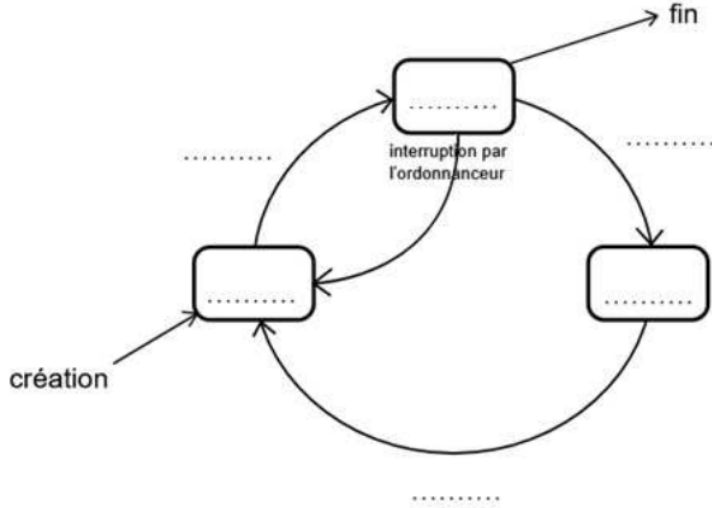


Exercice n°1 _____

1. Qu'est ce qu'un processus ?
2. Qu'est ce qu'un ordonnanceur ?
3. Compléter le schéma ci-dessous en faisant apparaître les différents états que peut prendre un processus depuis sa création jusqu'à sa terminaison. Les termes désignant les changements d'état devront être apparents.



4. Donner un exemple de situation qui contraint un processus à passer de l'état élu et l'état bloqué.

Exercice n°2

Sept processus P_i sont dans la situation suivante par rapport aux ressources R_i :

- P_1 a obtenu R_1 et demande R_2
- P_2 demande R_3
- P_3 demande R_2
- P_4 a obtenu R_2 et R_4 puis demande R_3
- P_5 a obtenu R_3 et demande R_5
- P_6 a obtenu R_6 et demande R_2
- P_7 a obtenu R_5 et demande R_2 .

On voudrait savoir s'il y a interblocage.

1. Construire un graphe orienté où les sommets sont les processus et les ressources, et où :
 - La présence de l'arc $R_i \longrightarrow P_j$ signifie que le processus P_j a obtenu la ressource R_i .
 - La présence de l'arc $P_j \longrightarrow R_i$ signifie que le processus P_j demande la ressource R_i .
2. Y-a-t-il interblocage ? si oui précisez le (ou les) processus concerné(s).

Exercice n°3 _____

On considère les cinq exécutions de processus suivants (la durée est exprimée en cycle CPU) :

Processus	Date d'arrivée	Durée
P1	0	7
P2	1	4
P3	2	2
P4	3	2
P5	4	1

PARTIE A : Ordonnancement

1. Compléter les diagrammes de Gantt (ci-dessous) à l'aide des algorithmes d'ordonnancement FIFO (Premier arrivé - Premier servi), PCTER (plus court temps d'exécution restant) et Tourniquet (avec un quantum de 2).

FIFO

[illegible]

PCTER

P1															
----	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

Tourniquet (quantum : 2)

P1															
----	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

2. Pour chaque algorithme, calculer le temps de traitement moyen.

Aide : Le temps de traitement d'un processus (appelé aussi temps de séjour, d'exécution ou de rotation) est la différence entre la date de fin d'exécution et la date d'arrivée de ce processus.

PARTIE B : Modélisation du tourniquet

Nous allons implémenter l'algorithme d'ordonnancement du tourniquet dont le fonctionnement est résumé ci-dessous :

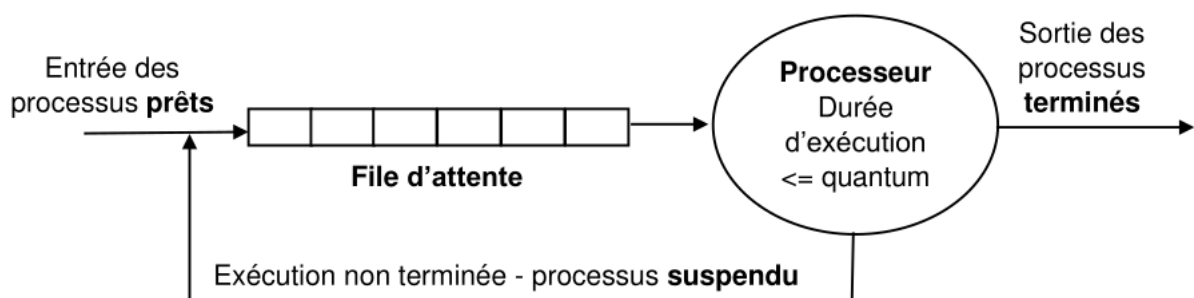


Schéma d'ordonnancement du tourniquet

- Les processus prêts à être exécutés sont placés dans une file d'attente selon leur ordre d'arrivée ;
- L'ordonnanceur alloue le processeur à chaque processus de la file d'attente un même nombre de cycles CPU, appelé quantum ;
- Si le processus n'est pas terminé au bout de ce temps, son exécution est suspendue et il est mis à la fin de la file d'attente ;
- Si le processus est terminé, il sort définitivement de la file d'attente.

