

## EXERCICE 1 (6 points)

*Cet exercice porte sur la programmation Python, la programmation orientée objet, les structures de données (file), l'ordonnancement et l'interblocage.*

On s'intéresse aux processus et à leur ordonnancement au sein d'un système d'exploitation. On considère ici qu'on utilise un monoprocesseur.

1. Citer les trois états dans lesquels un processus peut se trouver.

On veut simuler cet ordonnancement avec des objets. Pour ce faire, on dispose déjà de la classe `Processus` dont voici la documentation :

### Classe `Processus`:

```
p = Processus(nom: str, duree: int)
    Crée un processus de nom <nom> et de durée <duree> (exprimée en
    cycles d'ordonnancement)

p.execute_un_cycle()
    Exécute le processus donné pendant un cycle.

p.est_fini()
    Renvoie True si le processus est terminé, False sinon.
```

Pour simplifier, on ne s'intéresse pas aux ressources qu'un processus pourrait acquérir ou libérer.

2. Citer les deux seuls états possibles pour un processus dans ce contexte.

Pour mettre en place l'ordonnancement, on décide d'utiliser une file, instance de la classe `File` ci-dessous.

### Classe `File`

```
1 class File:
2     def __init__(self):
3         """ Crée une file vide """
4         self.contenu = []
5
6     def enqueue(self, element):
7         """ Enfile element dans la file """
8         self.contenu.append(element)
9
10    def dequeue(self):
11        """ Renvoie le premier élément de la file et l'enlève de
12        la file """
13        return self.contenu.pop(0)
14
15    def est_vide(self):
```

```

15         """ Renvoie True si la file est vide, False sinon """
16         return self.contenu == []

```

Lors de la phase de tests, on se rend compte que le code suivant produit une erreur :

```

1 f = File()
2 print(f.defile())

```

3. Rectifier sur votre copie le code de la classe `File` pour que la fonction `defile` renvoie `None` lorsque la file est vide.

On se propose d'ordonnancer les processus avec une méthode du type *tourniquet* telle qu'à chaque cycle :

- si un nouveau processus est créé, il est mis dans la file d'attente ;
- ensuite, on défile un processus de la file d'attente et on l'exécute pendant un cycle ;
- si le processus exécuté n'est pas terminé, on le replace dans la file.

Par exemple, avec les processus suivants

Liste des processus		
processus	cycle de création	durée en cycles
A	2	3
B	1	4
C	4	3
D	0	5

On obtient le chronogramme ci-dessous :

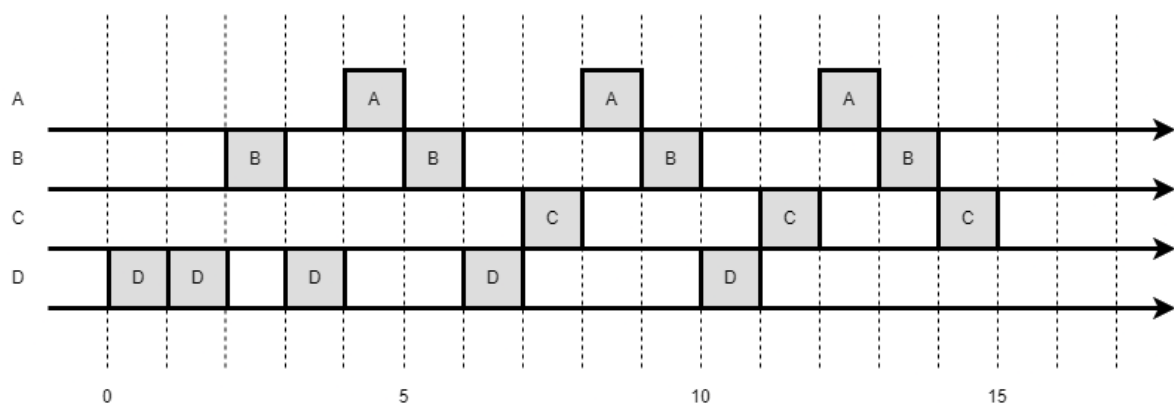


Figure 1. Chronogramme pour les processus A, B, C et D

Pour décrire les processus et le moment de leur création, on utilise le code suivant, dans lequel `depart_proc` associe à un cycle donné le processus qui sera créé à ce moment :

```

1 p1 = Processus("p1", 4)
2 p2 = Processus("p2", 3)
3 p3 = Processus("p3", 5)
4 p4 = Processus("p4", 3)
5 depart_proc = {0: p1, 1: p3, 2: p2, 3: p4}

