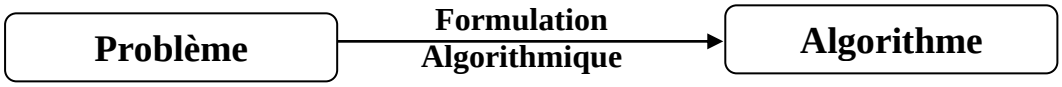
CHAPITRE 2

Mécanismes de Base d'Exécution des Programmes

II.1. Introduction

L'homme a utilisé l'ordinateur pour résoudre ses problèmes d'une manière automatique. Ces problèmes sont formulés en programmes, puis exécute par la machine matérielle.

Pour qu'un problème puisse être interprété, exécuté et donne les résultats attendus par le processeur de la machine, il faut qu'il soit formulé dans un langage compréhensible par la machine et il faut que le système d'exploitation pilote son exécution. Pour le faire, le programmeur commence au début d'écrire son problème d'une manière structurée sous forme d'un algorithme.

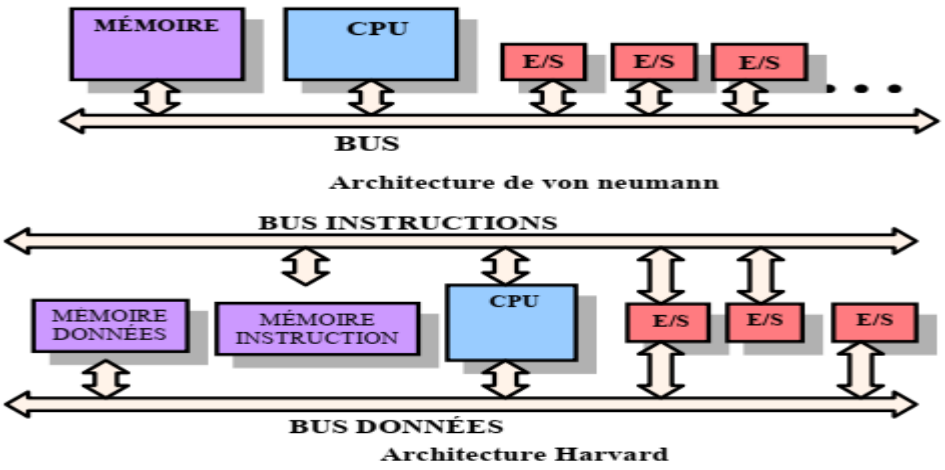


Cet algorithme est écrit en langage humain et pour le rendre exécutable par l'ordinateur, le programmeur fait le passer par plusieurs étapes tout en utilisant des outils de langage de programmation et des outils système.

Afin de comprendre les mécanismes de base utilisés pour exécuter un programme, nous rappelons donc quelques concepts nécessaires de l'architecture matérielle d'une machine, puis nous donnons les différent étapes de cheminement d'un programme dans un système, ainsi que les mécanismes utilisés par les systèmes d'interruption.

II.2. Architecture et technologie des ordinateurs

Au niveau d'architecture et technologie des ordinateurs nous distinguons deux types des architectures HARVARD & VON NEUMANN.



L'architecture la plus connus est de Von Neumann (la même architecture de notre PC), l'architecture de Harvard est utilisée dans des appareils plus spécifiques au domaines de calcule (DSP (digital signal processor): processeur à archi dédiée pour traitement de signal (opération sur les matrices) utilisent l'architecture Harvard). Dans nos études nous intéressons sur l'architecture de Von Neumann.

II.3. Machine de VON-NEUMANN

Un ordinateur est constitué d'un processeur qui effectue les traitements, d'une mémoire centrale où ce processeur range les données et les résultats de ces traitements et de périphériques permettant l'échange d'informations avec l'extérieur. Tous ces constituants sont reliés entre eux par l'intermédiaire d'un bus, qui est l'artère centrale et leur permet de s'échanger des données. Pratiquement, tous les ordinateurs actuels ont cette architecture, que ce soient les micro-ordinateurs personnels ou les gros ordinateurs des entreprises.

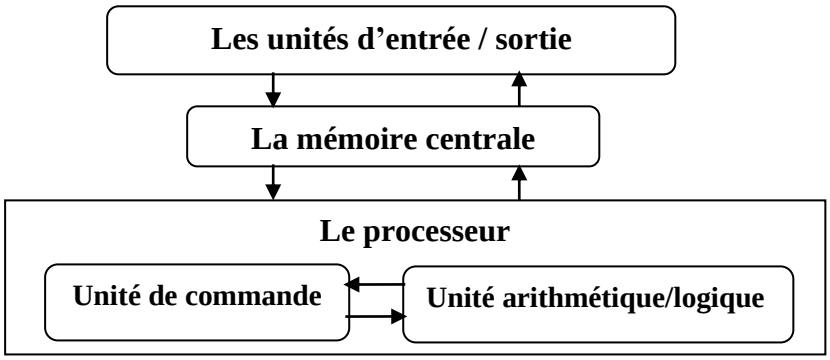
Définition 2.1

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

→ Mémoire Centrale (MC).

→ Processeur ou Unité Centrale (UC) ; pour effectuer les calculs et exécuter les instructions.

→ Unités périphériques ou d'Entrée/Sortie (E/S).



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

II.3.1. Unité Centrale de Traitement (CPU, Central Process Unit)

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

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

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

II.3.1.1.L'unité arithmétique et logique

L'UAL est composée des circuits dont le but est d'effectuer un traitement (les calculs) sur les opérandes sous le contrôle de l'unité de commande.

II.3.1.2.L'unité de contrôle et de commande (UC)

Sert à contrôler le bon fonctionnement de traitement et commande le déroulement des instructions, elle se compose de :

Compteur Ordinal (CO, Program Counter (PC), Instruction Pointer (IP)) : registre contenant l'adresse du mot mémoire où se trouve l'instruction suivante à exécuter.

Registre d'Instruction (RI, Instruction Register (IR)) : conserve l'instruction courante pendant son interprétation.

Décodeur d'instruction (DI) : analyse le code opération de l'instruction pour distribuer les différentes commandes élémentaires.

Séquenceur : il ordonne l'ensemble des microcommandes à l'ensemble des unités (mémoire, UAL, …) pour réaliser le traitement de l'instruction.

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).

L'horloge : est un circuit qui transmet régulièrement selon une périodicité déterminée des impulsions électriques, elle définit le fonctionnement séquentiel du processeur de sorte que les cycles machines sont synchronisés avec l'horloge. (L'horloge est un signal carré avec une fréquence fixe comme 3Ghz).

II.3.1.3. 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 contient d'autres registres dans la plupart des machines, citons :

Les registres généraux : sauvegardent les instructions fréquemment utilisées et les résultats intermédiaires. Dans la plupart des machines, on dispose de 7 registres.

