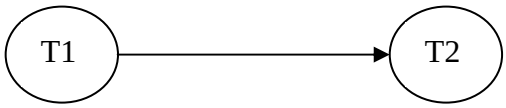
TD : synchronisation de processus par des Sémaphores

Graphe des dépendances –
Un nœud est une tâche. Une flèche entre tâches indique un ordre d'exécution.
Exemple : la flèche entre T1 et T2 indique que l'exécution de la tâche T2 doit débuter uniquement après que l'exécution de la tâche T1 soit terminée.

Graphe des dépendances 1 :



On a une seule exécution possible : `dT1fT1dT2fT2`.
Notation : `di` est le début de la tâche `i` et `fi` est la fin de la tâche `i`.

Exo1

Complétez le code pour réaliser le graphe des dépendances 1. Ce code correspond à la création en parallèle de deux processus indépendants ; un processus exécute le programme ProgA et l'autre le programme ProgB.
Vous devez utiliser **un** sémaphore (n'oubliez pas son initialisation).

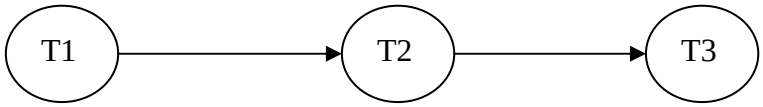
```
Debut

      Parbegin ProgA ; ProgB ; Parend
Fin
```

ProgA:	**ProgB:**
Debut T1 ; **Fin**	**Debut** T2 ; **Fin**

Exo2

Complétez le code pour réaliser le graphe des dépendances 2.
Vous pouvez utiliser deux sémaphores (n'oubliez pas leur initialisation). Une solution utilisant un seul sémaphore existe.

Graphe des dépendances 2 :




```
Debut

      Parbegin ProgA ; ProgB ; Parend
Fin
```

ProgA:	**ProgB:**
Debut T1 ; T3 ; **Fin**	**Debut** T2 ; **Fin**

Inter-blocage - (deadlock en anglais, appelé aussi étreinte fatale) est un phénomène qui peut survenir en programmation concurrente. L'inter-blocage se produit lorsqu'une suite de processus se bloque mutuellement. Les processus bloqués le sont définitivement.
Exemple : le processus P1 attend le processus P2 ; le processus P2 attend le processus P1.

Exo3

1. Donnez le graphe des dépendances correspondant au code Codea.
2. L'exécution du code Code4a provoque-t-il l inter-blocage des deux processus ?

Codea :

```
Debut
     Sémaphore S1, S2;
     Init(S1, 0); Init(S2,0);
     Parbegin ProgA ; ProgB ; Parend
Fin
```

ProgA:	**ProgB:**
Debut P(S1) ; TA ; V(S2) ; **Fin**	**Debut** P(S2) ; TB ; V(S1) ; **Fin**

3. Donnez le graphe des dépendances correspondant au code Codeb.
4. L'exécution du code Codeb provoque-t-il l inter-blocage des deux processus ?

Codeb :

```
Debut
     Sémaphore S1, S2;
     Init(S1, 0); Init(S2,1);
     Parbegin ProgA ; ProgB ; Parend
Fin
```

ProgA:	**ProgB:**
Debut P(S1) ; TA ; V(S2) ; **Fin**	**Debut** P(S2) ; TB ; V(S1) ; **Fin**

Section critique – Une section critique est une portion de code qui doit être exécutée par au plus un processus à un instant donné. Une section critique peut être protégée par un sémaphore (généralement nommé Mutex).

Exo4

Code C1 :

```
Debut
      Sémaphore Mutex ;

      Parbegin ProgA ; ProgA ; Parend
Fin
```

```
ProgA:
      Debut
            T1;
            T_SC;
            T2;
      Fin
```

1. Utilisez le sémaphore `Mutex` pour protéger la section critique `T_SC` dans le code C1.
2. Utilisez le sémaphore `Mutex` pour protéger la section critique `T_SC` dans le code C2.

