REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
Ministère de l'enseignement supérieur et de la recherche scientifique
Université de Jijel
Faculté des Sciences exactes et Informatique
Département d'informatique

Support de Cours

SYSTÈMES D'EXPLOITATION

Pour 2ème Année Licence - informatique

Présenté Par Mme Bouainah Madiha

mbouainah@yahoo.fr

Chapitre I

Introduction aux systèmes d'exploitation

I. Introduction :

Le système d'exploitation d'un ordinateur ou d'une installation informatique est un ensemble de programmes qui remplissent deux grandes fonctions :

- gérer les ressources de l'installation matérielle en assurant leurs partages entre un ensemble plus ou moins grand d'utilisateurs
- assurer un ensemble de services en présentant aux utilisateurs une interface mieux adaptée à leurs besoins que celle de la machine physique

Définition : Le programme « système d'exploitation » est le programme fondamental des programmes systèmes. Il contrôle les ressources de l'ordinateur et fournit la base sur la quelle seront construits les programmes d'application.

II. Les rôles d'un système d'exploitation :

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

a) Machine étendue ou virtuelle :

Le système d'exploitation fournit aux applications, une machine virtuelle, les appels au superviseur qui étendent les capacités des jeux d'instruction proposés par la machine. Son rôle est de masquer des éléments fastidieux liés au matériel, comme les interruptions, les horloges, la gestion de la mémoire, la gestion des périphériques (déplacement du bras du lecteur de disquette) ...

Le système d'exploitation offre à l'utilisateur une interface destinée à masquer les caractéristiques matérielles. Cette interface est composée d'un ensemble de primitives qui gèrent elles-mêmes lescaractéristiques matérielles sous-jacentes et

offrent un service à l'utilisateur. Un utilisateur souhaitant réaliser une opération d'entrées sorties fait appel à une primitive unique ECRIRE sans se soucier du type de gestion associée au périphérique. C'est la primitive qui prendra en charge la spécificité du périphérique. L'ensemble des primitives offertes par le système d'exploitation crée une machine virtuelle au-dessus de la machine physique plus simple d'emploi et plus conviviale. On distingue deux types de primitives : les appels systèmes et les commandes.

b) La gestion des ressources : Le second rôle du système d'exploitation est la gestion des ressources.Un ordinateur se compose de ressources (périphériques, mémoires , terminaux, disques ...).Le système d'exploitation permet l'ordonnancement et le contrôle de l'allocation des processeurs, des mémoires et des périphériques d'E/S entre les différents programmes qui y font appel. Le système d'exploitation peut être découpé en plusieurs grandes fonctions, ces fonctions qui seront étudiées plus en détails dans les chapitres suivants du cours

III. Historique des systèmes d'exploitation :

Pour comprendre ce que sont les SE, nous devons tout d'abord comprendre comment ils sesont développés. Cela permettra de voir comment les composants des SE ont évolué comme des solutions naturelles aux problèmes des premiers SE.

A. Porte ouverte ou exploitation self service (1945-1955)

Les machines de la première génération appelées **Machines à Tubes**, étaient dépourvues de tout logiciel. Les programmes utilisateurs étaient chargés en mémoire, exécutés et mis au point depuis un pupitre de commande.Ces machines étaient énormes, remplissaient les salles avec des centaines de tubes à vide (**Vacuum Tubes**), coûteuses, très peu fiables et beaucoup moins rapides car le temps de cycle se mesurait en secondes. Les programmes étaient écrits directement en langage machine : ils étaient chargés en mémoire, exécutés et mis au point à partir d'un **pupitre de commande** (Figure 1.1). Au début de 1950, la procédure s'est améliorée grâce à l'introduction de **cartes perforées**.

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

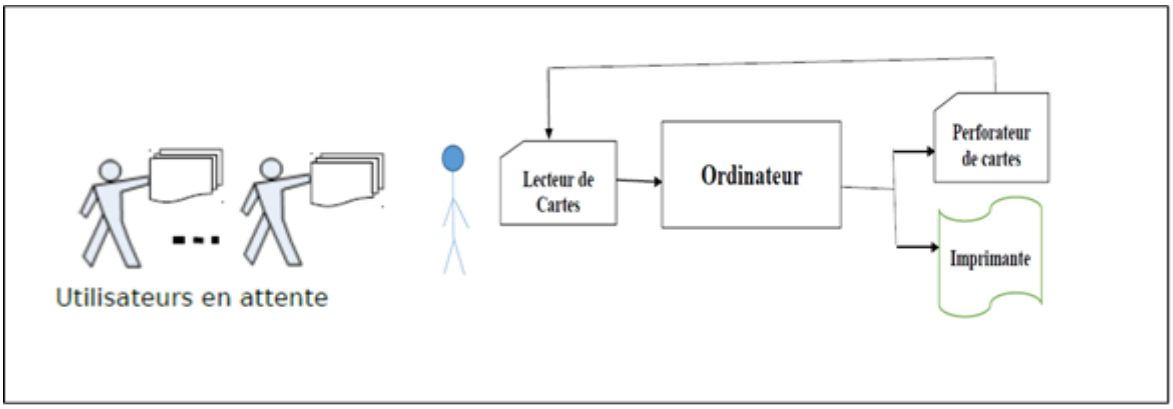


Figure 1. 1 : Utilisation de la machine en Porte Ouverte

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

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

Inconvénients

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

B. Traitement par lots (Batch Processing, 1955-1965) : Ce sont des systèmes réalisant le séquencement des jobs ou travaux selon l'ordre des cartes de contrôle à l'aide d'un moniteur d'enchaînement. L'objectif était de réduire les pertes de temps occasionnées par l'oisiveté du processeur entre l'exécution de deux jobs ou programmes (durant cette période,il y a eu apparition des machines à transistor avec unités de bandes magnétiques, donc évolution des ordinateurs).L'idée directrice était de collecter un ensemble de travaux puis de les transférer sur une bande magnétique en utilisant un ordinateur auxiliaire (Ex.IBM 1401). Cette bande sera remontée par la suite sur le lecteur de bandes de l'ordinateur principal (Ex. IBM 7094) afin d'exécuter les travaux transcrits en utilisant un programme spécial (l'ancêtre des S.E. d'aujourd'hui. Ex. FMS :Fortran Monitor System, IBSYS). Les résultats seront récupérés sur une autre bande pour qu'ils soient imprimés par un ordinateur auxiliaire. Cette situation est illustrée à la Figure 1.2.

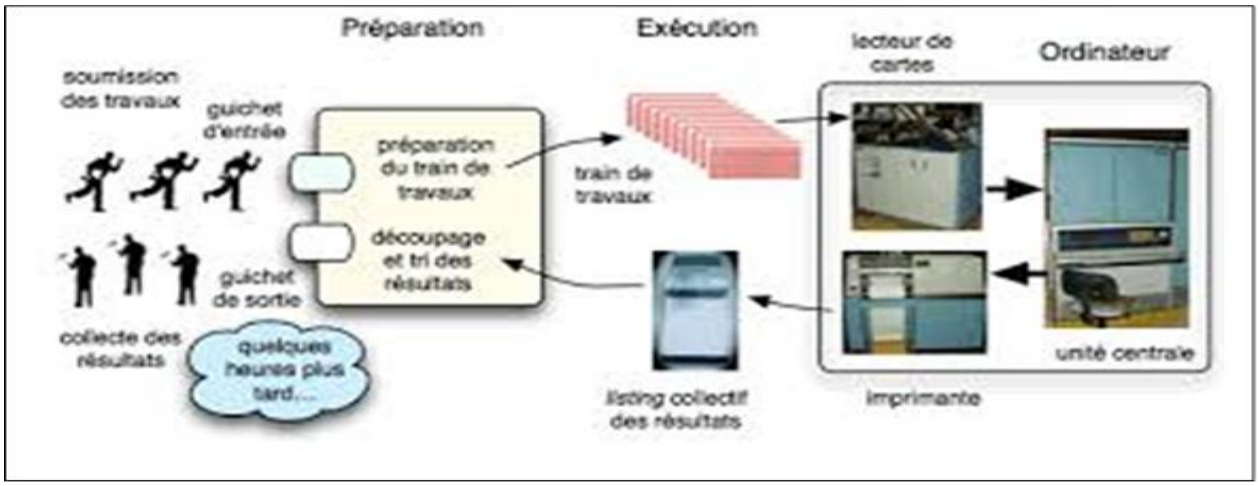


Figure 1.2 : Un système de traitement par lots

Quand le moniteur rencontre une carte de contrôle indiquant l'exécution d'un programme, il charge le programme et lui donne le contrôle. Une fois terminé, le programme redonne le contrôle au moniteur d'enchaînement. Celui-ci continue avec la prochaine carte de contrôle, ainsi de suite jusqu'à la terminaison de tous les jobs. La structure d'un travail soumis est montrée sur la Figure 1.3 :

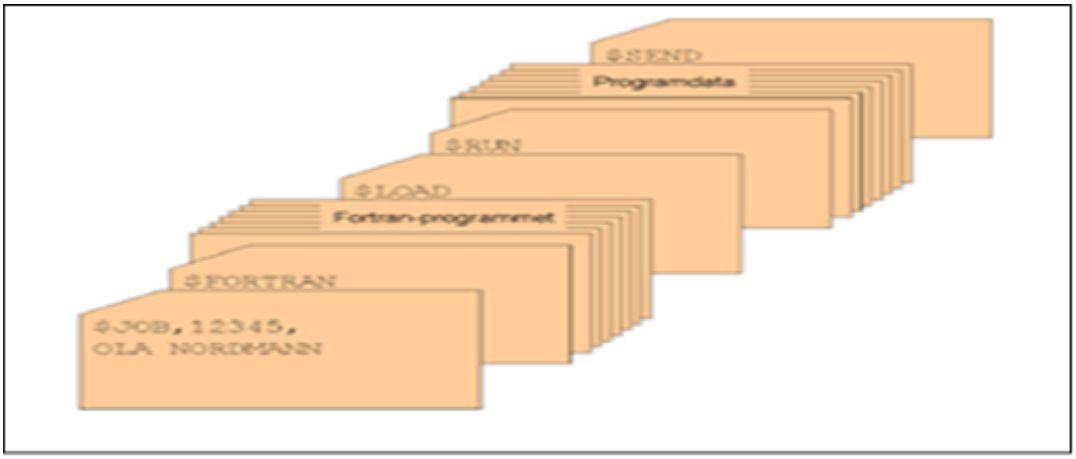


Figure 1.3: Structure d'un travail FMS typique

Inconvénients

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

C. Multiprogrammation (Multiprogramming, 1965-1970) :

L'introduction des **circuits intégrés** dans la construction des machines a permis d'offrir un meilleur rapport coût/performance. L'introduction de la technologie des **disques** a permis au système d'exploitation de conserver tous les travaux sur un disque, plutôt que dans un lecteur de cartes (Arrivée des unités disques à stockage important et introduction de **canaux d'E/S**).L'idée était alors, pour pallier aux inconvénients du traitement par lots, de maintenir en mémoire plusieurs travaux ou jobs prêts à s'exécuter, et partager efficacement les ressources de la machine entre ces jobs.

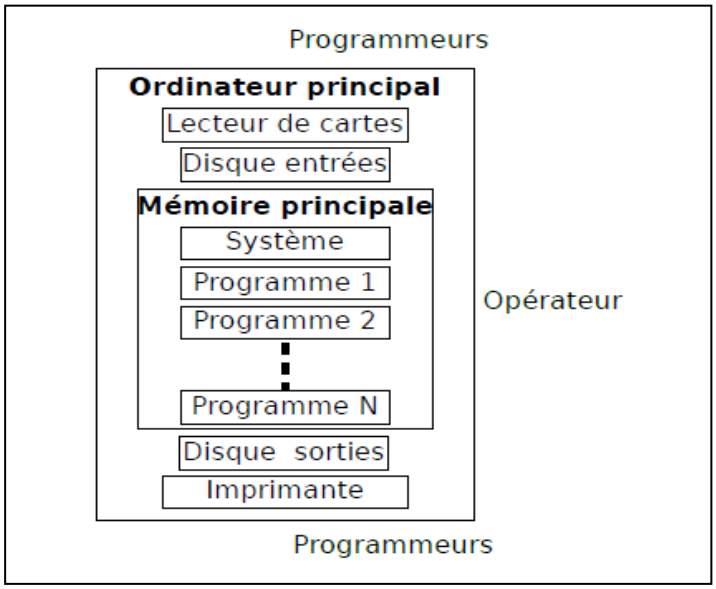


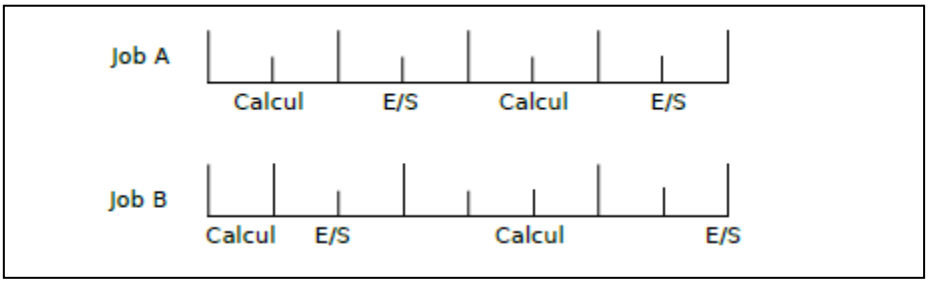
Figure 1.4 : Un système de multiprogrammation

- **Le principe de Spooling :** L'arrivée sur le marché des unités de disques (et des tambours magnétiques à des prix abordables) Possibilité de transférer les travaux vers les disques dès leur arrivée dans la salle machine.