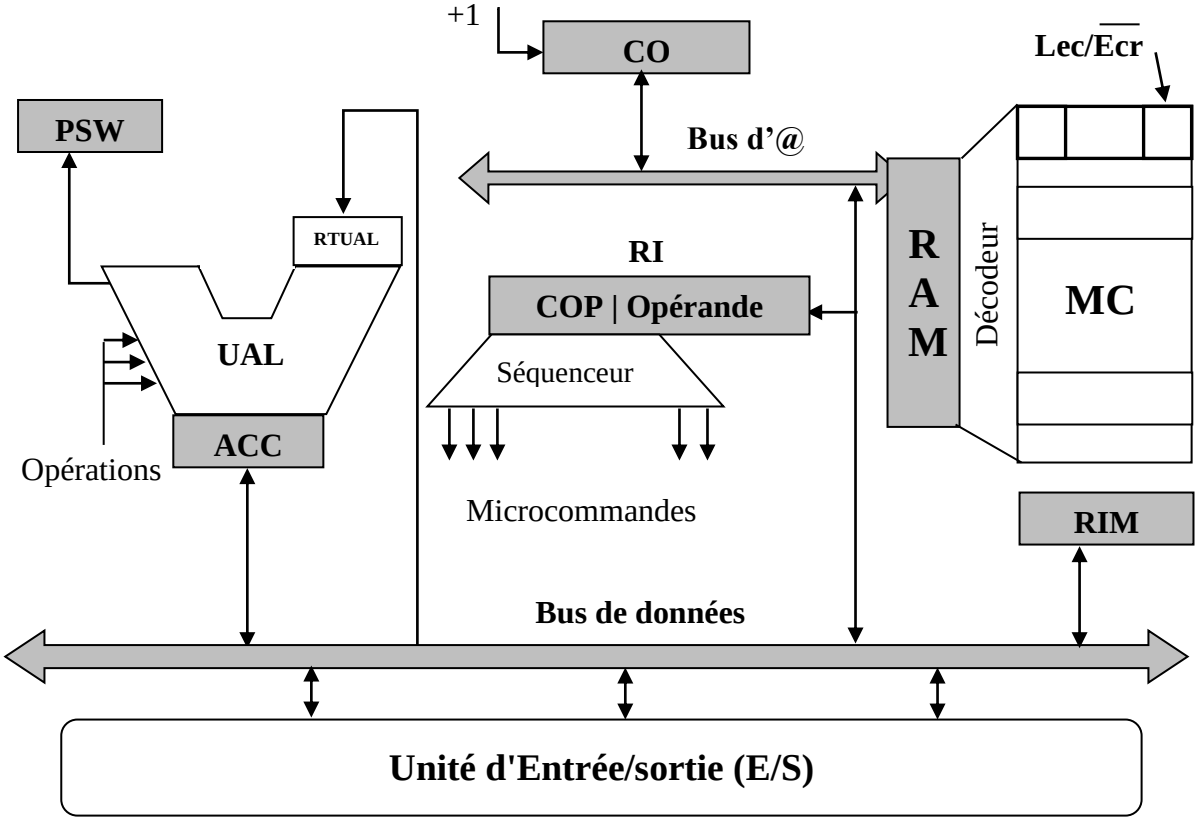
Registre d'index : utilisé pour comptabiliser le nombre d'itération dans une boucle (par exemple les tableaux).

Registre de base : utilisé pour calculer l'adresse effective dans la mémoire centrale pour un bloc d'instructions.

Registre pointeur de pile : utilisé pour l'accès à la pile.

Accumulateur : utilisé pour sauvegarder les résultats surtout dans la multiplication et l'addition.

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.



II.3.1.4. Cycle d'exécution d'une instruction

Supposons maintenant que la mémoire centrale de notre ordinateur contienne un programme et des données, et que l'on souhaite exécuter ce programme sur ces données. Lancer cette exécution revient à mettre dans le compteur ordinal (CO) l'adresse où se trouve stockée la première instruction du programme. A partir de là, le programme est exécuté étape par étape, instruction par instruction. Pratiquement, toutes les instructions élémentaires fait suivant un cycle comprenant trois phases principales : **Fetch → Decode → Execute.**

Phase 1 (Recherche de l'instruction à traiter)

1. Le CO contient l'adresse de l'instruction suivante du programme. Cette valeur est recopiée dans le registre d'adresse (RAM) en transitant par le bus «adresse»
2. Une impulsion de lecture généré par l'UC qui copier le contenu de la case mémoire sélectionnée dans le registre de donnée (RIM) par le bus «données» (RIM fonctionnait comme un registre tampon pour toutes les lectures ou écritures en mémoire)
3. L'instruction du registre de donnée (RIM) est stockée dans le registre instruction du processeur (RI) en transitant par le bus «instruction».

Phase 2 (Décodage de l'instruction et recherche de l'opérande)

1. Cette instruction courante est décodée à destination de l'UAL ; Le registre d'instruction contient maintenant le premier mot de l'instruction qui peut être codée sur plusieurs mots. Ce premier mot contient le code opératoire qui définit la nature de l'opération à effectuer (addition, rotation, ...) et le nombre de mots de l'instruction.
2. L'unité de commande transforme l'instruction en une suite de commandes élémentaires nécessaires au traitement de l'instruction.
3. Si l'instruction nécessite une donnée en provenance de la mémoire, l'unité de commande récupère sa valeur dans le registre de donnée (RIM) sur le bus «donnée».
4. L'opérande est stocké dans le registre tampon de l'UAL (RTUAL), qui stocke temporellement l'un des deux opérandes d'une instruction arithmétique.
5. Le code opération est transmis au décodeur qui détermine le type d'opération et le transmis en séquenceur

Phase 3 (Exécution de l'instruction)

1. Le séquenceur commence à envoyer des signaux de commandes vers la mémoire pour lire l'opérande à l'adresse déjà stockée dans le RIM et transmet le contenu du RIM à l'UAL [ACC, (RTUAL), …]
2. L'UAL exécute l'opération qui lui est demandée en mettant à jour son registre résultat «Accumulateur» et transfère ce résultat dans la mémoire centrale, à l'adresse référencée dans l'instruction, en utilisant le bus « données/résultats »;
3. Par ailleurs, le CO est automatiquement incrémenté (c'est-à-dire qu'il est augmenté de 1), pour signifier que l'instruction suivante à exécuter doit se trouver normalement à l'adresse qui suit immédiatement la précédente. Un nouveau cycle peut commencer alors pour la nouvelle instruction courante.

Définition 2.2

Le cycle d'exécution du processeur est divisé en deux niveau : **cycle de recherche (Fetch) + cycle d'exécution (Execute) = cycle indivisible l'arrêt ne peut se faire qu'à la fin de la phase Execute**. (Points observable qui correspondent en général au début et fin d'instruction.)

Le processeur central exécute continuellement le cycle suivant :

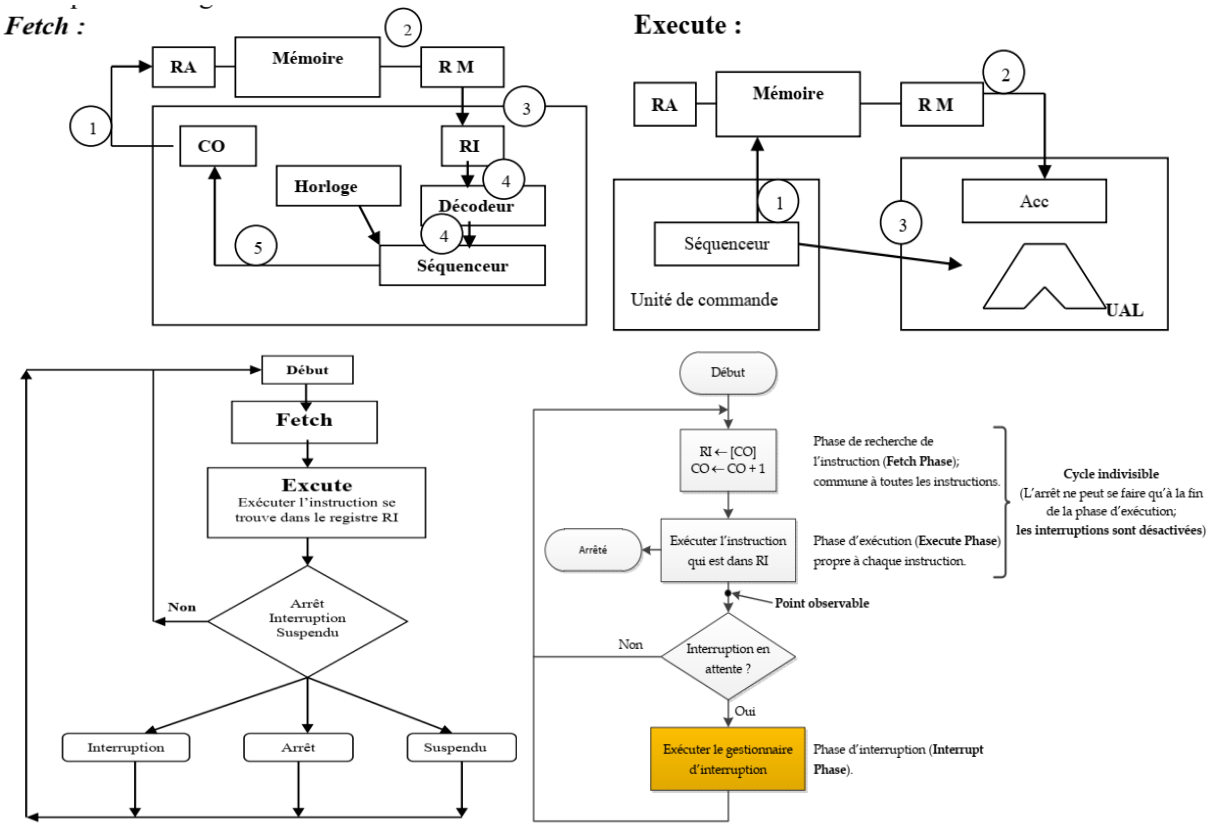


Figure 2.1 : Cycle d'exécution du processeur

II.3.1.5. Etat du processeur (Processor State Information)