```

Il s'agit d'une modélisation de la situation précédente où un seul processus peut être créé lors d'un cycle donné.

4. Recopier et compléter sur votre copie le chronogramme ci-dessous pour les processus p1, p2, p3 et p4.

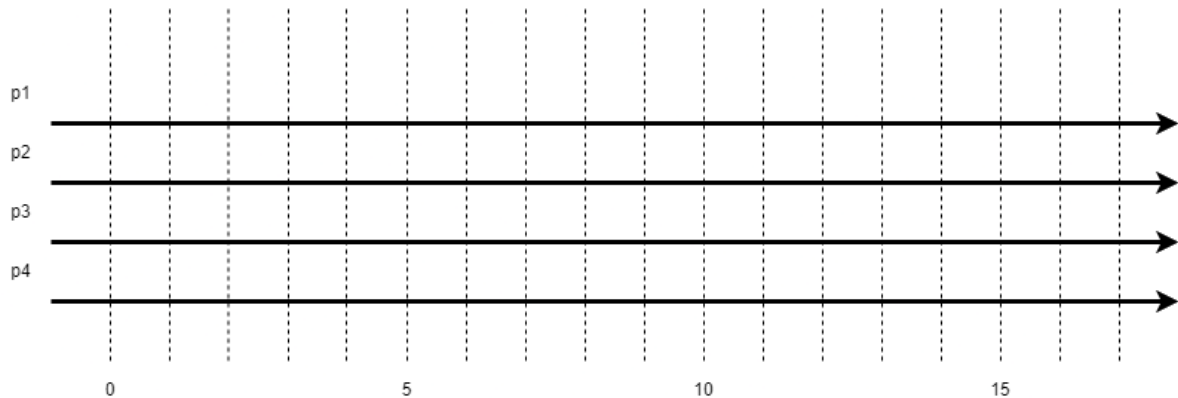


Figure 2. Chronogramme pour les processus p1, p2, p3 et p4

Pour mettre en place l'ordonnancement suivant cette méthode, on écrit la classe `Ordonnanceur` dont voici un code incomplet (l'attribut `temps` correspond au cycle en cours) :

```

1  class Ordonnanceur:
2
3      def __init__(self):
4          self.temps = 0
5          self.file = File()
6
7      def ajoute_nouveau_processus(self, proc):
8          '''Ajoute un nouveau processus dans la file de
9             l'ordonnanceur. '''
10         ...
11
12     def tourniquet(self):
13         '''Effectue une étape d'ordonnancement et renvoie le nom
14            du processus élu.'''
15         self.temps += 1
16         if not self.file.est_vide():
17             proc = ...
18             ...
19             if not proc.est_fini():
20                 ...
21             return proc.nom
22         else:
23             return None

```

5. Compléter le code ci-dessus.

À chaque appel de la méthode `tourniquet`, celle-ci renvoie soit le nom du processus qui a été élu, soit `None` si elle n'a pas trouvé de processus en cours.

6. Écrire un programme qui :

- utilise les variables `p1`, `p2`, `p3`, `p4` et `depart_proc` définies précédemment ;
- crée un ordonnanceur ;
- ajoute un nouveau processus à l'ordonnanceur lorsque c'est le moment ;
- affiche le processus choisi par l'ordonnanceur ;
- s'arrête lorsqu'il n'y a plus de processus à exécuter.