  - Les jobs sont lus et stockés sur disque au fur et à mesure de leur soumission aux opérateurs : C'est la notion de SPOOLING (SimulatneousPeripheralOperation On-Line). Le SPOOL consiste à créer des fichiers-disque qui correspondent aux jobs soumis et qui sont utilisés par le moniteur au moment de l'exécution, à la place des supports d'informations lents.

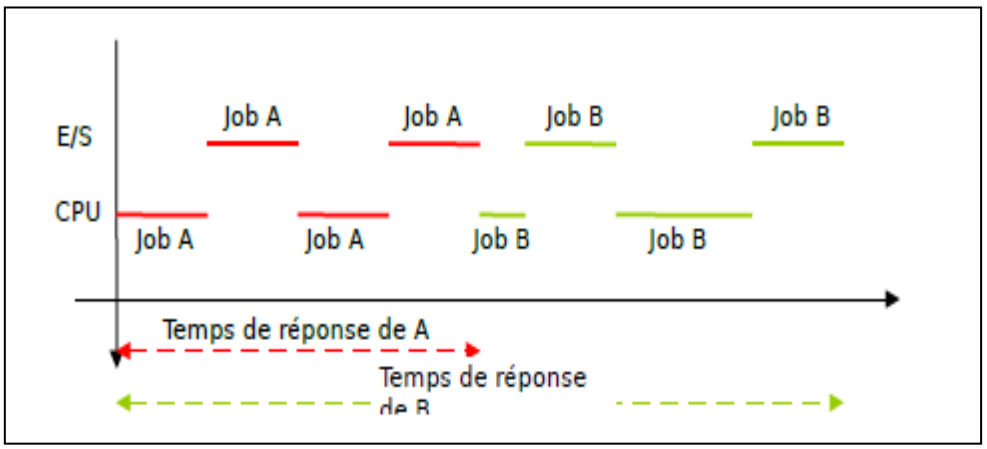
- **Exemple de multiprogrammation :**

  Pour les deux jobs Aet B :

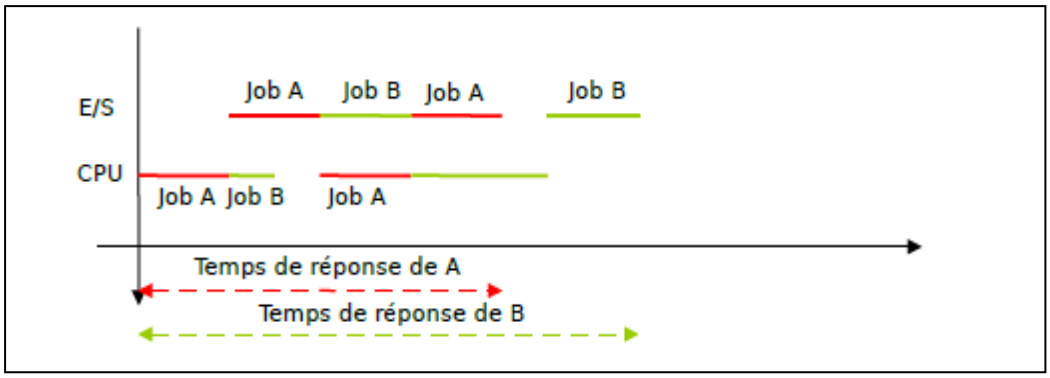


On suppose qu'on a un seul périphérique d'E/S.

- **Système mono-programmé :**



- **Système multiprogrammé :**



D. Temps partagé (Time Sharing, 1970-) :

C'est une variante du mode multiprogrammé où le temps CPU est distribué en petites tranches appelées quantum de temps.L'objectif est d'offrir aux usagers une interaction directe avec la machine par l'intermédiaire de terminaux de conversation, et de leur allouer le processeur successivement durant un quantum de temps, chaque utilisateur aura l'impression de disposer

de la machine à lui tout seul. Il peut aussi contrôler le job qu'il a soumis directement à partir du terminal (corriger les erreurs,recompiler, resoumettre le job, …).

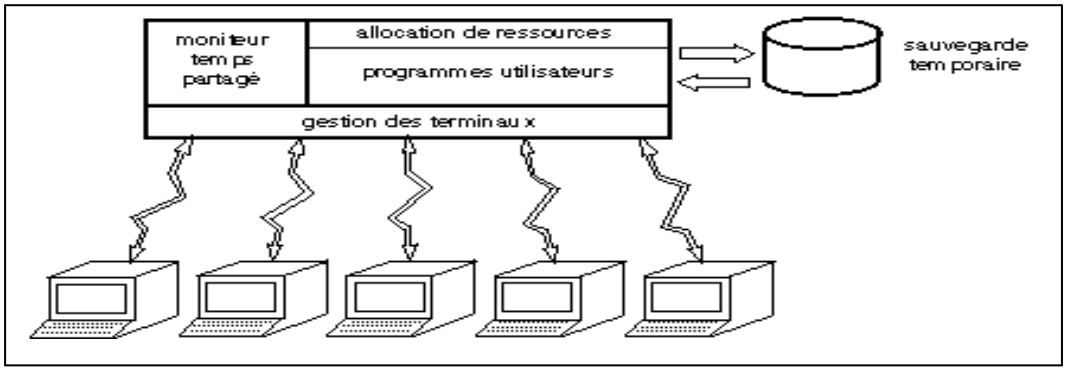


Figure 1.5 : Un système temps partagé

IV. Composants d'un système d'exploitation :

Pour créer un système aussi grand et complexe qu'un système d'exploitation, il est nécessaire de le découper en pièces plus petites. Chacune d'entre elles devrait être une portion bien délimitée du système, avec des entrées, des sorties et des fonctions soigneusement définies. Bien entendu les systèmes d'exploitation ne possèdent pas tous la même structure. Cependant, plusieurs systèmes modernes possèdent les composants suivants :

- Gestion de processeurs.
- Gestion mémoire principale.
- Gestion mémoire auxiliaire.
- Gestion du système d'entrée/sortie.
- Gestion des fichiers.
- Système de protection.
- Gestion de réseaux.
- Système interpréteur de commande.

V. Classification des SE :

a) Selon les services rendus :

- *Système monotâche* : A tout instant, un seul programme est exécuté. Un autre programme ne démarrera –sauf conditions exceptionnelles- que lorsque le premier sera terminé.

- *Multitâche* : Capacité du système à pouvoir exécuter plusieurs processus (i.e programmes en cours d'exécution) simultanément (système multiprocesseurs) ou en pseudo-parallélisme (système en time sharing).
- *Multi-utilisateurs* : Capacité du système à pouvoir gérer un ensemble d'utilisateurs utilisant simultanément les mêmes ressources matérielles. Un SE ne possédant pas cette fonctionnalité est dit mono-utilisateur.
- *Systèmes temps réel :*Les systèmes à temps réel sont une autre forme de SE spécialisés. Un système à temps réel est utilisé quand il existe des exigences impérieuses de temps de réponse pour le fonctionnement d'un processeur ou pour le jeu de données. Il est souvent employé comme dispositif de contrôle dans une application dédiée. Les capteurs amènent des données à l'ordinateur. Celui-ci doit analyser les données et éventuellement régler le contrôle pour modifier les entrées des capteurs.

  Exemple de système à temps réel : systèmes contrôlant les expérimentations scientifiques, les systèmes d'imagerie médicale, les systèmes de contrôle industriel.

b) **Selon leur architecture**

- *Systèmes centralisés (traditionnels*) : L'ensemble du système est entièrement présent sur la machine considérée. Les machines éventuellement reliées sont vues comme des entités étrangères disposant elles aussi d'un système centralisé. Le système ne gère que les ressources de la machine sur laquelle il est présent.
- *Systèmes distribués (répartis) (DistributedSystems*) : Une tendance récente dans les systèmes informatiques consiste à répartir le calcul entre plusieurs processeurs. A l'opposé des systèmes centralisés, dans les systèmes répartis on ne partage pas de mémoire ou d'horloge. Au lieu de cela, chaque processeur possède sa propre mémoire locale. Les processeurs communiquent entre eux à travers des lignes de communication, comme des bus rapides ou des lignes téléphoniques. Les systèmes sont habituellement appelés répartis (ou faiblement couplés).

c) **Selon leur capacité à évoluer :**

- *Systèmes fermés (ou prioritaires)* : Extensibilité réduite : Quand on veut rajouter des fonctionnalités à un système fermé, il faut remettre en cause sa conception et refaire une archive (système complet). C'est le cas d'UNIX, MSDOS.
- *Systèmes ouverts :*Extensibilité accrue : Il est possible de rajouter des fonctionnalités et des abstractions sans avoir à repenser le système et même sans avoir à l'arrêter sur une machine. Cela implique souvent une conception

modulaire basée sur le modèle«client/serveur». Cela implique aussi une communication entre systèmes, nécessitant des modules spécialisés.

d) **Selon l'architecture matérielle qui les supporte :**
   - *Architecture monoprocesseur* (time sharing ou multiprogrammation) :
   - *Architecture multiprocesseurs* (parallélisme).

e) **Autre classification (par grandes fonctions) :**
   - *Systèmes pour machines clientes.*
   - *Systèmes pour serveurs*
   - *Systèmes de réseaux*

VI. Structuration des SE : La conception d'un SE est basée sur une structure à couches. Elle consiste à découper le SE en un certain nombre de couches (niveaux), chacune d'entre elles étant construite au-dessus des couches inférieures. La couche inférieure (couche 0) est le matériel ; la couche supérieure (couche N) est l'interface utilisateur.

Les couches permettent d'identifier clairement les niveaux d'interdépendance entre les fonctions fournies par le SE.

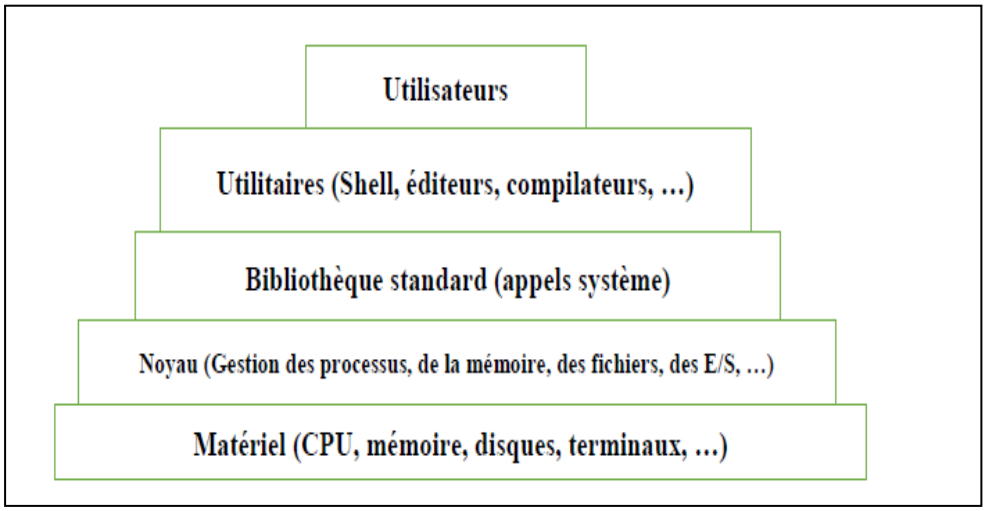


Figure 1.6 :le Modèle en couches d'un SE

Ainsi une couche i peut utiliser tous les objets (variables et routines) des couches inférieure j Le principal avantage de l'approche en couches est la modularité. Les couches sont sélectionnées de telle sorte que chacune utilise seulement les fonctions (opérations) et services des couches de niveau inférieur. Cette approche simplifie le débogage et la vérification du système.

Chapitre II
Mécanismes de base d'exécution des programmes

I. Introduction :

Dans cette partie du cours , nous étudierons les mécanismes de base utilisés par les systèmes d'exploitation pour réaliser leurs fonctions fondamentales .

II. Architecture matérielle d'une machine Von Neumann

John Von Neumann est à l'origine (1946) d'un modèle de machine universelle (non spécialisée) qui caractérise les machines possédant les éléments suivants :

- une mémoire contenant programme (instructions) et données,
- une unité arithmétique et logique (UAL ou ALU),
- une unité permettant l'échange d'information avec les périphériques : l'unité d'entrée/sortie (E/S ou I/O),
- une unité de commande (UC).

Ces dispositifs permettent la mise en œuvres des fonctions de base d'un ordinateur : le stockage de données, le traitement des données, le mouvement des données et le contrôle.

