Synchronisation et workflow

jeudi 5 septembre 2024 13:12

Cas 1 : concurrence intra (ou parallélisme)
Cas 2 concurrence inter (répartis)

Outils :
Thread (processus)
Partager un objet de manière sécurisé (synchronized) pour garantir que rien ne change dans l'objet

On veut des mécanismes plus souples qui vont permettre de ne pas verrouiller tout un objet

Mécanismes de verrouillage/déverrouillage :

- wait()
- notify() (ou notifyAll())

Quand on a un wait(), il faut forcément avoir un autre processus, sinon l'objet ne sera jamais réveillé

Quand il y a une notification, il faut laisser un jeton pour signaler qu'il y a eu un jeton (c'est un avis de passage)

Principe de la section critique SC) : verrouiller juste une partie du code au lieu de bloquer toute une méthode
=> utilisation de jeton (ou sémaphore)
=> c'est ce jeton qui permettra de rentrer dans la section critique
=> P() permet d'entrer dans la SC (correspond au wait())
=> V() permet de sortir de la SC (correspond au notify)

Il faut vérifier si le jeton est disponible avant d'entrer dans la SC
On utilise if(n==0) seulement si il y a deux processus, sinon on utilise while(n==0) pour vérifier que le jeton est toujours disponible après la notification
=> n==0 signifie que le jeton n'est pas utilisé
Pareil, s'il y a deux processus on utilise notify(), sinon notifyAll()

Outils :

- Notify()
- notifyAll()
- Wait()

Méthodes :

- Méthode pour verrouiller un processus
- Méthode pour notifier le(s) processus

Pour chaque processus P, pour chaque action Y de P tq X->Y (X action de P') : on applique la méthode de verrouillage
Pour chaque processus P, pour chaque action X de P tq X->Y (Y action de P') : on applique la méthode qui notifie
La -> signifie que Y dépend de X
On utilise un contrôleur par flèche

La méthode join() permet d'attendre qu'un processus termine.
Donc dans le main(…), si on fait P1.join(), tant que P1 n'a pas terminé, on exécute pas le suite de ce qui est dans le main.