Code C2 :

```
Debut
      Sémaphore Mutex ;
      Parbegin ProgA ; ProgA ; ProgA ; ProgA ; ProgA ; ProgA ; Parend
Fin
```

```
ProgA:
          Debut
                T1;
                T_SC;
                T2;
          Fin
```

Exo5a Soient 2 processus séquentiels qui exécutent le programme suivant :

```
Debut

      Parbegin ProgA ; ProgB ; Parend
Fin
```

```
ProgA:
  Debut
     Répéter Tant que « Vrai »
         T1 ;
     Fin Tant que
  Fin
```

```
ProgB:
  Debut
     Répéter Tant que « Vrai »
         T2 ;
     Fin Tant que
  Fin
```

1. Utilisez un sémaphore pour synchroniser les 2 processus de telle manière que l'exécution de la tâche T1 ne soit jamais simultanée avec l'exécution de la tâche T2.

Exo5.b Soient 2 processus séquentiels qui exécutent le programme suivant :

```
Debut

      Parbegin ProgA ; ProgB ; Parend
Fin
```

ProgA:	ProgB:
Debut **Répéter Tant que « Vrai »** T1 ; **Fin Tant que** **Fin**	**Debut** **Répéter Tant que « Vrai »** T2 ; **Fin Tant que** **Fin**

1. Utilisez deux sémaphores pour synchroniser les 2 processus de telle manière que les tâches se déroulent toujours dans l'ordre : T1T2T1T2T1T2...

Exo5.c Soient 2 processus séquentiels qui exécutent le programme suivant :

```
Debut

      Parbegin ProgA ; ProgB ; Parend
Fin
```

ProgA:	ProgB:
Debut **Répéter Tant que « Vrai »** T1 ; **Fin Tant que** **Fin**	**Debut** **Répéter Tant que « Vrai »** T2 ; **Fin Tant que** **Fin**

1. Utilisez deux sémaphores pour synchroniser les 2 processus de telle manière que les tâches se déroulent toujours dans l'ordre : T1T2T2T1T2T2T1T2T2...

Exo5.d Soient 2 processus séquentiels qui exécutent le programme suivant :

```
Debut

      Parbegin ProgA ; ProgB ; Parend
Fin
```

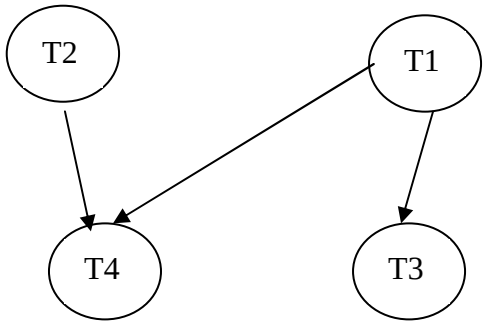
ProgA:	ProgB:
Debut **Répéter Tant que « Vrai »** T1 ; **Fin Tant que** **Fin**	**Debut** **Répéter Tant que « Vrai »** T2 ; **Fin Tant que** **Fin**

1. Utilisez deux sémaphores pour synchroniser les 2 processus de telle manière que les tâches se déroulent toujours dans l'ordre : T2T2T1T1T1T2T2T1T1T1T2T2T1T1T1...

Exo6

1. Ecrivez le code implémentant le graphe des dépendances 3 en utilisant 4 processus indépendants chacun exécutant une tâche et **deux** sémaphores

Graphe des dépendances 3 :



2. Les exécutions suivantes respectent-elles le graphe des dépendances 3 ou non ?

```
dT1fT1dT2fT2fT3dT3fT4dT4          dT1dT2fT2fT1dT3dT4fT3fT4
dT1dT2fT1fT2dT3dT4fT3fT4          dT1dT2fT2fT1dT3fT3dT4fT4
dT1dT2fT2dT3fT1dT4fT4fT3          dT1dT2fT2dT3fT3fT1dT4fT4
dT1dT2fT1dT3fT2fT3dT4fT4          dT2fT2dT1fT1dT4fT4dT3fT3
```

Exo7

Il y a 3 processus A, B et C. Les processus A, B et C exécutent séquentiellement 3 tâches. Le processus A exécute les tâches A1, A2 et A3. Le processus B exécute 3 tâches dans l'ordre suivant :B1, B2 et B3. …

1. Donnez le graphe des dépendances.

Nous considérons les dépendances suivantes :

- la tâche A2 doit être exécutée après la tâche C2 ;
- la tâche A3 doit être exécutée avant la tâche B3 ;
- tâche C3 doit être exécutée avant la tâche A3 ;
- la tâche B2 doit être exécutée avant la tâche C1 et avant A1.