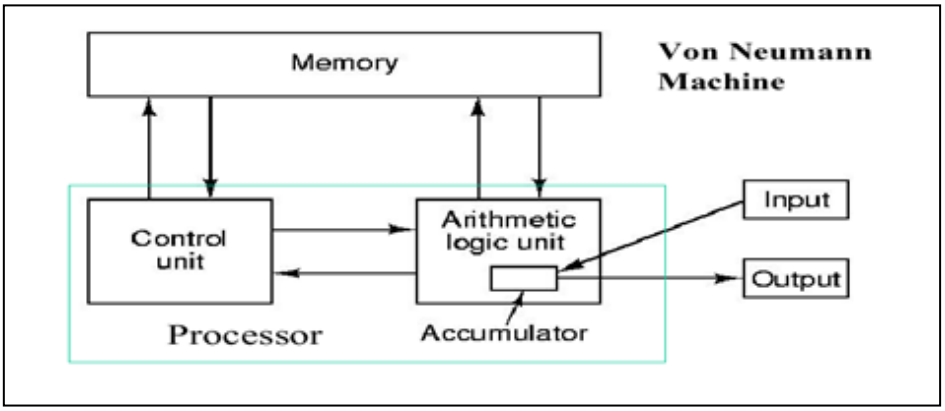


Figure 2.1:la Machine de Von Neumann

III. Les registres du processeur central :

Les registres sont une sorte de mémoire interne à la CPU, à accès très rapide qui permettent de stocker des résultats temporaires ou des informations de contrôle. Le nombre et le type des registres implantés dans une unité centrale font partie de son architecture et ont une influence

importante sur la programmation et les performances de la machine. Le processeur central dispose d'un certain nombre de registres physiques :

a) **Le Compteur Ordinal (CO, Program Counter (PC), Instruction Pointer (IP)) :** Il contient l'adresse de la prochaine instruction à exécuter.

b) **Le Registre d'Instruction (RI, Instruction Register (IR)) :** Il contient l'instruction en cours d'exécution.

c) **Les Registres Généraux (General-purposeRegisters):** Ils permettent de faire le calcul,la sauvegarde temporaire de résultats, …etc. Il s'agit de :

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

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

e) **Registre Mot d'Etat (PSW, Program Status Word) :** Il contient plusieurs types d'informations ; à savoir :

1. Les valeurs courantes des **codes conditions** (**Flags**) qui sont des bits utilisés dans les opérations arithmétiques et comparaisons des opérandes.

2. Le **mode d'exécution**. Pour des raisons de protection, l'exécution des instructions et l'accès aux ressources se fait suivant des modes d'exécution. Ceci est nécessaire pour pouvoir

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

− *Mode **privilégié** ou **maître** (ou **superviseur***). Il permet : L'exécution de tout type d'instruction. Les instructions réservées au mode maître sont dites **privilégiées** (Ex. instructions d'E/S, protection, …etc.).L'accès illimité aux données.

− *Mode **non privilégié** ou **esclave** (ou **usager**).* Il permet : Exécution d'un répertoire limité des instructions. C'est un sous ensemble du répertoire correspondant au mode maître. Accès limité aux données d'usager.

3. masque d'interruptions (seront détaillés dans la suite).

IV. Cycle principal du CPU : Un processeur ne peut exécuter qu'une seule instruction à la fois, Le cycle de processeur est le suivant :

- récupérer l'adresse de la prochaine instruction à exécuter (dans un registre spécifique)
- récupérer l'instruction elle-même (par le bus)
- exécuter l'instruction
- incrémenter le compteur ordinal pour pointer vers la prochaine instruction.

Pratiquement, toutes les instructions se déroulent en quatre étapes principales , Exécute les instructions selon un cycle : Fetch ->Decode ->Execute->suivante

- **fetch** : la recherche de l'instruction en mémoire
- **decod**e : le décodage de l'instruction
- **excute** : l'exécution de l'instruction
- le passage à l'instruction suivante

V. Etat du processeur (Processor State Information) :

Il est décrit par le **contenu** des **registres** du processeur. Le contenu des registres concerne le processus en cours d'exécution. Quand un processus est interrompu, l'information contenue dans les registres du processeur doit être sauvegardée afin qu'elle puisse être restaurée quand le processus interrompu reprend son exécution.

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

IV. cheminement d'un programme dans un système :

La chaine de production de programme transforme un programme écrit dans un langage de haut niveau en un programme dit exécutable, écrit en langage machine. Cette transformation s'effectue à l'aide de plusieurs étapes :

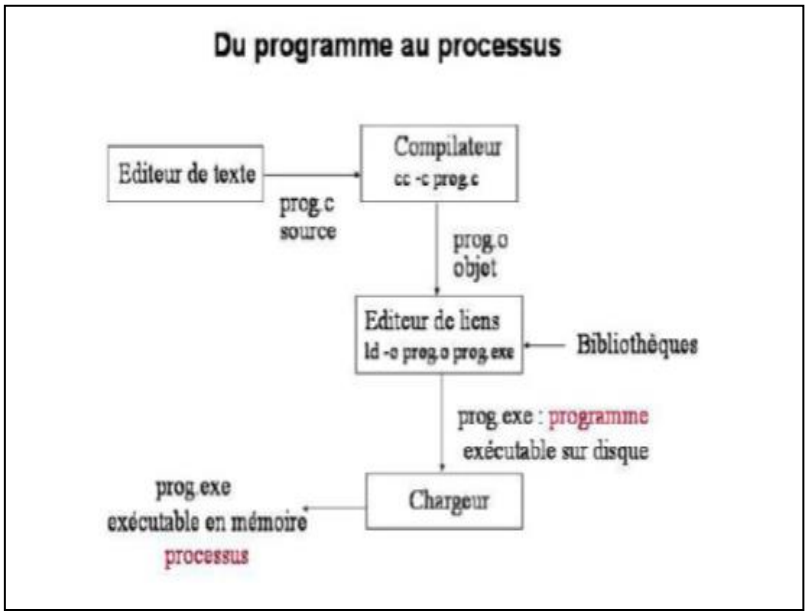


Figure 2.2: Cheminement d'un programme dans un système

a) **Etape de l'éditeur de texte :**L'utilisateur saisit son programme à l'aide de l'éditeur de texte qui crée un fichier sur le disque que l'on appelle le fichier source.

b) **Etape de compilation :**Ce fichier source est ensuite compilé à l'aide d'un compilateur dépendant du langage de programmation utilisé et dont le rôle est de vérifier que la syntaxe du langage est respectée et de générer alors l'équivalent du programme source en langage machine : on obtient alors sur disque le fichier objet.

c) **Etape de l'éditeur de lien :**Ce fichier objet est ensuite soumis à l'éditeur de liens dont le rôle est de résoudre les références externes, c'est-à-dire par exemple, d'associer les appels système inclus dans le programme à leur adresse dans le système. L'éditeur de liens produit sur disque le fichier exécutable.

d) **Etape de chargement :**Lorsque l'utilisateur demande l'exécution de son programme, le fichier exécutable est alors monté en mémoire centrale : c'est l'étape de chargement. Le système alloue de la place mémoire pour placer le code et les données du programme.

VI. le processus :

Les processus correspondent à l'exécution de tâches : les programmes des utilisateurs, les entrées-sorties, … par le système. Un système d'exploitation doit en général traiter plusieurs tâches en même temps. Comme il n'a, la plupart du temps, qu'un processeur, il résout ce

problème grâce à un pseudo-parallélisme. Il traite une tâche à la fois, s'interrompt et passe à la suivante. La commutation des tâches étant très rapide, l'ordinateur donne l'illusion d'effectuer un traitement simultané.

> *Un processus est une entité dynamique correspondant à l'exécution d'une suite d'instructions : un programme qui s'exécute, ainsi que ses données, sa pile, son compteur ordinal, son pointeur de pile et les autres contenus de registres nécessaires à son exécution.*

Attention : ne pas confondre un processus (aspect dynamique, exécution qui peut être suspendue, puis reprise), avec le texte d'un programme exécutable (aspect statique).

- **Rôle du SE pour les processus :**
    a. (Ré)Activer un processus
    b. Suspendre un processus
    c. Tuer un processus
    d. Contrôler l'exécution d'un processus De manière optimale pour le CPU
- **Les segments d'un processus : Les** processus sont composés d'un espace de travail en mémoire formé de 3 segments : la pile, les données et le code et d'un contexte.

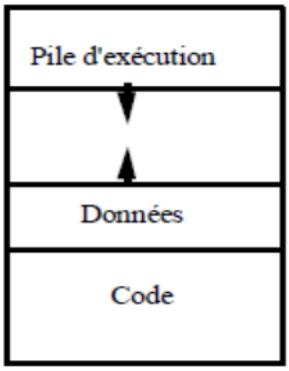


Figure 2.3: Les segments d'un processus

1. Le code correspond aux instructions, en langage d'assemblage, du programme à exécuter.
2. La zone de données contient les variables globales ou statiques du programme ainsi que les allocations dynamiques de mémoire, Parfois, on partage la zone de données en données elles-mêmes et en tas. Le tas est alors réservé aux données dynamiques.
3. Pile d'exécution : Enfin, les appels de fonctions, avec leurs paramètres et leurs variables locales, viennent s'empiler sur la pile. Les zones de pile et de données ont des frontières mobiles qui croissent en sens inverse lors de l'exécution du programme.

- **Cycle de vie d'un processus :**

  Un processus peut être dans l'´état actif si il est effectivement en cours, d'exécution bloqué en attente d'un évènement (fin d'E/S, d´elai,...) prêt si ni bloqué, ni actif.

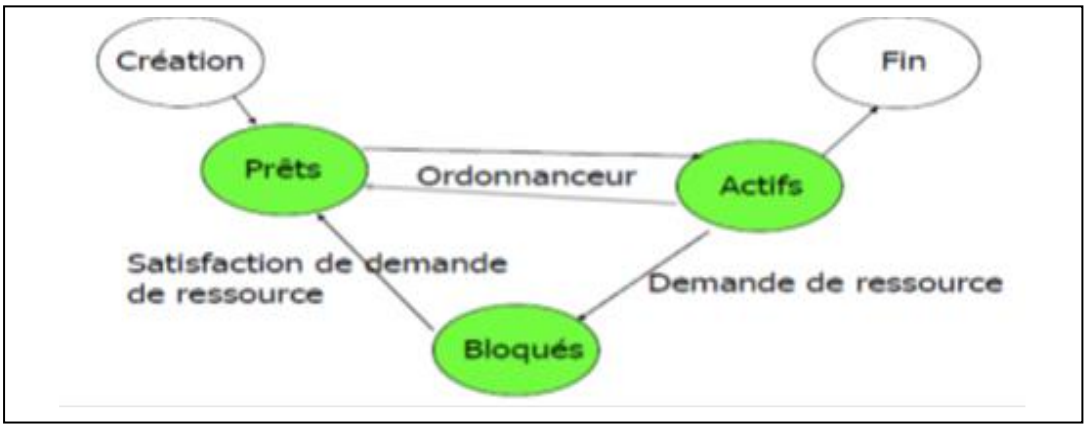


Figure 2.4: Le cycle de vie d'un processus

a) **L'état actif** : lorsque le processus obtient le processeur et s'exécute, il est dans l'état actif. L'état élu(actif) est l'état d'exécution du processus. lors de cette exécution, le processus peut demander à accéder à une ressource (réalisation d'une entrée/sortie, accès à une variable protégée) qui n'est pas immédiatement disponible : le processus ne peut pas poursuivre son exécution tant qu'il n'a pas obtenu la ressource (par exemple, le processus doit attendre la fin de l'entrée/sortie qui lui délivre les données sur lesquelles il réalise les calculs suivants dans son code) : le processus quitte alors le processeur et passe dans l'état bloqué.

b) **L'état bloqué** : est l'état d'attente d'une ressource autre que le processeur. lorsque le processus a enfin obtenu la ressource qu'il attendait, celui-ci peut potentiellement reprendre son exécution. Cependant, nous nous sommes placés dans le cadre de systèmes multiprogrammés, c'est-à-dire qu'il y a plusieurs programmes en mémoire centrale et donc plusieurs processus.. Lorsque le processus est passé dans l'état bloqué, le processeur a été alloué à un autre processus. Le processeur n'est donc pas forcément libre. Le processus passe alors dans l'état prêt.

c) **L'état Prêt** :est l'état d'attente du processeur. Le passage de l'état prêt vers l'état élu constitue l'opération d'élection. Le passage de l'état élu vers l'état bloqué est

l'opération de blocage. Le passage de l'état bloqué vers l'état prêt est l'opération de déblocage.

- **Le contexte d'un processus (PCB) :**

  Le contexte est formé des données nécessaires à la gestion des processus. Une table contient la liste de tous les processus et chaque entrée conserve leur contexte. C'est L'ensemble des informations nécessaire et suffisante pour la reprise d'un processus au cas où il aurait été interrompu, Le contexte est utilisé :

  - ✓ En lecture lors de la restauration.
  - ✓ En écriture lors de la sauvegarde

  A cet effet, à tout processus, on associe un bloc de contrôle de processus (BCP). Il comprend généralement :

  a. une copie du PSW au moment de la dernière interruption du processus
  b. l'état du processus : prêt à être exécuté, en attente, suspendu, ...
  c. des informations sur les ressources utilisées
  d. mémoire principale
  e. temps d'exécution
  f. périphériques d'E/S en attente
  g. files d'attente dans lesquelles le processus est inclus, etc...
  h. toutes les informations nécessaires pour assurer la reprise du processus en cas d'interruption

  Les BCP sont rangés dans une table en mémoire centrale à cause de leur manipulation fréquente.

