TD n° 6 : Sémaphores et moniteurs

Exercice 1. On considère un processus séquentiel $\mathtt{p}_1$, qui lit et écrit alternativement dans un fichier et qui à tout moment, peut exécuter un nombre quelconque de fois l'action `attendre`. Pour des raisons de simplicité, on suppose qu'une action `lire, écrire` ou `attendre` dure une unité de temps.

1. Représenter ce processus par un automate fini sur l'alphabet {`lire, écrire, attendre`}.

On considère ensuite un deuxième processus séquentiel $\mathtt{p}_2$ identique au premier.

2. En supposant que les deux processus sont indépendants, représenter par un automate fini le système $\mathtt{p}_1 \| \mathtt{p}_2$.

Enfin, on synchronise les deux processus s'exécutant en parallèle par la contrainte que deux écritures simultanées, ou une lecture et une écriture simultanées sont interdites et que deux attentes simultanées sont aussi interdites.

3. Représenter les comportements de $\mathtt{p}_1 \| \mathtt{p}_2$ qui satisfont cette contrainte de synchronisation par un troisième automate d'états fini.

Exercice 2. Écrire un programme qui calcule l'expression

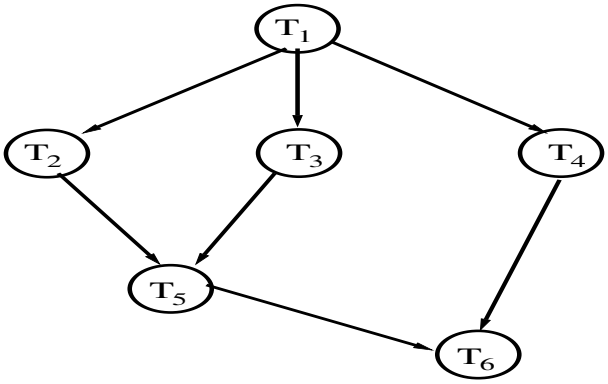
$$y := 2 * ((a + b) / (c - d) + e * f) + (a + b) * (c - d)$$

en évaluant en parallèle des sous-expressions contenant une opération et en utilisant les instructions `parbegin` et `parend`. On pourra utiliser un nombre quelconque de variables intermédiaires.

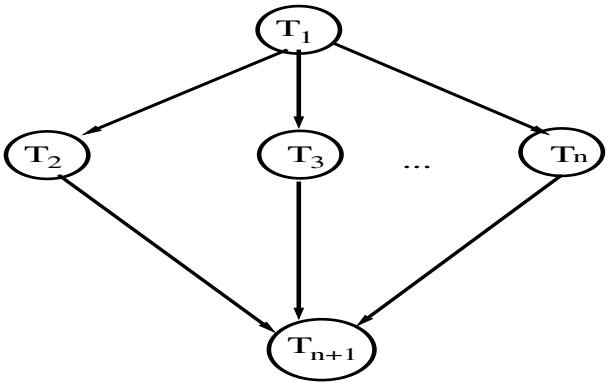
Exercice 3. Écrire un programme parallèle qui fait intervenir 3 processus pour multiplier des matrices 3×3.

Exercice 4.

1. Implanter le graphe de précédence suivant en utilisant 6 chaînes de tâches s'exécutant en parallèle, à l'aide de la structure `parbegin/parend` et de 3 sémaphores.



2. Implanter le graphe de précédence suivant en utilisant $n+1$ chaînes de tâches s'exécutant en parallèle, à l'aide de la structure `parbegin/parend` et de 2 sémaphores.



Exercice 5. On considère une situation pour laquelle deux processus p_1 et p_2 utilisent tous deux deux ressources critiques R_1 et R_2 selon le code suivant :

Processus p_1	Processus p_2
début	début
P(SR_1) ;	P(SR_2) ;
P(SR_2) ;	P(SR_1) ;
Utilisation de R_1 et R_2 ;	Utilisation de R_1 et R_2 ;
V(SR_2) ;	V(SR_1) ;
V(SR_1) ;	V(SR_2) ;
fin	fin

R_1 et R_2 sont gérées par le biais de deux sémaphores SR_1 et SR_2 initialisés à 1.

