Chapitre2 : Mécanismes de base d'exécution de programmes

2.1- Introduction :

Dans le S.E, on peut distinguer deux types de fichiers :

- Les fichiers binaires : Ils sont écrits en langage machine et interprétés directement par le CPU et le S.E. Exemple : Fichiers .exe,
- Les fichiers textes : ils sont compréhensibles par l'être humain car ils sont écrits en code ASC2. Généralement, les fichiers textes sont écrits par des éditeurs de textes (bloc note, word....etc). Exemple : fichiers .bat, .doc, .txt....etc.

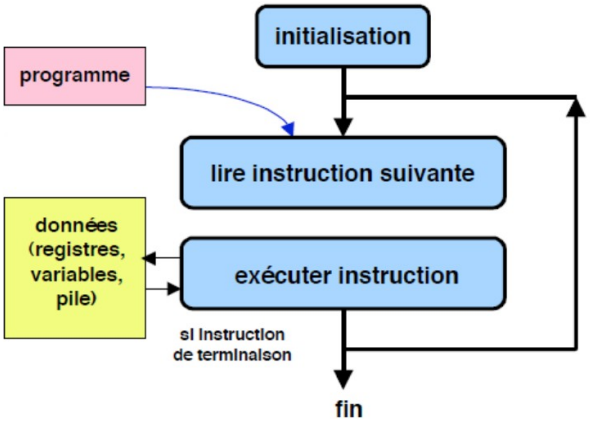
2.2 Cheminement d'un programme dans un système:

Un programme informatique passe par plusieurs étapes avant son Exécution:

2.2.1- Écriture du programme Dans cette étape, il s'agit de traduire l'algorithme du problème à résoudre en un programme écrit en langage évolué. En effet, aucun ordinateur n'est apte à exécuter les instructions telles qu'elles sont rédigées dans tel ou tel langage ; l'ordinateur, lui, ne comprend qu'un seul langage, qui est un langage codé en binaire (à la rigueur en hexadécimal) et qui s'appelle le langage machine (ou assembleur). Généralement, l'écriture du programme est réalisé en utilisant un éditeur de texte qui génère un fichier texte avec une extension selon le langage utilisé (.pas, .c, java....etc).

2.2.2- Compilation ou Interprétation : dans cette étape, il s'agit de traduire le programme en langage machine afin qu'il soit exécutable. Il existe deux schémas de traduction, ces deux schémas étant parfois disponibles au sein du même langage.

- **Schéma d'interprétation :** Dans ce schéma, un programme appelé « interpréteur » traduit et exécute les instructions au fur et à mesure qu'elles se présentent. Un interpréteur est un programme qui étant donnée un programme P et une entrée x, calcule la sortie s de P(x).



- **Schéma de compilation :** Dans ce schéma, le programme sera traduit une fois pour toutes par un programme annexe, appelé « compilateur », afin de générer un nouveau fichier qui sera autonome, c'est-à-dire qui n'aura plus besoin d'un programme autre que lui pour s'exécuter; on dit d'ailleurs que ce fichier est exécutable.

Ce schéma nécessite les étapes suivantes :

1. **Compilation :** Le compilateur vérifie lexicalement et syntaxiquement le programme. Deux cas de figure peuvent être distingués :
   - **Compilation vers programme objet absolu :** Dans ce cas, le programme est traduit en un programme objet avec des adresses fixes en mémoire. Ainsi le programme ne peut pas être déplacé en mémoire.

   - **Compilation vers programme objet translatable :** Dans ce cas, le programme est traduit en un programme objet dit « translatable » dont les adresses mémoire sont définies à une translation prés. Le programme objet translatable n'est donc pas exécutable.

Exemple

- gcc -c prog.c : produit un programme objet translatable dans le fichier Prog.o.
- gcc -o prog prog.c : produit un programme objet absolu dans le fichier prog.

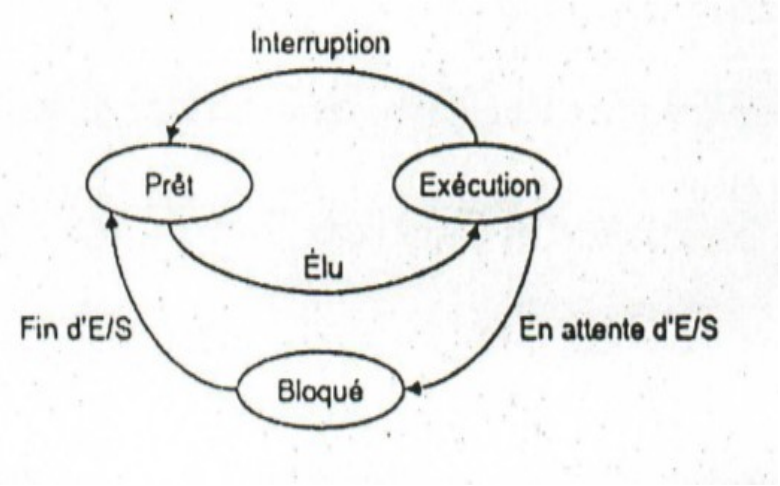
2. **Édition de liens :** l'objectif principal de l'édition de lien est de rassembler les fichiers objets générés dans l'étape de compilation au sein d'un seul fichier. En effet, si un programme P fait appel à une procédure p incluse dans un programme Q construit séparément. Quelle sera l'adresse de la procédure p dans le programme P ? C'est l'éditeur de lien qui s'en charge de faire la liaison entre l'appel de la procédure p et son adresse dans le programme Q.

