Processus et changements de contextes

SYE

4 - Processus & changement de context

Résumé du document

Le document explique la gestion des processus et des changements de contexte dans un système multitâche. Le PCB stocke l'état du processus. `fork()` crée un processus fils, `waitpid()` libère ses ressources, et `exec()` remplace l'image d'un processus. Les changements de contexte s'effectuent lors de l'attente, la fin, ou par décision de l'ordonnanceur.

Table des matières

1. Structure PCB (Process Control Block)

- PCB (Process Control Block)
  - Fiche signalétique d'un processus
  - Structure de données gérée et visible que par le noyau
  - Contient toutes les informations système liées au processus
  - Utilisé lors de changements de contexte
- Chaque processus dispose d'un PCB propre et unique.

1.1. Composition

Un PCB est une strucutre de donnée contenant un certains nombre d'informations:

- état du processus
- espace d'adressage
- pointeur vers le tas (heap)
- références vers les processus fils
- descripteur des fichiers ouverts
- contexte(s) d'exécution
  - etat du contexte d'exécution
  - pointeur sur instruction courante (PC)
  - registre de données
  - pile

L'allocation du PCB s'effectue à la création d'un nouveau processus.

2. Création de processus

L'appel système **fork()** permet de créer un nouveau processus. ce processus sera un **processus fils** de celui qui exécute l'appel à **fork()**.

On fera une copie intégrale du processus parent dans le processus fils y compris:
- le code
- les données
- la position courante

2.1. Valeur de retour de `fork()`

- **0** : Indique que le code s'exécute dans le processus fils.
- **> 0** : Indique que le code s'exécute dans le processus parent, et la valeur est le PID du fils (utile pour des opérations comme `waitpid()`).
- **< 0** : Indique un échec de création du processus fils (par ex. nombre maximal de processus atteint, manque de mémoire).
- **Erreur détaillée** : La cause de l'échec est disponible dans la variable globale `errno`.

2.2. exit() & waitpid()

- **Appel système `exit()` :**
  - Utilisé par le processus pour signaler sa terminaison.
  - Accepte un entier (`status`) comme argument, qui représente le code de sortie.
  - Ce code est conservé dans le PCB (Process Control Block) et pourra être récupéré par le parent.
- **Appel système `waitpid()` :**
  - Utilisé par le processus parent pour :
    - Attendre la fin d'un processus fils.
    - Récupérer le code de sortie (status) donné par le fils via `exit()`.
    - Libérer définitivement les ressources et la mémoire associées au processus fils.
  - Pendant l'attente, le processus fils se trouve dans un état « zombie » jusqu'à ce que `waitpid()` soit appelé.
  - **Obligation pour le parent** : le parent doit exécuter `waitpid()` pour supprimer proprement le processus fils du système.