En général, il est nécessaire d'empêcher le programme d'un utilisateur de perturber le fonctionnement global du système ou les programmes des autres utilisateurs. Cela implique qu'il n'ait pas accès à l'ensemble des ressources de la machine, mais seulement à celles qui lui ont été allouées en propre. La solution la plus couramment adoptée est de distinguer deux modes de fonctionnement du processeur, le mode maître et le mode esclave (asservi).

- **Mode utilisateur (ou esclave)** : réservé aux programmes usagers qui ont des droits d'actions limités. Certaines opérations sont interdites dans ce mode (manipulation directe des E/S, masquage d'IT, accès à une zone système, modification de priorité d'un process, modification directe de l'heure, manipulation de l'horloge, …).
- **Mode superviseur (maître, noyau ou privilégié)** : réservé au noyau du système d'exploitation, c.à.d. le propriétaire du programme en cours d'exécution est le SE. Il a tous les droits d'actions sur les objets du SIQ (le processeur a accès à toutes les ressources de la machine) (modifier les variables systèmes, faire les E/S, actionner le CPU, l'heure système, …)

Remarque :

- ✓ L'indicateur du mode de fonctionnement maître/esclave fait partie du mot d'état programme PSW.
- ✓ 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.
- ✓ L'état d'un processeur n'est observable qu'entre deux cycles du processeur.

II.3.2. Mémoire Centrale (MC)

Une mémoire adressable est un ensemble de supports d'informations de même capacité appelé **Emplacement**. Chaque emplacement est repéré dans la mémoire par une information appelée **adresse** qui permet de le désigner et de le distinguer des autres emplacements. L'interface entre un processeur et une mémoire adressable est schématisé par la figure ci-dessus.

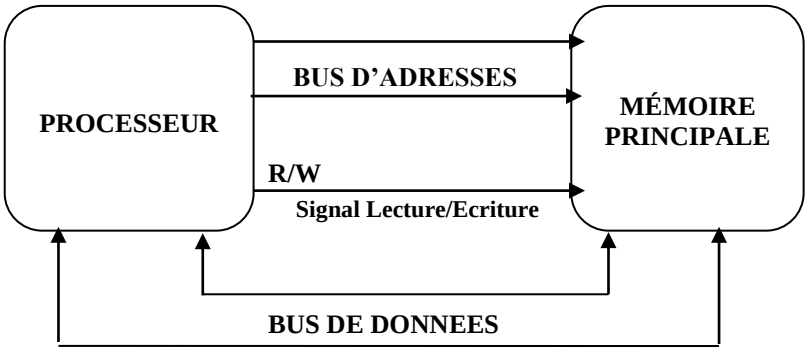
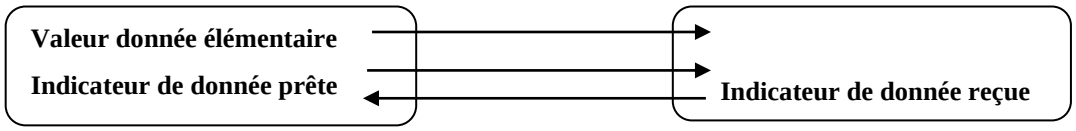


Figure 2.2 : Liaisons processeur-mémoire

Cycle mémoire : c'est le temps minimal qui doit séparer deux accès successifs à une mémoire

II.3.3. Les Unités périphériques ou d'Entrée/Sortie (E/S)

L'unité d'E/S est un dispositif capable de transférer une quantité d'information entre la mémoire d'un ordinateur (mémoire centrale) et un support d'information externe (périphérique). Le principe élémentaire mis en œuvre pour l'échange de données entre deux constituants physiques est représenté dans la figure suivante :



Différents modes d'entrée/sortie peuvent être distingués :

- ✓ Entrée/ Sortie Directe (programmée)
- ✓ E/S par Accès Direct à la Mémoire (DMA).
- ✓ Les entrées-sorties par processeur spécialisé (Canal d'E/S).

II.4. Les étapes de cheminement d'un programme dans un système

Le développement d'un programme, depuis l'analyse du problème jusqu'à sa mise au point, nécessite de nombreux outils logiciels qui constituent un environnement de programmation. Pour fonctionner, ces outils utilisent les services du système d'exploitation. La chaine de production de programme transforme un programme écrit dans un langage de haut niveau (Pascal, C, VB, Java, etc.) en un programme dit **exécutable**, écrit en langage machine. Cette transformation s'effectue à l'aide de plusieurs étapes (Voir Figure 2.3):

Ces étapes permettent de bien traduire le code source en code machine tout en permettant au programmeur de corrige les erreurs apparues dans chaque étape.

1) L'Éditeur de texte ou l'édition de programme ou (Text Editor)

C'est l'étape dont on saisit l'algorithme établit pour résoudre le problème tout en le traduisant à un programme écrit dans un langage évolué (exemple : Java, Pascal, Delphi, C++, …) ou dans le langage Assembleur. L'éditeur de texte est l'outil qui nous permet de réaliser cette étape, c'est un

logiciel interactif soit associé au système, soit associé au langage de programmation. Il nous permet d'appliquer toutes les opérations nécessaires sur un fichier nommé « **Code source** ». Ce code source à une extension spécifique tout dépend du langage utilisé (exemple : *.pas, *.java, *.cpp, …)

Exemple : Bloc-notes, Wordpad, Edit de MS-DOS, Editeur du compilateur C, JEDPlus pour java, …

