Chapitre 2: mécanismes de base d'exécution des programmes

1 Machine de Von Neumann

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

- Mémoire centrale;
- Processeur ou Unité Centrale (UC) pour effectuer les calculs et exécuter les instructions;
- Unités périphériques ou d'Entrées/Sorties.

1.1 Unité centrale de traitement (CPU, Central Processing Unit

Appelée aussi *Processeur Centrale* (PC). Elle est composée de:

- L'unité de commande (UC);
- L'unité arithmétique et logique (UAL);
- Les registres (données, instructions, contrôles);
- Bus interne.

1.1.1 Registres du processeur central

Les registres sont des mémoires internes à la CPU à accès très rapide qui permettent de stocker des résultats temporaires ou des informations de contrôle. Le processeur central dispose d'un certain nombre de registres physiques:

- **Le compteur ordinal (CO) (Program Counter (PC), Instruction Pointer (IP))** il contient l'adresse de la prochaine instruction à exécuter;
- **Le registre d'intruction (RI, Intruction register (IR))** il contient l'instruction en cours d'exécution;
- **Les registres généraux** ils permettent de faire le calcul, la sauvegarde temporaires de résultats, etc. Il s'agit de:
  - *Registres accumulateurs (Accumulator registers)*: ils servent aux opérations arithmétiques;
  - *Registres index (Index registers)*: ils sont utilisés pour l'adressage indexé. L'adresse effective d'un opérande est obtenue en ajoutant le contenu du registre index à l'adresse contenue dans l'instruction;

  - *Registre de base (Base register)*: il 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;
  - *Registre banalisés*: sont des registres généraux pouvant servir à divers opérations (stockages des résultats intermédiaires, sauvegarde des informations fréquemment utilisées, etc.). Ils permettent de limiter les accès à la mémoire, ce qui accélère l'exécution d'un programme.

- **Le registre pile (Stack pointer (SP))** il pointe vers la tête de la pile du processeur (Pile système). Elle est réservé à l'usage de l'unité centrale, en particulier pour sauvegarder les registres et l'adresse de retour en cas d'interruption ou lors d'un appel d'une procédure ou de fonction. Le pointeur de pile est accessible au programmeur, ce qui souvent source d'erreur;

- **Le registre mot d'état (Program status word (PSW))** il contient plusieurs types d'informations, à savoir:
  - Les valeurs courantes des flags qui sont des bits utilisés dans les opérations arithmétiques et comparaisons d'opérandes;
  - Le mode d'exécution. Deux modes d'exécution existent généralement : mode utilisateur et mode noyau;
  - Masque d'interruptions.

1.2 Cycle d'exécution du processeur

Le processeur central exécute continuellement le cycle suivant:

1.3 Etat du processeur (Processor State Information)

Il est décrit par le contenu des registres du processeur d'un 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 restituée quand le processus interrompu reprend son exécution.

2 Concept de processus

2.1 Définition

Un processus (Process) est un programme en d'exécution. Il est constitué:

- Un code exécutable du programme en exécution;
- Un contexte qui est l'ensemble des données qui permettent de reprendre l'exécution d'un processus qui a été interrompu. Il est formé des contenus de:
  - Compteur ordinal (CO);
  - Mot d'état PSW;
  - Registres généraux;

