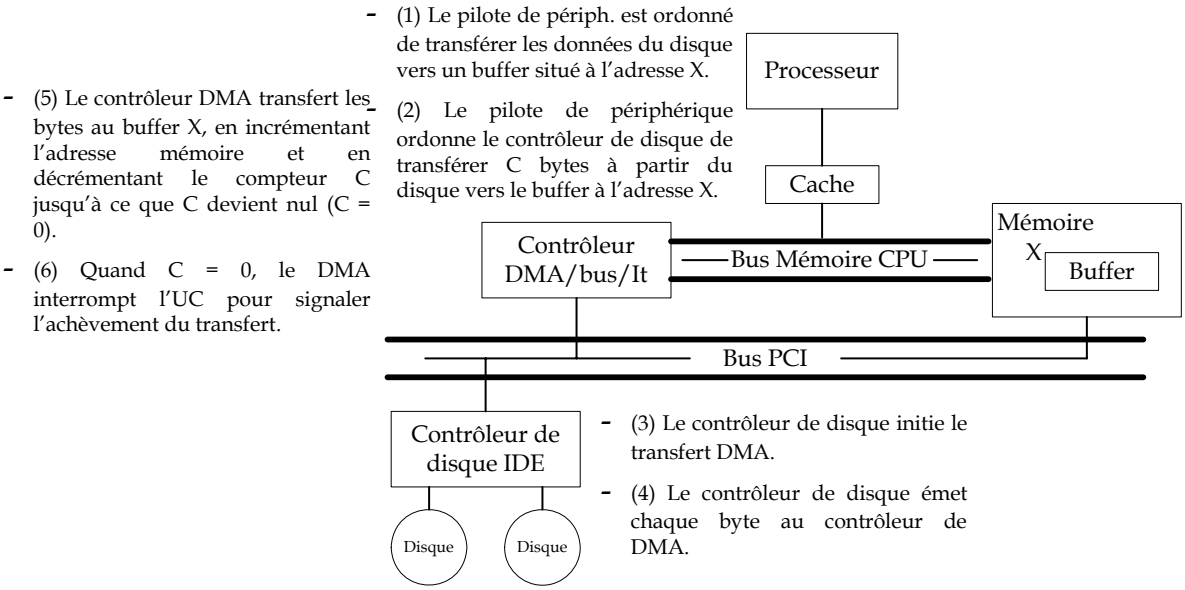


Figure 3.7 : Processus d'un transfert DMA

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).

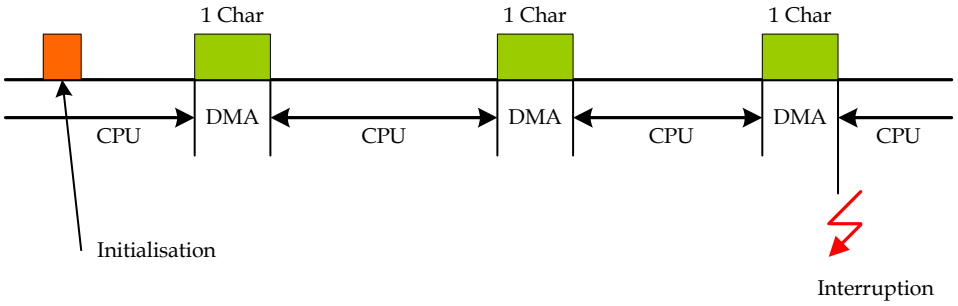


Figure 3.8 : Principe de vol de cycles

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.

5 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**).

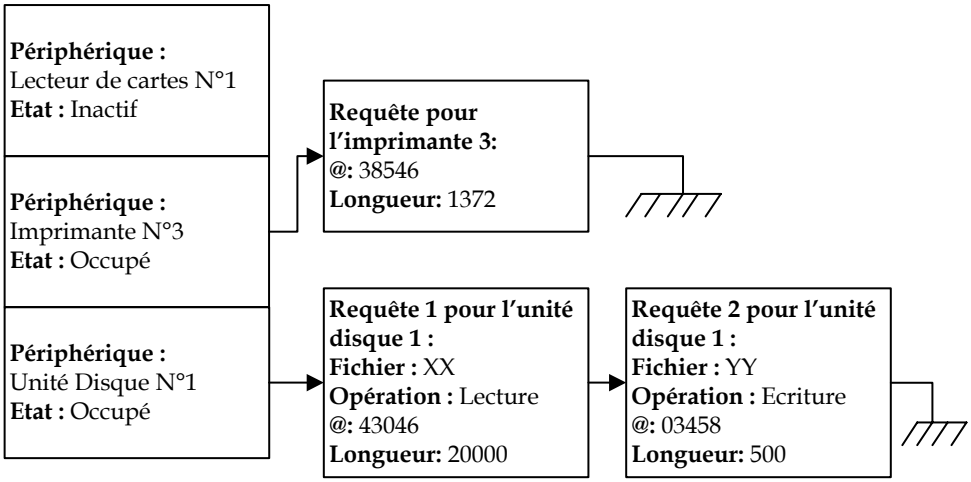


Figure 3.9 : Table d'état des périphériques

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.

6 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.

7 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.

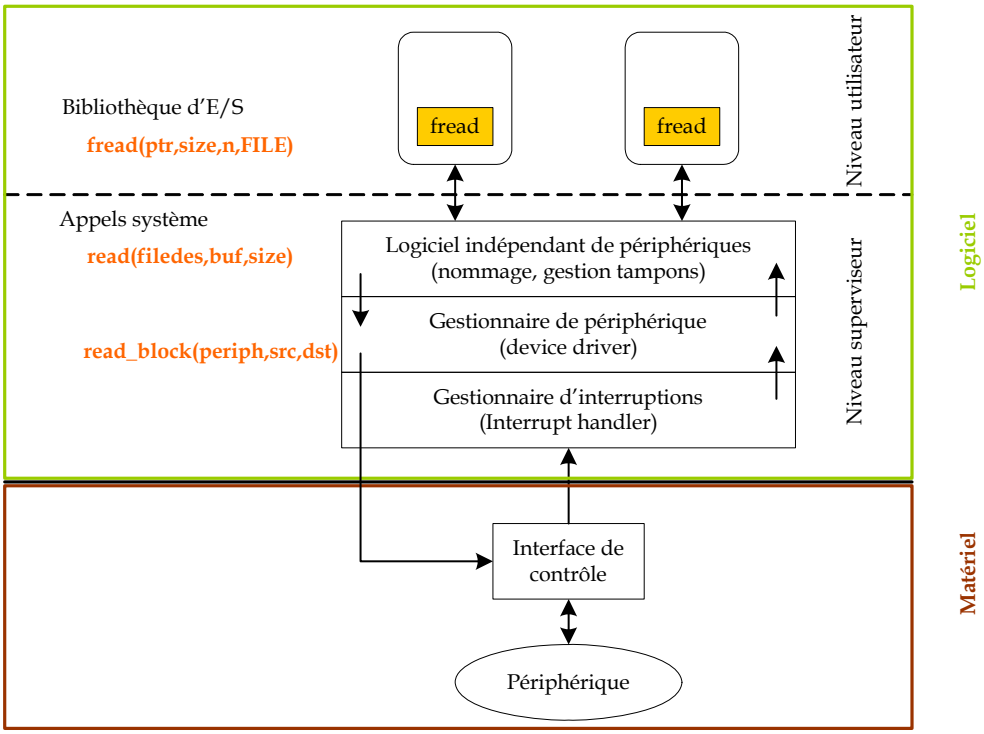


Figure 3.10 : Structure en couches d'un système d'E/S

CHAPITRE IV : INTERFACE DE SYSTEME 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 :

2.1 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**.

2.2 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.

2.3 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.

2.4 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.