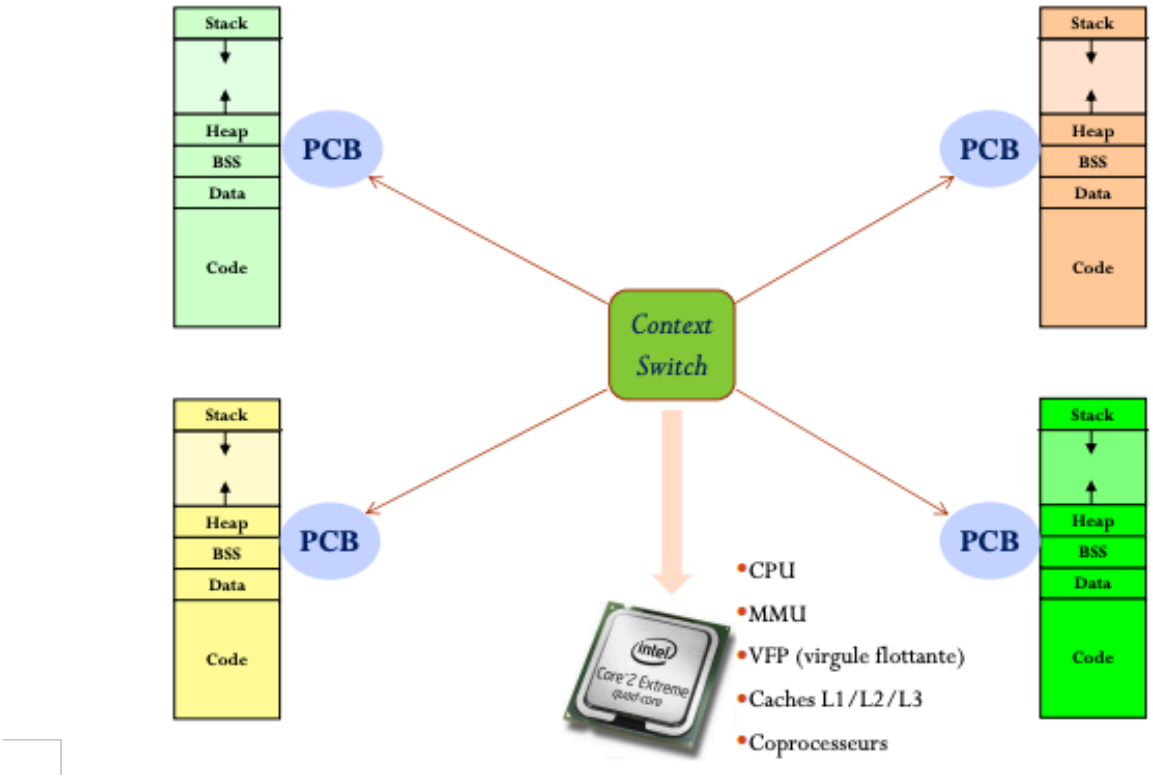
Afin d'éviter l'apparition de processus orphelins, il est nécessaire d'attendre la terminaison des processus-enfants dans tous les cas avec l'appel système «waitpid()», comme le montre le code ci-dessus.

2.3. exec()

- **Appel système `exec()` :**
  - Remplace l'image binaire actuelle d'un processus par une nouvelle.
  - Lance l'exécution à une adresse spécifique (point d'entrée), où la fonction `main()` est appelée.
  - Permet de passer des arguments à l'image binaire (ex. chemin de fichier, nom, etc.).
- **Famille `exec`** :
  - Regroupe plusieurs fonctions de la librairie C (Posix), avec `execve()` comme véritable appel système.
  - Utilisée pour charger et démarrer une nouvelle application.
- **Processus de lancement** :
  - **Étape 1** : Création du processus avec `fork()` par le parent.
  - **Étape 2** : Chargement et démarrage de la nouvelle image avec `exec()` par le fils.

3. Changement de contexte

- **Système multitâche et changement de contexte :**
  - Un système multitâche permet à plusieurs processus de résider en mémoire, impliquant des changements de contexte.
- **Scénarios déclenchant un changement de contexte :**
  - **Terminaison d'un processus** : Appel à `exit()` pour signaler au parent que le processus se termine, libérant ainsi ses ressources.
  - **Mise en attente** : En cas de blocage sur une opération, par exemple un accès disque, le processus est suspendu en attendant la réponse du périphérique.
  - **Décision de l'ordonnanceur** : Lorsque le temps alloué à un processus expire, l'ordonnanceur peut le remplacer par un autre processus prêt à exécuter.