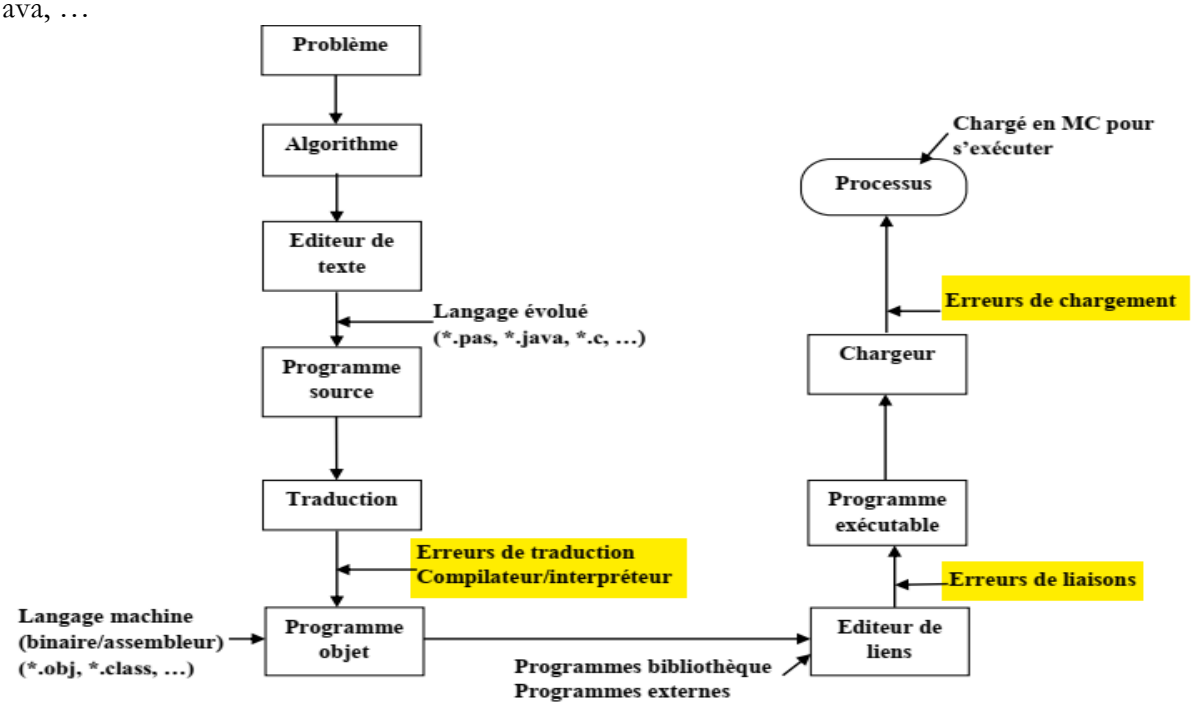


Figure 2.3 : Cheminement d'un programme dans un système

2) La traduction en langage machine

Le traducteur est un outil système qui permet de traduire les instructions écrites dans un langage de programmation vers des instructions écrites dans le langage machine (Binaire), on distingue deux types de traducteurs :

a) Le compilateur : Est un logiciel de complexité grandissante, permettant de traduire un programme source vers le langage machine (programme objet) en passant par :

- **L'analyse lexicale** : le rôle de cette phase est la reconnaissance des unités lexicales (tokens), en inspectant le code source caractère par caractère. Ces unités peuvent être des nombres, des identificateurs, des mots clés, des opérateurs, … Chaque unité est décrite par un type (mot clé, identificateur, constante, opérateur, …) et une valeur (le nom qui figure dans le code source). Alors le résultat de cette étape :
  - ✓ La création de la table de symbole.
  - ✓ Elimination de blancs et de commentaires.
  - ✓ Signalisation des erreurs lexicales.
- **L'analyse syntaxique :** vérifie que le code source à compiler respecte bien les règles syntaxiques du langage. Le résultat est la création de l'arbre syntaxique en se basant sur les unités lexicales et l'ensemble des règles syntaxiques du langage. Elle met à jour la table des

symboles en ajoutant les informations manquantes comme les types des identificateurs, et elle signale les erreurs dues au non-respect de la syntaxe du langage.

- **L'analyse sémantique** : qui vérifie la sémantique du programme. Cet analyseur utilise l'arbre syntaxique pour identifier les opérandes et les opérations et il utilise la table des symboles pour identifier les types des opérandes. Il signale les erreurs sémantiques qui peuvent être faites dans le code.
- **L'optimisation de code**.
- **Transformation en code objet** : c'est la partie qui réalise effectivement la traduction du code source en code objet.

Lors de ces étapes, le compilateur signale des erreurs à corriger qui peuvent apparaître dans chaque étape. Le résultat de cette phase est un fichier nommé **code objet** a une extension spécifique selon le langage de programmation (**Exemple** : *.obj Pascal, *.class Java, …)

b) L'interpréteur : L'interpréteur est un logiciel qui permet de traduire le programme vers le langage machine et de l'exécuter en même temps.

Ce type de traducteur ne résulte pas un fichier contenant le code objet ce qui impose l'interprétation à chaque fois d'exécution (**Exemple** : l'interpréteur HTML).

N.B : le compilateur définit deux types de codes dans le code objet :

- ✓ Le code absolu qui a une adresse fixe dans la mémoire.
- ✓ Le code translatable qui peut se mettre dans n'importe quel emplacement dans la mémoire.

L'opération de translation consiste à ajouter à chaque emplacement qui contient une adresse, la valeur de l'adresse effective de l'implantation finale dans la mémoire.

3) L'édition de liens

Lors de la traduction d'un programme en code objet, le compilateur associe à chaque instruction son type, dont on peut avoir :

- ✓ Des données variables
- ✓ Des données constantes
- ✓ Des instructions
- ✓ Des appels à des procédures, des modules ou des variables externes.

Dans le dernier type, le programmeur peut référencer des structures ou des modules externes par rapport au module en question (il s'agit des références externes de la bibliothèque du langage ou personnelles). Alors le compilateur marque ces références sans les ramener et les ajouter au code objet. Donc il associe à chaque référence soit :

- Un lien interne au programme, mais qui peut être accédé par d'autre modules → **lien utilisable (définition externe).**
- Un lien externe appartient à un autre module appelé dans ce programme → **lien à satisfaire (référence externe).**

Nous appellerons **lien à satisfaire** la partie du lien située chez l'accédant, et **lien utilisable** la partie située chez l'accédé.

L'éditeur de lien est un outil système qui a comme objectif d'accomplir la tâche du compilateur en ajoutant les codes objets de tous les liens à satisfaire dans le code objet principal. Alors l'éditeur de liens cherche l'origine de chaque référence externe et établit une table globale des modules contenant les informations : nom du module, taille et adresse d'implantation.

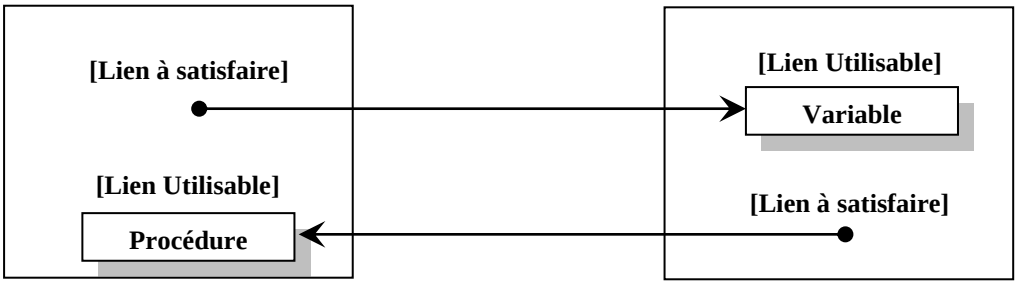
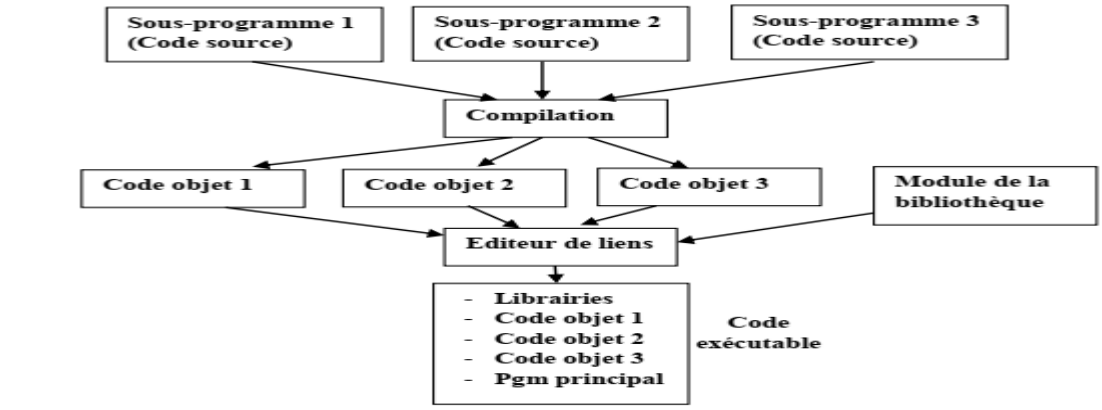


Figure 2.4 : La notion de lien.