Pour représenter le processus, on utilise une classe `Processus` qui possède les variables d'instances `PID` (l'identifiant du processus), `date_arrivee`, `temps_utilisation_CPU` sur le CPU et le temps nécessaire à son exécution `temps_necessaire`.

1. Compléter le constructeur de la classe `Processus` :

```

class Processus:

    .....(self, PID , ..... , temps_necessaire):

        self.PID = ...

        self..... = .....

        self.temps_utilisation_CPU = 0

        .....
  
```

2. Le processus P1 est :

```
P1 = Processus(PID=1, date_arrivee=0, temps_necessaire=7)
```

En utilisant le tableau au début de l'exercice, créer les processus `P2`, `P3`, `P4` et `P5` : quatre instances de la classe `Processus`.

3. Compléter les trois méthodes suivantes de la classe `Processus` :

```

def execute_un_cycle(self):
    """Met à jour le temps_utilisation_CPU après l'exécution d'un cycle."""

    .....
  
```

```
def est_termine(self):
    """Renvoie True si le processus est terminé, False sinon,
    en se basant sur temps_utilisation_CPU."""
```

```
.....
```

4. Compléter la fonction `is_finish` qui tester si tous les processus de la liste : `liste_process` sont terminés

```
liste_process=[P1,P2,P3,P4,P5]
def is_finish(liste_process):
    for p in liste_process:
        .....
        .....

    return .....
```

5. Compléter la fonction `tourniquet` qui renvoie la liste des PID des processus dans l'ordre de leur exécution.

```
def tourniquet(liste_process, quantum):
    temps=0
    ordre_execution = []
    liste_attente = []
    # Tant que les processus ne sont pas terminés

    while .....:

        if liste_attente == []:

            temps+=1

            for p in liste_process:

                # tester si la date_arrivee est inférieur au temps

                if p not in liste_attente and .....:

                    ..... # Ajouter p à liste_attente

            else:

                processus = liste_attente.pop(0)

                compteur_tourniquet = 0

                while ..... and .....:

                    ordre_execution.append(.....)

                    ..... # On execute un cycle de processus

                    compteur_tourniquet = .....

                    temps+=1

                    for p in liste_process:

                        if p != processus and p not in liste_attente and p.date_arrivee<=temps:

                            liste_attente.append(p)

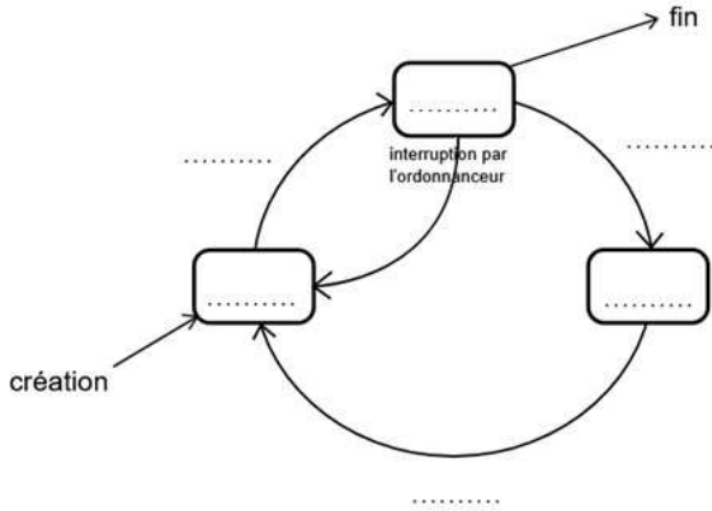
                    if .....:

                        liste_attente.append(.....)

                return .....
```

Exercice n°4

1. Qu'est ce qu'un processus?
2. Compléter le schéma ci-dessous en faisant apparaître les différents états que peut prendre un processus depuis sa création jusqu'à sa terminaison. Les termes désignant les changements d'état devront être apparents.



3. Donner un exemple de situation qui contraint un processus à passer de l'état élu et l'état bloqué.
4. Qu'est ce qu'un ordonnanceur?

Exercice n° 5 _____

Sept processus P_i sont dans la situation suivante par rapport aux ressources R_i :

- P_1 a obtenu R_1 et demande R_2
- P_2 demande R_3
- P_3 demande R_2
- P_4 a obtenu R_2 et R_4 puis demande R_3
- P_5 a obtenu R_3 et demande R_5
- P_6 a obtenu R_6 et demande R_2
- P_7 a obtenu R_5 et demande R_2 .

On voudrait savoir s'il y a interblocage.

1. Construire un graphe orienté où les sommets sont les processus et les ressources, et où :
 - La présence de l'arc $R_i \longrightarrow P_j$ signifie que le processus P_j a obtenu la ressource R_i .
 - La présence de l'arc $P_j \longrightarrow R_i$ signifie que le processus P_j demande la ressource R_i .
2. Y-a-t-il interblocage ? si oui précisez le (ou les) processus concerné(s).

