

EXERCICE 1 – Somme des éléments d'une liste

Pour calculer la somme des éléments d'une liste **L** à **N** entiers on utilise deux processus qui s'exécutent en parallèle. Le processus **P1** parcourt les éléments d'indice impair et le processus **P2** parcourt les éléments d'indice pair. Le processus père lance les 2 processus **P1** et **P2**. A la fin d'exécution de **P1** et **P2**, le processus père affiche le résultat stocké dans la variable partagée.

```
#Processus P1
i = 1
SommImpairs = 0
Tant que (i ≤ N)
faire
    SommImpairs = SommImpairs + L[i]
    i = i + 2
FinTantque
Somme = Somme + SommImpairs
```

```
#Processus P2
i = 0
SommePairs = 0
Tant que (i ≤ N)
faire
    SommePairs := SommePairs + L[i]
    i = i + 2
FinTantque
Somme = Somme + SommePairs
```

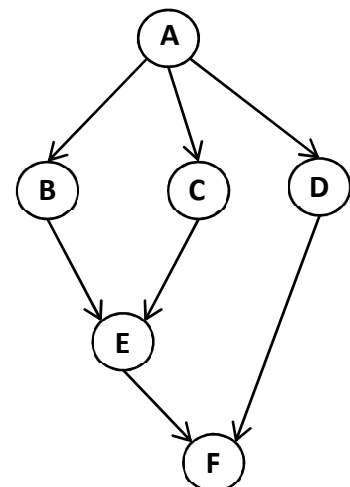
EXERCICE 2 - Précédence de tâches

On considère un ensemble de six tâches séquentielles : **A**, **B**, **C**, **D**, **E** et **F** avec les contraintes suivantes :

- La tâche **A** doit précéder les tâches **B**, **C**, **D** [ces trois tâches ne peuvent démarrer qu'à la fin d'exécution de la tâche **A**].
- Les tâches **B** et **C** doivent précéder la tâche **E**.
- Les tâches **D** et **E** doivent précéder la tâche **F**.

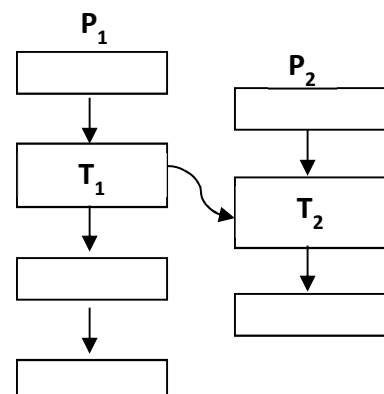
Réaliser la synchronisation de ces tâches en utilisant les **sémaphores**.

Chaque tâche est exécutée par un processus. Chaque processus se contente d'afficher son état.



EXERCICE 3 - Précédence de tâches

Un système est composé de deux tâches **T1** et **T2** soumises à la contrainte de précédence **T1 < T2** (**T2 doit être exécutée après T1**). Ces deux tâches appartiennent à deux processus **P1** et **P2** qu'on doit synchroniser ⇒ **Le processus P2 doit retarder l'exécution de la tâche T2 jusqu'à ce que le premier processus P1 termine sa tâche T1**.



EXERCICE 4 – Rendez-vous de deux processus

On considère deux processus producteurs P_1 et P_2 qui produisent des messages (nombres entiers tirés aléatoirement) et les déposent dans deux queues (files de messages vues en cours) Q_1 et Q_2 respectivement (Q_i pour P_i , $i=1,2$). Deux processus consommateurs C_1 et C_2 consomment les messages : C_1 ceux déposés dans Q_1 , C_2 ceux déposés dans Q_2 ; avec la contrainte que lorsqu'un processus C_i ($i=1,2$) consomme un message, il attendra que l'autre processus C_j ($j=3-i$) ait consommé un message lui aussi pour continuer à consommer un autre message (**Rendez-vous entre C_1 et C_2 après chaque consommation**). Synchroniser ces processus en utilisant les sémaphores.

EXERCICE 5 – Rendez-vous à 2

Deux processus P_1 et P_2 souhaitent établir un rendez-vous avant l'exécution de la fonction `rdv1()` pour l'un et `rdv2()` pour l'autre. En utilisant les sémaphores, écrire les scripts P_1 et P_2 permettant d'établir ce rendez-vous.

EXERCICE 6 - Le Rendez-vous – Exercice supplémentaire

Pour réaliser un mécanisme de communication un à plusieurs, on utilise un ensemble de processus composé d'émetteurs et de récepteurs. Un émetteur produit un message (à simuler par l'affichage d'un message sur l'écran) et se met en attente jusqu'à ce qu'il y ait $N-1$ récepteurs au rendez-vous. Un récepteur lancé attend l'émission et l'arrivée des autres récepteurs. Prenons l'exemple d'un rendez-vous 1 à 2 - On exécute ce script avec 4 récepteurs et 2 émetteurs :

\$> python3 exercice6.py R E E R R R

Voici un exemple des affichages de ce script :

```
Le processus 1 récepteur se met en attente
Le processus 2 émetteur produit un message et se met en attente
Le processus 3 émetteur produit un message et se met en attente
Le processus 4 récepteur débloque les processus 1 et 2 et consomme le le message.
Le processus 5 récepteur au Rendez-vous avec le processus 3 émetteur - Attente
Le processus 6 récepteur débloque les processus 3 et 5.
```

<https://docs.python.org/fr/3/library/multiprocessing.html>