Exemple :

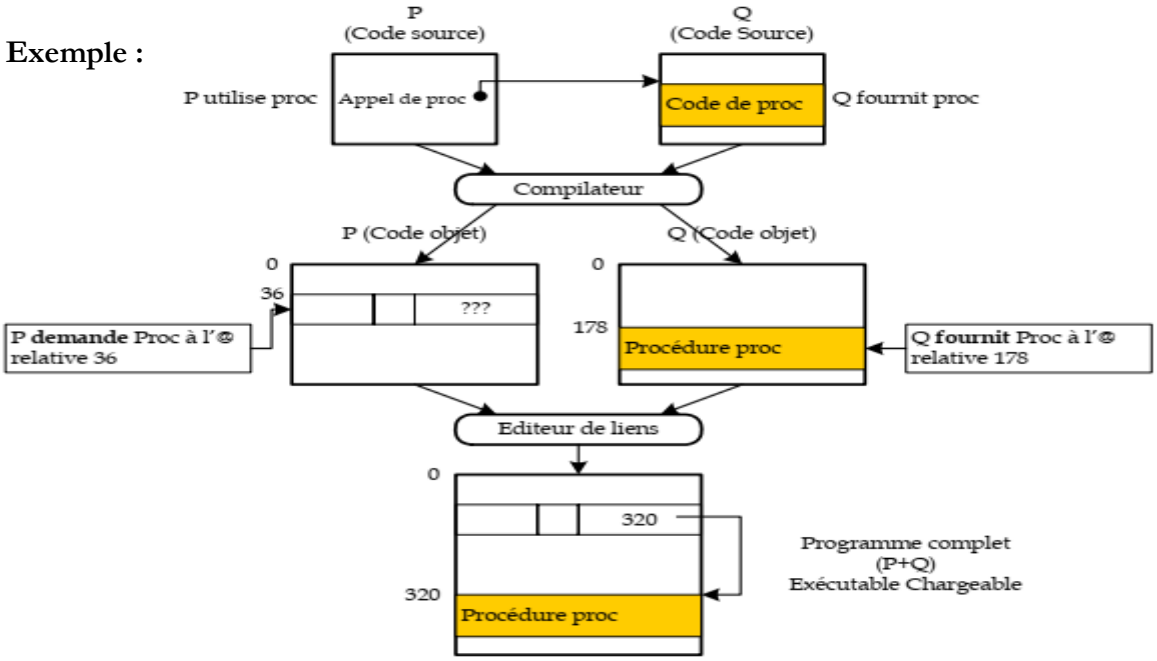


Figure 2.5 : Principe d'édition de liens

Le travail de l'éditeur de liens est basé sur la table de fonctions qui contient toutes les informations concernant la bibliothèque. On peut distinguer entre deux types des éditeurs de liens :

- **Editeur de lien statique** : c'est l'édition de liens qui s'établit une fois pour toute et elle engendre un fichier résultat liant toutes les références externes avec le programme principal, nommé le fichier exécutable. Ce type des éditeurs de liens ne se refait pas à chaque exécution.

  Exemple : l'éditeur de liens du langage Pascal → des fichiers *.exe.
- **Editeur de liens dynamiques** : c'est l'édition de liens qui s'établit à chaque demande d'exécution du programme, elle n'engendre pas un fichier exécutable.

  Exemple : l'éditeur de liens du langage Java

Exemple : considérons les deux programmes en langage C suivants : prog1.c et prog2.c

Prog1.c

```
extern int v1 ;
extern int proc1(int x) ;
float v2 ;
main()
{
int a, x ;
x = v1 + proc1(a) ;
}
```

Prog2.c

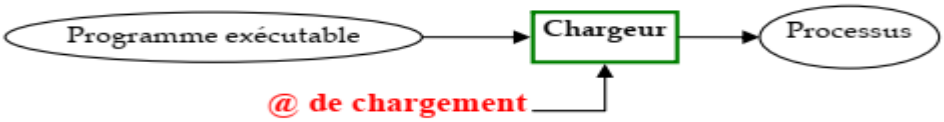
```
int v1 ;
extern float v2 ;
int proc1(int x) ;
main()
{
float b ;
b = v2 *2;
}
int proc1(int x)
{
int y ;
return (x * y /2) ;
}
```

On remarque dans cet exemple que pour le programme **Prog1.c** les liens à satisfaire sont la variable **v1** et la procédure **proc1**, alors que le lien utilisable est la variable **v2**. par contre pour **Prog2.c**, le lien à satisfaire est la variable **v2** alors que les liens utilisables sont la variable **v1** et la procédure **proc1**.

Module	Liens à satisfaire	Liens utilisables
Prog1.c	Variable v1 Procédure Proc1	Variable v2
Prog2.c	Variable v2	Variable v1 Procédure proc1

4) Le chargement

Après l'édition de liens, le programme est prêt à s'exécuter, pour faire, il faut le charger dans la mémoire centrale. Cette phase est faite par un outil système nommé le **chargeur**. Cette étape consiste à lire les instructions en mémoire secondaire, et les transférer en mémoire centrale tout en lisant au début l'adresse de chargement fournit par l'éditeur de liens. Ainsi, le chargeur est un programme qui installe ou charge un exécutable en MC à partir d'une adresse déterminée par le S.E. Dans ce cas le programme devient un **processus**.



On distingue deux types de chargement :

- **Le chargement absolu** : le code chargé ne doit être pas met dans un autre bloc de données différent de celui indiqué dans le code objet, (d'une autre façon recopier le code).
- **Le chargement relogeable (translatable)** : le chargement peut se réaliser à n'importe quelle adresse dans la mémoire centrale, la façon de translater les adresses est basée sur les informations fournies par l'éditeur de liens.

5) Le débogueur

C'est un logiciel qui permet d'exécuter le programme pas à pas, d'afficher les valeurs des variables à tout moment et mettre en place des points d'arrêts sur des conditions ou des lignes du programme. Il offre au programmeur la possibilité de contrôler l'exécution et de détecter l'origine des erreurs non corrigées.

II.5. Le processus

Définition 2.3

Un processus est un programme en cours d'exécution qui est exécuté par un processeur. Aussi, plusieurs processus peuvent-ils être associés à un programme. Chaque processus possède un espace de travail en mémoire, son compteur ordinal et ses registres.

Définition 2.4

On peut définir un processus (process) comme un programme en exécution. Autrement dit, un programme par lui-même n'est pas un processus. Un programme est une entité passive, comme le contenu d'un fichier stocké sur disque, tandis qu'un processus est une entité active, avec un compteur d'instructions spécifiant l'instruction suivante à exécuter et un ensemble de ressources associées.

Définition[1] 2.5

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, ses ressources et les autres contenus de registres nécessaires à son exécution.

II.5.1. *Ce qui se passe après le lancement d'un programme*

Quand l'utilisateur lance l'exécution d'un programme, et quand la machine utilise le principe de la mémoire virtuelle et le multitâche, alors ce programme passe par les étapes suivantes :

- Le système crée un **Job** au niveau de la mémoire virtuelle tout en chargeant le programme.
- Après, selon un critère ou autre ce job est admis dans le système tout en le chargeant vers la mémoire centrale dans sa totalité ou une partie, ce chargement crée ce qu'on appelle un **processus**.
- Le processus est dirigé, après, vers l'exécution dans le processeur selon des conditions spécifiques, où ce processus peut changer son état, durant cette exécution, plusieurs fois jusqu'à sa terminaison.

[1] Cours, Imene Sghaier, introduction aux Système d'exploitation. (Technologue pro - Ressources pédagogiques pour l'enseignement technologique).

II.5.2. Caractéristiques du processus

1. Un processus possède un identifiant unique qui est généralement un entier incrémental (le premier processus 1, le second 2, ... etc.) et qui désigne de façon unique le processus dans le système (PID : Process Identifier)
2. Les instructions à exécuter sont stockées dans une pile de données contenant les valeurs des variables du programme
3. Un espace d'adressage (code, données, piles d'exécution);
4. Un état principal (prêt, en cours d'exécution (élu), bloqué, …);
5. Les valeurs des registres lors de la dernière suspension (CO, sommet de Pile...);
6. Une priorité;
7. Les ressources allouées (fichiers ouverts, mémoires, périphériques …);
8. Les signaux à capter, à masquer, à ignorer, en attente et les actions associées;
9. Autres informations indiquant le processus père, les processus fils, le groupe, les variables d'environnement, les statistiques et les limites d'utilisation des ressources...