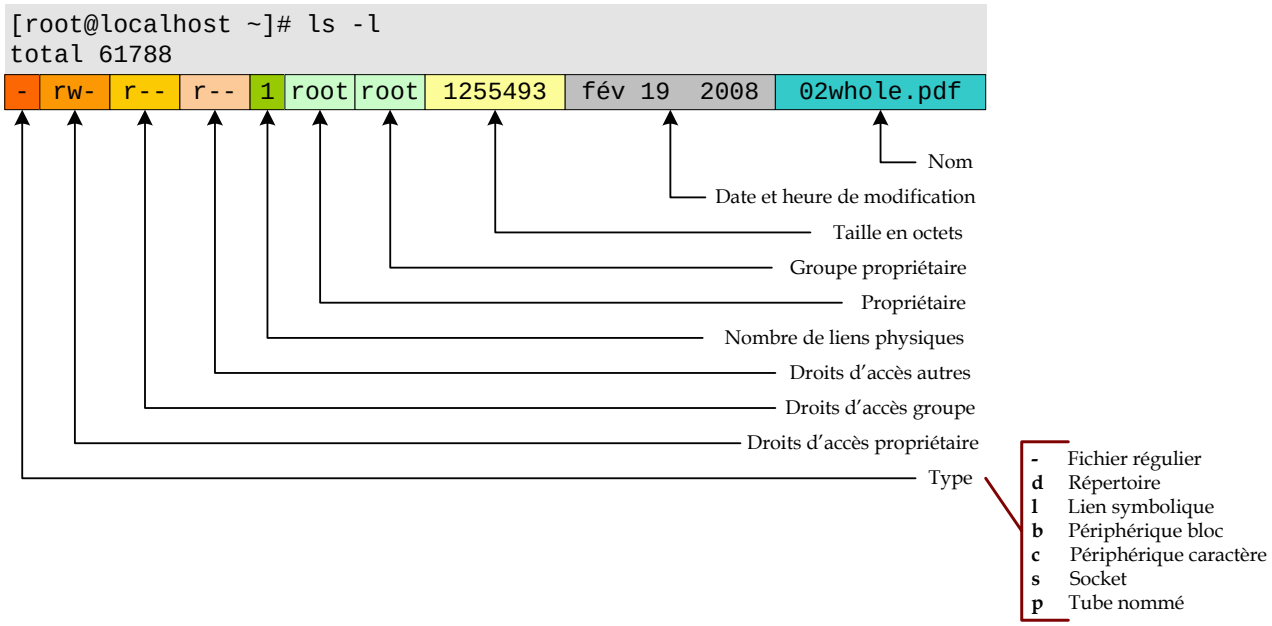
- Les **tubes nommés** (**Named Pipe**, **FIFO**) permettant la communication entre processus.
- Les **sockets** internes qui sont destinées à la communication bidirectionnelle entre processus.

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**ist) 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 fichiers** (**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 fichiers 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

10.1 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.

10.2 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.

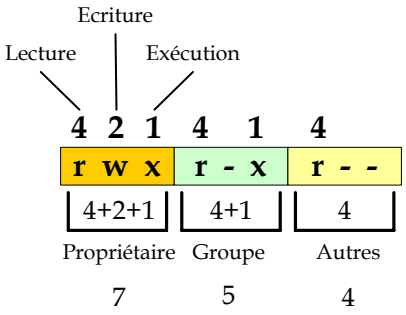


Figure 4.1 : Représentation des droits d'accès

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

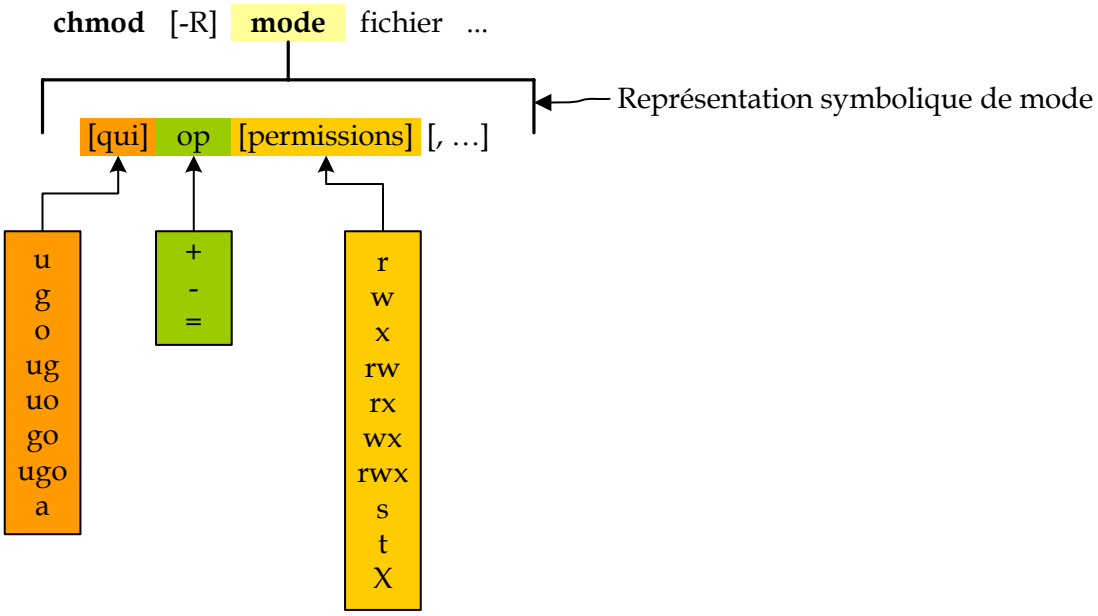


Figure 4.2 : Représentation symbolique de la commande chmod

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).

10.3 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

11.1 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.

11.2 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).

11.3 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.

11.4 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.

11.5 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	**mkfs -t ext3 /dev/hdb2**

	mke2fs –j /dev/device	**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.

