Travaux Dirigés n°3

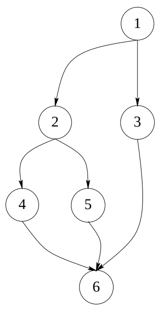
Cours de Parallélisme

—Maîtrise Informatique—

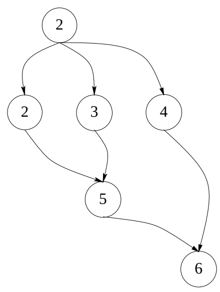
Ce TD porte sur l'expression de la concurrence avec une introduction au problème d'ordonnancement de tâches.

Le problème de l'ordonnancement de tâches peut se formuler ainsi : ordonnancer en temps minimum un ensemble de tâches soumises à des contraintes de précédence fixées, sur un ensemble de processeurs identiques.

Lorsque l'on cherche à résoudre des problèmes d'ordonnancement, on représente les contraintes de précédence imposées pour les différentes tâches, par un graphe de précédence dont les sommets sont les tâches mises en concurrence. Les arêtes orientées du graphe spécifient l'ordre d'exécution des différentes tâches, le graphe étant par nature acyclique.



▸ **Exercice 1.** (expression de la concurrence) *On considère le graphe de précédence suivant :*



1. *Spécifier une exécution parallèle de ce graphe en utilisant les constructions* `parbegin` *et* `parend`.
2. *Spécifier une exécution parallèle de ce graphe en utilisant les constructions* `fork` *et* `join`.

▸ **Exercice 2.** (expression de la concurrence) *Partant du graphe de précédence, on exprime la concurrence à l'aide des primitives* `fork` *et* `join` *ou du formalisme* `parbegin` *et* `parend`. *On considère la portion de programme suivante :*

```
 1.  a=b-c+h          2.  b=d-e
 3.  h=c-i            4.  a=c+e
 5.  h=2*h            6.  d=d*3
 7.  i=i-c            8.  b=e
 9.  e=b-d           10.  i=h-c+a-e
```

et on définit les ensembles $\mathcal{R}(i)$ et $\mathcal{W}(i)$ d'une instruction i comme étant :

① *$\mathcal{R}(i)$ est l'ensemble des variables dont les valeurs sont utilisées par i ;*

② *$\mathcal{W}(i)$ est l'ensemble des variables dont les valeurs sont modifiées par i.*

1. *Rappeler les conditions que Bernstein a données sur les ensembles $\mathcal{R}$ et $\mathcal{W}$ pour que deux instructions i_1 et i_2 soient en concurrence.*
2. *Donner les ensembles $\mathcal{R}$ et $\mathcal{W}$ pour chacune des lignes du programme.*
3. *En déduire le graphe de précédence de cette portion de programme.*
4. *Spécifier une exécution parallèle de ce graphe en utilisant les constructions* `fork` *et* `join`.
5. *Pourquoi l'opération de* `join` *doit-elle être atomique ?*
6. *Spécifier une exécution parallèle de ce graphe en utilisant les constructions* `parbegin` *et* `parend`.

▸ **Exercice 3.** (utilisation d'un sémaphore) *On considère maintenant une modification de la portion de programme précédente, modification visant à changer l'instruction* `6` *afin de modifier le graphe de précédence :*

```
 1.  a=b-c+h          2.  b=d-e
 3.  h=c-i            4.  a=c+e
 5.  h=2*h            6.  d=i-d
 7.  i=i-c            8.  b=e
 9.  e=b-d           10.  i=h-c+a-e
```

1. *Peut-on paralléliser cette portion de programme en utilisant uniquement les constructions* `parbegin` *et* `parend` *?*
2. *Spécifier une exécution parallèle du programme en utilisant les constructions* `fork` *et* `join`.
3. *Utiliser un sémaphore et la construction* `parbegin` *et* `parend` *pour spécifier une exécution parallèle de ce programme.*