Introcution aux threads

PCO

1 - Introduction

Résumé du document

Table des matières

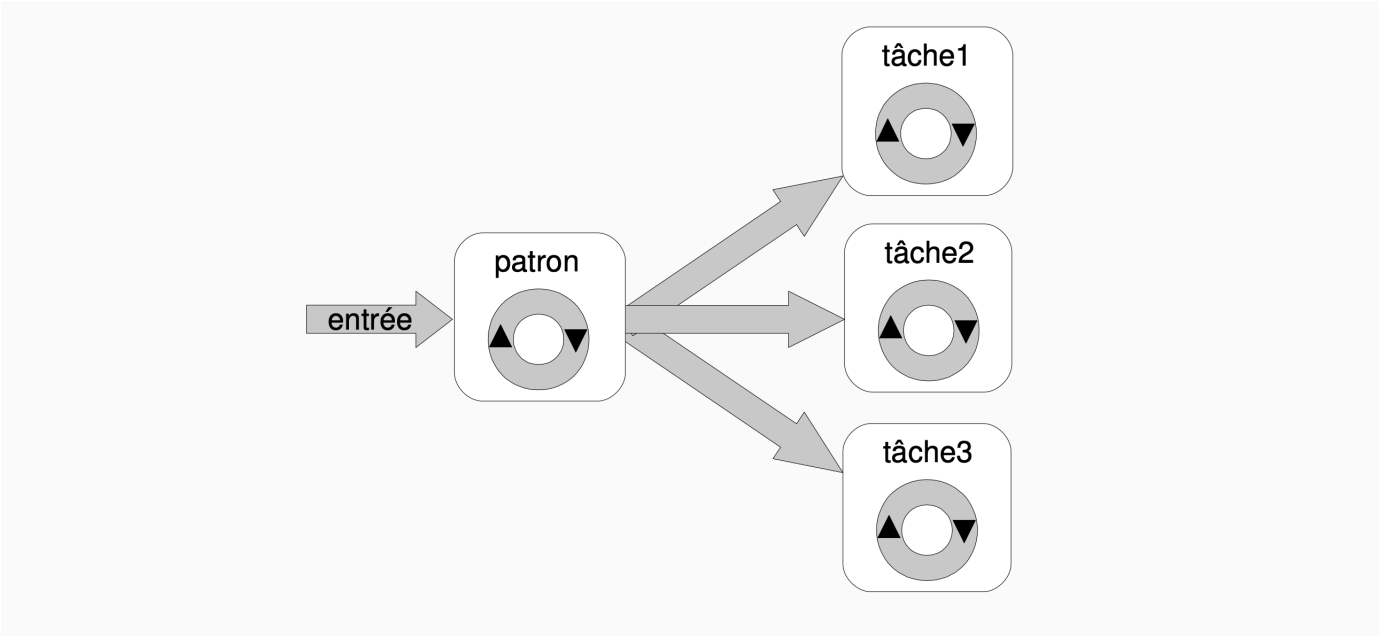
1. Modèles de délégation 2
1.1. Modèle de délégation (boos-worker model / delegation model) 2
1.2. Modèle pair (peer model) 3
1.3. Modèle pipeline (pipeline model) 4
2. Anatomie d’un processus 5
2.1. Propriétés d’un processus 5
2.2. Adressage d’un processus 5
2.3. Etat et transition d’un processus 5
3. Anatomie d’un thread 6
3.1. En commun processus et thread 6
3.2. Non commun processus et thread 6

1. Modèles de délégation

1.1. Modèle de délégation (boos-worker model / delegation model)

Le modèle de délégation est un modèle de conception qui permet à un objet de déléguer une partie de son comportement à un autre objet. Ce modèle est utilisé pour déléguer des tâches à un autre objet. Il est utilisé pour réutiliser le code. Il est un moyen de réutiliser le code en utilisant la composition.