11.6 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

```
$ umount /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.

11.7 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 V : 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.

1 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**.
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.

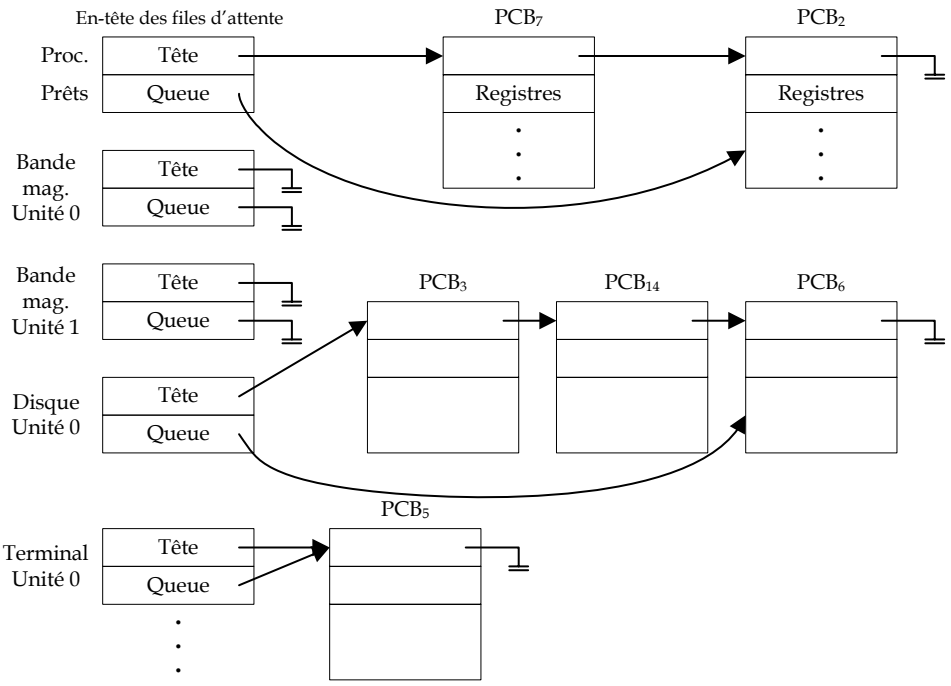


Figure 5.1 : La file d'attente des proc. Prêts et plusieurs files d'attente des périph. d'E/S

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

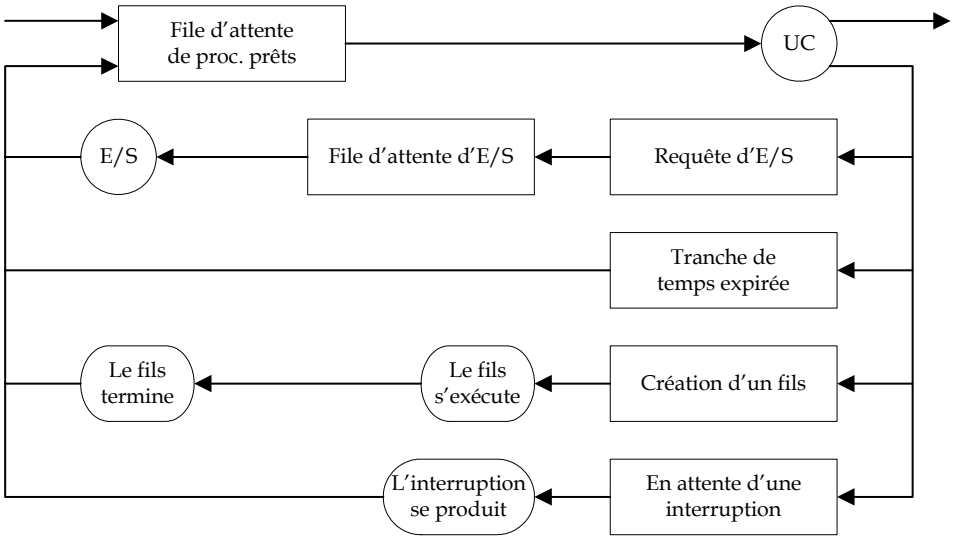


Figure 5.2 : Représentation du diagramme des files d'attente du scheduling des processus

2 Concept de Scheduling

2.1 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.

2.2 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) –** le temps qui s'écoule entre la soumission d'une requête et la première réponse obtenue. Cette métrique est importante dans le cas des systèmes interactifs.
- **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.

2.3 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).

2.4 Priorité de scheduling

L'allocation du processeur central aux programmes peut se faire suivant certaines valeurs de priorité, qui sont 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).

3 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.

4 Politiques de Scheduling

4.1 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 d'exécution et les dates d'arrivée respectives sont données dans la table ci-après. Faites un schéma qui illustre leur exécution et calculez le **temps de rotation** de chaque processus, le **temps moyen de rotation**, le **temps d'attente** et le **temps moyen d'attente** en utilisant la politique FCFS.

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

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

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

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

Le schéma d'exécution est : AAABBBBBBCCCCDDE

Processus	Durée d'Exe.	Temps d'arrivée	Date Début d'Exe.	Date fin d'Exe.	Temps de rotation	Temps d'attente
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 rotation 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.

4.2 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 rotation	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 rotation 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			