Dans la situation donnée en exemple (voir Figure 1), il s'avère qu'en fait les processus utilisent des ressources comme :

- un fichier commun aux processus ;
- le clavier de l'ordinateur ;
- le processeur graphique (GPU) ;
- le port 25000 de la connexion Internet.

Voici le détail de ce que fait chaque processus :

Liste des processus			
A	B	C	D
acquérir le GPU	acquérir le clavier	acquérir le port	acquérir le fichier
faire des calculs	acquérir le fichier	faire des calculs	faire des calculs
libérer le GPU	libérer le clavier	libérer le port	acquérir le clavier
	libérer le fichier		libérer le clavier
			libérer le fichier

7. Montrer que l'ordre d'exécution donné en exemple aboutit à une situation d'interblocage.

## EXERCICE 2 (6 points)

Cet exercice porte sur les systèmes d'exploitation, les commandes UNIX, les structures de données (de type LIFO et FIFO) et les processus.

*“Linux ou GNU/Linux est une famille de systèmes d'exploitation open source de type Unix fondée sur le noyau Linux, créé en 1991 par Linus Torvalds. De nombreuses distributions Linux ont depuis vu le jour et constituent un important vecteur de popularisation du mouvement du logiciel libre.”*

Source : Wikipédia, extrait de l'article consacré à GNU/Linux.

*“Windows est au départ une interface graphique unifiée produite par Microsoft, qui est devenue ensuite une gamme de systèmes d'exploitation à part entière, principalement destinés aux ordinateurs compatibles PC. Windows est un système d'exploitation propriétaire.”*

Source : Wikipédia, extrait de l'article consacré à Windows.

1. Expliquer succinctement les différences entre les logiciels libres et les logiciels propriétaires.
2. Expliquer le rôle d'un système d'exploitation.

On donne ci-dessous une arborescence de fichiers sur un système GNU/Linux (les noms encadrés représentent des répertoires, les noms non encadrés représentent des fichiers, / correspond à la racine du système de fichiers) :

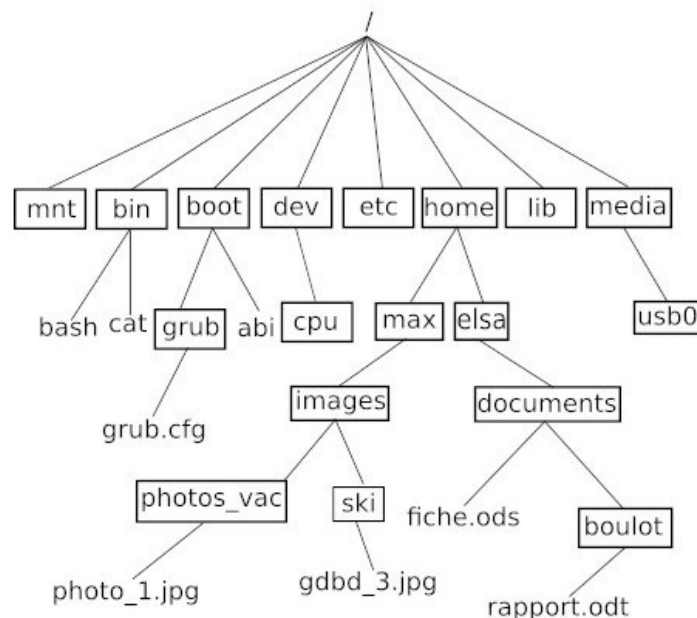


Figure 1. Arborescence de fichiers

3. Indiquer le chemin absolu du fichier `rapport.odt`.

On suppose que le répertoire courant est le répertoire `elsa`.

- Indiquer le chemin relatif du fichier `photo_1.jpg`.