  Le passage d'un processus à l'autre s'effectue par un changement de contexte

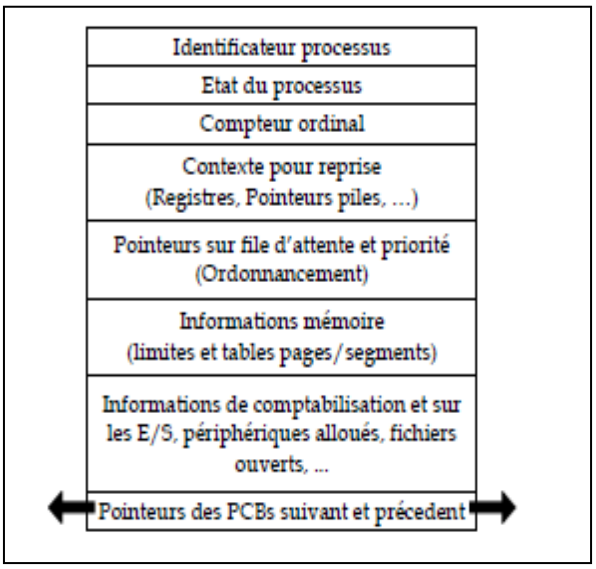


Figure 2.5: Bloc de contrôle de processus (PCB)

VII. Mécanisme de commutation de contexte :

La commutation de contexte est le mécanisme qui permet au système d'exploitation de remplacer le processus élu par un autre processus éligible. Une commutation de contexte se produit quand l'exécution d'un processus s'interrompt, ou reprend.

A tout moment, un processus est caractérisé par ces deux contextes: le contexte d'unité centrale qui est composé des mêmes données pour tous les processus et le contexte qui dépend du code du programme exécuté. Pour pouvoir exécuter un nouveau processus, il faut pouvoir sauvegarder le contexte d'unité centrale du processus courant (mot d'état), puis charger le nouveau mot d'état du processus à exécuter. Cette opération délicate réalisée de façon matérielle est appelée commutation de mot d'état. Elle doit se faire de façon non interruptibleCette "Super instruction" utilise 2 adresses qui sont respectivement:

- l'adresse de sauvegarde du mot d'état
- l'adresse de lecture du nouveau mot d'état

Le compteur ordinal faisant partie du mot d'état.

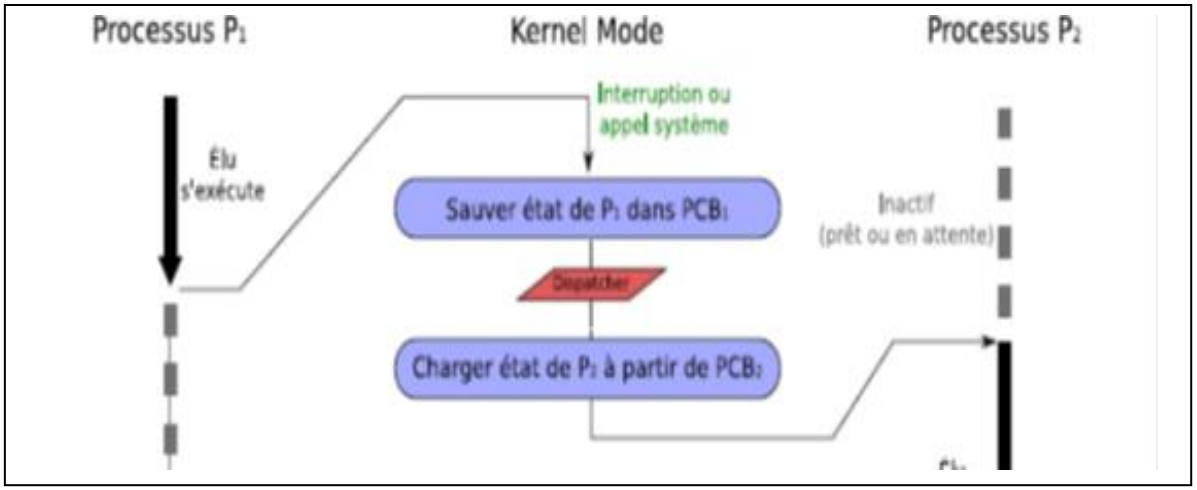


Figure 2.6: Le Mécanisme de commutation de contexte

la commutation du contexte se fait en deux phases :

− La première phase consiste à commuter le petit contexte (CO, PSW) par une instruction indivisible.

− La deuxième phase consiste quant à elle à commuter le grand contexte par celui du nouveau processus.

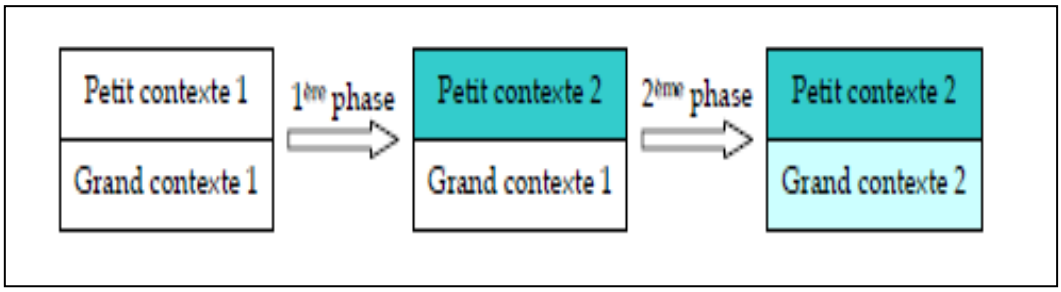


Figure 2.7: Les phases d'une commutation de contexte

VIII. La cause d'une commutation de contexte :

La commutation de contexte constitue la réponse du système à l'occurrence d'une interruption qui est un signal généré par le matériel comme conséquences à un événement qui s'est produit durant l'exécution du programme. Les événements pouvant se produire durant l'exécution d'un programme sont nombreux. Certains d'entre eux sont internes et résultent de son exécution et d'autres sont externes et indépendants de cette exécution. On parle ainsi de : l'interruption .

IX. Les interruptions :

Une interruption est un signal déclenché par un événement interne à la machine ou externe, provoquant l'arrêt d'un programme en cours d'exécution à la fin de l'opération

courante, au profit d'un programme plus prioritaire appelé programme d'interruption. Ensuite, le programme interrompu reprend son exécution à l'endroit où il avait été interrompu ou un autre programme.

- **Types d'interruptions :**

Les interruptions ne jouent pas seulement un rôle important dans le domaine logiciel, mais aussi pour l'exploitation de l'électronique.

On distingue donc :

1. **interruptions internes ou déroutement** : protection du système et des processus, appelées par une instruction à l'intérieur d'un programme (erreur d'adressage, code opération inexistant,problème de parité…)
2. **interruptions logiques ou appel système (SVC)**:permet à un processus utilisateur de faire un appel au système(software),demande d'entrée-sortie par exemple.
3. **interruptions matérielles ou externe** : déclenchées par une unité électronique (lecteur, clavier, canal,contrôleur de périphérique, panne de courant,....) ou par l'horloge (hardware externes)

Remarque : déroutements et les appels au superviseur sont déclenchés par une cause interne au programme en cours, à la différence des interruptions qui sont d'origine externe, autre processus ou par le matériel

- **Le traitement d'une interruption :** Lorsqu'une interruption survient, le processeur achève l'exécution de l'instruction en cours pour les interruptions, puis il se produit :

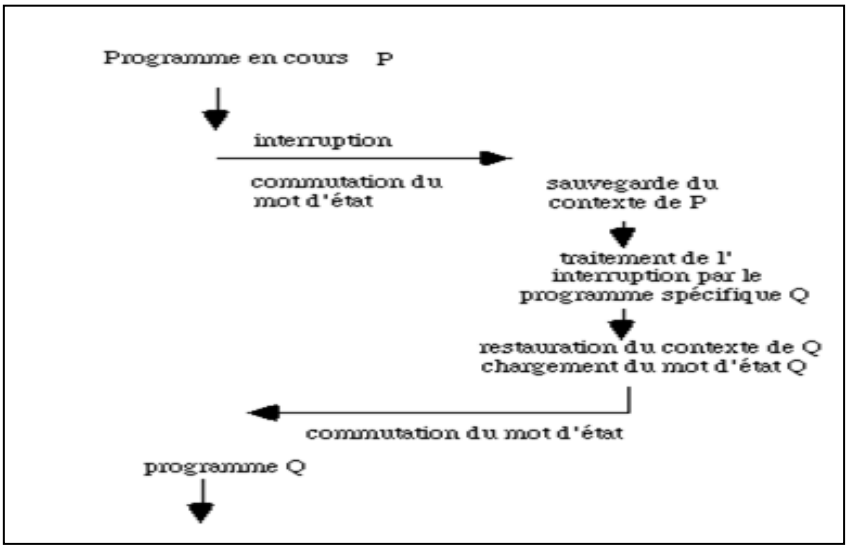


Figure 2.8: Mécanisme de gestion des interruptions

a) Sauvegarde du contexte dans une pile : - adresse de la prochaine instruction à exécuter dans le programme interrompu, - contenu des registres qui seront modifiés par le programme d'interruption, - contenu du mot d'état (registre de drapeaux) rassemblant les indicateurs (tout cela forme le contexte sauvegardé)
b) Chargement du contexte du programme d'interruption (contexte actif) et passage en mode système (ou superviseur)
c) Exécution du programme d'interruption
d) Retour au programme interrompu en restaurant le contexte et en repassant en mode utilisateur ou le choix d'un autre .

- **Les priorités des interruptions :**

Une fois que le signal est détecté dans **un point observable**, il faut déterminer la cause de l'interruption. Pour cela on utilise un indicateur, pour les différentes causes, On parle alors du vecteur d'interruptions.Pour cela on peut envisager diverses méthodes :

a) si l'indicateur d'interruption est unique, le code de l'interruption est stocké quelque part dans la mémoire et doit être lu par le programme de traitement.
b) s'il existe des indicateurs multiples, chacun est appelé niveau d'interruption. On attache un programme différent de traitement à chacun de ces niveaux.
c) on peut utiliser simultanément ces deux méthodes. Chaque niveau d'interruption est accompagné d'un code qui est lu par le programme de traitement de ce niveau.

A chaque interruption, est associée une priorité (système d'interruptions hiérarchisées) qui permet de regrouper les interruptions en classes. Chaque classe est caractérisée par un degré d'urgence d'exécution de son programme d'interruption.

Règle : Une interruption de priorité j est plus prioritaire qu'une interruption de niveau i si j > i.

On peut schématiser les différents niveaux d'interruptions comme dans le schéma suivant. La priorité va en décroissant du haut vers le bas.

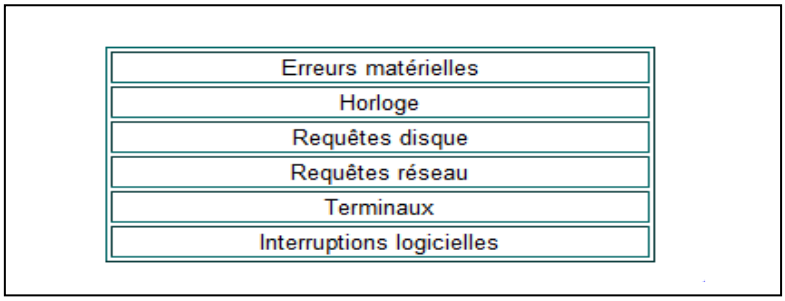


Figure 2.9: les différents niveaux d'interruptions

L'intérêt de ce système est la solution de problèmes tels que :

- ✓ arrivée de plusieurs signaux d'interruption pendant l'exécution d'une instruction,
- ✓ arrivée d'un signal d'interruption pendant l'exécution du signal de traitement d'une interruption précédente.

On peut utiliser un contrôleur d'interruptions pour regrouper les fils d'interruptions, gérer les priorités entre les interruptions et donner les éléments de calcul d'adresse au processeur.

- **Les opérations sur les interruptions :**
  - **A. Masquage des interruptions :** Certaines interruptions présentent tellement d'importance qu'il ne doit pas être possible d'interrompre leur traitement. On masquera alors les autres interruptions pour empêcher leur prise en compte. Certaines interruptions sont non-masquables : on les prend obligatoirement en compte. Une interruption masquée n'est pas ignorée : elle est prise en compte dès qu'elle est démasquée.
  - **B. Désarmer une interruption :**elle sera ignorée. Par défaut, les interruptions sont évidemment armées.

Chapitre III

Gestion des processus

I. Introduction :

Le S.E choisit un processus qui deviendra actif parmi ceux qui sont prêts. Tout processus qui se bloque en attente d'un événement (par exemple l'attente de la frappe d'un caractère au clavier lors d'un scanf) passe dans l'état bloqué tant que l'événement attendu n'est pas arrivé. Lors de l'occurrence de cet événement, le processus passe dans l'état prêt. Il sera alors

Susceptible de se voir attribuer le processeur pour continuer ses activités. Les processus en attente sont marqués comme bloqués dans la table des processus. Ils se trouvent généralement dans la file d'attente liée à une ressource (imprimante, disque, ...).

Le changement d'état d'un processus peut être provoqué par :

- un autre processus (qui lui a envoyé un signal, par exemple)
- le processus lui-même (appel à une fonction d'entrée-sortie bloquante,...)
- une **interruption** (fin de quantum, terminaison d'entrée-sortie, ...)