4.3 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.	Préempté A	Repris A	Date fin d'Exe.	Temps de rotation	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 rotation 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			5					

4.4 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. Faites un schéma qui illustre leur exécution et calculez le **temps de rotation** de chaque processus, le **temps moyen de rotation**, le **temps d'attente** et le **temps moyen d'attente** 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 rotation	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 rotation 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 $\Rightarrow$ le processus est moins prioritaire.

4.5 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, un processus aura 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 μs).

Exemple 1

Considérons trois processus A, B, et C dont les durées d'exécution et les dates d'arrivée respectives sont données dans la table ci-après. Faites un schéma qui illustre leur exécution et calculez le **temps de rotation** de chaque processus, le **temps moyen de rotation**, le **temps d'attente** et le **temps moyen d'attente** en utilisant la politique RR.

On suppose que le Quantum = 3 et que la durée de commutation = 0.

Le schéma d'exécution : AAABBBAAABBBCCCAABBC

Processus	Durée d'Exe.	Temps d'arrivée	Date Début d'Exe.	Préempté A	Repris à	Date fin d'Exe.	Temps de rotation	Temps d'attente
A	8	0	0	3	6	17	17	9
				9	15			
B	8	2	3	6	9	19	16	8
				12	17			
C	4	7	12	15	19	20	13	9
Temps de rotation moyen			(17+16+13)/3= 46/3 = 15,33 UT					
Temps d'attente moyen			(9+8+9)/3 = 26/3 = 8,66 UT					

Exemple 2

Considérons trois processus A, B, et C dont les durées d'exécution et les dates d'arrivée respectives sont données dans la table ci-après. Faites un schéma qui illustre leur exécution et calculez le **temps de rotation** de chaque processus, le **temps moyen de rotation**, le **temps d'attente** et le **temps moyen d'attente** ainsi que le **nombre de changements de contexte** effectués en utilisant la politique RR.

La notation **x(y)z** signifie que le processus fait **x** UT calcul, ensuite **y** UT E/S et enfin **z** UT calcul.

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 rotation	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 rotation 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			7					

4.6 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 → FCFS.

- Processus interactifs → Round Robin.

