Processus et threads

PCO

2 - Threads et Contexte

Résumé du document

Definition

Table des matières

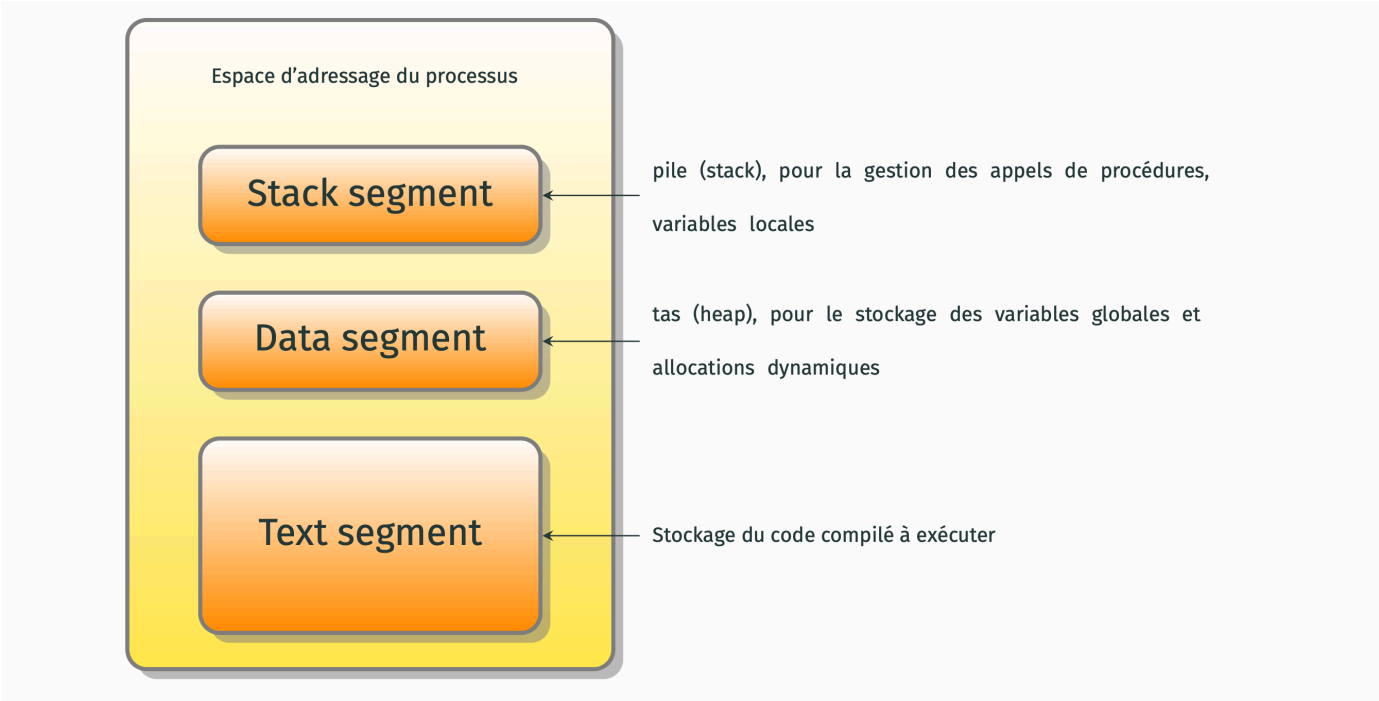
1. Anatomie d’un processus

1.1. Propriétés d’un processus

- Un code à exécuter
- Un espace d’adressage
- Une priorité
- Un identifiant
- Un contexte d’exécution (PC + registres)

Les processus sont gérés par le système d’exploitation.

1.2. Adressage d’un processus



1.3. Etat et transition d’un processus

Etat	Description
Prêt (ready)	Le processus est prêt à être exécuté. Cas d'un processus nouvellement créé, débloqué ou, d'un ou plusieurs processus occupant le ou les processeurs disponibles
Elu (running)	Le processus est en cours d'exécution sur un processeur. Plusieurs processus peuvent être en exécution dans le cas d'une machine multiprocesseur.
Bloqué (waiting)	Le processus est en attente sur une synchronisation ou sur la fin d'une opération d'entreé/sortie par exemple.
Zombie	Le processus a terminé son exécution, mais son processus parent doit encore récupérer sa valeur de terminaison.
Terminé	Le processus a terminé son exécution ou a été annulé (cancelled). Les ressources du processus seront libérées et le processus disparaîtra. Il s'agit d'un pseudo-état.

2. Anatomie d’un thread

Un thread est un fil d’exécution dans un processus

- Les threads d’un même processus se partagent l’espace d’adressage du

processus
- Ils possèdent:
  - leur propre pile (stack)
  - leur propre contexte d’exécution (PC + registres)
- Ils ont un cycle de vie semblable à celui d’un processus

2.1. En commun processus et thread

Processus et thread
Processus et Thread
Possèdent un ID, un ensemble de registres, un état, et une priorité
Possèdent un bloc d'information
Partagent des ressources avec les processus parents
Sont des entités indépendantes, une fois créés
Les créateurs de processus et thread ont contrôle sur eux
Peuvent changer leurs attributs après création, et créer de nouvelles ressources
Ne peuvent accéder aux ressources d'autres threads et processus non reliés

2.2. Non commun processus et thread

Processus	**Thread**
Propre espace d'adressage	Pas d'espace d'adressage propre
Les processus parents et enfants doivent utiliser les mécanismes de communication inter-processus	Les threads d'un même processus communiquent en lisant et modifiant les variables de leur processus
Les processus enfants n'ont aucun contrôle sur les autres processus enfants	Les threads d'un processus sont considérés comme des pairs, et peuvent exercer un contrôle sur les autres threads du processus
Les processus enfants ne peuvent pas exercer de contrôle sur le processus parent	N'importe quel thread peut exercer un contrôle sur le thread principal, et donc sur le processus entier

2.3. Changement de contexte d’exécution

L’opération de changement de contexte d’un processus (ou thread) comporte les séquences suivantes :

1. Mise en attente du processus actif dans la liste des processus bloqués ou prêts
2. Sauvegarde de son contexte d’exécution
3. Recherche du processus éligible ayant la plus haute priorité
4. Restauration du contexte d’exécution du processus élu ⟹ restauration de la valeur de ses registres lorsqu’il s’exécutait précédemment
5. Activation du processus élu

Tout se apsse comme si le processus préalablement interrompu n’avait pas cessé de s’exécuter.

2.4. Green threads et threads natifs

- **Green Threads**
  - Ordonnancé par une machine virtuelle ou librairie
    - Emulation de l’ordonnanceur natif
  - Cela est utile sur système bare metal (système sans ordonnanceur)
- **Thread natif**

- Ordonnancé par la plateforme (OS)
- Seule façon d’exploiter du multi-coeur

3. Librairie PcoThread

La librairie PcoThread est une librairie maison utilisée dans ce module pour gérer la concurence.

3.1. Jointure

Une jointure permet à un thread d'attendre qu'un autre se termine. Il s'agit d'une fonction dite **bloquante**, car elle bloque l'exécution du thread appelant la fonction `PcoThread::join()` jusqu'à que la tâche soit du thread attendu soit terminée.

3.2. Terminaison

La terminaison d'un thread peut être exécutée depuis:

- Le thread lui-même:
  - Fin de l'exécution de la fonction lancée
  - `return`;
- Un autre thread?
  - Pas une excellente idée
  - Pourquoi?
  - Mal terminer un thread peut laisser le système dans un état incohérent!

Pour terminer l'application (destruction de tous les threads): • exit()