

Les mécanismes d'exécution

Processus, ordonnancement

Jérôme David - Florian Rodriguez

Introduction

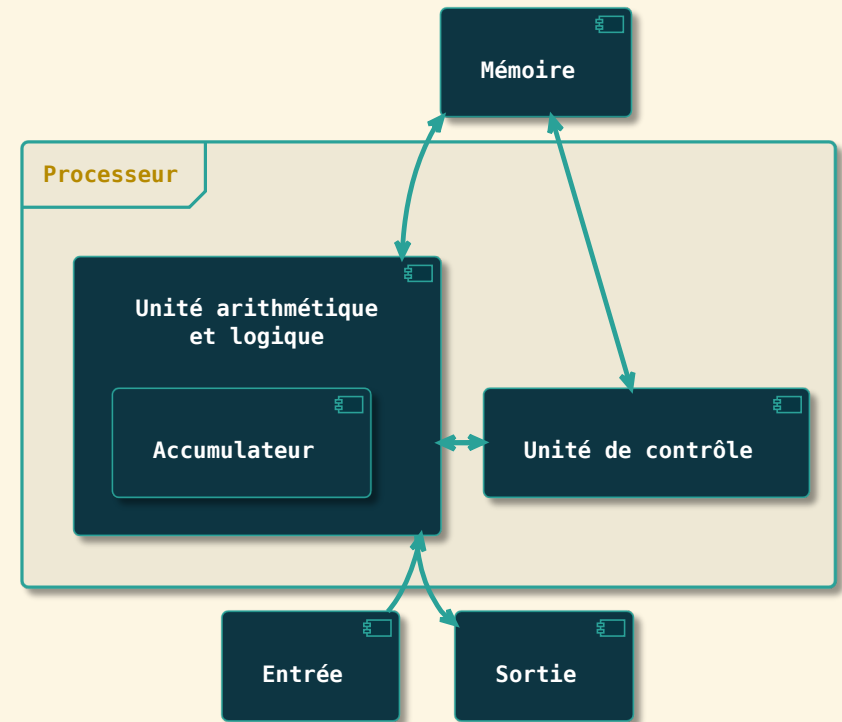
- La notion de processus est centrale
 - C'est une abstraction d'un programme en cours d'exécution
 - Elle permet de simuler des opérations concurrentes même si il n'y a qu'un seul CPU
 - Elle fournit une vue virtuelle de la mémoire, des CPU (plusieurs CPU virtuels avec un seul CPU physique)
 - Elle permet de donner « l'illusion » que plusieurs programmes s'exécutent en même temps (la multiprogrammation)
- Programme != processus
 - Un programme est une entité passive (un fichier exécutable)

Principes de la multiprogrammation

- Il n'y a qu'un seul processus en cours d'exécution à la fois (si 1 seul processeur)
 - Un processus n'est pas exécuté en une seule fois
 - On doit être capable d'enregistrer l'état d'un processus pour y revenir plus tard
 - Il faut des stratégies pour répartir l'allocation du processeur aux processus

Modèle de processeur

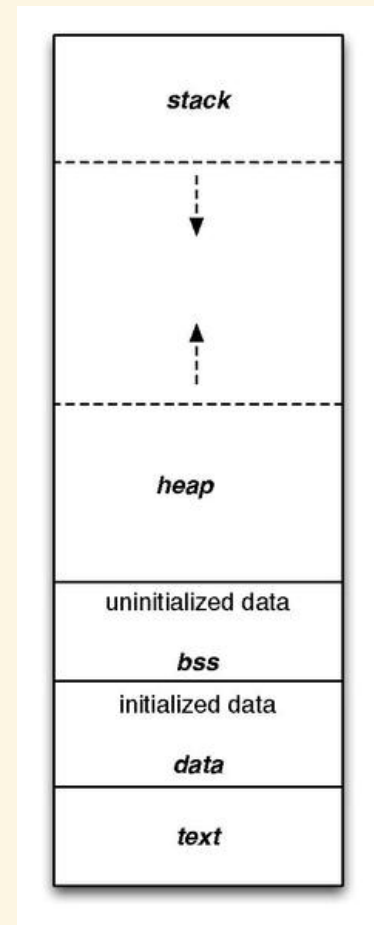
- Instructions = opérations
 - Accès à la mémoire, opérations arithmétiques & logiques, contrôle (branchements)
- Registres (mémoire interne du processeur)
 - Accumulateur : pour stocker le résultat
 - **Registre d'état (PSW)** : zéro, retenue, dépassement, signe, etc.
 - Registre d'instruction (IR) : instruction en cours
 - **Compteur ordinal (PC)** : adresse de la prochaine instruction à charger
 - Pointeur de pile (SP) : prochain emplacement dispo dans la pile mémoire