II.5.3. Contexte d'exécution d'un processus

Le noyau d'un système d'exploitation maintient une table pour gérer l'ensemble des processus, chaque processus est identifié par son PID, entier de 0 à 215 retourné par le noyau. Le contexte d'un processus est l'ensemble des données qui permettent de reprendre l'exécution d'un processus qui a été interrompu. Le contexte d'un processus est l'ensemble de :

- son état
- son mot d'état: en particulier
  - ✓ La valeur des registres actifs
  - ✓ Le compteur ordinal
- les valeurs des variables globales statiques ou dynamiques
- son entrée dans la table des processus
- La pile
- les zones de code et de données.

Quand il y a changement de processus courant, il y a réalisation d'une **commutation de mot d'état** et d'un **changement de contexte.** Le noyau s'exécute alors dans le nouveau contexte.

II.5.4. Image mémoire d'un processus

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

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

- Adresse de retour de fonctions.

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

II.5.5. Descripteur de Processus (PCB)

Chaque processus est représenté dans le SE par un **bloc de contrôle de processus** (**Process Control Bloc, PCB**). Le contenu du PCB (Figure 2.6) varie d'un système à un autre suivant sa complexité. Il peut contenir :

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

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

Alors, les files d'attente utilisées dans chaque état contiennent les PCBs des processus, tel que :

- Le PCB d'un nouveau processus est mis dans la file d'attente des processus prêts.
- Attente jusqu'à sélection pour entrer dans le processeur → le processus devient actif, et on utilise un pointeur vers son PCB dans la mémoire.
- Si le processus attend une ressource ou un évènement, son PCB entre dans la file d'attente des processus bloqués.
- S'il est suspendu, son PCB se mis dans la file d'attente des processus suspendu.

II.5.6. Etats de processus

Les processus se partagent le processeur et changent d'états selon qu'ils possèdent ou non le processeur. Durant son exécution le processus peut être (Figure 2.7):

- **Nouveau** : le processus en cours de création.
- **Actif (Elu)** : c'est-à-dire, le processus dispose de toutes les ressources nécessaires et il est en train de s'exécuter au niveau du processeur.
- **Prêt (Eligible)** : c'est-à-dire, le processus dispose de toutes les ressources nécessaires sauf le processeur → processus en attente du processeur.
- **Bloqué** : le processus est en attente d'une autre ressource que le processeur ou d'un évènement.
- **Terminé** : le processus a fini son exécution.

Pour chacun des états prêt, bloqué et suspendu, on associe une file d'attente pour les processus dans cet état et elle est gérée par un outil du SE.

Les différentes transitions d'un état à un autre sont effectuées comme illustré dans le diagramme ci-après.

- La transition **Actif ➔ Prêt** a lieu quand on retire le processeur au processus car son quantum a expiré ou suite à un appel système ou une interruption.
- La transition **Actif ➔ Bloqué** a lieu lorsque le processus attend une ressource autre que le processeur (un événement extérieur).
- La transition **Bloqué ➔ Prêt** a lieu lorsque le processus a toutes les ressources qu'il faut (autres que la CPU).
- La transition **Prêt ➔ Actif** a lieu lorsque le processeur lui est alloué (à son tour).

Identificateur processus
Etat du processus
Compteur ordinal
Contexte pour reprise (Registres, Pointeurs piles, …)
Pointeurs sur file d'attente et priorité (Ordonnancement)
Information mémoire (limites et tables pages/segments)
Informations des comptabilisation ets sur les E/S, périphériques alloues, fichiers ouverts, ...
Pointeurs des PCBs suivant et précèdent

Figure 2.6 : Bloc de Contrôle de Processus (BCP).

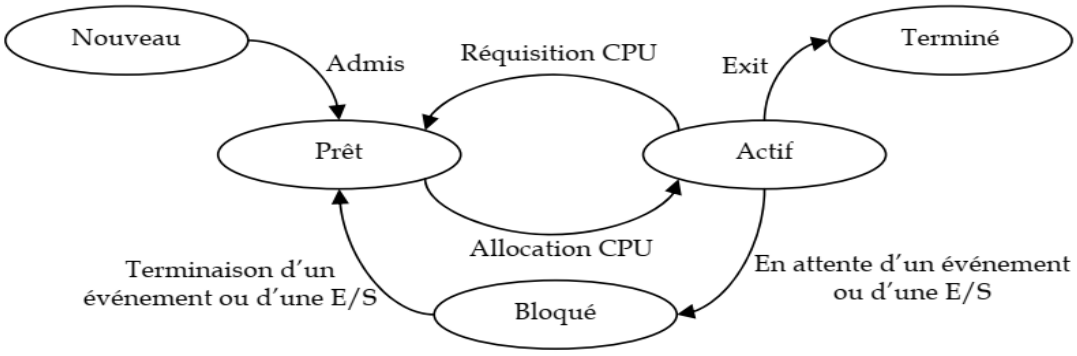


Figure 2.7 : Diagramme des transitions d'états d'un processus

Donc, un processus en cours d'exécution peut être schématisé comme suit (Figure 2.8) :

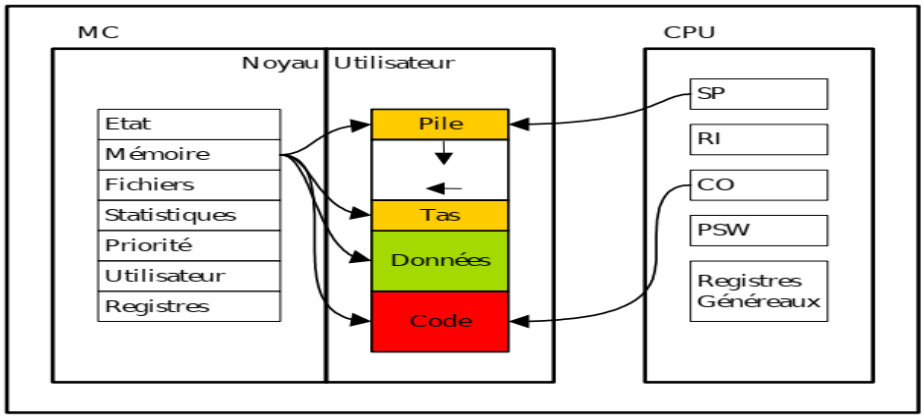


Figure 2.8 : Schéma général d'un processus en cours exécution.

II.5.7. Les primitives de manipulation de processus :

La manipulation d'un processus se fait par des primitives nécessaires qui utilisent son PCB et se sont des procédures systèmes. Ces primitives d'exécutent d'une manière indivisible :

- La création d'un processus.
- Activation d'un processus prêt.
- Suspension d'un processus.
- Destruction d'un processus

II.5.8. Mécanisme de commutation de contexte

Dans un système multiprogrammé, le processeur assure l'exécution de plusieurs processus en parallèle (pseudo-parallélisme). Le passage dans l'exécution d'un processus à un autre nécessite une opération de sauvegarde du contexte du processus arrêté, et le chargement de celui du nouveau processus. Ceci s'appelle la **commutation du contexte** (**Context Switch**).

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.

