## EXERCICE 2 (6 points)

*Cet exercice porte sur la programmation Python, la gestion des processus.*

On souhaite élaborer un programme système permettant de gérer l'ordre d'exécution des processus sur le processeur.

1. Donner le nom de ce type de programme.
2. Donner les différents états possibles d'un processus.

Chaque processus dispose d'une valeur de priorité. Un processus est prioritaire sur un autre processus si sa valeur de priorité est plus petite. Ainsi pour rendre un processus moins prioritaire, il faut augmenter sa valeur de priorité, par exemple en la faisant passer de 2 à 3.

Fonctionnement du programme gérant l'ordre d'exécution des processus :

On dispose d'une liste dont les éléments sont des files de processus. La première file contient les processus ayant la valeur de priorité la plus élevée 0, la seconde ceux ayant la valeur de priorité 1, etc.

À l'arrivée d'un nouveau processus :

- Attribuer au nouveau processus la valeur de priorité la plus élevée 0 ;
- Placer le nouveau processus dans la file d'attente correspondant à sa valeur de priorité (c'est-à-dire la première file de la liste).

À chaque cycle d'horloge :

- S'il n'y a pas de processus en cours d'exécution et s'il reste des processus en attente :
  - Sélectionner un processus avec la priorité la plus élevée dans l'une des files d'attente non vides ;
  - Élire ce processus comme nouveau processus en cours d'exécution ;
- Sinon si un processus est en cours d'exécution :
  - Si le processus a terminé son exécution, le retirer du processeur ;
  - Sinon,
    - incrémenter le temps d'utilisation du processus ;
    - Si des processus de priorité supérieure ou égale attendent :
      - Retirer le processus en cours d'exécution du processeur ;
      - Réduire sa priorité de 1 et le mettre dans la file d'attente correspondant à sa priorité ;
      - Élire un processus dont la priorité est la plus élevée parmi les processus des files d'attente non vides;
    - Sinon, réduire sa priorité de 1 et continuer à exécuter le processus en cours d'exécution.

3. Parmi les propositions suivantes, donner la structure la plus adaptée pour stocker les processus d'une même priorité :

- Proposition 1 : Liste
- Proposition 2 : File
- Proposition 3 : Pile

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

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

```
1 class Processus:
4   ...(self, ..., priorite, temps_CPU):
5       ... priorite = priorite
7       ... PID = ...
8       self.temps_utilisation = 0
9       self.temps_CPU = temps_CPU
```

5. On considère les trois processus suivants :

```
P1 = Processus(PID=1,priorite=0,temps_CPU=10)
P2 = Processus(PID=2,priorite=0,temps_CPU=7)
P3 = Processus(PID=3,priorite=0,temps_CPU=5)
```

On a donc `liste_files=[[P3, P2, P1], [], []]`.

Compléter la simulation suivante, dans laquelle la variable `CPU` contient le processus en cours d'exécution :

```
Cycle 1: CPU=P1  liste_files=[[P3, P2], [], []]

Cycle 2: CPU=P2  liste_files=[[P3],[P1],[]]

Cycle 3: CPU=P3  liste_files=[[], [...],[]]

Cycle 4: CPU=P3  liste_files=[[], [...], [...]]

Cycle 5: CPU=... liste_files=[[], [...], [...]]
```

Dans les questions 6 et 7, on dispose :

- d'un processus qui nécessite un temps d'utilisation de 1000 pour terminer ;
- d'un nombre important de processus dont le temps d'utilisation pour terminer est de 4 où l'on suppose de plus que chaque processus terminé est remplacé par un nouveau processus similaire.

6. Expliquer pourquoi le processus qui nécessite un long temps d'utilisation du CPU risque de ne jamais terminer avec le programme de gestion de l'ordre des processus ci-dessus (indiquer notamment la priorité du processus long au bout de quelques temps).

Pour régler ce phénomène, on décide d'ajouter la variable d'instance `temps_d_attente` au processus, et on définit une constante appelée `Max_Temps` qui correspond au temps maximum qu'un processus attend avant de remonter sa priorité. L'idée est qu'à chaque cycle, le `temps_d_attente` augmente. Ainsi, si sa valeur dépasse `Max_Temps`, alors sa priorité augmente.

7. Expliquer pourquoi le processus qui nécessite un temps long d'utilisation du CPU ne risque plus de ne jamais terminer avec ce nouveau programme de gestion de l'ordre des processus.

8. Écrire une fonction `meilleur_priorite` qui renvoie `None` s'il n'y a plus de processus et la priorité de l'un des processus les plus prioritaires de la liste des files d'attente dans le cas contraire.

```
1 def meilleur_priorite(liste_files):
2     ...
```

Exemple :

```
# p1, p2 et p3 sont des instances de la classe 'Processus'
>>> liste_files = [[], [p2], [p3, p1]]
>>> meilleur_priorite(liste_files)
1
```

9. Écrire une fonction `prioritaire` qui renvoie `None` si aucune des files d'attente de la liste ne contient un processus et qui renvoie l'un des processus parmi les plus prioritaires sinon (dans ce cas la fonction `prioritaire` supprimera le processus choisi de la file d'attente dans laquelle il se trouvait).

```
1 def prioritaire(liste_files):
2     ...
```

On pourra utiliser `liste.pop(i)` pour renvoyer l'élément de la liste à la position i, tout en le supprimant de la liste.

10. Écrire une fonction `gerer` qui récupère le processus en cours d'exécution `p` ainsi que la liste des files d'attente `liste_files` et qui implémente le programme donné en début d'énoncé pour gérer les processus.

```
1 def gerer(p, liste_files):
2     ...
```