Architecture de von Neumann
Source: Wikipedia

Le modèle de processus

- Chaque processus possède un état
 - La valeur du compteur ordinal (PC)
 - Le contenu des registres
- Et une image mémoire (segment de données)
 - *stack* (La pile) : stockage temporaire pour les param de fonction, retour, variables locales
 - *heap* (Le tas) : mémoire allouée dynamiquement
 - *data* et *bss* Les variables globales
 - *text* Code exécutable



Exemple

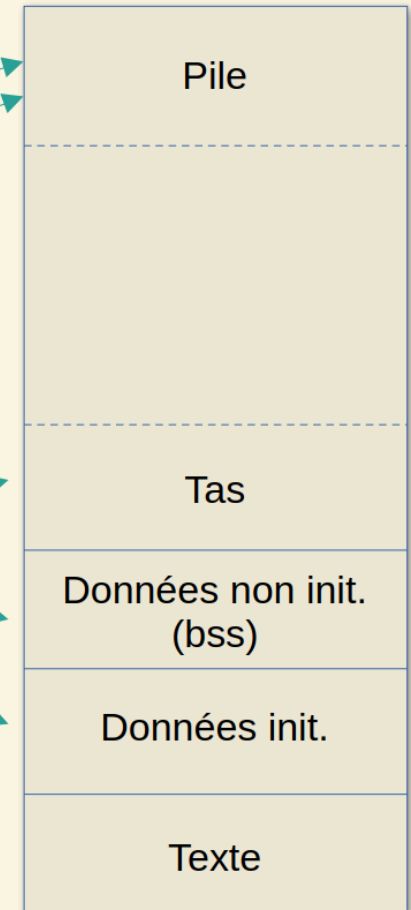
```
#include <stdio.h>
#include <stdlib.h>

int nb=10;
int res;

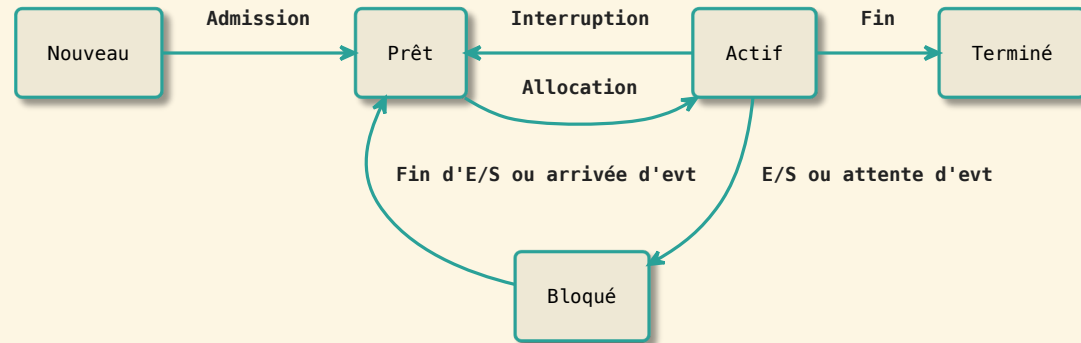
int main(int argc, char* argv[]) {
    // un pointeur vers un tableau d'entier
    int* tab;
    int i;

    // allocation d'un tableau de nb entiers
    tab = (int*) malloc(sizeof(int)*nb);

    for (i=0 ; i<nb ; i++) {
        tab[i]=i;
    }
    return 0;
}
```



Le cycle de vie d'un processus - ses états



Bloc de contrôle de processus - PCB

- Contient toutes les infos permettant de démarrer ou redémarrer un processus
 - sauvegardé à chaque passage « actif » vers « prêt » ou « bloqué »
 - restauré au passage « prêt » vers « actif »
- Process bloc control (PCB) :
 - L'identifiant du processus (PID), du parent (PPID), de l'utilisateur (UID)
 - Les valeurs des registres processeur
 - Le compteur ordinal (PC)
 - Le pointeur de pile (SP)
 - L'espace d'adressage (mémoire virtuelle)
 - La listes de descripteurs de fichiers (fichiers ouverts)
 - Information d'ordonnancement
 - ... (ca dépend des systèmes)

La création des processus

- Cela revient à « lancer un programme »
 - Par clic ou en tapant une commande dans le shell
 - Certains processus sont lancés au démarrage
- La création du processus est une opération système (exécutée en mode noyau)
 - Réalisée par l'appel système `fork( )` sous Unix

La création des processus (cont.)