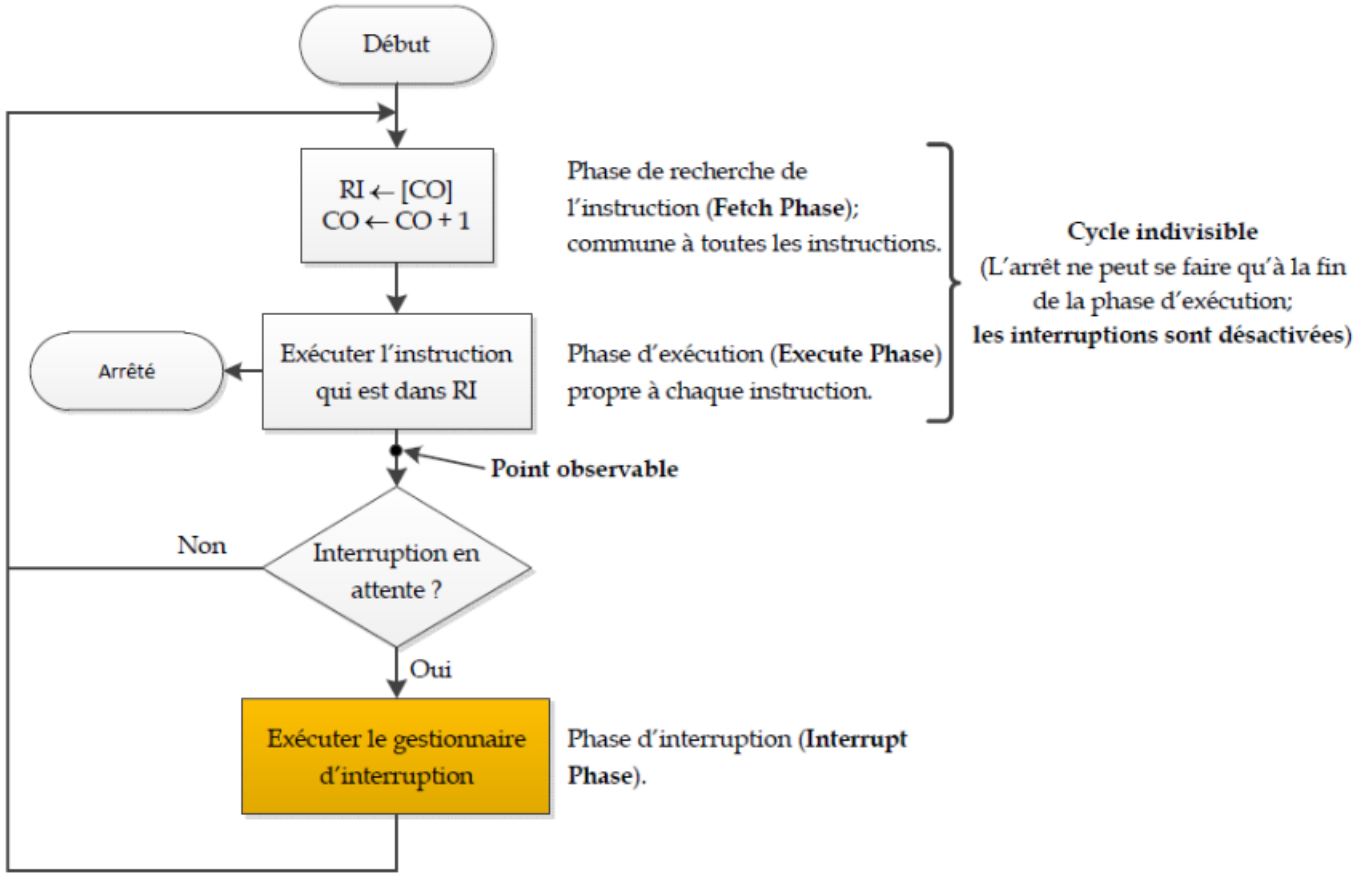

Figure 1: Cycle d'exécution du processeur

– Pile.

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

2.2 Image mémoire d'un processus

L'image mémoire (Memory map) d'un processus c'est l'espace mémoire alloué pour ce processus. Il est divisé en un ensemble de parties:

- **Code (Text)**: qui correspond au code des instructions du programme à exécuter. L'accès à cette zone se fait en lecture seulement (Read-only);
- **Données (Data)**: qui contient l'ensemble des constantes et variables déclarées;
- **Pile (Stack)**: qui permet de stocker les valeurs des registres, les variables locales, les paramètres de fonctions et l'adresse de retour de fonctions;
- **Tas (Heap)**: une zone à partir de laquelle l'espace peut être alloué dynamiquement en cours d'exécution, en utilisant par exemples les fonctions new et malloc (langage C/C++).