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

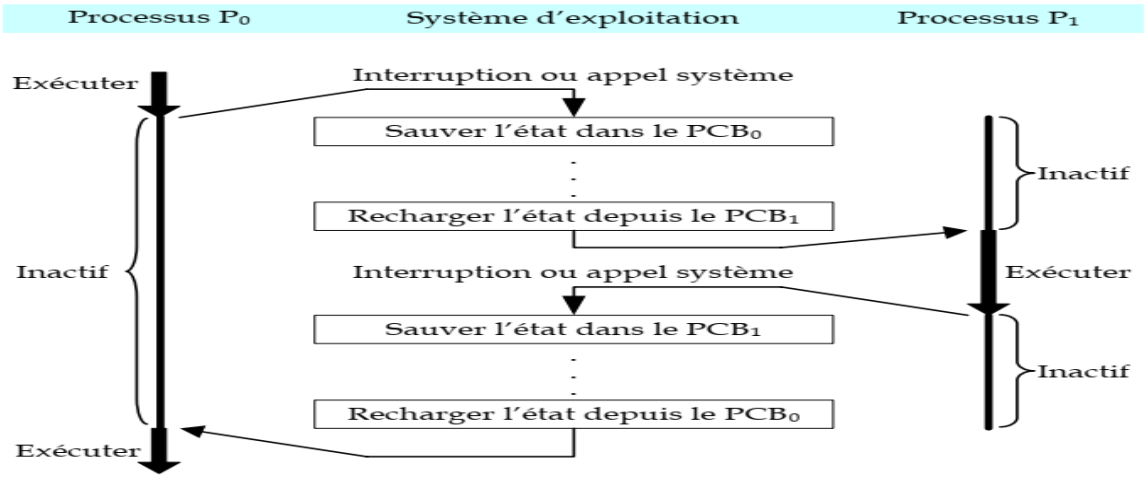


Figure 2.9 : Le Mécanisme de commutation de contexte

La commutation du contexte se fait en deux phases (Voir Figure 2.10) :

- 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.

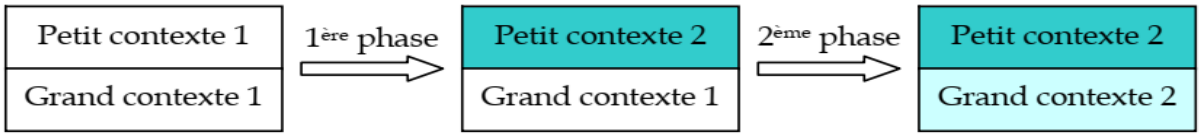


Figure 2.10 : Les phases d'une commutation de contexte.

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

II.6. Les Systèmes d'Interruption

II.6.1. Problématique

Dans un ordinateur, deux types de programmes coexistent :

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

Ces deux types de programmes se partagent les ressources communes de la machine et plus **particulièrement le processeur**. Lorsqu'un programme est en cours d'exécution, plusieurs événements peuvent arriver :

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

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

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

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

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

Définition 2.6

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.

Définition 2.7

Une **interruption** constitue un mécanisme pour lequel les modules (E/S, mémoire, processus) peuvent interrompre le traitement normal du processeur.

L'interruption est une réponse à un évènement qui interrompe l'exécution des programmes en cours à un point observable (interruptible) du processeur central, elle se traduit par un signal envoyé au processeur, elle permet de forcer le processeur à suspendre l'exécution du programme en cours et à déclencher l'exécution d'un programme prédéfini appelé « **routine d'interruption** ».

Définition 2.8

Point de vue matérielle : **Une interruption** est un signal physique (électronique) émis par un périphérique au CPU pour lui signaler l'occurrence d'un évènement.

Point de vue logicielle : **Une interruption** est un évènement extérieur provoquant la suspension temporaire (la préemption) du process actif (lui retirer le CPU) au profit d'un programme spécial appelé programme (**routine** ou **handler**) d'interruption chargé de l'évènement arrivé. Ou un appelle à un service de système d'exploitation (**appel système**)

II.6.2. Causes d'interruption

On distingue plusieurs types d'événements pouvant provoquer des interruptions, ils peuvent être regroupés en deux principales causes :

1) Les interruptions externes ou matérielles

Elles sont provoquées par les périphériques connectés à la machine suite à une fin d'E/S ou à une anomalie ou pour signaler d'autres événements.

A titre d'exemples : Branchement ou retrait d'un Flash disque au port USB, le signal émis par le clavier suite à une frappe d'une touche, signal émis par l'horloge suite à un tick ou encore le signal généré par un capteur de température dans une installation industrielle.

2) Les interruptions internes ou logicielles

La cause principale de ce type d'IT est le process actif (le programme en cours d'exécution). On en distingue deux principales causes :

a) les déroutements : c'est une exception levée suite à une anomalie (une erreur) générée par l'exécution de l'instruction courante. Il peut être provoqué par :

- ✓ une erreur arithmétique : division par zéro, débordement, …
- ✓ référence d'une adresse mémoire hors inexistante (limites de la mémoire disponible).
- ✓ une violation de la protection mémoire.
- ✓ une instruction illégale
- ✓ tentative d'exécution d'une instruction privilégiée en mode esclave.
- ✓ les défauts de page, débordement de pile, …

A la suite d'un déroutement, l'utilisateur sera averti de l'erreur commise et le process actif est arrêté immédiatement.

b) les Appels au superviseur (SVC) : Un appel au superviseur (SVC : SuperVisor Call) est une instruction qui a pour fonction de provoquer le changement d'état du processeur. Un appel au superviseur provoque l'exécution d'un programme spécifié avec un changement du mot d'état du processeur et sauvegarde de l'état au moment de l'appel. Cette instruction est principalement utilisée pour réaliser l'appel, à partir d'un programme utilisateur exécuté en mode esclave (asservi), par exemple des fonctions d'accès aux fichiers (ouvrir, lire, fermer…etc.). ces appels offrent des services de système d'exploitation.

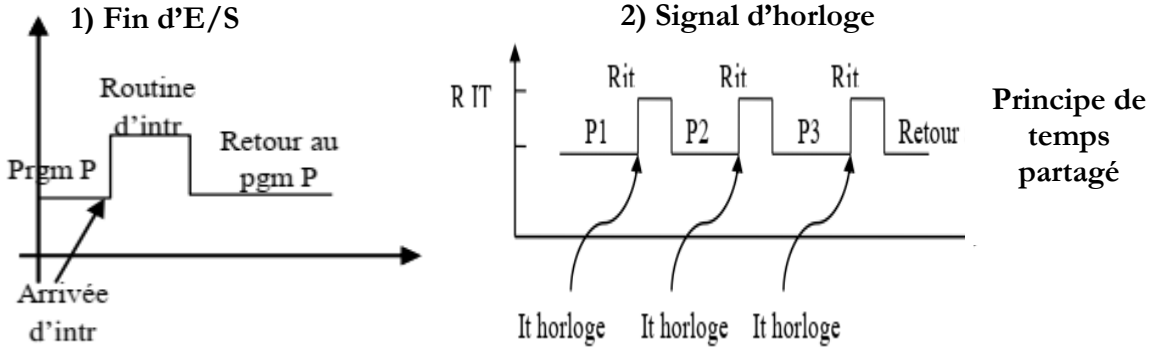
Un appel au superviseur (SVC) est une instruction au moyen de laquelle un programme qui s'exécute en mode esclave peut solliciter du système une fonction ou un service auquel il n'a pas le droit d'accéder directement.

Remarque 2.1

Pour certains auteurs, un appel SVC est considéré comme étant un déroutement. La différence principale est qu'un **déroutement** exprime **une erreur non volontaire** alors qu'un **SVC** est une **instruction voulue en elle-même par l'utilisateur**.
La distinction entre interruption, déroutement, appel au superviseur est **fondée sur la fonction** et **non sur le mécanisme** qui est commun.

Exemple:

Le cas 1 représente un événement de fin d'E/S, et le cas 2 est un événement de Signal d'horloge, qui se déclenche dans le cas d'un système à temps partagé. Trois processus P1, P2 et P3 s'exécutent en temps partagé. A chaque interruption d'horloge, la routine d'interruption passe à l'exécution d'un autre processus, en chargeant son contexte.