D'une manière générale :

- le passage de l'état actif à l'état prêt est provoqué **par le système** en fonction de sa politique d'ordonnancement (fin de quantum, préemption du processus actif si un processus plus prioritaire devient prêt dans le cas des politiques préemptives, …),
- le passage de l'état actif à l'état bloqué est provoqué par le **Programme :** exécuté par le processus.

II. Qu'est-ce que l'ordonnancement de processus ?:

La figure 1 schématise le fonctionnement d'une machine multiprocessus. Plusieurs processus sont présents en mémoire centrale. P1 est élu et s'exécute sur le processeur. P2 et P4 sont dans l'état bloqué car ils attentent tous les deux une fin d'entrée/sortie avec le disque. Les processus P3, P5 et P6 quant à eux sont dans l'état prêt : ils pourraient s'exécuter (ils ont à leur disposition toutes les ressources nécessaires) mais ils ne le peuvent pas car le processeur est occupé par P1. Lorsque P1 quittera le processeur parce qu'il a terminé son exécution, les trois processus P3, P5 et P6 auront tous les trois le droit d'obtenir le processeur. Mais le processeur ne peut être alloué qu'à un seul processus à la fois : il faudra donc choisir entre P3, P5 et P6 : c'est le rôle de l'ordonnancement qui élira un des trois processus.

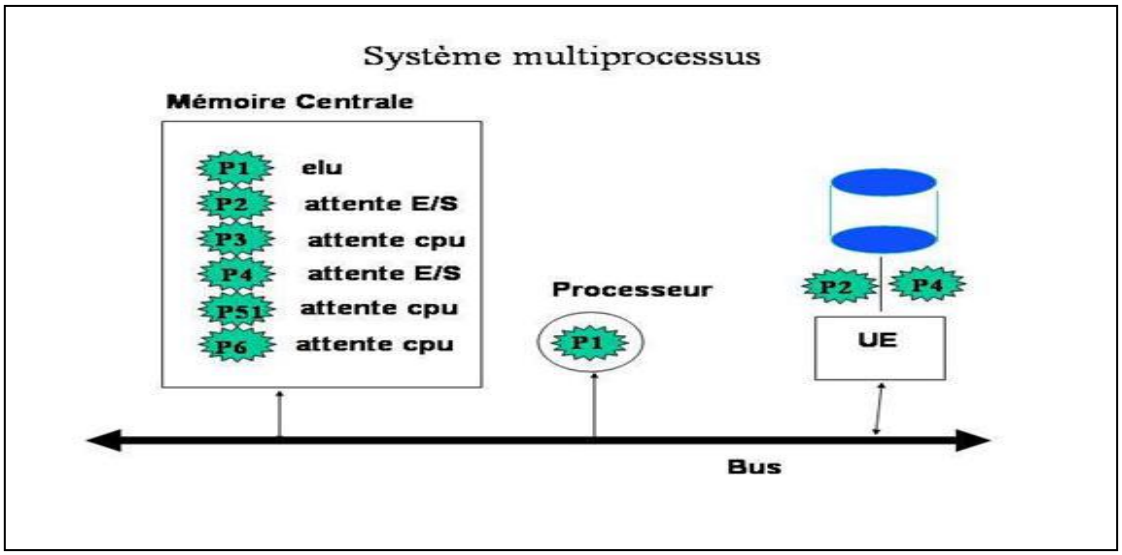


Figure 3.1: le rôle de l'ordonnanceur des processus

Définition *: La fonction d'ordonnancement gère le partage du processeur entre les différents processus en attente pour s'exécuter, c'est-à-dire entre les différents processus qui sont dans l'état prêt. L'opération d'élection consiste à allouer le processeur à un processus.*

Chaque fois, que le processeur devient inactif, le système d'exploitation doit sélectionner un processus de la file d'attente des processus prêts, et lui passe le contrôle. D'une manière plus concrète, cette tâche est prise en charge par deux routines système en l'occurrence le ***Dispatcheur*** et le ***Scheduleur (Ordonnanceur)***.

a) **Le Dispatcheur :** Il s'occupe de l'allocation du processeur à un processus sélectionné par l'Ordonnanceur du processeur. Une fois allouer, le processeur doit réaliser les tâches suivantes
   - ***Commutation de contexte :*** sauvegarder le contexte du processus qui doit relâcher le processeur et charger le contexte de celui qui aura le prochain cycle processeur.
   - ***Commutation du mode d'exécution :*** basculer du mode Maître (mode d'exécution du dispatcheur) en mode utilisateur (mode d'exécution du processeur utilisateur).
   - ***Branchement :*** se brancher au bon emplacement dans le processus utilisateur pour le faire démarrer.

b) **L'Ordonnanceur :** Certains systèmes d'exploitation utilisent une technique d'ordonnancement à deux niveaux qui intègre deux types d'Ordonnanceurs :
   - ***Ordonnanceur du processeur :*** c'est un Ordonnanceur court terme opère sur une ensemble du processus présents en mémoire. Il s'occupe de la sélection

du processus qui aura le prochain cycle processeur, à partir de la file d'attente des processus prêts.

- ***Ordonnanceur de travail :*** ou Ordonnanceur long terme, utilisé en cas d'insuffisance de mémoire, son rôle est de sélectionné le sous ensemble de processus stockés sur un disque et qui vont être chargés en mémoire. En suite, il retire périodiquement de la mémoire les processus qui sont restés assez longtemps et les remplace par des processus qui sont sur le disque depuis trop de temps.

III. Ordonnancement préemptif ou non préemptif

. Une transition a été cependant ajoutée entre l'état élu et l'état prêt : c'est la transition de préemption. La préemption correspond à une opération de réquisition du processeur, c'est-à-dire que le processeur est retiré au processus élu alors que celui-ci dispose de toutes les ressources nécessaires à la poursuite de son exécution. Cette réquisition porte le nom de préemption

Selon si l'opération de réquisition du processeur est autorisée ou non, l'ordonnancement sera qualifié d'ordonnancement préemptif ou non préemptif :

A. si l'ordonnancement est non préemptif, la transition de l'état élu vers l'état prêt est interdite : un processus quitte le processeur si il a terminé son exécution ou si il se bloque.
B. si l'ordonnancement est préemptif, la transition de l'état élu vers l'état prêt est autorisée : un processus quitte le processeur si il a terminé son exécution , si il se bloque ou si le processeur est réquisitionné.

IV. Objectifs des politiques d'ordonnancement

Les buts de l'ordonnancement diffèrent en fonction du type de systèmes.

- Systèmes de traitements par lots. Le but est de maximiser le débit du processeur ; c'est-à-dire le nombre moyen de processus traités par unité de temps.
- Systèmes interactifs. Les buts principaux de l'ordonnancement sont premièrement de maximiser le taux d'occupation du processeur, c'est-à-dire le rapport entre le temps où le processeur est actif et le temps total. En théorie, ce taux peut varier entre 0% et 100 % ; dans la pratique, on peut observer
- un taux d'occupation variant entre 40 % et 95 %. Deuxièmement, on va chercher à minimiser le temps de réponse des processus, c'est-à-dire la durée séparant l'instant de soumission du processus au système de la fin d'exécution du processus. Au mieux, le temps de réponse peut être exactement égal au temps d'exécution du

processus, lorsque le processus a immédiatement été élu et s'est exécuté sans être préempté.

- Systèmes temps réel. Le but principal est d'assurer le respect des contraintes de temps liées aux processus.

V. Critères d'évaluation de performances :

a) **Le *rendement* d'un système** : est le nombre de processus exécutés par unité de temps.

b) Le temps de réponse : est le temps qui s'écoule entre le moment où un processus devient prêt à s'exécuter et le moment où il finit de s'exécuter (temps d'accès mémoire + temps d'attente dans la file des processus éligibles + temps d'exécution dans l'unité centrale + temps d'attente + temps d'exécution dans les périphériques d'entrée/sortie).

Le temps de réponse moyen décrit la moyenne des dates de fin d'exécution :

$$TRM = \sum_{i=1}^{n} TRi \,/\, n \text{ , avec } TR_i = \text{date fin} - \text{date arrivée.}$$

c) ***Le temps d'attente :***est le temps passé dans la file des processus éligibles. On utilise en général un temps moyen d'attente calculé pour tous les processus mis en jeu pendant une période d'observation donnée.

Le temps d'attente moyen est la moyenne des délais d'attente pour commencer une exécution :

$$TAM = \sum_{i=1}^{n} TAi \,/\, n \text{ , avec } TA_i = TR_i - \text{temps d'exécution}$$

Remarque :*Pour représenter schématiquement l'évolution dans le temps des processus, on recourthabituellement à des diagrammes de Gantt*.

VI. Ordonnancement non préemptif :

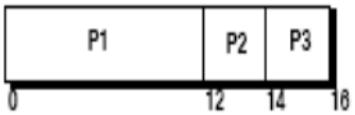
a) Ordonnancement FCFS (first come first Served) : Dans cet algorithme ; connu sous le nom FIFO (First In, First Out) ; les processus sont rangés dans la file d'attente des processus prêts selon leur ordre d'arriver. Les règles régissant cet ordonnancement sont :

→ Quand un processus est prêt à s'exécuter, il est mis en queue de la file d'attente des processus prêts.

→ Quand le processeur devient libre, il est alloué au processus se trouvant en tête de file d'attente des processus prêts.

→ Le processus élu relâche le processeur s'il se termine ou s'il demande une entrée sortie.

Exemple : On a trois processus 1, 2, 3 à exécuter de durées d'exécution respectives 12 , 2 et 2 unités de temps.



→ Si l'ordre d'arrivée est 1, 2, 3, les temps de réponse respectifs sont 12, 14, et 16 unités de temps, et le temps de réponse moyen est de 14 unités de temps.

→ Mais si l'ordre d'arrivée est 2,3,1, le temps de réponse moyen est 7,6 (2+4+16 / 3).

b) L'algorithme du Plus Court d'abord (SJF) : Cet algorithme (en anglais Shortest Job First : SJF) affecte le processeur au processus possédant le temps d'exécution le plus court. Si plusieurs processus ont la même durée, une politique FIFO sera alors utilisée pour les départager.

→ Problème : comment prédire la durée d'exécution des processus a priori ?

→ Solutions :

✓ faire de la prédiction sur la durée d'exécution des processus.

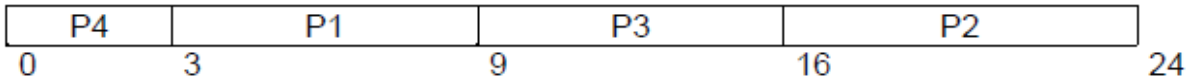
✓ faire accompagner la demande d'exécution de chaque programme d'une durée maximum d'exécution autorisée qui est utilisée comme durée d'exécution.

→ Autre problème : risque de famine (les processus de longue durée peuvent n'avoir jamais accès au processeur si des processus de courte durée arrivent en permanence)

Exemple :On soumet au système quatre processus P1, P2, P3 et P4 dont les durées d'exécution sont données par le tableau suivant :

processus	Durée d'exécution
P1	6
P2	8
P3	7
P4	3

L'algorithme du travail le plus court donnera alors le résultat suivant : (Diagramme de Gantt)



Le temps moyen d'attente est = (0+3+9+16)/4=7. Alors que si on avait choisi une politique FCFS, le temps moyen serait de : 10.25 unités de temps.

c) Ordonnancement basé sur les priorités :

Dans cet algorithme, les processus sont rangés dans la file d'attente des processus prêt par ordre décroissant de priorité. L'ordonnancement dans ce cas est régit par les règles suivantes :

1. Quand un processus est admis par le système, il est inséré dans la file d'attente des processus prêts à sa position approprie (dépend de la valeur de priorité).

2. Quand le processeur devient libre, il est alloué au processus se trouvant en tête de file d'attente des processus prêts (le plus prioritaire).

3. Un processus élu relâche le processeur s'il se termine ou se bloque (E/S ou autre).

Exemple : On dispose de 5 processus ayant des priorités différentes, comme le montre cetableau :

processus	Durée d'exécution	Priorité
P1	10	2
P2	1	4
P3	2	2
P4	1	1
P5	5	3

Diagramme de Gantt :

P2	*P5*	*P1*	*P3*	*P4*

Le temps moyen d'attente est = (0+1+6+16+18)/5=8.2 unités de temps.

VII. Ordonnancement préemptif :

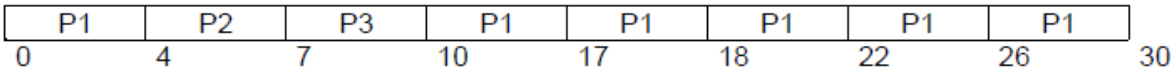
a) L'algorithme de Round Robin (Tourniquet) :

L'algorithme de scheduling du tourniquet, appelé aussi **Round Robin**, a été conçu pour des systèmes à temps partagé. Il alloue le processeur aux processus à tour de rôle , pendant une tranche de temps appelée **quantum**. Dans la pratique le quantum s'étale entre 10 et 100 ms.