2. Donnez le nouveau graphe des dépendances
3. Retirez dans le graphe des dépendances une dépendance qui est redondante.
4. Donnez deux exécutions respectant les dépendances.
5. Complétez le code suivant pour réaliser les dépendances en utilisant des sémaphores (3 sont suffisants).

```
Debut

      Parbegin ProgA ; ProgB ; ProgC ; Parend
Fin
```

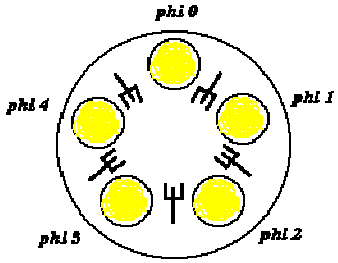
ProgA:	**ProgB:**	**ProgC:**
Debut TA1 ; TA2 ; TA3 ; **Fin**	**Debut** TB1 ; TB2 ; TB3 ; **Fin**	**Debut** TC1 ; TC2 ; TC3 ; **Fin**

6. Est-il possible d'ajouter la condition « la tâche B3 s'exécute avant la tâche C3 » ? Si oui donnez une exécution respectant les dépendances.
7. Comment se traduit l'inter-blocage dans le graphe des dépendances ?

Problème de la famine : il y a possibilité de famine si un processus peut attendre indéfiniment d'obtenir une ressource alors que d'autres processus prennent la ressource infiniment souvent. Autrement dit, un processus est privé d'une ressource aux profits d'autres processus (ou à cause d'autre processus)

Exo8

Problème ultra classique des philosophes " mangeurs de spaghettis " : 5 philosophes autour d'une table circulaire (voir l'image jointe) passent leur temps à penser, puis à manger, pour penser à nouveau avant de manger … . Un philosophe a besoin pour manger (des spaghettis) de ses deux fourchettes : la fourchette à sa droite et la fourchette à sa gauche. Un philosophe ne peut manger que si ces deux voisins immédiats ne mangent pas. A la fin de son repas, un philosophe repose les deux fourchettes.



Code Naïf :

```
Debut
      Semaphore fourchette[5];
      i := 0 ; Tant que i < 5 Init(fourchette[i], 1); i++ ; fin Tant que
      Parbegin
            Philosophe[0] ; Philosophe[1] ;  Philosophe[2] ; Philosophe[3] ;  Philosophe[4] ;
      Parend
Fin
```

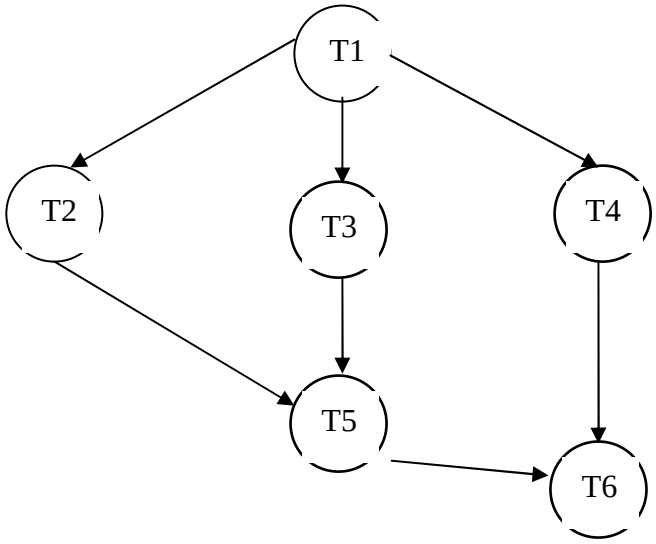
Philosophe[i]:

```
Debut
    Répéter Tant que « Vrai »
          Penser ;
          S'assoir à sa place ;
          P(fourchette[i]) ;
          P(fourchette[(i+1)%5]) ;
          Manger;
          V(fourchette[i]) ;
          V(fourchette[(i+1)%5]) ;
          quitter sa place;
     Fin Tant que
 Fin
```

1. Expliquez le code naïf.
2. L'exécution du code naïf peut-telle conduire à un inter-blocage des Philosophes ? Si oui, proposez une telle exécution.
3. Complétez le code naïf pour avoir une solution sans inter-blocage. Indice: si les 5 philosophes ont chacun une fourchette, combien de philosophes peuvent manger ? Si au plus 4 philosophes ont chacun une fourchette, combien de philosophes peuvent manger ?
4. Votre solution évite-t-elle la famine ? Indice : essayez de proposez une exécution infinie où le Philosophe[0] ne peut jamais manger.