- Un thread principal
- Des threads travailleurs

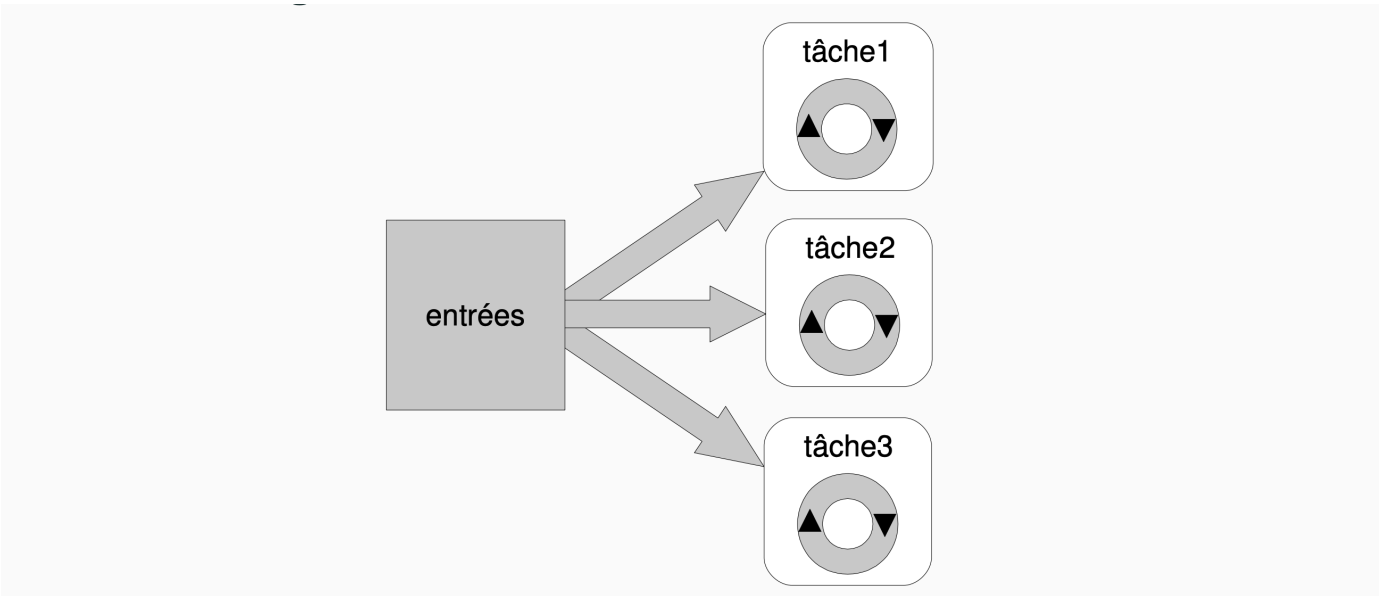



```
void tachePatron(void *ptr) {
  boucle infinie {
    attend une requête
      switch (requete) {
        case requeteX: startThread( ... tacheX); break;
        case requeteY: startThread( ... tacheY); break;
        ...
      }
    }
  }
void tacheX(void *ptr) {
  exécuter le travail demandé, puis se terminer
}
void tacheY(void *ptr) {
  exécuter le travail demandé, puis se terminer
}
```

1.2. Modèle pair (peer model)

Le modèle pair est un modèle de conception qui permet à un objet de partager une partie de son comportement avec un autre objet. Ce modèle est utilisé pour partager des tâches avec un autre objet. Il est utilisé pour partager des tâches entre deux objets. Il est un moyen de partager des tâches entre deux objets.

- Pas de thread principal
- Tous égaux
- Chacun s'arrange avec ses entrées/sorties




```
int main() {
  startThread( ... tache1);
  startThread( ... tache2);
  ...
  signale aux threads qu'ils peuvent commencer à travailler
}
void tache1() {
  attend le signal de commencement
  effectue le traitement, et synchronise avec les autres threads
  si nécessaire
}
void tache2() {
  attend le signal de commencement
  effectue le traitement, et synchronise avec les autres threads
  si nécessaire
}
```

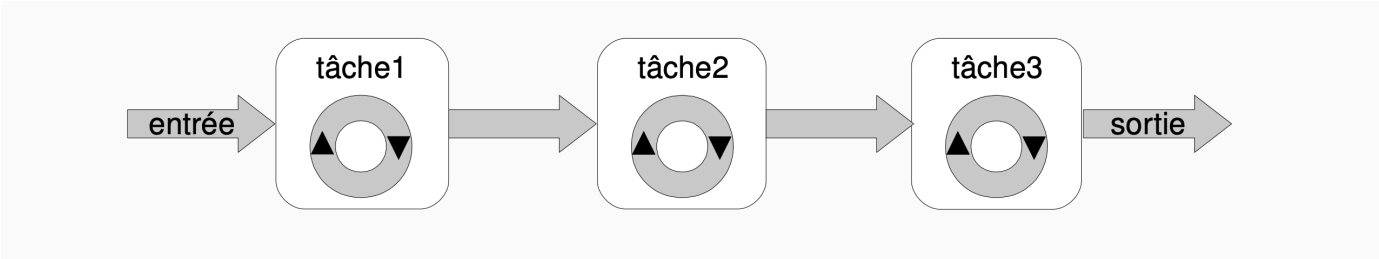
1.3. Modèle pipeline (pipeline model)

Le modèle pipeline est un modèle de conception qui permet à un objet de transmettre une partie de son comportement à un autre objet. Ce modèle est utilisé pour transmettre des tâches à un autre objet.