→ ***Principe***: le temps est découpé en tranches ou quantum de temps. Un quantum de temps est alloué à chaque processus de la file des processus éligibles à tour de rôle. Le processus peut quitter l'unité centrale avant échéance du temporisateur s'il est terminé ou en attente d'événement. Sinon, à échéance le système reçoit une interruption et la traite en sauvegardant le contexte du processus et en le plaçant à la fin de la file des processus éligibles. La qualité de cet algorithme va beaucoup dépendre de la taille

choisie pour le quantum : celle-ci doit être grande par rapport au temps nécessaire pour le changement de contexte des processus.

Exemple : On dispose de 3 processus P1, P2 et P3 ayant comme durée d'exécutions, respectivement 24, 3 et 3 ms. En utilisant un algorithme Round Robin, avec un quantum de4 ms, on obtient le diagramme de Gantt suivant :



Le temps moyen d'attente est de : (6+4+7)/3 = 17/3 = 5.66 ms

b) **Ordonnacement SRTF (ShortestRemaining Time first) (plus court temps d'exécution restant PCTER)** :SRTF choisit le processus dont le temps d'exécution restant est le plus court. C'est une variante de l'algorithme SJF Cet algorithme est non implantable parce qu'il suppose, entre autres, connu le temps d'exécution réel de chaque processus pour pouvoir calculer le temps restant.

c) avec priorité (avec réquisition) :

On associe une priorité à chaque processus et on choisit dans la file des processus prêts le processus qui a la priorité la plus haute. Si 2 processus ont la priorité la plus haute égale, on choisit le 1er dans la file d'attente. Lorsqu'un processus arrive dans l'état prêt, le système d'exploitation regarde si la priorité de ce processus est strictement supérieure à celui du processus en exécution.

d) Priorités dynamiques :

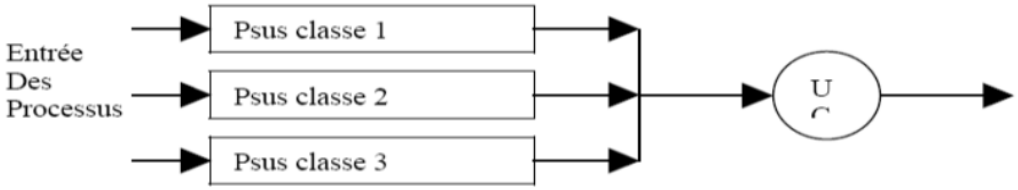
Si le système est très chargé, les processus de faible priorité n'obtiennent jamais le CPU(famine ou *starvation*), pour pallier à ce défaut, la priorité doit être un paramètre dynamique qui permettra de changer le processus de file (files réactives), l'ajustement dynamique de la priorité d'un processus est calculé par l'ordonnanceur.

- **Politique d'allocation avec priorité (Recyclage avec priorité variable) :** Les priorités sont calculées de deux façons différentes: Si un processus est arrêté par un événement E (entrée-sortie, fonctions wait, pause ...), il

est réactivé avec une priorité qui dépend du type de cet événement. La priorité des processus prêts est directement liée à la consommation de temps CPU et autemps qui s'est écoulé depuis leur dernier accès au processeur.

e) Les files multi niveaux :

Les processus sont regroupés par classe, et à chacune est associée une priorité et politique d'allocation.



L'ordonnanceur gère plusieurs politiques (c.-à-d. plusieurs queues)

Caractéristiques :

- En général préemptif : priorité préemptive
- Nombre de files utilisées
- Ordonnancement utilisé pour chaque file
- Dans quelle file est passé un processus tout neuf

Exemple:

- file 1 (ou politique 1): round-robin
- file 2 (ou politique 2): premier arrivé-premier servi (allocation du CPU uniquement si la queue 1 est vide)

f) Les files multi niveaux avec feedback:

Plusieurs queues (politiques) les processus peuvent passer d'une queue à une autre processus prêt sont gérer dans plusieurs files de priorités,et il est possible de faire passer un processus d'une file à une autre.

Caractéristiques :

- En général préemptif : priorité préemptive
- Nombre de files utilisées
- Ordonnancement utilisé pour chaque file
- Conditions/méthodes de passage d'une file à l'autre
- Dans quelle file est passé un processus tout neuf

Chapitre IV

Gestion de la mémoire

I. Introduction :

La mémoire principale est le lieu où se trouvent les programmes et les données quand le processeur les exécute. On l'oppose au concept de mémoire secondaire, représentée par les disques, de plus grande capacité, où les processus peuvent séjourner avant d'être exécutés. La nécessité de gérer la mémoire de manière optimale est toujours fondamentale, car en dépit de sa grande disponibilité, elle n'est, en général, jamais suffisante. Ceci en raison de la taille continuellement grandissante des programmes. Nous verrons qu'il y a plusieurs schémas pour la gestion de la mémoire,avec leurs avantages et inconvénients.

II. 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 busd'adresse et le bus de contrôle doit déclencher l'opération. Les données àl'adresse choisie sont remplacées par celles sur le bus de données.

III. Allocation de la mémoire centrale et multiprogrammation :

On définit le degré de multiprogrammation comme étant le nombre de processus présents en mémoire centrale. Dans un système multiprogrammé, trois problèmes sont à résoudre vis-à-vis de la mémoire centrale :

- → Il faut définir un espace d'adressage indépendant pour chaque processus.
- → Il faut protéger les espaces d'adressage des processus les uns vis-à-vis des autres
- → Il faut allouer de la mémoire physique à chaque espace d'adressage.

IV. Différentes méthodes d'allocation mémoire :

Les méthodes d'allocation mémoire peuvent être divisées en deux grandes familles :

A. ***Allocation contiguë (Contiguous Allocation)*** : pour la première famille, un programme est un ensemble de mots contigus insécable. L'espace d'adressage du processus est linéaire. On trouve ici les méthodes d'allocations en partitions variables que nous allons étudier en premier.

B. ***Allocation non contiguë (Non Contiguous Allocation)*** : pour la seconde famille, un programme est un ensemble de mots contigus sécable, c'est-à-dire que le programme peut être divisé en plus petits morceaux, chaque morceau étant lui-même un ensemble de mots contigus. Chaque morceau peut alors être alloué de manière indépendante. On trouve ici les mécanismes de segmentation et de pagination..

V. Caractéristiques liées au chargement d'un programme :

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

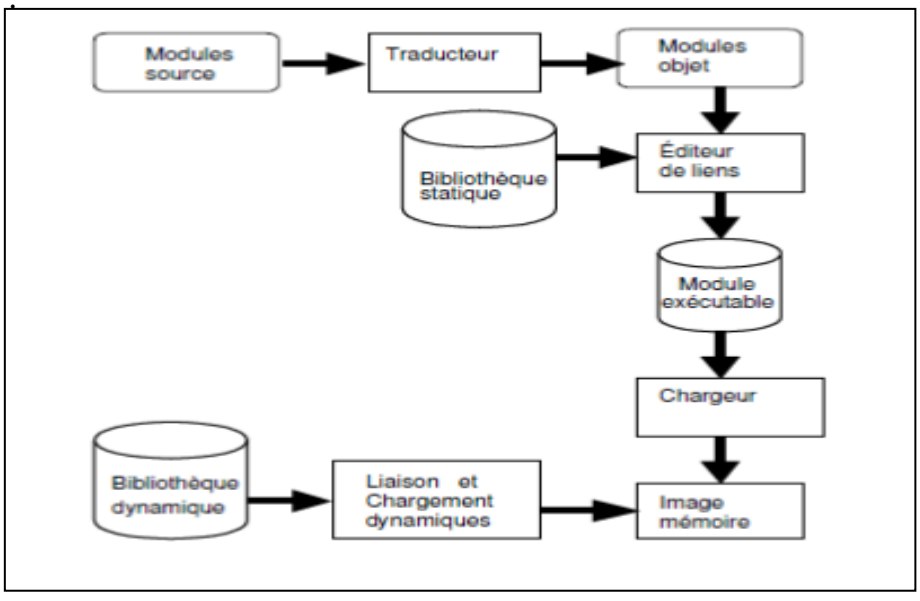


Figure 4.1: Traitement en plusieurs Phases d'un Programme Utilisateur

Le programme réside sur disque comme Fichier exécutable. Il est ensuite chargé en mémoire pour être exécuté. Il peut résider, le plus souvent, dans une partie quelconque de la mémoire. On les appelle Programmes translatables.

→ **Type d'adresse :**

a) *adresses symboliques* : au niveau de Code source Par exemple : int compteur

b) *adresses traduites* : pour le Module objet : Par exemple le 50ème mot depuis le début d'espace mémoire.

c) *adresses absolues* : Module exécutable, chargement : Par exemple l'emplacement mémoire situé à l'adresse 7777

→ **Espace d'adressage logique ou physique :**

1) **L'espace d'adressage logique :** 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.

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

→ **Le passage d'une adresse logique vers une adresse physique :**

A. ***Pendant la compilation*** : Si l'emplacement du processus en mémoire est connu, le compilateur génère des adresses absolues.

B. ***Pendant le chargement*** : après la compilation un Code translatable si l'emplacement du processus en mémoire n'est pas connu.

C. ***Pendant l'exécution*** : déplacement dynamique du processus possible.Utilise des fonctions du matériel.

VI. La gestion de la mémoire : La gestion de la mémoire a deux objectifs principaux : d'abord le partage de mémoire physique entre les programmes et les données des processus prêts, et ensuite la mise en place des paramètres de calcul d'adresses, permettant de transformer une adresse virtuelle en adresse physique. Pour ce faire le gestionnaire de la mémoire doit remplir plusieurs tâches :

- ✓ Connaître l'état de la mémoire (les parties libres et occupées de la mémoire).
- ✓ Allouer de la mémoire à un processus avant son exécution.
- ✓ Récupérer l'espace alloué à un processus lorsque celui-ci se termine.

- ✓ Traiter le va-et-vient (swapping) entre le disque et la mémoire principale lorsque cette dernière ne peut pas contenir tous les processus.
- ✓ Posséder un mécanisme de calcul de l'adresse physique (absolue) car,en général, les adresses générées par les compilateurs et les éditeurs de liens sont des adresses relatives ou virtuelles.

VII. 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.

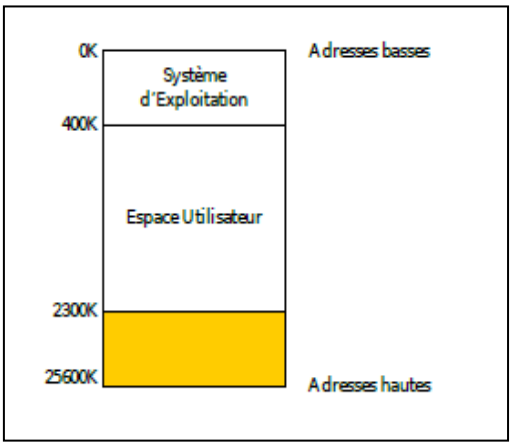


Figure 4.2: Organisation de la mémoire en Monobloc

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.

→ **Partitions fixes:**

Dans ce schéma, plusieurs programmes peuvent partager la mémoire. Pour chaque utilisateur il existe une partition. La méthode la plus simple consiste à diviser la mémoire en partitions qui peuvent être de tailles égales ou inégales. On appelle ceci Mutiprogramming with a Fixed Number of Tasks (**MFT).**Avec MFT, le nombre et la taille des partitions sont fixés à l'avance. Par exemple, avec une mémoire de 32 Ko, et un moniteur utilisant 10 Ko, le reste de 22 Ko peut être divisé en 3 partitions fixes de 4Ko, 6 Ko et 12 Ko.

La décision sur le nombre et la taille des partitions dépend du niveau de multiprogrammation voulu. En tout cas, le programme avec le plus grand besoin de mémoire doit avoir au moins, une partition pour y mettre ses données et instructions.MFT doit implanter des mécanismes de protection pour permettre à plusieurs programmes de s'exécuter sans envahir l'espace d'adresses d'autre programme. Pour cela il existe deux registres qui indiquent la plage des adresses disponibles pour chaque programme. Les deux interprétations courantes sont les **registres limite** et les **registres base-limite.**

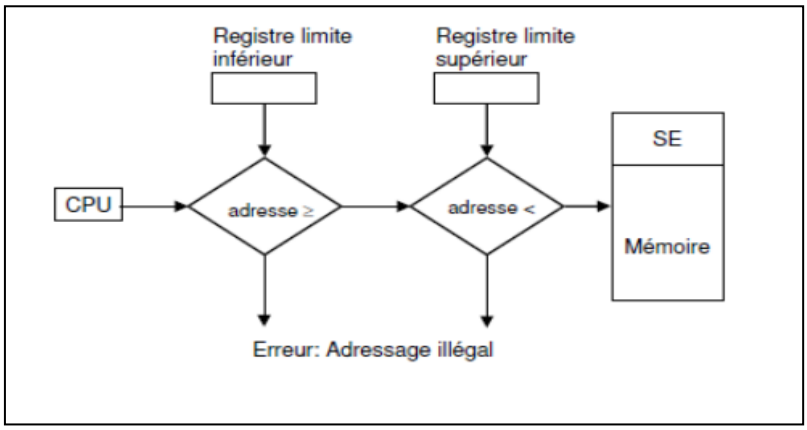


Figure 4.3 : Adresse avec registre limite

Dans ce cas, les valeurs des registres indiquent la valeur de l'adresse de mémoire la plus petite et la plus grande qu'un programme puisse atteindre. Il y a une paire de valeurs de ces registres limite pour chaque programme en exécution. Chaque accès en mémoire entraîne deux vérifications : la première pour ne pas aller en dessous de la limite inférieure, et la seconde pour ne pas aller au-dessus de la limite supérieure de mémoire.