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

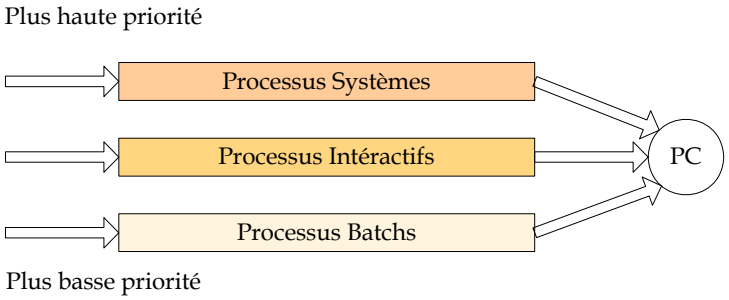
- Soit par priorité.

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

- Soit par partage du temps du processeur.

  Ex. 80% → Interactifs, 20% → Batch.

Exemple



4.7 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 consommation 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

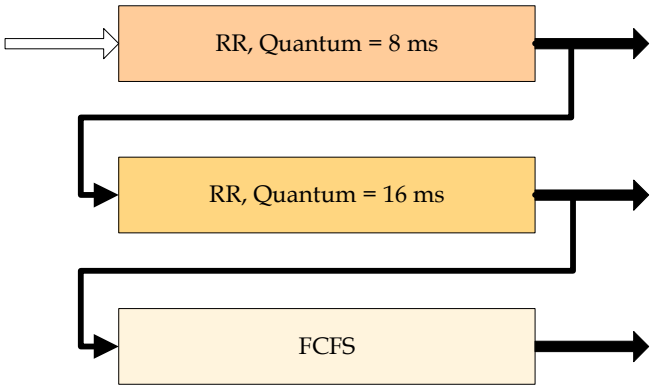
Soient Trois files:

- 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.



5 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** (**Load Balancing**).

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 de travail = Load Sharing**).
- **Approche asymétrique** où certaines fonctions sont réservées à une seule UC. Files d'attentes séparées pour chaque UC.

6 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 IV : GESTION DE LA MEMOIRE

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.

1 Hiérarchie

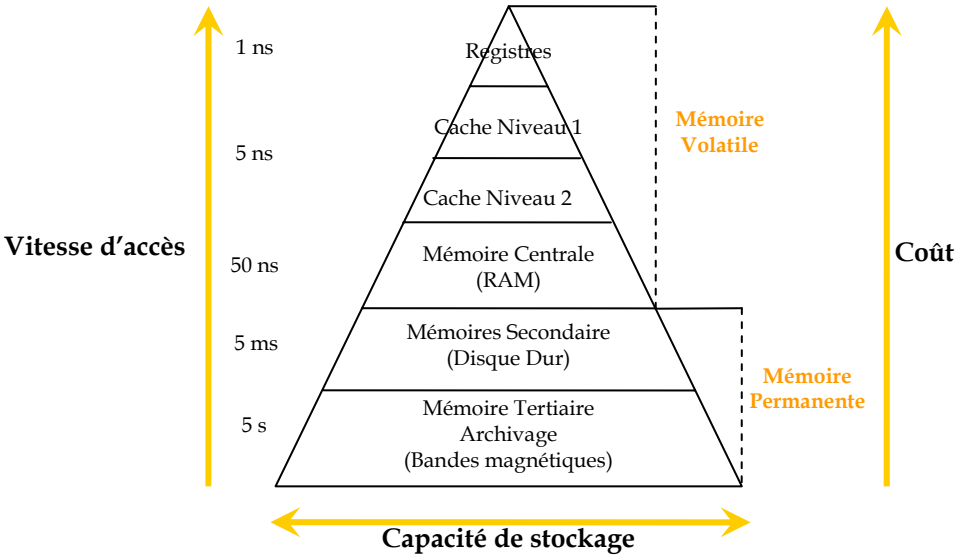


Figure 6.1 : Hiérarchie des Mémoires

- **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.

2 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.

3 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 viser les objectifs suivants :

3.1 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 :

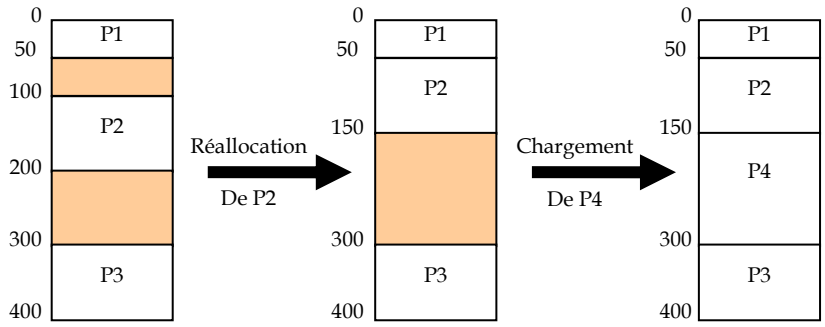


Figure 6.2 : Principe de Réallocation

3.2 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.

3.3 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.

3.4 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.

4 Caractéristiques liées au chargement d'un programme

4.1 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.

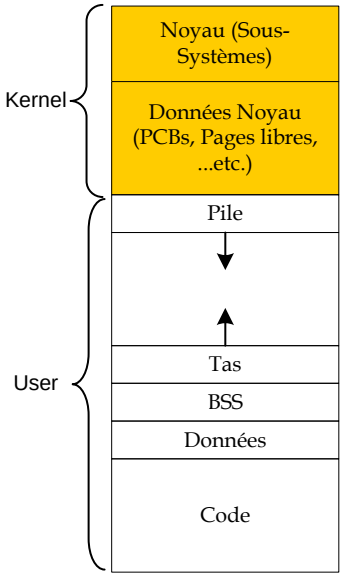


Figure 6.3 : Espace d'adressage (User/Kernel)

4.2 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.

4.3 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). **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.

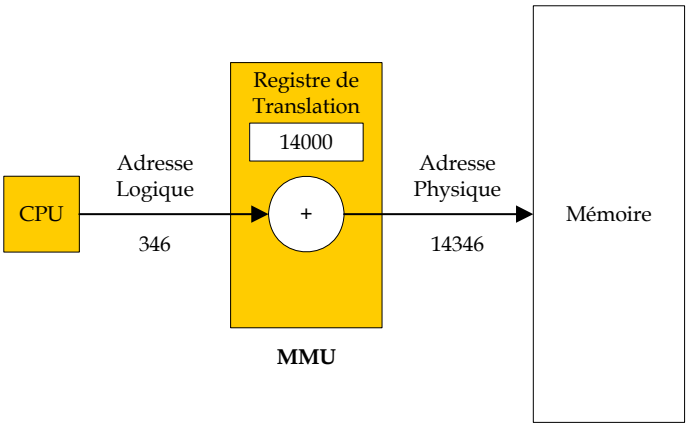


Figure 6.4 : Translation Dynamique en Utilisant un Registre de Translation

5 Stratégie d'allocation

5.1 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.

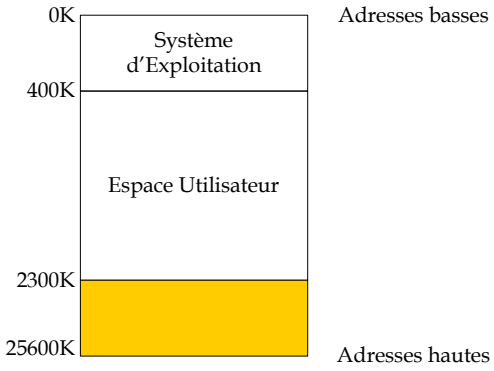


Figure 6.5 : Organisation de la mémoire en Monobloc

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 :

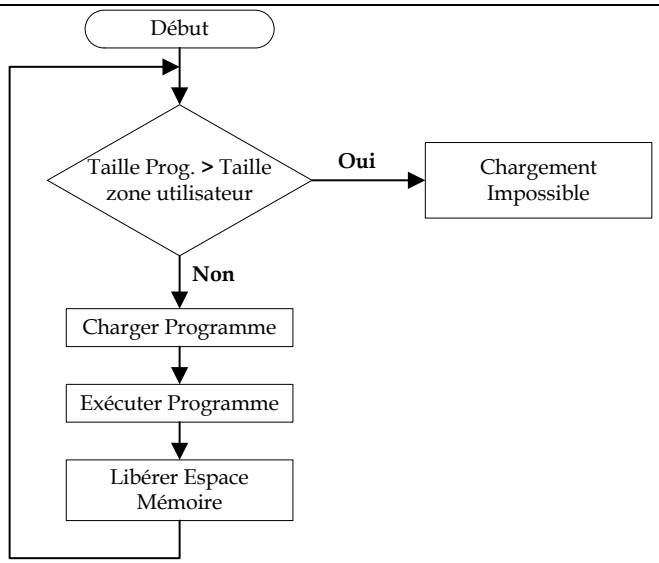


Figure 6.6 : Algorithme d'Allocation mémoire d'un programme utilisateur

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)