3. **Chargement:** Dans cette étape, un programme appelé « chargeur » est responsable du chargement du programme dans la mémoire. Son rôle est d'effectuer la translation des adresses.

2.3- Concepts de processus et multiprogrammation:

2.3.1- Concept de processus : Un processus est une entité dynamique qui matérialise un programme en cours d'exécution avec ses données, sa pile, son compteur ordinal, son pointeur de pile et les autres valeurs de registres nécessaires à son exécution. Contrairement à un programme (texte exécutable) qui a une existence statique, un processus constitue l'aspect dynamique du programme.

2.3.2- États d'un processus : Comme toute entité dynamique, un processus peut changer d'état au cours 'de son exécution. La transition d'un état à un autre est effectuée selon le diagramme de transitions ci-dessous :



- **Élu** : en cours d'exécution.
- **Prêt :** en attente du processeur.
- **Bloqué:** en attente d'un événement.

Initialement, un processus est à l'état prêt. Il passe à l'état exécution, lorsque le processeur entame son exécution. Un processus passe de l'état exécution à l'état prêt, lorsqu'il est suspendu provisoirement pour permettre l'exécution d'un autre processus. Il passe de l'état exécution à l'état bloqué, si le processus ne peut plus poursuivre son exécution (demande d'une E/S). Il se met alors en attente d'un événement (fin de l'E/S). Lorsque l'événement survient, il redevient prêt.

2.3.3- Contexte de processus : Le contexte d'un processus regroupe l'ensemble des informations que les actions du processus peuvent consulter ou modifier ainsi que les informations nécessaire au système d'exploitation pour la gestion des processus. Pour gérer les processus, le système d'exploitation sauvegarde dans des structures de données adéquates.

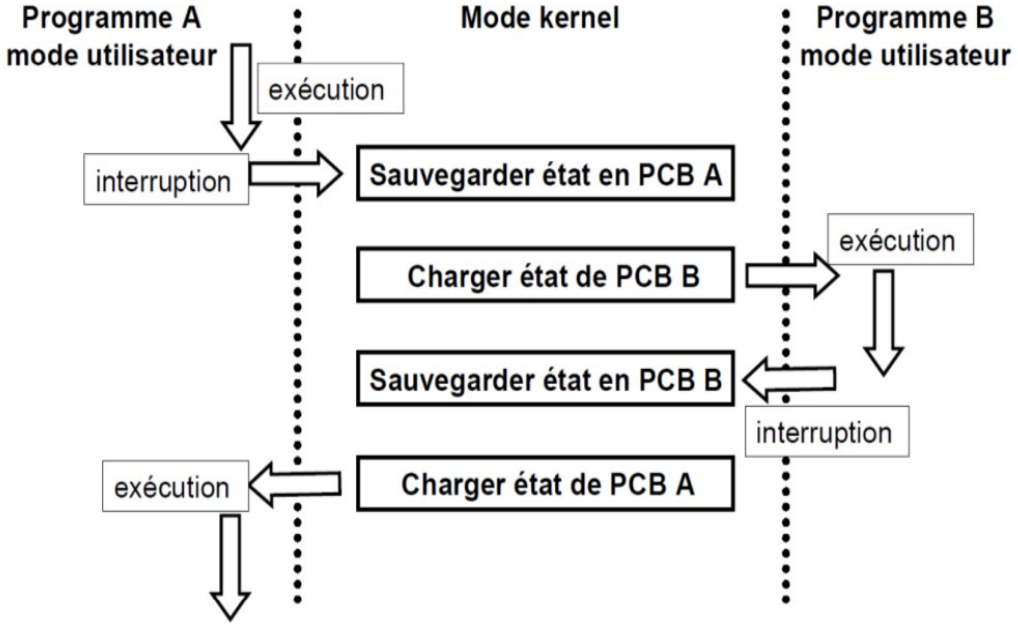
Il existe une table des informations concernant tous les processus créés. Il y a une entrée par processus dans la table, appelée le **Bloc de Contrôle de Processus (PCB).**
Chaque entrée de la table **PCB** comporte des informations sur :

- Le pid du processus (identificateur du processus).
- L'état du processus (Prêt, en attente, suspendu,...).
- Son compteur ordinal (adresse de la prochaine instruction devant être exécutée par ce processus).
- Contexte mémoire ou espace de travail (segment procédure, données, pile d'exécution).
- Les fichiers ouverts.
- Contexte processeur (mot d'état, les valeurs contenues dans les registres du processeur).
- Tout ce qui doit être sauvegardé lorsque l'exécution d'un processus est suspendue.