Initialement, les deux ressources R1 et R2 sont libres.

1. Montrer que cette solution présente une situation d'interblocage.
2. Peut-on y remédier en modifiant a minima le code donné ?

Exercice 6. N ($N \geq 5$) philosophes sont assis autour d'une table ronde. Chaque philosophe a devant lui un plat de spaghetti tellement glissants qu'il lui faut 2 fourchettes pour pouvoir les manger. Une fourchette sépare 2 assiettes consécutives :

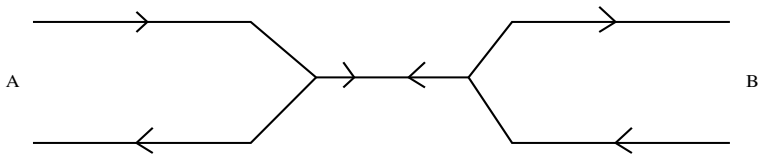
Un philosophe passe son temps à penser, et quand il a faim, il tente de s'emparer des fourchettes de part et d'autre de son assiette pour manger pendant un temps déterminé et fini. S'il obtient les 2 fourchettes, il mange pendant un certain temps, puis repose les fourchettes et se remet à penser. Un tel comportement serait produit par le programme suivant :

```
Philosophe(i) :
   while(true) {
      penser();
      prendre_fourchettes(i);
      manger();
      poser_fourchettes(i);
   }
}
```

1. Montrer que cette situation peut mener à un interblocage ou à un phénomène de famine.
2. Proposer une solution qui permet le maximum de parallélisme quelque soit le nombre N de philosophes, $N \geq 5$.
   - On utilisera 2 fonctions `Gauche` et `Droite` qui désignent les voisins gauche et droit d'un philosophe quelconque. Ex. si `i`$= 2$, `Gauche(i) = 1` et `Droit(i) = 3`.
   - On suppose que chaque philosophe peut être dans l'état `penser, faim` ou `manger`.
3. Cette solution garantit l'absence de blocage mais ne préserve pas de la famine. Proposez une situation de fonctionnement dans laquelle se produit un phénomène de famine pour l'un des philosophes, (prendre $N = 5$).
4. Améliorer la solution en gérant une file d'attente FIFO, qui assure que les philosophes sont servis dans l'ordre :
   - lorsqu'un philosophe veut prendre les fourchettes pour manger, et si la file n'est pas vide, il s'y place en queue ;
   - si la file est vide, il tente de prendre ses fourchettes. S'il y parvient, il mange, sinon il est placé dans la file ;
   - lorsqu'il pose ses fourchettes, on regarde si cela libère le premier de la file.

   Quel inconvénient cette solution présente-t-elle ?

Exercice 7. Une ligne de chemin de fer reliant deux villes `A` et `B` comporte une section à voie unique.



On représente les trains par des processus selon le schéma ci-après :

```
trains A -> B
...
voie_unique.entrée_ouest
<trajet voie unique>
voie_unique.sortie_est
...
```

```
trains B -> A
          ...
 voie_unique.voie_est
 <trajet voie unique>
 voie_unique.sortie_ouest
  ...
```

Le rôle du moniteur `voie_unique` est de garantir que tous les trains engagés à un instant donné sur la voie unique circulent dans le même sens.

1. Écrire le programme du moniteur `voie_unique` en supposant que le nombre de trains présents sur la voie unique n'est pas limité.
   Vous pourrez utiliser les variables `nt_BversA` (respec. `nt_AversB`) pour comptabiliser le nombre de trains circulant de `B` vers `A` (respec. de `A` vers `B`).
   Il est demandé d'écrire les procédures `entrée_ouest, sortie_ouest, entrée_est` et `sortie_est`.
2. Même question, avec une limite fixe `N` au nombre de trains présents sur la voie unique.