Exercice n° 6 _____

On considère les cinq exécutions de processus suivants (la durée est exprimée en cycle CPU) :

Processus	Date d'arrivée	Durée
P1	0	6
P2	1	5
P3	2	2
P4	3	2
P5	4	1

PARTIE A : Ordonnancement

1. Compléter les diagrammes de Gantt (ci-dessous) à l'aide des algorithmes d'ordonnancement FIFO (Premier arrivé - Premier servi), PCTER (plus court temps d'exécution restant) et Tourniquet (avec un quantum de 2).

FIFO

[illegible]

PCTER

[illegible]

Tourniquet (quantum : 2)

[illegible]

2. Pour chaque algorithme, calculer le temps de traitement moyen.

Aide : Le temps de traitement d'un processus (appelé aussi temps de séjour, d'exécution ou de rotation) est la différence entre la date de fin d'exécution et la date d'arrivée de ce processus.

PARTIE B : Modélisation du tourniquet

Nous allons implémenter l'algorithme d'ordonnancement du tourniquet dont le fonctionnement est résumé ci-dessous :

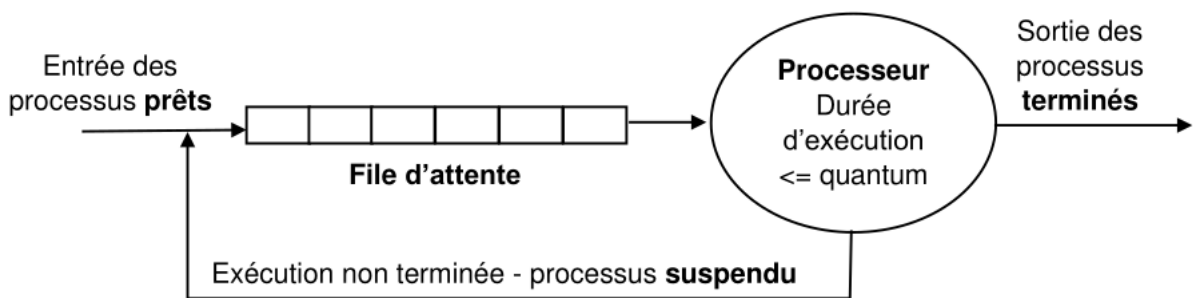


Schéma d'ordonnancement du tourniquet

- Les processus prêts à être exécutés sont placés dans une file d'attente selon leur ordre d'arrivée ;
- L'ordonnanceur alloue le processeur à chaque processus de la file d'attente un même nombre de cycles CPU, appelé quantum ;
- Si le processus n'est pas terminé au bout de ce temps, son exécution est suspendue et il est mis à la fin de la file d'attente ;
- Si le processus est terminé, il sort définitivement de la file d'attente.

Pour représenter le processus, on utilise une classe `Processus` qui possède les variables d'instances `PID` (l'identifiant du processus), `date_arrivee`, `temps_utilisation_CPU` sur le CPU et le temps nécessaire à son exécution `duree`.

1. Compléter le constructeur de la classe Processus :

```
class Processus:
```

```
.....(self, PID , ..... , duree):

    self.PID = ...

    self..... = .....

    self.temps_utilisation_CPU = 0

    .....
```

2. Le processus P1 est :

```
P1 = Processus(PID=1, date_arrivee=0, duree=6)
```

En utilisant le tableau au début de l'exercice, créer les processus $P2$, $P3$, $P4$ et $P5$: quatre instances de la classe `Processus`.

3. Compléter les trois méthodes suivantes de la classe Processus :

```
def execute_un_cycle(self):
    """Met à jour le temps_utilisation_CPU après l'exécution d'un cycle."""
    .....

```

```
def est_termine(self):
    """Renvoie True si le processus est terminé, False sinon,
    en se basant sur duree."""
```

```
.....
```

4. Compléter la fonction `is_finish` qui tester si tous les processus de la liste : `liste_process` sont terminés

```
liste_process=[P1,P2,P3,P4,P5]
def is_finish(liste_process):
    for p in liste_process:

        .....

        .....

    return .....
```

5. Compléter la fonction `tourniquet` qui renvoie la liste ordonnancement des PID des processus dans l'ordre de leur exécution.

```
def tourniquet(liste_process, quantum):
    temps=0
    ordonnancement = []
    liste_attente = []
    # Tant que les processus ne sont pas terminés

    while .....:

        if liste_attente == []:

            temps+=1

            for p in liste_process:

                # tester si la date_arrivee est inférieur au temps

                if p not in liste_attente and .....:

                    ..... # Ajouter p à liste_attente
            else:

                processus = liste_attente.pop(0)

                compteur_tourniquet = 0

                while ..... and .....:

                    ordonnancement.append(.....)

                    ..... # On execute un cycle de processus

                    compteur_tourniquet = .....

                    temps+=1

                for p in liste_process:

                    if p != processus and p not in liste_attente and p.date_arrivee<=temps:

                        liste_attente.append(p)

            if .....:

                liste_attente.append(.....)
```

```
return .....
```