Exercice 2 (6 points)

Cet exercice porte sur les systèmes d'exploitation, les processus, les structures de données linéaires, la programmation en Python et en particulier la programmation orientée objet.

Partie A

"Le système d'exploitation est chargé d'allouer les ressources (mémoires, temps processeur, entrées/sorties) nécessaires aux processus et d'assurer que le fonctionnement d'un processus n'interfère pas avec celui des autres."

Source : Wikipédia, extrait de l'article consacré aux processus.

1. Expliquer succinctement, dans ce contexte, ce qu'est un processus.

On rappelle qu'un processus peut-être soit élu, soit bloqué, soit prêt.

2. Recopier et compléter le schéma ci-dessous avec les termes suivants :

   élu, *bloqué*, *prêt*, *élection*, *blocage*, *déblocage*.

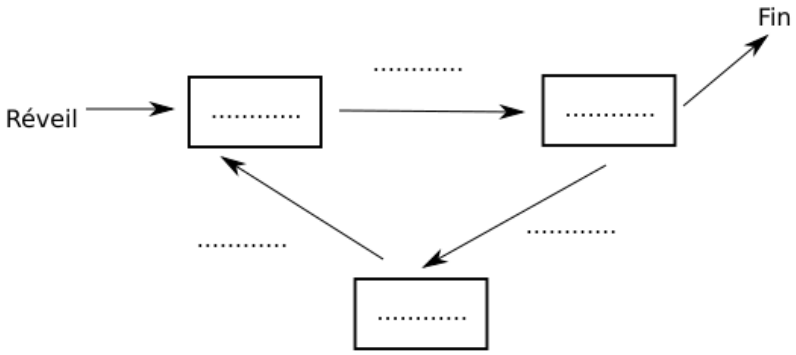

Figure 1. Schéma processus

On considère qu'un monoprocesseur est utilisé. Le système d'exploitation tel un chef d'orchestre, gère l'accès au processeur selon la règle du « premier arrivé, premier servi ». À chaque processus, on associe un instant d'arrivée (instant où le processus demande l'accès au processeur pour la première fois) et une durée d'exécution (durée d'accès au processeur nécessaire pour que le processus s'exécute entièrement).

3. Donner la structure de données la plus adaptée pour gérer l'accès des processus au processeur selon la règle du « premier arrivé, premier servi ».

Le tableau ci-dessous présente les instants d'arrivées et les durées d'exécution de quatre processus :

4 processus		
Processus	instant d'arrivée	durée d'exécution
P1	0	4
P2	2	2
P3	3	4
P4	4	2

4. Recopier et compléter, à l'aide du tableau, le schéma ci-dessous avec les processus P1 à P4 en utilisant la règle du « premier arrivé premier servi ».

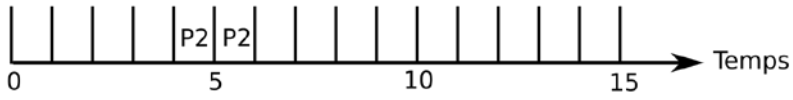

Figure 2. Utilisation du processeur

5. Déterminer le temps qu'a dû attendre le processus P4 avant de pouvoir accéder au processeur.

Partie B

6. Expliquer en quoi consiste la notion d'interblocage.

Afin d'éviter une situation d'interblocage, une solution consiste à attribuer à chaque processus un numéro de priorité.

On souhaite modéliser ce mode de fonctionnement mettant en jeu des numéros de priorité :

- en utilisant une liste de tuples, tuple constitué d'un entier représentant le numéro de priorité ainsi que d'une chaîne de caractères représentant le nom du processus ;
- le processus prioritaire est celui dont le numéro de priorité est le plus petit.

Il est donc important que la liste soit et reste triée dans l'ordre décroissant des numéros de priorités.

Exemple :

```
>>> exemple = [(10, 'Edge'), (8, 'Firefox'), (5, 'Chrome'), (1,
'Vivaldi')]
>>> # La liste est triée, le processus le plus prioritaire est
'Vivaldi'
```