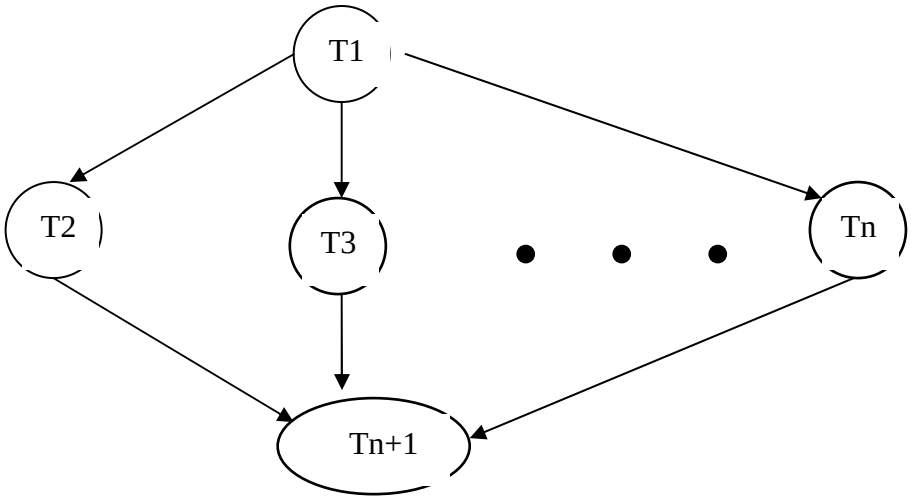
Exo9

Implémentez le graphe de précédence suivant par 6 processus indépendants chacun exécutant une tâche et des sémaphores. Une solution à deux sémaphores existe.



Exo10

Implémentez le graphe de précédence suivant par n+1 processus indépendants chacun exécutant une tâche et des sémaphores. Une solution à 1 sémaphore existe.



Exo11

Une route joignant la France à l'Espagne utilise un tunnel à voie unique. En utilisant des sémaphores, vous allez synchroniser les véhicules de manière qu'à tout instant, tous les véhicules circulent dans le même sens. Votre solution doit

- permettre le passage d'un véhicule venant de France ou d'Espagne dans le cas où le pont est vide,
- autoriser immédiatement le passage d'un véhicule venant de France si les véhicules sur le pont viennent d'Espagne (idem pour la France),
- permettre l'alternance : autoriser le passage d'un véhicule en attente, lorsque le pont est de nouveau vide. Par exemple, dans le cas où un véhicule venant de France est attente car il y a des véhicules venant d'Espagne sur le pont. Le Véhicule venant de France doit pouvoir passer une
- fois que le pont est vide.

Définissez les trois composantes du programme :

1. Variables globales utilisées (lues et modifiées) par des procédures communes à l'ensemble des voitures. N'oubliez pas de déterminer la valeur initiale de chaque variable ; initialisez les sémaphores.
2. La procédure d'entrée sur le pont par les voitures venant d'Espagne (`entreeParEspagne`) et la a procédure d'entrée sur le pont par les voitures venant de France (`entreeParFrance`).
3. La procédure de sortie du pont par les voitures venant d'Espagne (`sortieEnFrance`) et la a procédure d'entré sur le pont par les voitures venant de France (`sortieEnEspagne`).

Exo12

Un processus **producteur** ou plusieurs produisent des "objets " qui seront stockés dans une remise. Un objet occupe un emplacement dans la remise.

Un processus **consommateur** ou plusieurs utilisent les "objets" après les avoir retirés de la remise. Un fois un objet utilisé, l'emplacement occupé par cet objet dans la remise est de nouveau disponible. Si la remise est vide, un consommateur ayant besoin d'un objet est bloqué jusqu'au dépôt d'un objet dans la remise.

1. Proposez une solution au problème du producteur/consommateur utilisant les sémaphores dans le cas où la remise est de taille infinie.
2. Les objets produits sont stockés dans une remise de taille bornée (par N). La remise peut contenir au plus N objets. Si la remise est pleine, un producteur ne peut plus déposer d'objet, il doit attendre qu'un emplacement se libère.