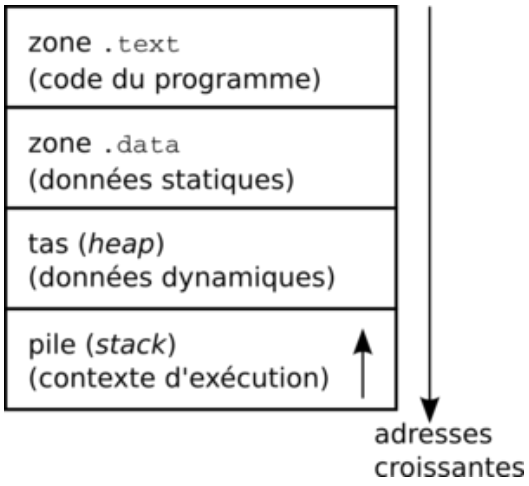

Figure 2: Image mémoire d'un processus

2.3 Etat d'un processus

Un processus prend un certain nombre d'états durant son exécution. Les trois principaux états d'un processus sont:

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

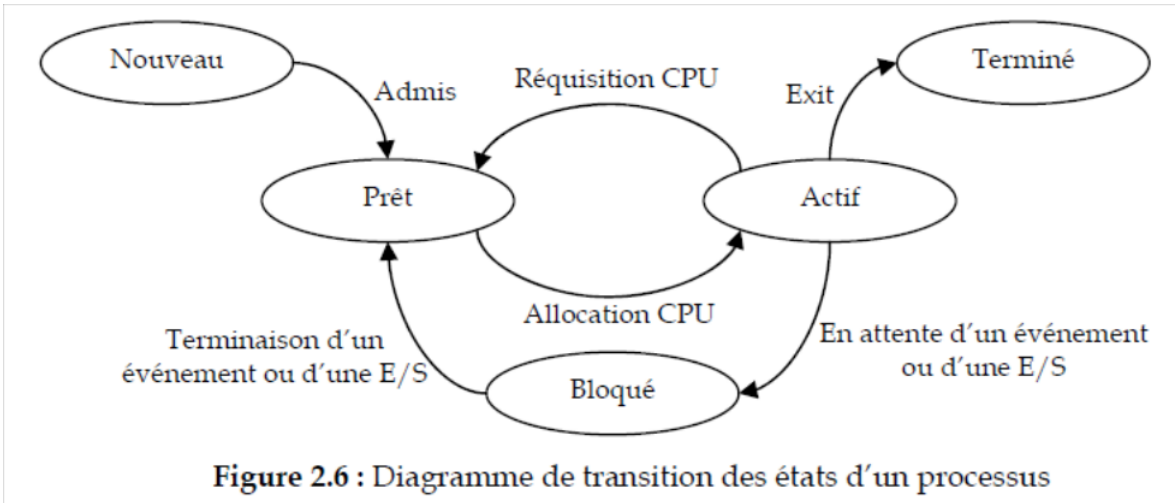

Figure 2.6 : Diagramme de transition des états d'un processus

Figure 3: Etat d'un proccessus

2.4 Processus sous Linux/Unix

2.4.1 Identification d'un processus

Un processus est identifié de manière unique par un numéro appelé **PID (Process IDentifier)**.

La commande **ps** donne la liste des processus en cours d'exécution et leurs priorités.

la fonction **getpid()** retourne le numéro du processus qui l'exécute.

Syntaxe

```
#include<sys/types.h>
#include<unistd.h>

pid_t getpid();
```

La fonction **getppid()** retourne le numéro du processus **père** (créateur) du processus qui l'exécute.

Syntaxe

```
#include<sys/types.h>
#include<unistd.h>

pid_t getppid();
```

2.4.2 Création d'un processus

Un processus est créé par la fonction **fork()**. Le processus créé (fils) est un clone (copie conforme) du processus créateur (père).

Syntaxe

```
#include<sys/types.h>
#include<unistd.h>

pid_t fork();
```

Le père et le fils ne se distinguent que par le résultat retourné par le fork. Pour le père, cette fonction renvoie le numéro du fils (-1 en cas d'erreur) et pour le fils, elle renvoie 0.

```
pid_t pid = fork();

switch(pid) {
    case -1: /* Erreur lors de l'appel de la fonction fork() */
    ...
    case 0: /* On est dans le programme fils */
    ...
    default: /* On est dans le programme père */
    ...
}
```