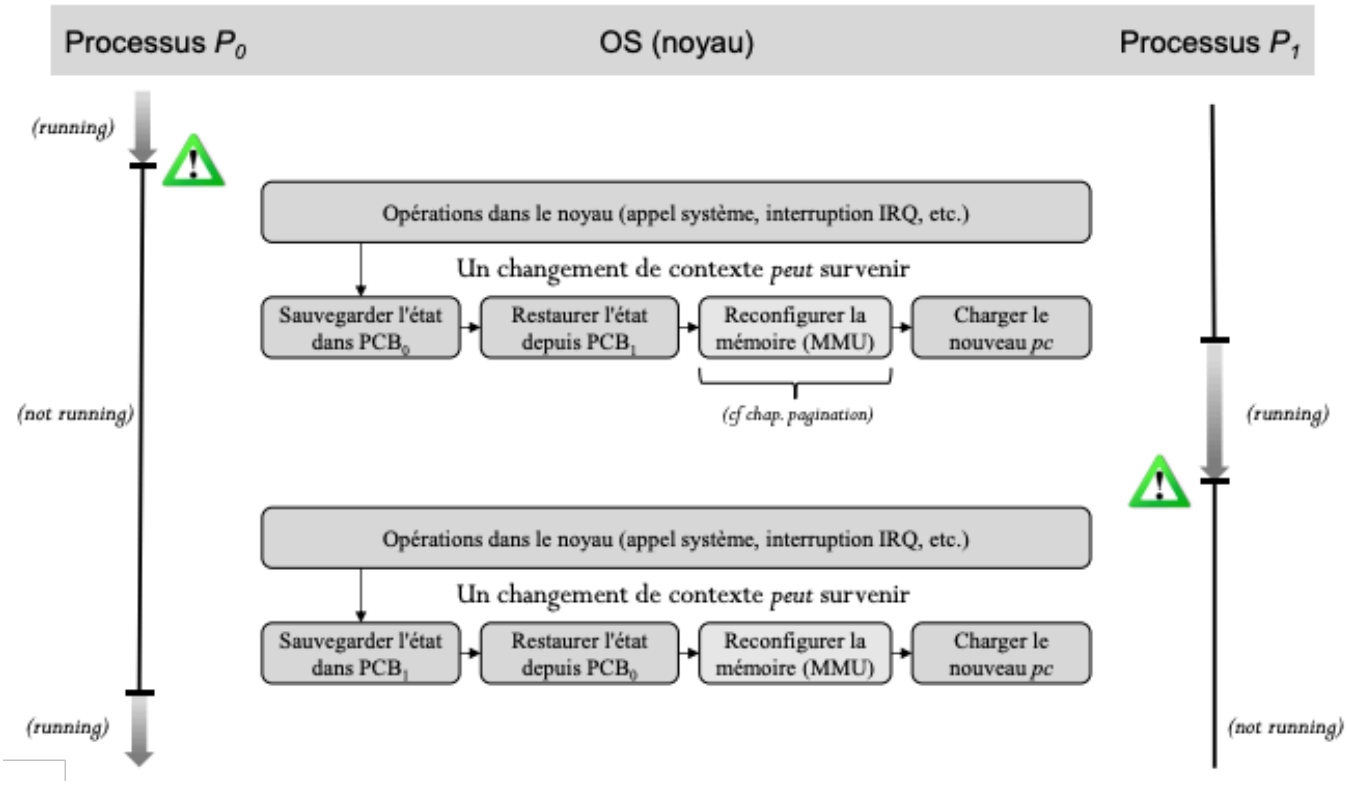


Un changement de contexte entre processus est une opération couteuse (ordre de la milliseconde) car elle fait intervenir beaucoup d'opérations sur le matériel.

L'état actuel du processus sortant sera stocké dans son PCB. De même, l'état du processus entrant sera récupéré depuis son PCB.

3.1. Execution d'un changement de contexte

- **Contexte d'exécution et de mémoire :**
  - Lors d'un changement de contexte, le noyau sauvegarde l'état du processus actuel en copiant les valeurs des registres processeur dans son PCB.
  - Il préserve aussi les pointeurs vers les sections mémoire du processus (pile, tas, etc.).
- **Rôle du matériel :**
  - Les unités matérielles comme les coprocesseurs (arithmétique, graphique) nécessitent la sauvegarde de leur état actuel.
  - Les caches doivent être synchronisées avec la RAM pour garantir la cohérence des données.
- **MMU et mémoire :**
  - La MMU doit être reconfigurée pour correspondre à l'espace mémoire du processus entrant, assurant un accès mémoire correct.


Processus P_0
OS (noyau)
Processus P_1
(running)
Opérations dans le noyau (appel système, interruption IRQ, etc.)
Un changement de contexte *peut* survenir
Sauvegarder l'état dans PCB_0
Restaurer l'état depuis PCB_1
Reconfigurer la mémoire (MMU)
Charger le nouveau *pc*
(cf chap. pagination)
(not running)
(running)
Opérations dans le noyau (appel système, interruption IRQ, etc.)
Un changement de contexte *peut* survenir
Sauvegarder l'état dans PCB_1
Restaurer l'état depuis PCB_0
Reconfigurer la mémoire (MMU)
Charger le nouveau *pc*
(running)
(not running)