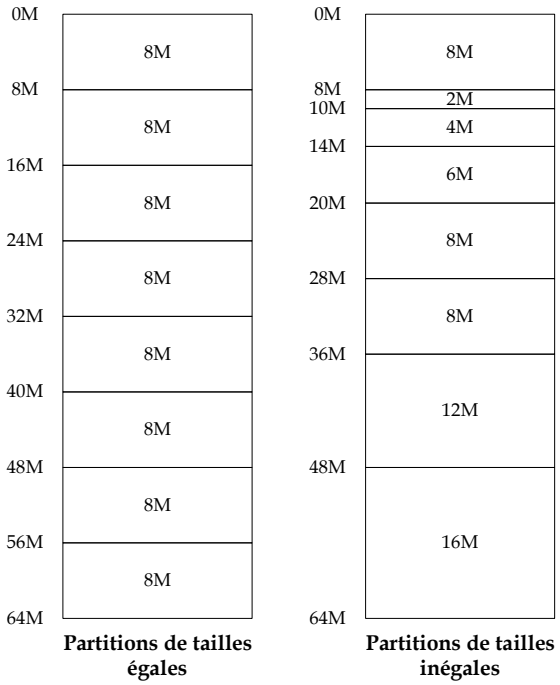
Cette stratégie constitue une technique simple pour la mise en œuvre de la **multiprogrammation**. La mémoire principale est divisée en régions séparées ou partitions mémoires ; chaque partition dispose de son espace d'adressage. Le partitionnement de la mémoire peut être **statique (fixe)** ou **dynamique (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**.

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.



Base de partition	Taille de Partition	Statut de Partition
0M	8M	Allouée
8M	2M	Libre
10M	4M	Allouée
14M	6M	Allouée
20M	8M	Allouée
28M	8M	Libre
36M	12M	Allouée
48M	16M	Libre

Table de description des partitions

Figure 6.7 : 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.

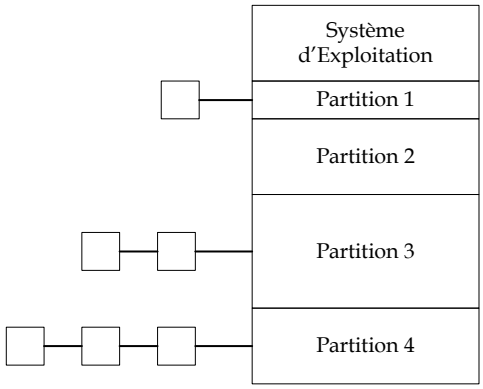


Figure 6.8 : Partitions fixes avec files multiples

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.

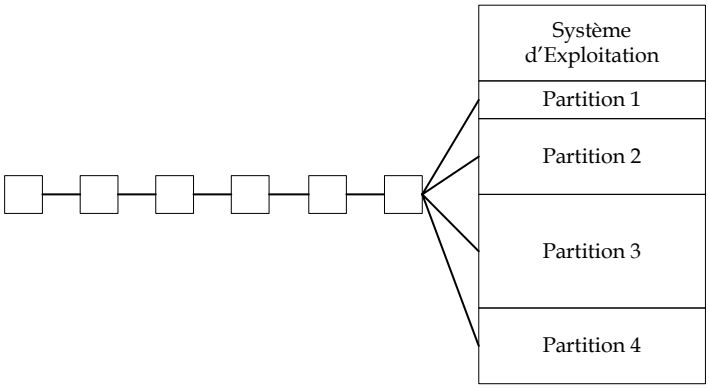


Figure 6.9 : Partitions fixes avec une seule file

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 :

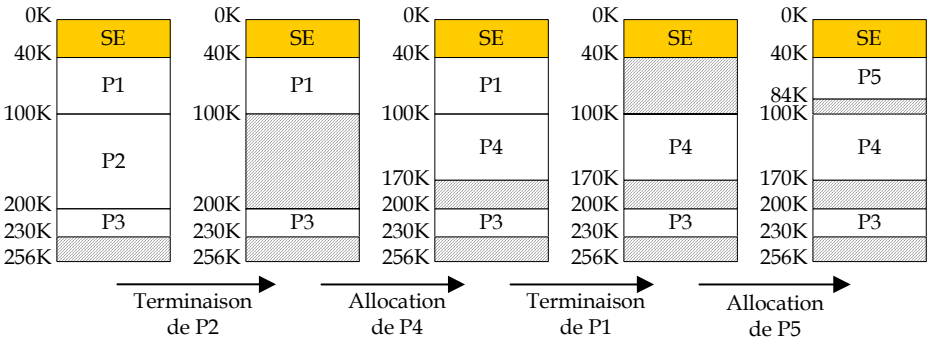


Figure 6.10 : Exemple d'allocation en utilisant des partitions variables

- **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.

  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.

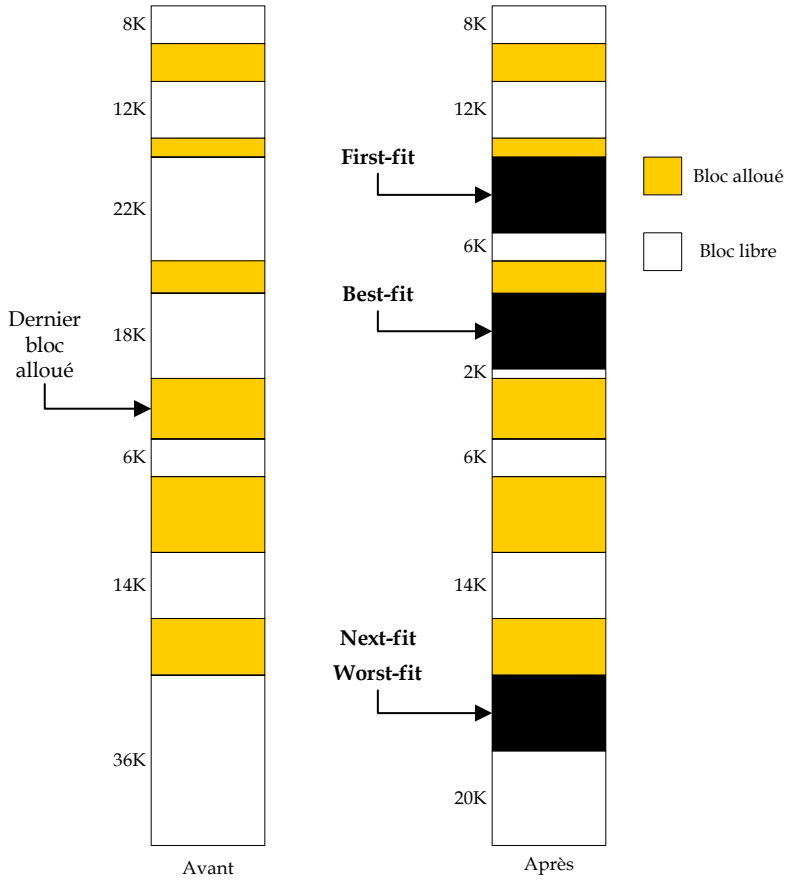


Figure 6.11 : Exemple illustrant les algorithmes de placement

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.

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). Cette opération est **très coûteuse** en temps CPU.

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.

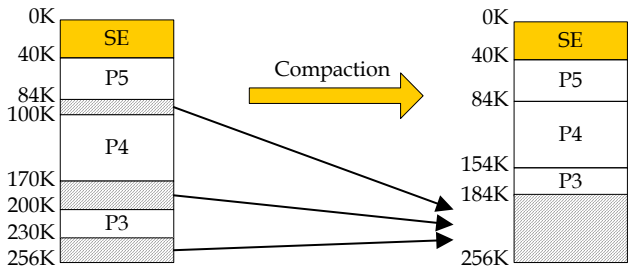


Figure 6.12 : Compactage

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**.

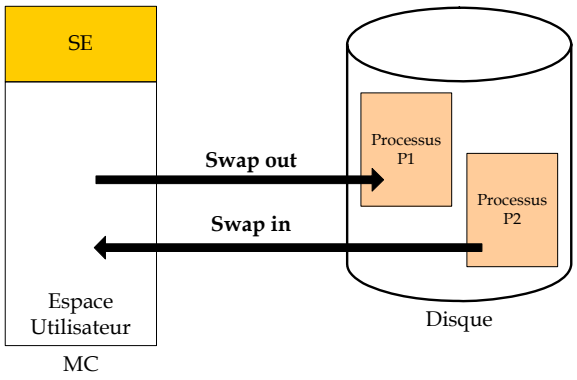


Figure 6.13 : Système de swapping

5.2 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 :

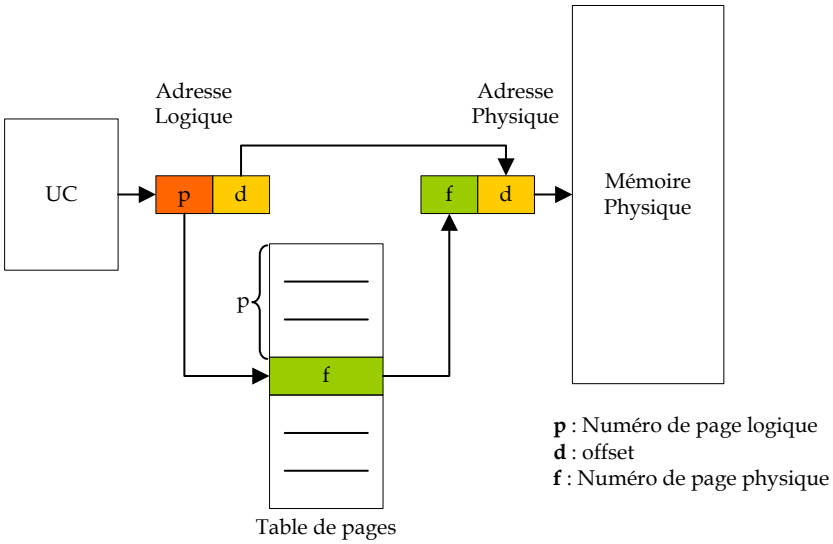
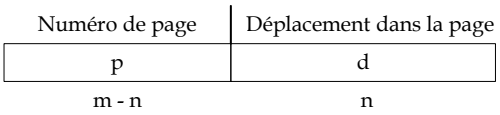


Figure 6.14 : Matériel pour la pagination

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.



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)

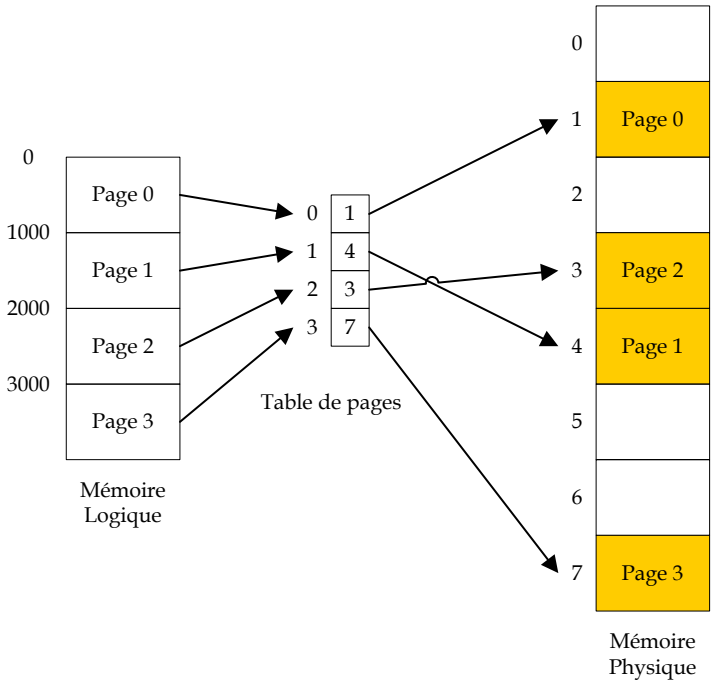


Figure 6.15 : Modèle de pagination de la mémoire logique et physique

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.

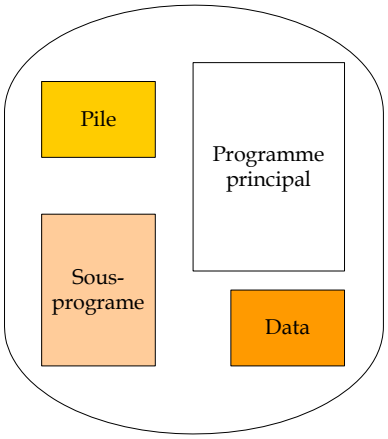


Figure 6.16 : Vue de l'utilisateur d'un programme

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 :

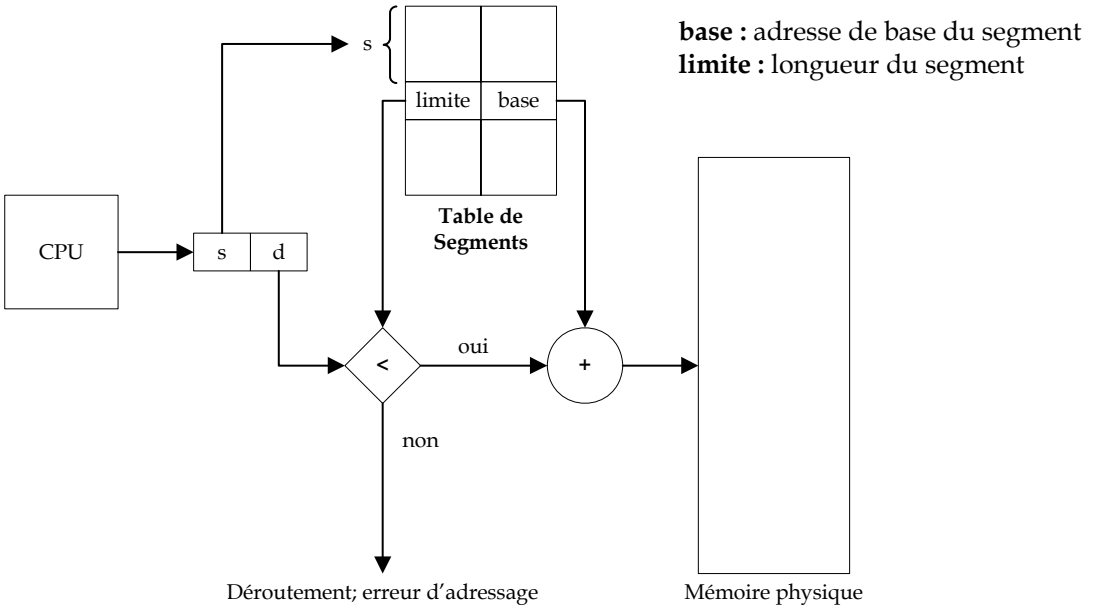