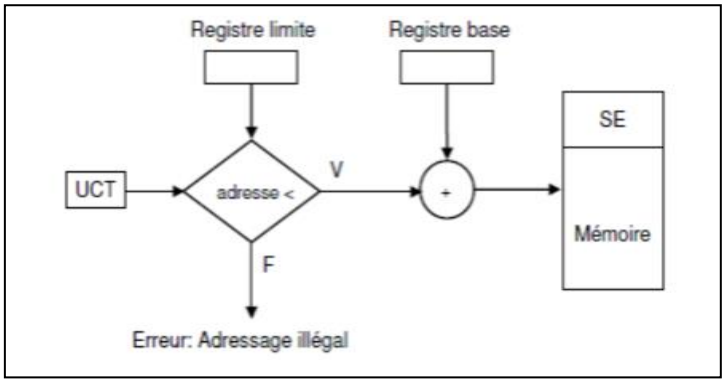


Figure 4.4 : Adresse avec registre base-limite

→ **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.

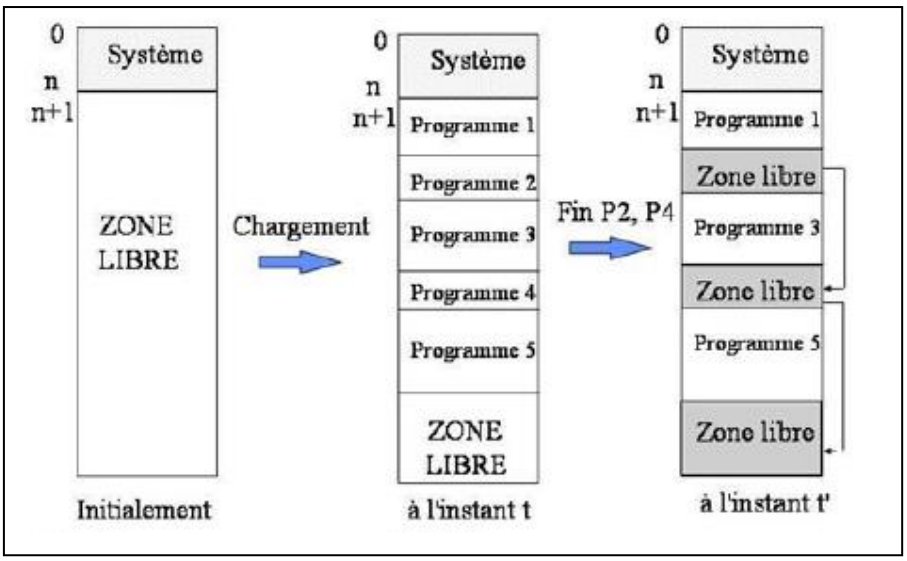


Figure 4.5 : Allocation par partitions variables

- **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.

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

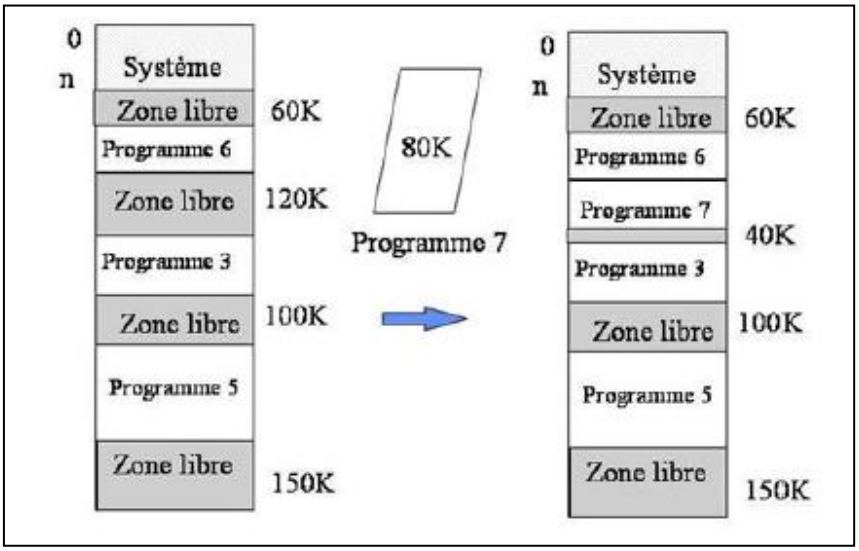


Figure 4.6 : stratégie First-fit

b) **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.

c) **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.

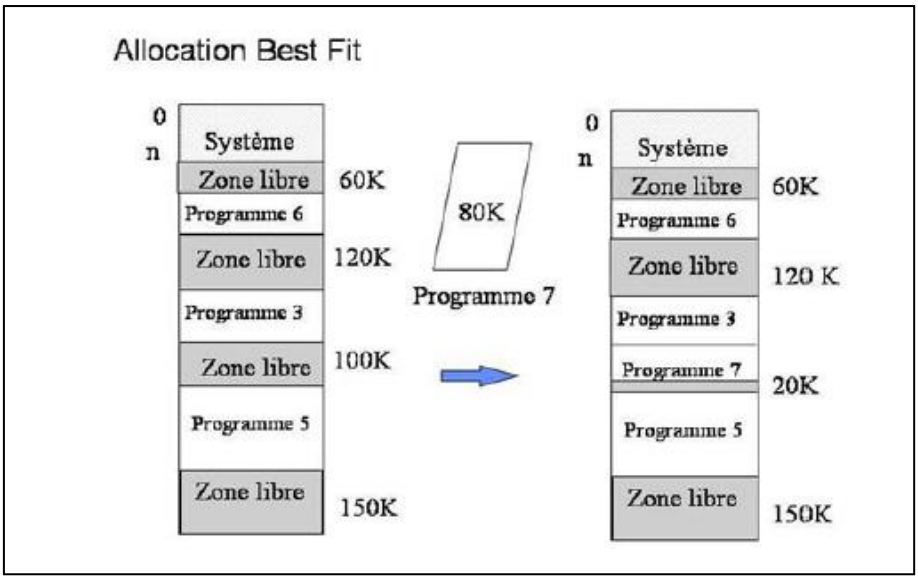


Figure 4.7: stratégie Best-fit

d) **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.

e) **Quick-fit (placement rapide) :** On utilise des listes de trous de même taille. On peut, par exemple, utiliser une table de n entrées, où la première entrée pointe sur la tête d'une liste chaînée des zones de4K, la deuxième sur la tête d'une liste chaînée des zones de 8K, …etc.

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

→ **La fragmentation mémoire:**

On fait face au problème de fragmentation, puisqu'on génère deux types d'espaces de mémoire non utilisés :

- ***La fragmentation interne :*** la partie d'une partition non utilisée par un processus est nommée fragmentation interne. Ceci est dû au fait qu'il est très difficile que tous les processus rentrent exactement dans la taille de la partition.
- ***La fragmentation externe :*** Les partitions qui ne sont pas utilisées, engendrent la fragmentation externe.

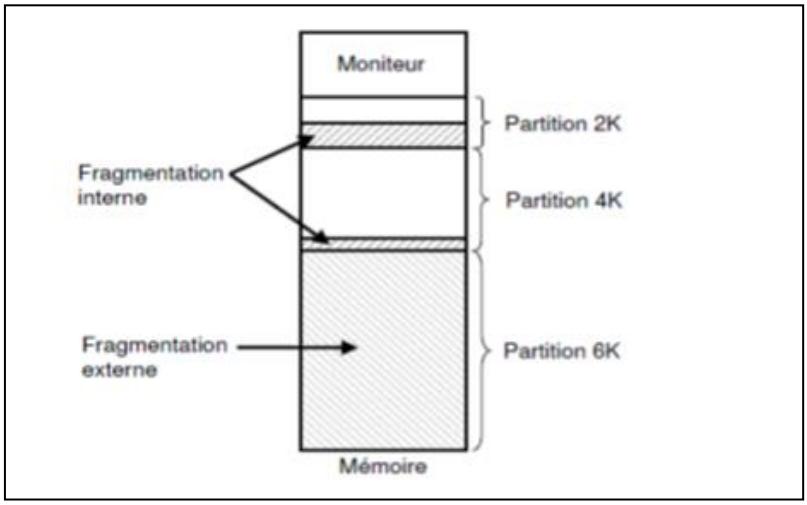


Figure 4.8: Problème de fragmentation

→ **Le compactage : une solution au problème de la fragmentation externe :**

Le compactage est une solution au problème de la fragmentation externe. Au fur et à mesure des opérations d'allocations et de désallocations, la mémoire centrale devient composée d'un ensemble de zones occupées et de zones libres éparpillées dans toute l'étendue de la mémoire centrale. Ces zones libres peuvent devenir trop petites pour permettre l'allocation de nouveaux programmes (problème de fragmentation de la mémoire).il faut donc réunir l'ensemble des zones libres pour ne former plus qu'une zone libre suffisante : c'est l'opération de compactage de la mémoire centrale.

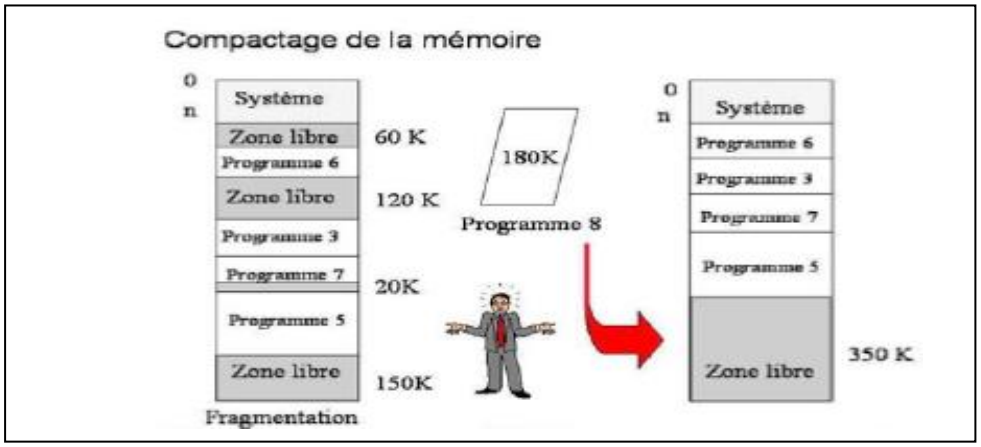


Figure 4.9: le compactage de la mémoire

→ **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 oeuvre lorsque tous les processus ne peuvent pastenir 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 **BackingStore**), 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**.

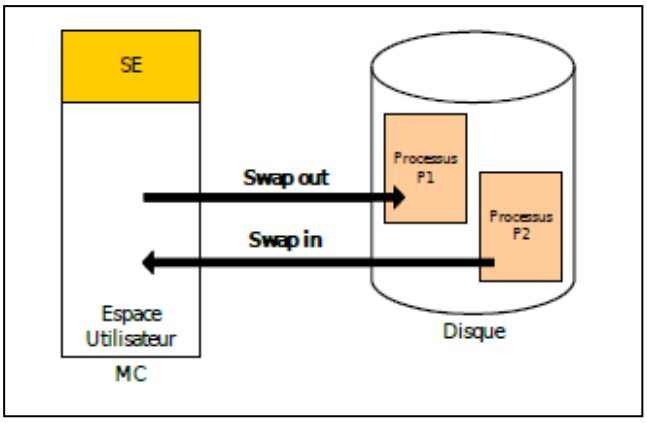


Figure 4.10: le système de swaping

VIII. 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. La pagination :

Les pgms sont découpés en parties que l'on nomme pages. La mémoire physique est, elle aussi, Découpée en parties appelés cadres ou cases, de même taille.Pour exécuter un pgm , le SE charge en mémoire centrale, uniquement, une ou plusieurs pages. Cette partie est dite résidente. Les pages sont chargées en MC à la demande, donc elles ne sont pas Forcément contiguës. Lorsqu'un pgm fait référence à une partie qui ne réside pas en MC, son exécution est suspendue (bloqué), le SE effectue une demande d'E/S pour ramener la partie référencée.

→ **Pages et cases :**La MV et la MC sont structurés en unités d'allocation appelées, respectivement, pages et cases. La taille d'une page est fixe et est égale à celle d'une case. Généralement, elle varie entre 2 ko et 16ko.