2.3.4- 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. L'opération de commutation de contexte, préalable au changement de processus actif, est l'enchaînement indivisible des deux opérations suivantes:

1- rangement du mot d'état du processus actif à un emplacement particulier de la mémoire et copie du contexte du processeur.
2- chargement d'un autre mot d'état, depuis un emplacement spécifique de la mémoire, vers le processeur et récupération du contexte du processeur correspondant.

Le nouveau processus peut alors être exécuté à partir de l'état où il se trouvait lorsqu'il a été lui-même interrompu. Insistons sur le fait que cette commutation de contexte ne peut être effectuée que lorsque le processeur se trouve dans un état observable, c'est à dire entre deux instructions. La notion d'instruction est relative au microcode de la machine qui a peu à voir avec celle utilisée par le programmeur qui pense aux instructions du langage évolué qu'il emploie. Une instruction de C ou Fortran, de Java encore plus, correspond à plusieurs instructions du microcode. Le temps d'exécution de ces instructions élémentaires représente le quantum de temps minimum pendant lequel le processeur ne peut être stoppé. Les points d'arrêt sont donc les instants entre les instructions. Ils sont désignés sous le nom de point interruptible.



2.4- Les systèmes d'interruption :

2.4.1- Interruption : Une interruption est un signal envoyé de façon asynchrone au processeur qui le force à suspendre l'activité en cours au profit d'une autre. La source peut être un autre processeur, un contrôleur d'entrées/sorties ou tout autre dispositif physique externe. Le programme en cours d'exécution suspend son activité au premier point interruptible. Le processeur exécute alors un programme prédéfini de traitement de l'interruption. Les causes d'interruption sont multiples. Il faut donc être capable de les distinguer et de les traiter chacune de façon spécifique. Pour cela, diverses méthodes peuvent être envisageables:

- 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.
- S'il existe des indicateurs multiples, chacun est appelé niveau d'interruption. On attache un programme différent de traitement à chacun de ces niveaux.
- On peut utiliser ces deux méthodes simultanément. Chaque niveau d'interruption est accompagné d'un code qui est lu par le programme de traitement de ce niveau. On distingue alors les interruptions matérielles qui sont les différents niveaux et les interruptions logicielles qui correspont4nt aux codes lus.

2.4.2- Organigramme générale d'une interruption : Lorsqu'une interruption survient, le processeur achève l'exécution de l'instruction en cours, puis il se produit

1- 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 peaux) rassemblant les indicateurs (Tout cela forme le contexte sauvegardé).

2- Chargement du contexte du programme d'interruption et passage en mode système (superviseur).

3- Exécution du programme d'interruption.

4- Retour au programme interrompu en restaurant le contexte et en repassant en mode utilisateur.

2.4.3- Type d'interruption : les interruptions se différencient selon leurs sources' (interne ou externe) et selon leurs intentions (volontaire ou involontaire); on distingue donc :

- **Interruption interne (déroutement):** est un événement qui se produit lorsque certaines conditions particulières apparaissent dans l'exécution d'un programme. C'est le cas d'un débordement d'une opération arithmétique, d'une division par zéro, ou d'un accès sur une zone interdite de la mémoire. Un déroutement provoque une rupture de la séquence d'exécution d'un programme vers une adresse mémoire spécifique à laquelle se trouve un branchement à la procédure de traitement du déroutement (Trap Handler). Cette dernière commence par sauvegarder le contexte du processeur qui sera restitué au retour du programme.
- **Interruption logique (appel système) :** permet à un processus utilisateur de faire un appel au système d'exploitation. Il a pour but : changer de mode d'exécution pour passer du mode utilisateur au mode maître, récupérer les paramètres et vérifier la validité de l'appel, de lancer l'exécution de la fonction demandée, de récupérer la (les) valeur(s) de retour et de retourner au programme appelant avec retour au mode utilisateur.
- **Interruptions matérielles :** déclenchées par une unité électronique (lecteur, clavier, canal, contrôleur de périphérique, panne de courant,....) ou par un autre processeur.

2.4.4- Priorité d'interruption : 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 $j > i$.

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.

2.4.5- 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.
Au contraire, une interruption peut-être désarmée : elle sera ignorée. Par défaut, les interruptions sont évidemment armées.