- Appliqué lorsque:
- L'application traite une longue chaîne d'entrée;
- Le traitement à effectuer sur ces entrées peut être décomposé en sous-tâches

(étages de pipeline) au travers desquelles chaque donnée d'entrée doit passer;

- Chaque étage peut traiter une donnée différente à chaque instant.
- Un thread attend les données du précédent
- Et les transmet ensuite au suivant




```
void etage1() {
  boucle infinie {
    récupérer une entrée du programme
    traiter cette donnée
    passer le résultat à l'étage suivant
  }
}
void etage2() {
  boucle infinie {
    récupérer une donnée de l'étage précédent
    traiter cette donnée
    passer le résultat à l'étage suivant
  }
}
void etageN() {
  boucle infinie {
    récupérer une donnée de l'étage précédent
    traiter cette donnée
    passer le résultat en sortie du programme
  }
}
```

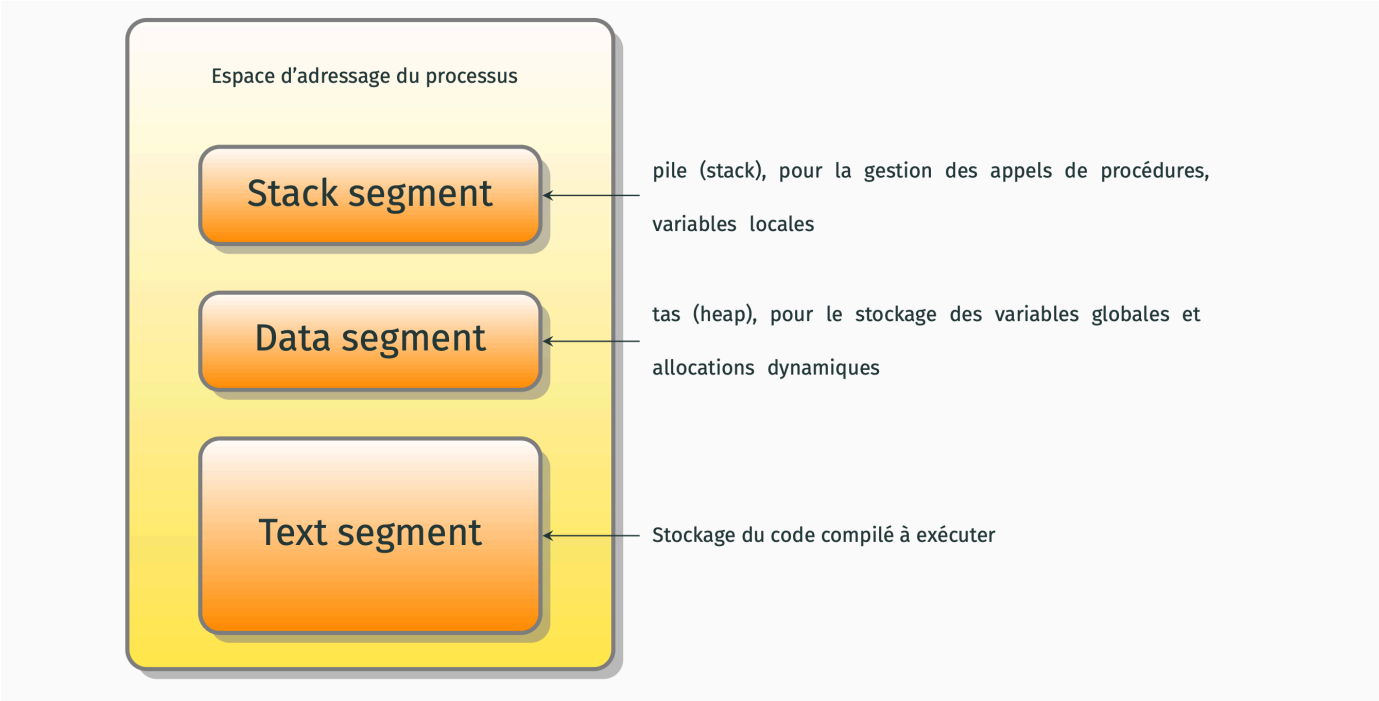
2. Anatomie d’un processus

2.1. 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.

2.2. Adressage d’un processus



2.3. 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.

3. 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

3.1. 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

3.2. 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