Figure 6.17 : Matériel pour la segmentation

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

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. Les programmes sont alors divisés en segments et chaque segment en pages. Cette technique a été inventée pour le système Multics.

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 :

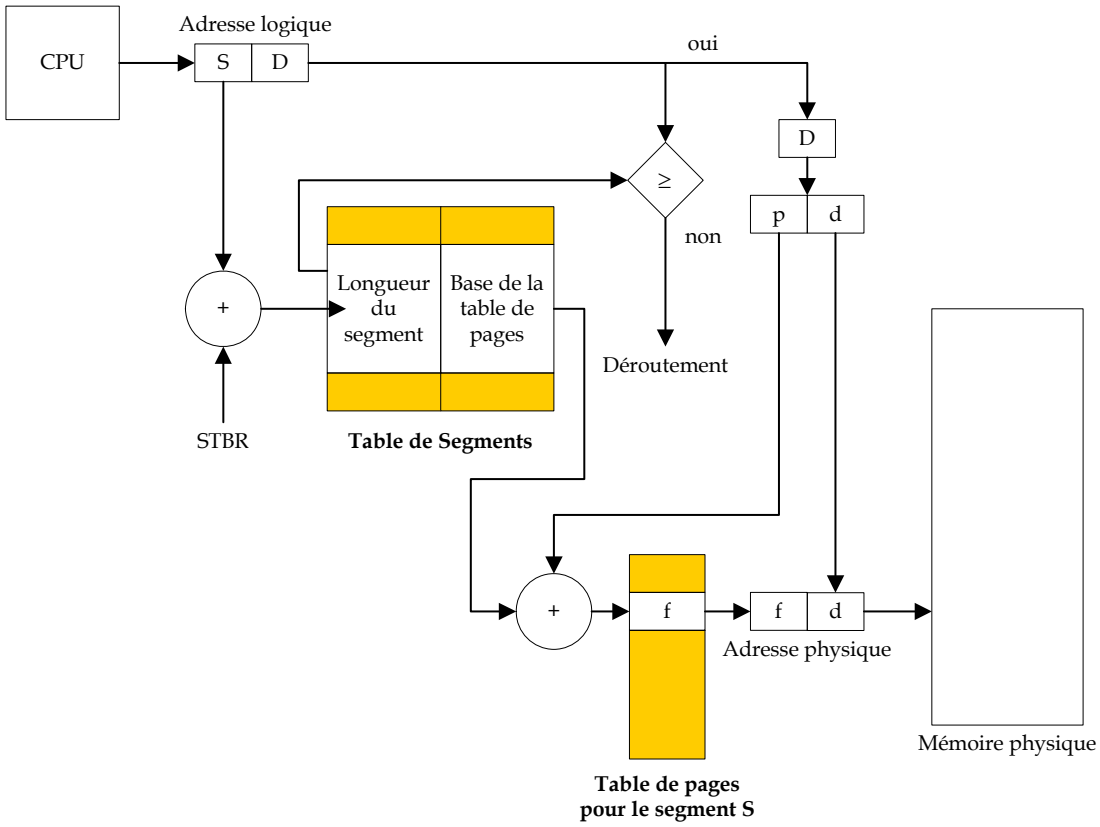


Figure 6.18 : Matériel pour la segmentation paginée

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

6 Mémoire virtuelle

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

Un processus est donc constitué de morceaux (pages ou segments) ne nécessitant pas d'être tous en MC durant l'exécution. Afin 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 MC 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.

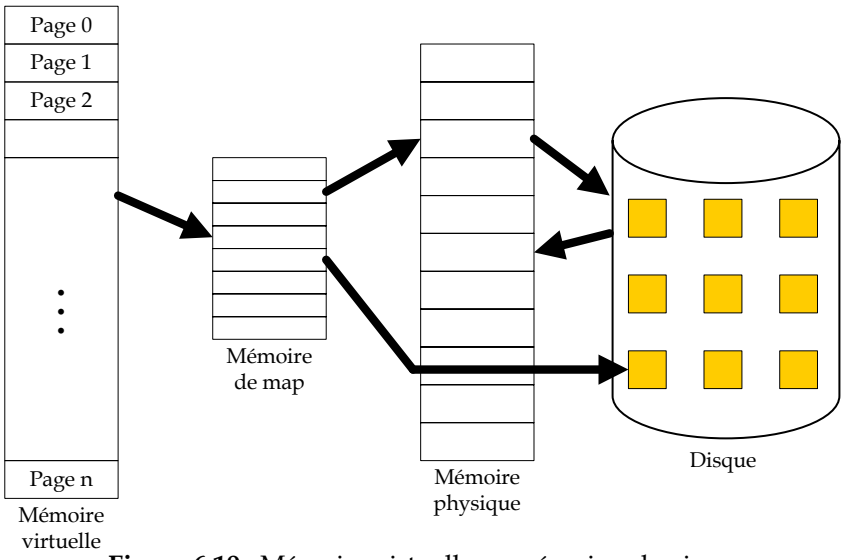


Figure 6.19 : Mémoire virtuelle vs mémoire physique

6.1 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 de 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.

6.2 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).

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.

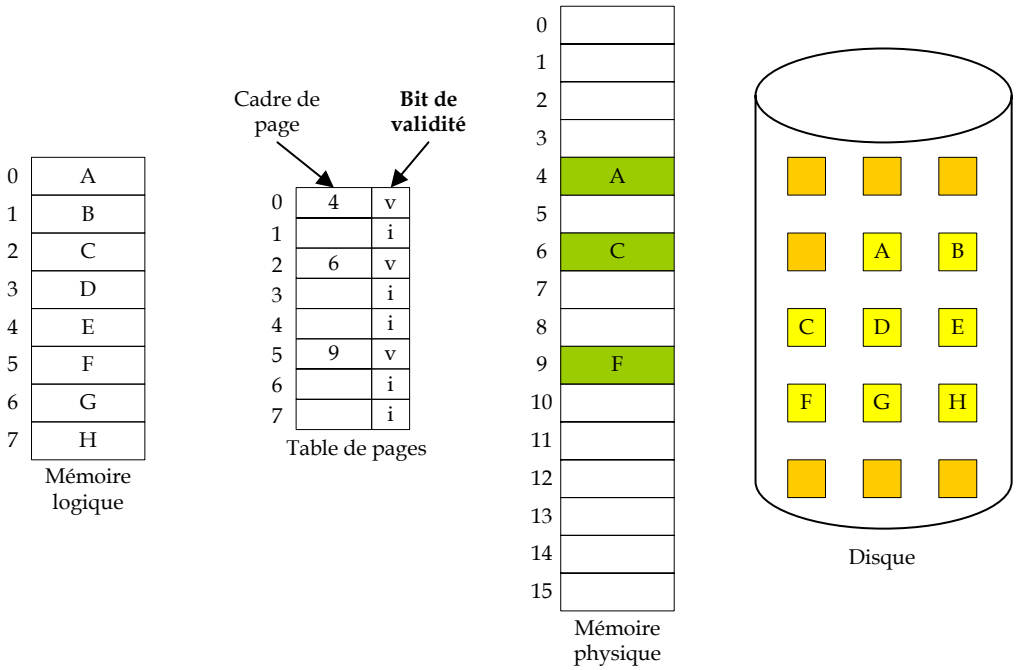


Figure 6.20 : Tables de pages ave pages absentes

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.

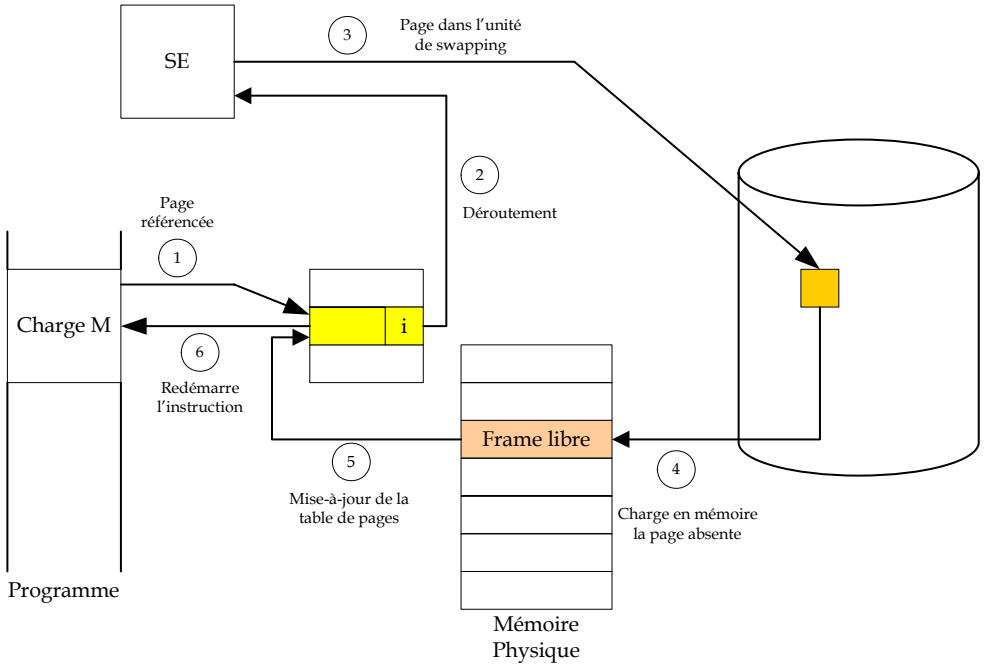


Figure 6.21 : Les étapes de prise en charge d'un défaut de page

6.3 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 ($0 \leq P \leq 1$), et par **Tdp** le temps de traitement d'un défaut de page, alors le temps d'accès effectif (**Tae**) est égal :

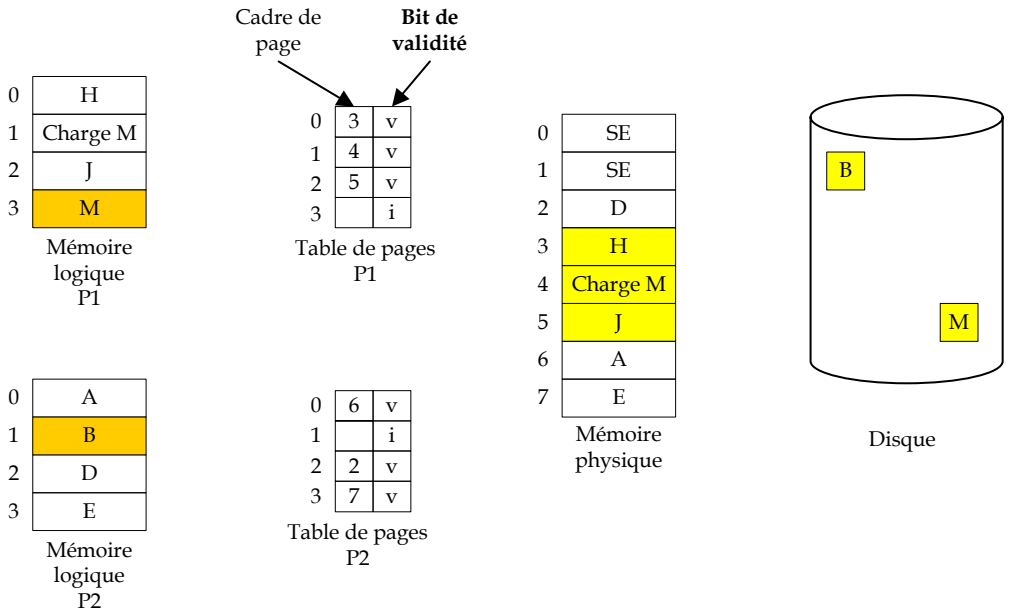
$$\mathbf{Tae = (1-}P\mathbf{) * ma + }P\mathbf{ * Tdp}$$

6.4 Remplacement de page

L'algorithme précédent suppose qu'il existe toujours une case libre en MC. 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.

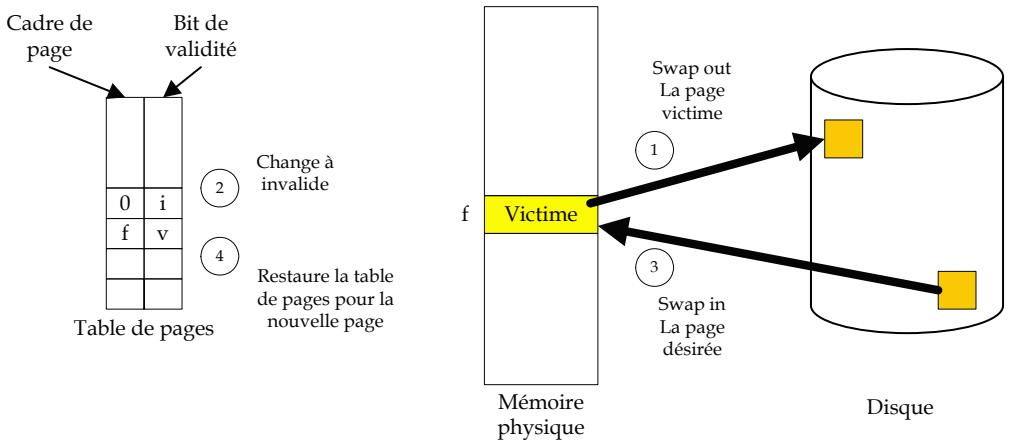


Figure 6.22 : Remplacement de pages

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.

- **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 dont 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)

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 placée 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%													

3. Algorithme LRU (Least Recently Used Algorithm)

Avec l'algorithme LRU, **la page la moins récemment utilisée est retirée**. Cette stratégie utilise le principe de la **localité temporelle** selon lequel les pages récemment utilisées sont celles qui seront référencées dans le futur.