II.6.3. Mécanismes de gestion des interruptions

a) Le traitement d'une interruption

Quand un processus utilisateur **P** est en cours d'exécution, dans un temps **T**, et le processeur reçoit une interruption quelconque plus prioritaire, alors :

1) Suspendre le process **P** en cours et sauvegarder son 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)

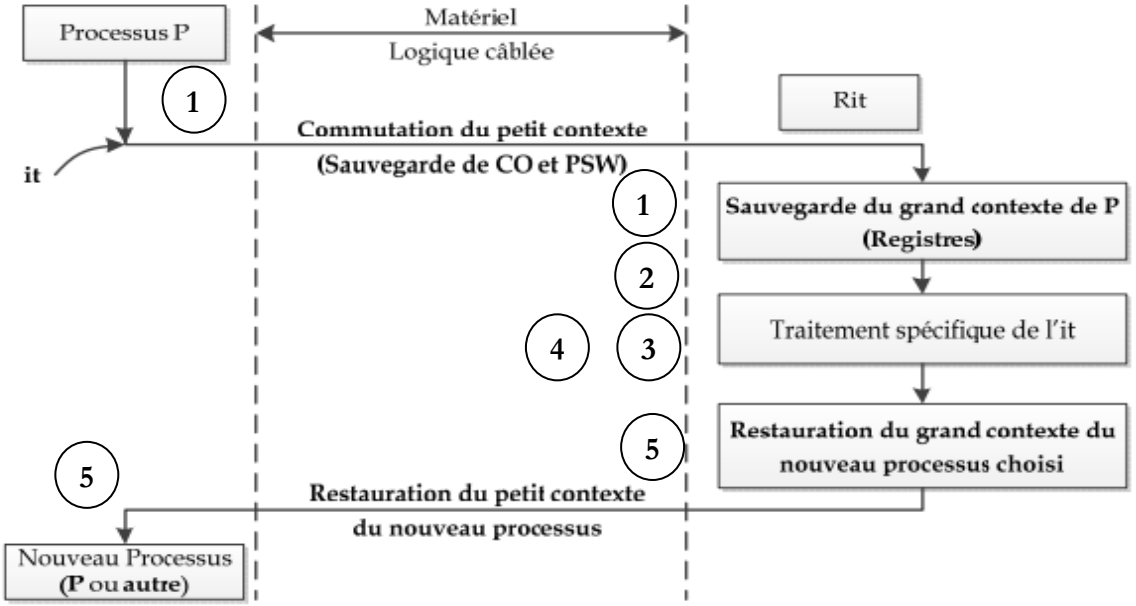
en sauvegardant son contexte pour pouvoir reprendre son exécution ultérieurement.

2) Identifier la cause d'interruption : pour détecter la source émettrice du signal d'IT.

3) Chargement du contexte du programme d'interruption (contexte actif) et passage en mode système (ou superviseur)

4) Exécuter le programme de traitement de cette **IT**

5) Restaurer le contexte du programme **P** suspendu et en repassant en mode utilisateur ou le choix d'un autre, et continuer son exécution.



b) Conditions d'arrivée d'une interruption

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

1) Le système d'interruption est **actif**.

2) L'UC est à un point observable (**interruptible**).

3) L'interruption est **armée**.

4) L'interruption est **démasquée**.

5) L'interruption est plus prioritaire que le programme en cours.

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

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

- **L'interruption est armée :** Une interruption désarmée ne peut interrompre l'UC. Ceci se passe comme si la cause de l'interruption était supprimée. Toute demande d'interruption faite durant son désarmement est perdue. On utilise ce procédé quand on désire qu'un élément ne doive plus interrompre.

- **L'interruption est démasquée** : Parfois, il est utile de protéger, contre certaines interruptions, l'exécution de certaines instructions (par exemple, les programmes

d'interruption eux-mêmes). Une interruption masquée ne peut alors interrompre l'UC, mais toute demande d'interruption faite durant le masquage est retardée (mémorisée) pour être traitée à la levée du masquage.

c) 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 :

1) 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.
2) S'il existe des indicateurs multiples, chacun est appelé niveau d'interruption. On attache un programme différent de traitement à chacun de ces niveaux.
3) 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.

Erreurs matérielles
Horloge
Requêtes disque
Requêtes réseau
Terminaux
Interruptions logicielles

Figure 2.10 : 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.

d) Hiérarchie des interruptions

Les interruptions peuvent être **hiérarchisées**, c'est-à-dire classées par ordre de priorités respectives. Un traitant d'interruption peut donc être lui-même interrompu par une demande d'interruption de niveau de priorité supérieure. Il passe alors à l'état d'attente. La figure suivante représente l'activité des programmes dans le temps pour un système hiérarchisé à 8 niveaux où le niveau 0 est le plus prioritaire, le niveau 7 correspondant au programme d'arrière-plan (Figure 2.11).

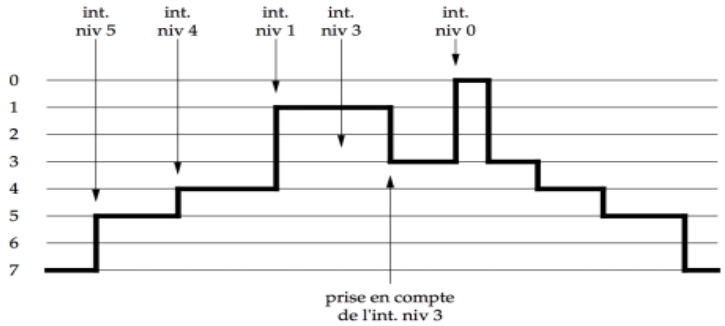


Figure 2.11 : Hiérarchie des interruptions.

Aussi, certaines opérations doivent être **atomiques** (elles ne doivent pas être interrompues), comme par exemple la sauvegarde ou la restauration du contexte doit toujours se faire entièrement (sinon l'on risque de perdre certaines informations).

e) Les opérations sur les interruptions

- 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.
- Désarmer une interruption :elle sera ignorée. Par défaut, les interruptions sont évidemment armées.

Exemple

Il est possible à un processus superviseur de masquer une interruption. Masquer (inhiber) une interruption revient à retarder son effet temporairement. Elle peut être démasquée après. À un instant donné, les interruptions sont soit masquées soit autorisées, suivant l'état d'un indicateur spécial du registre d'état, IF (Interrupt Flag).

- Si IF = 1 (interruptions autorisées), alors le processeur accepte les demandes d'interruptions masquables, c'est-à-dire qu'il les traite immédiatement ;
- si IF = 0 (interruptions masquées), alors le processeur ignore ces interruptions. L'état de l'indicateur IF peut être modifié à l'aide de deux instructions, CLI (CLear IF pour la mise à 0 de IF), et STI (SeT IF, pour la mise à 1 de IF).

f) La table des vecteurs d'interruption

Le vecteur d'interruption est une table qui contient les adresses des routines d'interruptions. Les interruptions sont numérotées et ces numéros (allant de 0 à 255 (FFh) dans un PC) servent d'index pour rechercher dans le vecteur d'interruption l'adresse de la routine à exécuter.

L'adresse de début d'une routine d'interruption est appelée **Vecteur d'interruption**. Toutes les adresses (logique CS:IP) de début de ces vecteurs sont rangées dans une table et constituent donc la Table des Vecteurs d'Interruption. Cette table se charge en **RAM** à partir de l'adresse la plus basse (0000h). Chaque vecteur d'interruption est stocké sur **4 octets** (2 pour CS et 2 pour IP)

Le tableau suivant montre la gestion des déroutements du processeur Pentium d'Intel.

Tableau 2.1 : Table des vecteurs d'interruption du processeur Pentium d'Intel

Numéro de vecteur	Usage
0	Division par zéro
1	Exception de débogage
2	Non-Maskable Interrupts (NMI)
3	Point d'arrêt
4	Dépassement de capacité de registre interne
5	Dépassement de limite
6	Code d'opération non valide
7	Périphérique indisponible
8	Erreur sur nombre double précision
9	réservé : gestion coprocesseur
10	segment d'état de tâche invalide
11	segment absent
12	Erreur dans la pile
13	Erreur de protection
14	Défaut de page
15	réservé
16	Erreur dans nombre en virgule flottante
17	Contrôle alignement
18	Contrôle hardware
19 - 31	réservé
32 - 255	Interruptions masquables