- Chaque processus est créé par un père
 - Hiérarchie de processus (vu précédemment)
- L'appel `fork` crée un processus fils, *clone* du processus père qui l'exécute
 - Même image mémoire, etc.
 - Par contre, ils ne partagent rien

Exemple de fork

Copy

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

int main() {
    pid_t pid = fork();
    if (pid == -1) {
        // Il y a une erreur
        perror("fork");
        return EXIT_FAILURE;
    } else if (pid == 0) {
        // On est dans le fils
        printf("Mon PID est %i et celui de mon père est %i\n", getpid(), getppid());
    } else {
        // On est dans le père
        printf("Mon PID est %i et celui de mon fils est %i\n", getpid(), pid);
    }
    return EXIT_SUCCESS;
}
```

Terminaison d'un processus

- Un processus se termine
 - Lorsqu'il a exécuté sa dernière instruction
 - Par appel à la fonction système `exit(n)` ou `n` est le code de retour
 - La terminaison d'un processus libère toutes les ressources du processus
- Le code de retour est envoyé au processus père qui le récupère via l'appel système `wait(&nb)`
 - Entre le moment où le processus est terminé et le moment où la fonction `wait()` du père est appelée, le processus est considéré comme un zombie
 - Il n'a plus d'image mémoire mais il a encore un PCB en mémoire (où est stocké) le code de retour

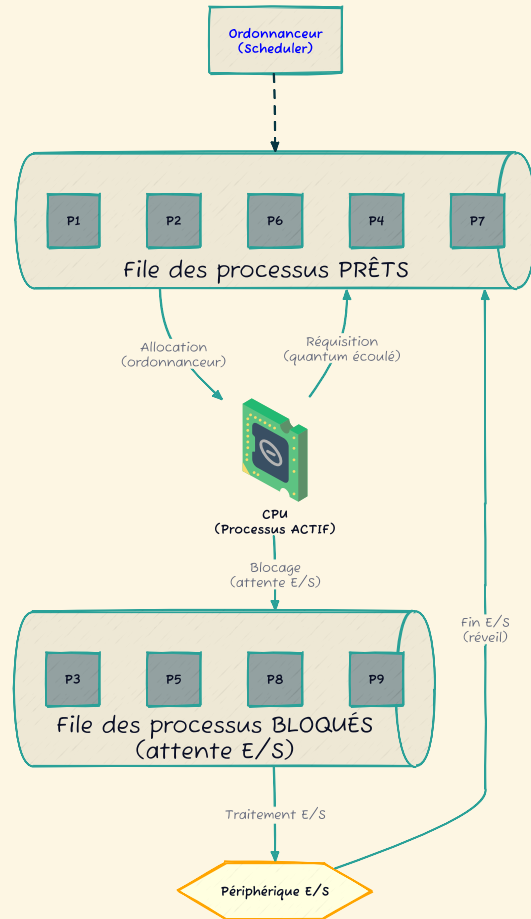
La gestion de l'exécution

- Il faut pouvoir arrêter et redémarrer un processus
 - Commutation de contexte, allocation du CPU
 - Rôle du « dispatcher »
- Il faut gérer des files d'attentes
 - Pour l'accès au CPU ou autres périphériques
- Réaliser une stratégie d'ordonnancement du CPU au processus prêts
 - Rôle du « scheduler »

Commutation de contexte

- Pour passer d'un processus à un autre
- Principe (simplifié) :
 - Sauvegarde du PCB du processus à interrompre
 - Chargement du PCB du processus à exécuter
- C'est généralement réalisé via des interruptions
 - Un mécanisme matériel du processeur
- Une commutation prend un certain temps...
 - C'est la latence de commutation (dispatcher latency)

Les files d'attentes



L'ordonnancement

- C'est le rôle du scheduler
 - Il doit choisir un processus parmi tous ceux qui sont prêts et lui allouer le CPU
- Il existe plusieurs stratégies
 - Non Préemptif
 - Laisse un processus actif tant qu'il ne bloque pas (état attente) ou qu'il n'est pas terminé (ou que le processus laisse volontairement passer son tour)
 - Utilisé dans les systèmes en mode « batch »
 - Préemptif
 - Laisse un processus dans l'état actif un temps maximum fixé.
 - Nécessite d'avoir une interruption d'horloge
