Ordonnancement et processus

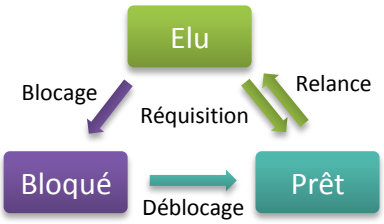
SE – Chapitre 5

I. Le processus

1. Définition

- Un processus est une unité d'exécution
- Connait : code du programme / pointeur d'instruction / état de la pile / variables

2. Etats d'un processus



- **Elu :** en cours d'exécution
- **Prêt :** Attente de disponibilité processeur
- **Bloqué :** Attente d'un évènement extérieur

II. L'ordonnancement

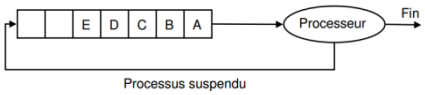
1. Critères

- **Equité :** même temps pour chaque process
- **Efficacité :** 100% utilisation UC
- **Temps de réponse :** Ne pas trop attendre
- **Temps d'exécution :** Pas trop long, pas trop court
- **Rendement :** plus d'opérations en un temps donné

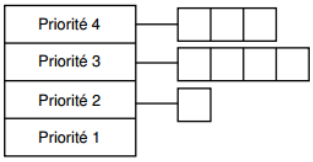
2. Types d'ordonnancement

- **Sans réquisition :** Processus exécuté jusqu'à la fin
- **Avec réquisition :** Le SE gère le partage du temps de calcul

3. Exemples d'ordonnancement



- **Ordonnancement circulaire :**
  - Un processus a un quantum d'action pendant lequel il travaille. (Ne doit pas être trop court ni trop long)
  - S'il n'a pas terminé à la fin, repasse en bout de liste.
- **Ordonnancement avec priorité :**
  - Plusieurs files plus ou moins prioritaires
  - Priorité décroit au cours du temps



- **Ordonnancement « plus court d'abord »**
- **Ordonnancement garanti (à l'utilisateur)**
  - Divise le temps également entre les n utilisateurs connectés.

III. Les processus UNIX

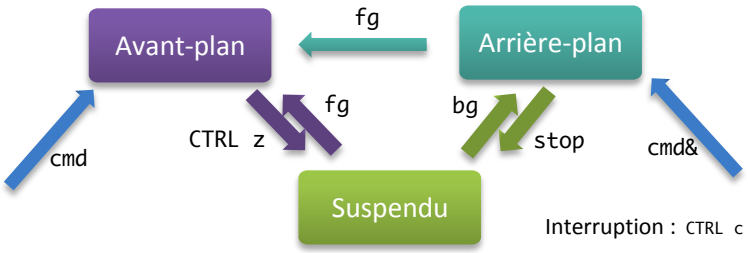
1. Caractéristiques

- Caractérisé par PID (Process ID) et PPID (Parent PID)
- 2 types : process daemons / process utiliseurs

2. Commandes

- **Plusieurs commandes :** `C1 ; C2 ; C3`
- **Redirections :**
  `commande < fichier_stdin`
  `commande > fichier_stdout`
  `commande >& fichier_stdout_stderr`
- **Connexion (tubes) :** `c1 | c2` *(stdout1 = stdin2)*
- **Exécution cond. :** `c1 && c2` *(c2 ssi c1 réussi)*
  `c1 || c2` *(c2 ssi c1 échec)*

3. Modes d'exécution



- **Différé :**
  - `at` : à une date donnée
  - `batch` : à une date donnée selon charge processeur
  - `crontab` : tâches exécutées régulièrement

4. Appels systèmes

- int fork() : duplique processus (return 0 dans fils, PID fils dans père, -1 si echec)
- void exit(int status) : termine processus, retourne status au père
- int sleep(int seconds) : bloque le processus
- int execlp() : remplacer l'exécutable d'un processus
- int wait(0) : bloqué jusqu'à l'arrêt d'un fils, retourne -1 si pas de fils, sinon pid du fils terminé
- int getpid() : numéro de pid du processus
- int kill(int pid, int signum) : envoie un signal signum au process pid

5. Pipes

Fichier permettant la communication entre processus : un accès en écriture, un accès en lecture destructrice (fifo).

6. File de messages

Boite à lettres « numérotée » avec clé.

7. Sémaphores

Un sémaphore S est une variable entière. **P(S)** attend que S soit ≥ 1 puis **décrémente** S. **V(S)** **incrémente** S. **Z(S)** attend que S soit **nul**. Permet la synchro des processus.

8. Segment de mémoire partagée

Partage de la mémoire physique entre processus. Nécessite synchro.