L'utilisatrice Elsa ouvre une console (aussi parfois appelée terminal), le répertoire courant étant toujours le répertoire `elsa`. On fournit ci-dessous un extrait du manuel de la commande UNIX `cp` :

NOM

`cp` - copie un fichier

UTILISATION

`cp fichier_source fichier_destination`

- Déterminer le contenu du répertoire `documents` et du répertoire `boulot` après avoir exécuté la commande suivante dans la console :

```
cp documents/fiche.ods documents/boulot
```

*“Un système d’exploitation est multitâche (en anglais : multitasking) s’il permet d’exécuter, de façon apparemment simultanée, plusieurs programmes informatiques. GNU/Linux, comme tous les systèmes d’exploitation modernes, gère le multitâche.”*

*“Dans le cas de l’utilisation d’un monoprocesseur, la simultanéité apparente est le résultat de l’alternance rapide d’exécution des processus présents en mémoire.”*

Source : Wikipédia, extraits de l’article consacré au Multitâche.

Dans la suite de l’exercice, on s’intéresse aux processus. On considère qu’un monoprocesseur est utilisé. On rappelle qu’un processus est un programme en cours d’exécution. Un processus est soit élu, soit bloqué, soit prêt.

- Recopier et compléter le schéma ci-dessous avec les termes suivants :  
*élu, bloqué, prêt, élection, blocage, déblocage.*

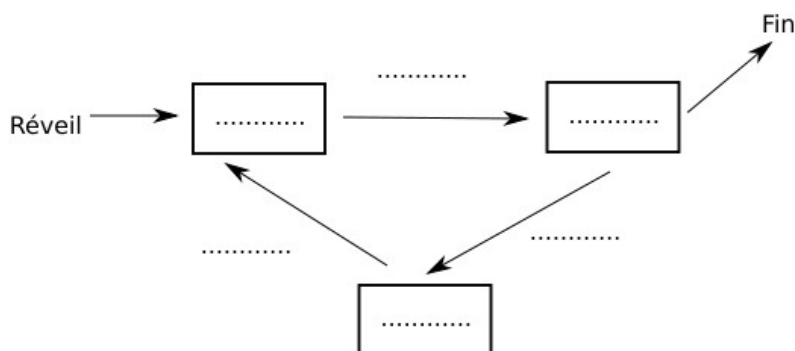


Figure 2. Schéma processus

- Donner l'exemple d'une situation qui contraint un processus à passer de l'état élu à l'état bloqué.

*“Dans les systèmes d’exploitation, l’ordonnanceur est le composant du noyau du système d’exploitation choisissant l’ordre d’exécution des processus sur le processeur d’un ordinateur.”*

*Source : Wikipédia, extrait de l’article consacré à l’ordonnancement.*

L’ordonnanceur peut utiliser plusieurs types d’algorithmes pour gérer les processus.

L’algorithme d’ordonnancement par “ordre de soumission” est un algorithme de type FIFO (First In First Out), il utilise donc une file.

8. Nommer une structure de données linéaire de type LIFO (Last In First Out).

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).

Par exemple, l’exécution d’un processus P4 qui a un instant d’arrivée égal à 7 et une durée d’exécution égale à 2 peut être représentée par le schéma suivant :

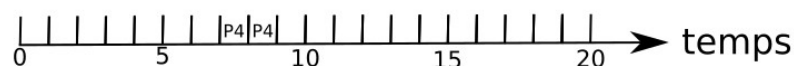


Figure 3. Utilisation du processeur

L’ordonnanceur place les processus qui ont besoin d’un accès au processeur dans une file, en respectant leur ordre d’arrivée (le premier arrivé étant placé en tête de file). Dès qu’un processus a terminé son exécution, l’ordonnanceur donne l’accès au processus suivant dans la file.

Le tableau suivant présente les instants d’arrivées et les durées d’exécution de cinq processus :

5 processus		
Processus	instant d’arrivée	durée d’exécution
P1	0	3
P2	1	6
P3	4	4
P4	6	2
P5	7	1

9. Recopier et compléter le schéma ci-dessous avec les processus P1 à P5 en utilisant les informations présentes dans le tableau ci-dessus et l’algorithme d’ordonnancement “par ordre de soumission”.