 - Utilisé dans tous les systèmes d'exploitation interactifs « modernes »

Buts des algorithmes d'ordonnancement

- L'équité entre les processus
- Maximisation
 - de l'utilisation du CPU
 - du rendement (tâches terminées / unité de temps)
- Minimiser
 - Les temps d'exécution des tâches
 - Le temps d'attente (dans l'état prêt)
 - Le temps de réponse (surtout dans les systèmes interactifs)

Critères de qualité de l'ordonnancement

- **Efficacité/Rendement:** le maximum de temps doit être consacré à l'application
- **Temps de réponse:** le plus faible possible (lié au temps de latence)
- **Impartialité:** partage équitable entre tâche
- **Débit:** le plus de tâches possibles dans un temps donné

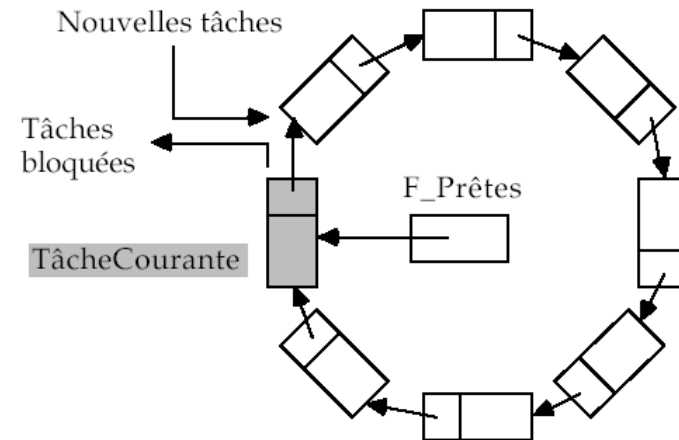
Quelques algo d'ordonnancement (non préemptifs)

- Premier arrivé, premier servi (FCFS - First Come First Served)
 - Facile à implémenter, mais pb lorsque qu'une tâche prend trop de temps...
- Le plus court d'abord (Shortest Job First)
 - Ca serait bien mais on ne sais pas a priori le temps que va mettre un processus à s'exécuter...
 - Variante : le plus court restant.

Le Tourniquet

- Round Robin
 - FCFS en version préemptive
 - On change de processus après un certain temps
 - Quantum de 10ms à 100ms

La file des tâches prêtes est circulaire et gérée en FIFO.

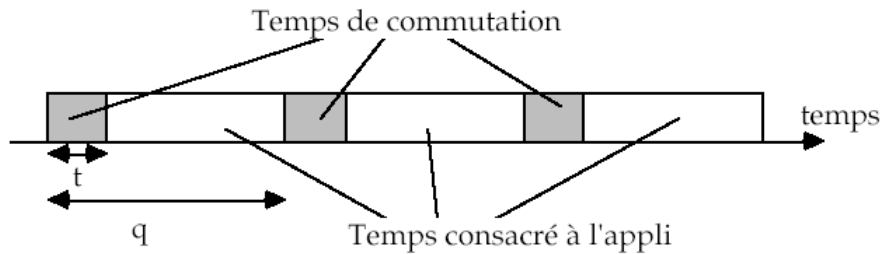


Sans réquisition : très efficace s'il y a peu de tâches qui se bloquent souvent (E/S)

Avec réquisition : c'est une méthode impartiale pour les tâches de même priorité.

Le Tourniquet - choix du quantum

Compromis entre débit et efficacité



$$\text{Efficacité } E = \frac{q - t}{q} = \text{rapport } \frac{\text{temps consacré à l'appli}}{\text{temps total}}$$

Exemple : $t = 1\text{ms}$

$$E = 0.8 \text{ si } q = 5\text{ms}$$

$$E = 0.98 \text{ si } q = 50\text{ms}$$

$$\text{Débit } D = \frac{1}{q} = \text{nombre de tâches traitées par seconde}$$

Exemple :

$$D = 200 \text{ si } q = 5\text{ms}$$

$$D = 20 \text{ si } q = 50\text{ms}$$

Meilleur débit = meilleur temps de réaction

Meilleure efficacité = temps total d'exécution plus court

Bien sûr il faut que t soit le plus petit possible.

La priorité pure

- Chaque processus est associé à une priorité
- L'ordonnanceur choisi la tâche la plus prioritaire
- PB de monopolisation du CPU
 - On décrémente la priorité à chaque interruption d'horloge
 - Le processus laissera donc sa place au bout d'un certain temps

Files d'attente multi-niveau

- Approche mixte : tourniquet + priorité

On a quelques niveaux de priorité mais on peut avoir plusieurs tâches au même niveau de priorité.