Exemple

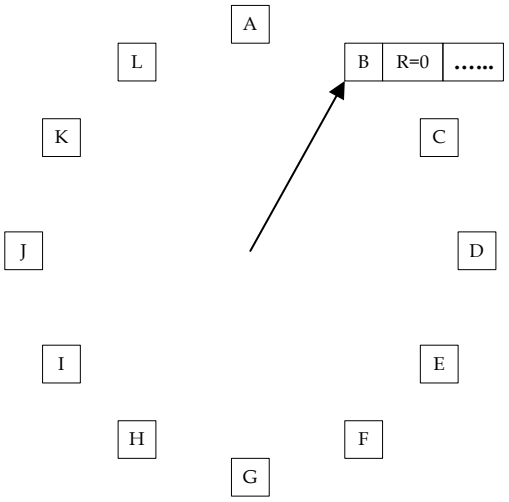
Appliquer l'algorithme LRU sur la chaîne de références ci-après, avec trois (03) blocs disponibles.

	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	4	4	4	0	0	0	1	1	1	1	1	1	1
Frame 2		0	0	0	0	0	0	0	0	3	3	3	3	3	3	0	0	0	0	0
Frame 3			1	1	1	3	3	3	2	2	2	2	2	2	2	2	2	7	7	7
Défaut de page	D	D	D	D		D		D	D	D	D			D		D		D		
Taux de défaut de page							(12/20) * 100 = 60%													

Le problème avec cet algorithme est la difficulté d'implémentation, qui requiert un support matériel. Il faut une manière de mémoriser le temps à chaque fois qu'une page est référencée.

4. Algorithme de seconde chance (Second Chance Algorithm)

Cet algorithme, aussi appelé **algorithme de l'horloge**, est une approximation de l'algorithme LRU. Dans cet algorithme, les pages en mémoire sont mémorisées dans une liste circulaire en forme d'horloge :



Un indicateur pointe sur la page la plus ancienne selon FIFO. Lorsqu'un défaut de page se produit, la page pointée par l'indicateur est examinée. Si le **bit de référence** (**Reference Bit**, **R**) de la page pointée par l'indicateur est à **0**, la page est retirée, la nouvelle page est insérée à sa place et l'indicateur avance d'une position. Sinon, il est mis à **0** et l'indicateur avance d'une position. Cette opération est répétée jusqu'à ce qu'une page, ayant **R** égal à **0** soit trouvée. À noter que lorsqu'une page est ajoutée, on met son bit de référence à 1.

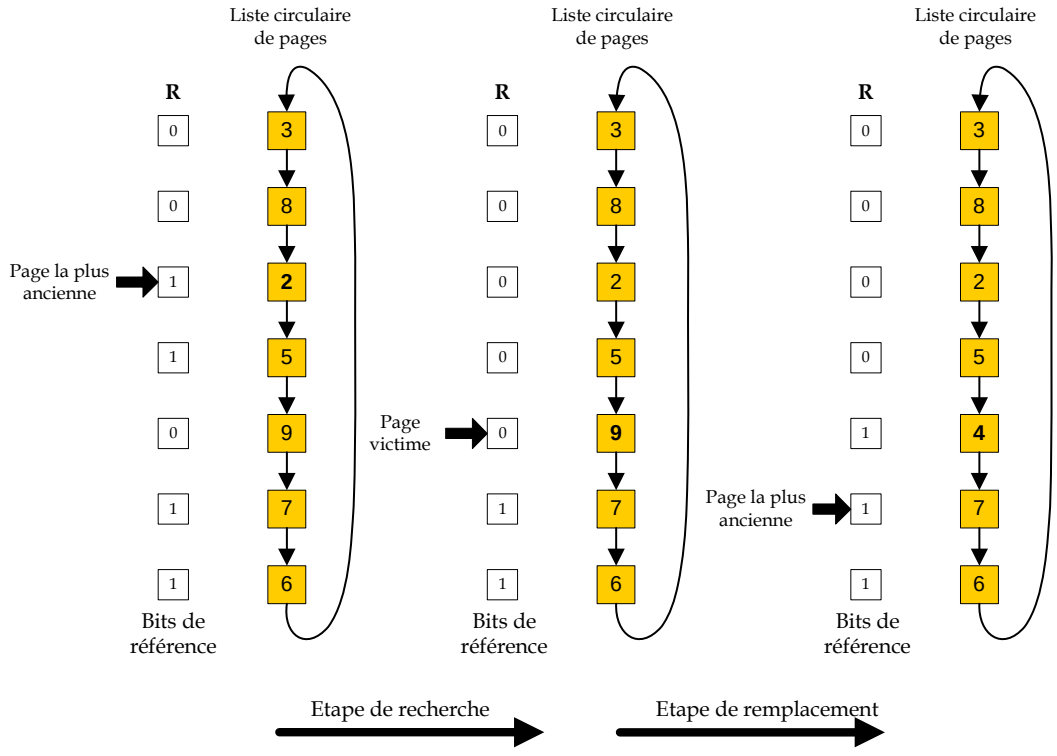


Figure 6.23 : Algorithme de la seconde chance (exemple de chargement de la page 4)

On remarquera que cet algorithme a pour effet de donner une seconde chance aux pages qui ont été référencées depuis la dernière exécution de l'algorithme. Ce n'est que si toutes les pages ont été référencées que l'on revient à la première page pour l'expulser.

Exemple

Appliquer l'algorithme de seconde chance 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	4	4	4	4	3	3	3	3	0	0	0	0	0
Frame 2		0	0	0	0	0	0	0	2	2	2	2	2	1	1	1	1	7	7	7
Frame 3			1	1	1	3	3	3	3	3	0	0	0	0	2	2	2	2	2	1
Défaut de page	D	D	D	D		D		D	D		D	D		D	D	D		D		D
Taux de défaut de page							(14/20) * 100 = 70%													

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.
- W. STALLINGS, "**Operating Systems: Internals and Design Principles**", 4th Edition, Prentice-Hall, 2001.
- 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.