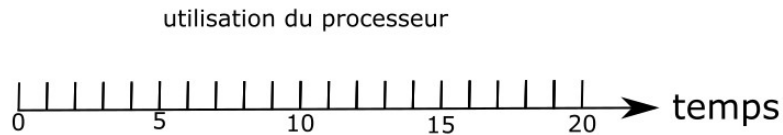


Figure 4. Utilisation du processeur

On utilise maintenant un autre algorithme d'ordonnancement : l'algorithme d'ordonnancement "par tourniquet". Dans cet algorithme, la durée d'exécution d'un processus ne peut pas dépasser une durée  $Q$  appelée quantum et fixée à l'avance. Si ce processus a besoin de plus de temps pour terminer son exécution, il doit retourner dans la file et attendre son tour pour poursuivre son exécution.

Par exemple, si un processus  $P1$  a une durée d'exécution de 3 et que la valeur de  $Q$  a été fixée à 2,  $P1$  s'exécutera pendant deux unités de temps avant de retourner à la fin de la file pour attendre son tour ; une fois à nouveau élu, il pourra terminer de s'exécuter pendant sa troisième et dernière unité de temps d'exécution.

10. Recopier et compléter le schéma ci-dessous, en utilisant l'algorithme d'ordonnancement "par tourniquet" et les mêmes données que pour la question 9, en supposant que le quantum  $Q$  est fixé 2.

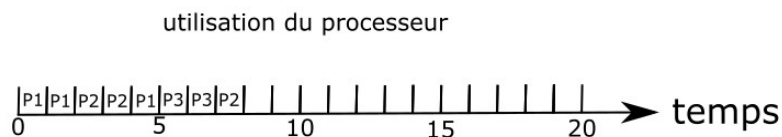


Figure 5. Utilisation du processeur

On considère deux processus  $P1$  et  $P2$ , et deux ressources  $R1$  et  $R2$ .

11. Décrire une situation qui conduit les deux processus  $P1$  et  $P2$  en situation d'interblocage.



### EXERCICE 3 (4 points)

Cet exercice traite du thème architecture matérielle, et plus particulièrement des processus et leur ordonnancement.

1. Avec la commande `ps -aef` on obtient l'affichage suivant :

PID	PPID	C	STIME	TTY	TIME	CMD
8600	2	0	17:38	?	00:00:00	[kworker/u2:0-fl]
8859	2	0	17:40	?	00:00:00	[kworker/0:1-eve]
8866	2	0	17:40	?	00:00:00	[kworker/0:10-ev]
8867	2	0	17:40	?	00:00:00	[kworker/0:11-ev]
8887	6217	0	17:40	pts/0	00:00:00	bash
9562	2	0	17:45	?	00:00:00	[kworker/u2:1-ev]
9594	2	0	17:45	?	00:00:00	[kworker/0:0-eve]
9617	8887	21	17:46	pts/0	00:00:06	/usr/lib/firefox/firefox
9657	9617	17	17:46	pts/0	00:00:04	/usr/lib/firefox/firefox -contentproc -childID
9697	9617	4	17:46	pts/0	00:00:01	/usr/lib/firefox/firefox -contentproc -childID
9750	9617	3	17:46	pts/0	00:00:00	/usr/lib/firefox/firefox -contentproc -childID
9794	9617	11	17:46	pts/0	00:00:00	/usr/lib/firefox/firefox -contentproc -childID
9795	9794	0	17:46	pts/0	00:00:00	/usr/lib/firefox/firefox
9802	7441	0	17:46	pts/2	00:00:00	ps -aef