On considère la classe `Priority_Queue` dont l'attribut `liste_priorite` est une liste de tuples, constitués d'un numéro de priorité et d'un nom de processus comme dans l'exemple ci-avant.

```
 1 class Priority_Queue:
 2     def __init__(self):
 3         self.liste_priorite = []
 4 
 5     def est_vide(self):
 6         """Renvoie Vrai si la liste_priorite
 7            est vide, Faux sinon
 8         """
 9         return self.liste_priorite == []
10 
11     def sortir(self):
12         """Retire et renvoie le dernier élément de
13            liste_priorite"""
14         assert ...
15         ...
16 
17     def index_insertion_element(self, element):
18         """Renvoie la position/index d'insertion
19            d'element dans liste_priorite triée
20            par ordre décroissant
21            de numéro priorité
22         """
23         if self.est_vide():
24             ...
25         else:
26             debut = 0
27             fin = len(self.liste_priorite) - 1
28             milieu = (debut + fin) // 2
29             while ... <= fin:
30                 if self.liste_priorite[milieu][0] > ...:
31                     debut = ...
32                 elif self.liste_priorite[milieu][0] < ...:
33                     fin = ...
34                 else:
35                     # cas d'égalité de priorité
36                     ... milieu
37                 milieu = ...
38             return milieu + 1
39 
40     def inserer(self, element):
41         """Modifie liste_priorite en insérant
42            element à la position adéquate
43            dans l'ordre décroissant de
44            numéro de priorité"""
```

7. Écrire l'instruction permettant d'instancier `navigateurs` un objet de la classe `Priority_Queue`.

On rappelle que la méthode `pop()`, appelée sans argument, supprime et renvoie le dernier élément d'une liste.

```
>>> fruits = ['pomme', 'pomme', 'raisin', 'orange', 'poire']
>>> fruits.pop()
'poire'
>>> fruits
['pomme', 'pomme', 'raisin', 'orange']
```

8. Recopier et compléter les lignes 14 et 15 du code de la méthode `sortir` qui après avoir vérifié, sous la forme d'une précondition, que l'objet n'est pas vide, retire et renvoie le dernier élément de `liste_priorite`.

Pour maintenir la liste de priorités triée dans l'ordre décroissant des numéros de priorités, il est indispensable de savoir à quelle position on doit insérer un nouvel élément en fonction de sa priorité.

Cette question ne porte que sur la détermination de la position à laquelle devrait être inséré un élément et cela sans effectuer d'insertion.

On considère, par exemple, que `navigateurs.liste_priorite` contient

`[(10, 'Edge'), (8, 'Firefox'), (5, 'Chrome'), (1, 'Vivaldi')]`.

Si on souhaite insérer :

- l'élément `(12, 'Opera')` on devrait l'insérer au tout début, à la position 0 de `navigateurs.liste_priorite` ;
- l'élément `(6, 'Brave')` on devrait l'insérer à la position 2 juste avant `(5, 'Chrome')` ;
- l'élément `(0, 'Safari')` on devrait l'insérer à la position 4 c'est-à-dire l'ajouter à la fin de la liste.

Pour déterminer la position d'insertion d'un nouvel élément on adapte la méthode dite de recherche dichotomique dans une liste triée dans l'ordre décroissant des numéros de priorités (voir la méthode `index_insertion_element`).

On compare la priorité du tuple `element` à la priorité du tuple se situant au milieu de la `liste_priorite`.

- si elle est strictement supérieure on recommence dans la moitié gauche de `liste_priorite` ;
- si elle est strictement inférieure on recommence dans la moitié droite de `liste_priorite` ;
- si elle est égale la position devra être le milieu.

9. Donner le coût en temps de la recherche dichotomique.

10. Recopier et compléter les huit lignes 24, 29, 30, 31, 32, 33, 36, et 37 du code de la méthode `index_insertion_element` qui prend en paramètre un élément `element` et qui renvoie la position d'insertion de l'élément `element` en utilisant une méthode dichotomique.

11. Écrire, sans utiliser la méthode `insert` des listes Python, une méthode `inserer` qui prend en paramètre un élément `element`, et modifie `liste_priorite` en insérant l'élément `element` à la position adéquate de la liste triée par ordre décroissant des numéros de priorités.

    Exemples :

```
>>> navigateurs.liste_priorite
[(10, 'Edge'), (8, 'Firefox'), (5, 'Chrome'), (1,
'Vivaldi')]
>>> navigateurs.inserer((16, 'Brave'))
>>> navigateurs.liste_priorite
[(16, 'Brave'), (10, 'Edge'), (8, 'Firefox'), (5,
'Chrome'), (1, 'Vivaldi')]
>>> navigateurs.inserer((6, 'Safari'))
>>> navigateurs.liste_priorite
[(16, 'Brave'), (10, 'Edge'), (8, 'Firefox'),(6,
'Safari'), (5, 'Chrome'), (1, 'Vivaldi')]
>>> navigateurs.inserer((0, 'Lynx'))
>>> navigateurs.liste_priorite
[(16, 'Brave'), (10, 'Edge'), (8, 'Firefox'),(6,
'Safari'), (5, 'Chrome'), (1, 'Vivaldi'), (0, 'Lynx')]
```