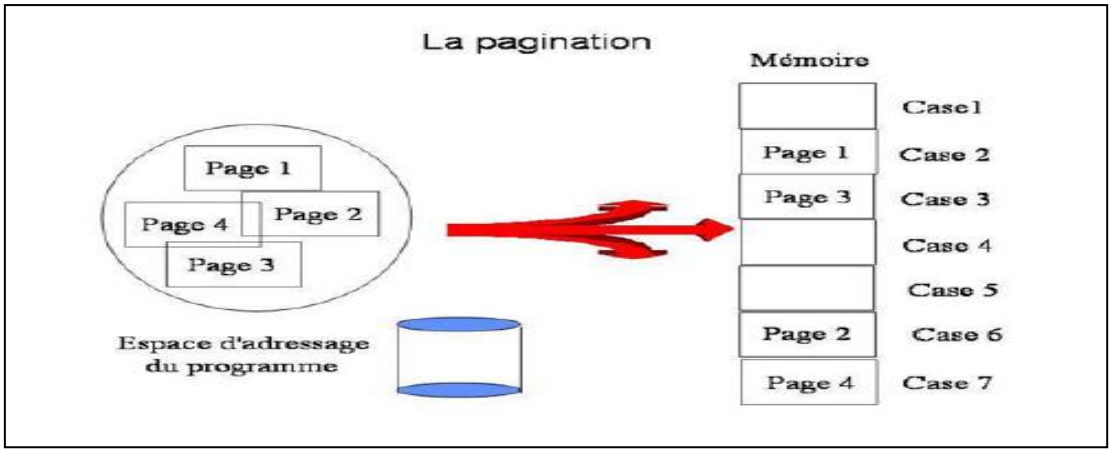
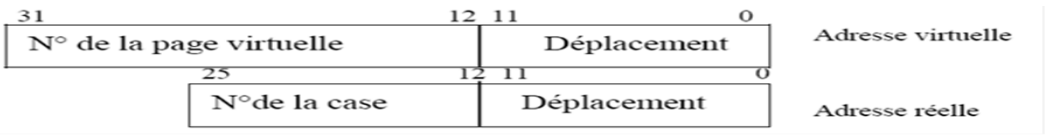


Figure 4.11 :principe de la pagination

→ **Traduction des adresses virtuelles en adresses réelles :** La conversion de l'adresse virtuelle en adresse réelle est effectuée, généralement par le circuit appelé MMU (circuit matériel de gestion mémoire).L'adresse virtuelle est 1 couple composé de n° de la page et d'un déplacement relatif au début de la page. L'adresse réelle est 1 couple composé d'un n° de la case et d'un déplacement au sein de la case. La correspondance entre les pages et les cases est mémorisée dans une table appelée Table des pages .Le nombre d'entrées (éléments) dans cette table est égale au nombre de pages virtuelles. A chaque page virtuelle présente correspond une case en MC.



A chaque pgm est associé une table de pages, l'adresse de celle-ci fait partie du PCB du pgm. Laquelle doit être en totalité ou en partie en MC lors de l'exécution du pgm.

- → **Structure de la table des pages :**Chaque entrée de la table des pages est composée de plusieurs champs :- P : bit de présence de la page en MC ;- R : bit de référence de la page ;- Pr : bits de protection (L/E, exécution, droits d'accès) ;- M : bit de modification de la page (nécessité de la réécrire sur le disque) ;- N° de la case correspondant à la page.

Si la page virtuelle n'est pas présente en MC, alors il se produit ce que l'on appelle un défaut de page.

- → **La pagination à la demande :** Lorsqu'une adresse référencée appartient à une page n'ayant pas de copie en mémoire , on dit qu'on à un défaut de page .le système cherche alors sur le disque une copie de la page demandée, la charge en mémoire en déplaçant une autre page et met à jour la table des pages. cette méthode de gestion de la mémoire, qui consiste à charger une page seulement si elle est référencée, est appelée pagination à la demande
- → **Traitement de Défauts de page :** Lorsque l'adresse virtuelle référence une page non présente en MC. Le mécanisme d'adressage (MMU) génère un défaut de page (déroutement).Si la MC est pleine : 1) Choisir une page victime, en fonction des algorithmes ; 2) Si elle est modifiée, il faut la réécrire.3) Charger la page référencée en MC (placement) ;4)Modifier les indicateurs de Présence et de Référence dans la table de pages
- → **Algorithmes de remplacement :**A la suite d'un défaut de page, le SE doit ramener en MC la page manquante à partir dudisque. S'il n'y a pas de cases libres, il doit retirer une page de la MC pour la remplacer par celle demandée. D'où la question : Quelle est la page à retirer de manière à minimiser le nombre de défauts de page ?
  - **Algorithme de remplacement de page FIFO** : Cet algorithme mémorise dans une file, de type FIFO, les pages présentent en MC. Lorsqu'un défaut de page (DP) se produit, il retire la plus ancienne, c'à d, celle qui se trouve en tête de file.

***Exemple* :** Soient une MC à 3 cases et la suite de références suivantes :

W= [7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1]

Case\Page	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
0	**7**	7	7	**2**	2	2	2	**4**	4	4	**0**	0	0	0	0	0	0	**7**	7	7
1		**0**	0	0	0	**3**	3	3	**2**	2	2	2	2	**1**	1	1	1	1	**0**	0
2			**1**	1	1	1	**0**	0	0	**3**	3	3	3	3	**2**	2	2	2	2	**1**

On constate 15 défauts de page. L'algorithme est rarement utilisé car il génère beaucoup de défauts de page.

- **Algorithme de remplacement de page la moins récemment utilisée (LRU):**Mémorise dans une file chaînée toutes les pages en MC. La page la plus utilisée est en tête de liste et la moins utilisée est en queue de liste. Lors qu' un défaut de page se produit, la page la moins utilisée est retirée.

Exemple: Soit l'exemple précédent :

Case\Page	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
0	**7**	7	7	**2**	2	2	2	**4**	4	4	**0**	0	0	**1**	1	1	1	1	1	1
1		**0**	0	0	0	0	0	0	0	**3**	3	3	3	3	3	**0**	0	0	0	0
2			**1**	1	1	**3**	3	3	**2**	2	2	2	2	2	2	2	2	**7**	7	7

L'algorithme est relativement coûteux et difficile d'implémentation, car il faut mémoriser le temps à chaque fois qu'une page est référencée, et utiliser une pile dans laquelle est chargée au sommet de pile la page à laquelle on fait référence. Donc, la page la moins récemment se trouve en queue de pile.

Remarque : plusieurs autres algorithmes exemple : OPTIMUM, RANDOM, Seconde chance, NRU.

B. La segmentation :

La pagination constitue un découpage de l'espace d'adressage du processus qui ne correspond pas à l'image que le programmeur a de son programme. Pour le programmeur, un programme est généralement constitué des données manipulées par ce programme, d'un programme principal, de procédures séparées et d'une pile d'exécution. La segmentation est un découpage de l'espace d'adressage qui cherche à conserver cette vue du programmeur. Ainsi, lors de la compilation, le compilateur associe un segment à chaque morceau du programme compilé. Un segment est un ensemble d'emplacements mémoire consécutifs non sécable. A la différence des pages, les segments d'un même espace d'adressage peuvent être de taille différente. D"une manière générale, on trouvera un segment de code, un segment de données et un segment de pile.

→ **Traduction de l'adresse segmentée vers l'adresse physique :**

D'une manière similaire à ce qui se passe avec la pagination, la segmentation de l'espace d'adressage d'un processus génère des adresses segmentées, c'est-à-dire qu'un octet est repéré par son emplacement relativement au début du segment auquel il appartient. L'adresse d'un octet est donc formée par le couple :

<n°de segment pour le mot, déplacement relativement au début du segment >.

Plus précisément la conversion d'une adresse segmentée <s,d> avec s, numéro de segment et d déplacement dans le segment suit les étapes suivantes, en mettant en jeu un registre processeur qui contient en partie haute, le nombre maximal de segments de l'espace d'adressage couramment actif (LT) et en partie basse l'adresse de la table des segments de l'espace d'adressage couramment actif.

- ✓ s est comparé à LT. Si s >= à LT alors il y a erreur : le segment adressé n'existe pas.
- ✓ sinon s est additionné à l'adresse de la table des segments de manière à indexer l'entrée de la table concernant le segment s. On récupère alors l'adresse d'implantation du segment s en mémoire centrale (adr début)
- ✓ Une information sur la taille du segment peut être conservée dans la table : d est alors comparé à cette information. Si d est supérieure à l'information taille, alors une erreur est générée car le déplacement est en dehors du segment. Sinon , le déplacement d est ajouté à l'adresse d'implantation du segment pour générer l'adresse physique.

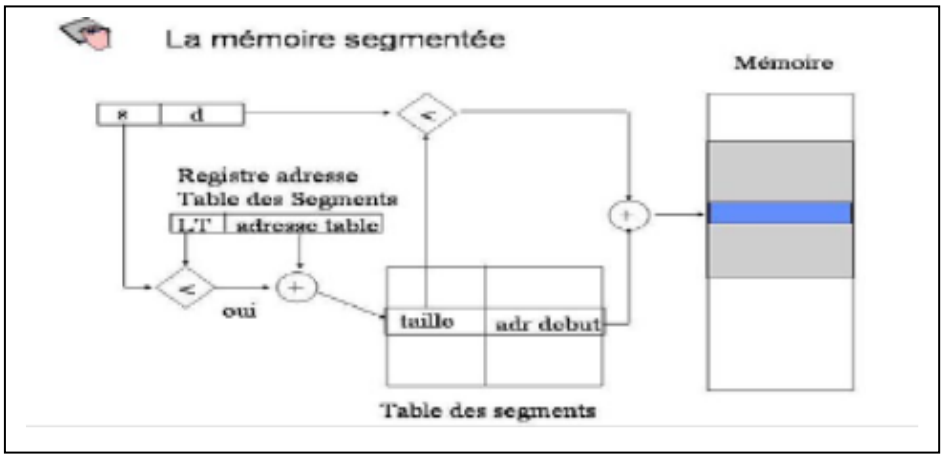


Figure 4.12: Matériel pour la segmentation

- **Le problème** de la segmentation est la fragmentation externe à nouveau .
- **La solution : la segmentation et pagination**

Le processus est découpé en segment, chaque segment est ensuite paginé

C. Segmentation et Pagination :

Dans la cas où pagination et segmentation sont simultanément employées, une table des segments est définie pour chaque segment de l'espace d'adressage du processus. Chaque segment est à son tour paginé, il existe donc une table des pages pour chaque segment. Ainsi une entrée de la table des segments ne contient plus l'adresse du segment correspondant en mémoire physique mais contient l'adresse de la table des pages en mémoire physique pour ce segment. L'adresse d'un octet dans l'espace d'adressage du processus est un couple <s,d>, le déplacement d étant à son tour interprété comme un couple numéro de page p, déplacement d' dans cette page. Les mécanismes de traduction d'adresses vus dans les paragraphes précédents se superposent l'un à l'autre.

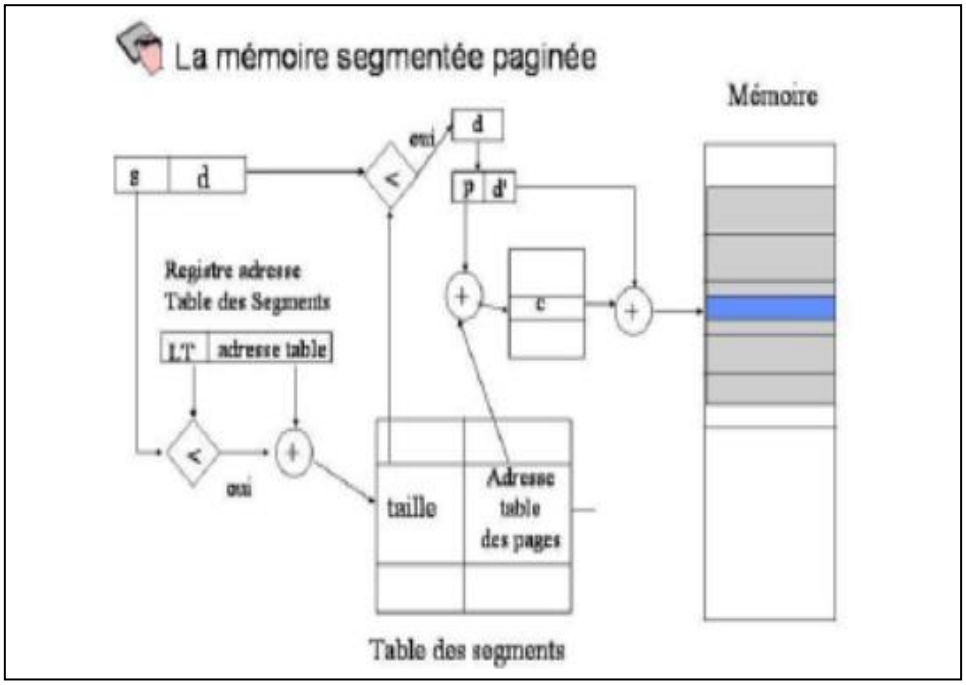


Figure 4.13 : Matériel pour la segmentation paginée

IX. La mémoire virtuelle :

La mémoire virtuelle (MV) est le support de l'ensemble des informations potentiellement La mémoire virtuelle est une technique qui permet d'exécuter des programmes dont la taille excède la taille de la mémoire centrale. Pourquoi une mémoire virtuelle ?

- La mémoire physique est coûteuse ;
- La mémoire secondaire (disque, mémoire étendue, …) peu coûteuse ;

Ce qui nous amène à utiliser la mémoire secondaire comme mémoire RAM.

Idée générale : Conserver une partie des pgms en cours d'exécution en MC.La taille d'un processus doit pouvoir dépasser la taille de la mémoire physique disponible, même si l'on enlève tous les autres processus. le principe de la mémoire virtuelle : le SE conserve en mémoire centrale les parties utilisées des processus et stocke, si nécessaire, le reste sur disque.