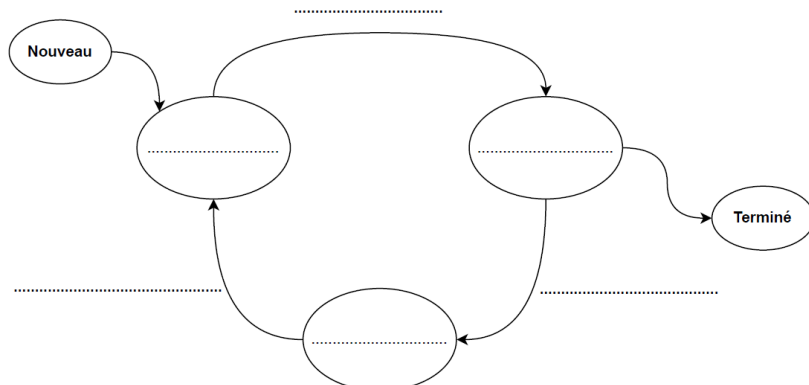
On rappelle que : *PID* = Identifiant d'un processus (*Process Identification*)

*PPID* = Identifiant du processus parent d'un processus (*Parent Process Identification*)

- Donner sous forme d'un arbre de PID la hiérarchie des processus liés à *firefox*.
- Indiquer la commande qui a lancé le premier processus de *firefox*.
- La commande *kill* permet de supprimer un processus à l'aide de son *PID* (par exemple *kill 8600*). Indiquer la commande qui permettra de supprimer tous les processus liés à *firefox* et uniquement cela.

2.

- Recopier et compléter le schéma ci-dessous avec les termes suivants concernant l'ordonnancement des processus : *Élu*, *En attente*, *Prêt*, *Blocage*, *Déblocage*, *Mise en exécution*



On donne dans le tableau ci-dessous quatre processus qui doivent être exécutés par un processeur. Chaque processus a un instant d'arrivée et une durée, donnés en nombre de cycles du processeur.

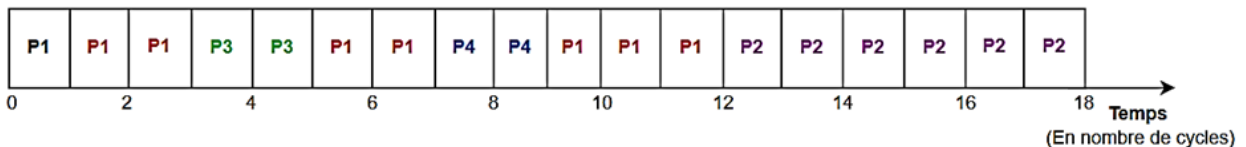
Processus	P1	P2	P3	P4
Instant d'arrivée	0	2	3	7
Durée	8	6	2	2

Les processus sont placés dans une file d'attente en fonction de leur instant d'arrivée.

On se propose d'ordonnancer ces quatre processus avec la méthode suivante :

- Parmi les processus présents en liste d'attente, l'ordonnanceur choisit celui dont la durée **restante** est la plus courte ;
- Le processeur exécute un cycle de ce processus puis l'ordonnanceur désigne de nouveau le processus dont la durée restante est la plus courte ;
- En cas d'égalité de temps restant entre plusieurs processus, celui choisi sera celui dont l'instant d'arrivée est le plus ancien ;
- Tout ceci jusqu'à épuisement des processus en liste d'attente.

On donne en exemple ci-dessous, l'ordonnancement des quatre processus de l'exemple précédent suivant l'algorithme ci-dessus.

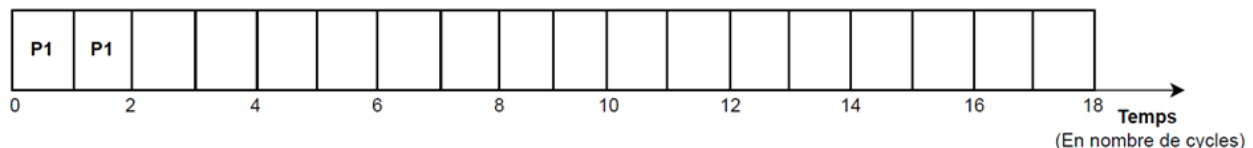


On définit le temps d'exécution d'un processus comme la différence entre son instant de terminaison et son instant d'arrivée.

**b.** Calculer la moyenne des temps d'exécution des quatre processus.

On se propose de modifier l'ordonnancement des processus. L'algorithme reste identique à celui présenté précédemment mais au lieu d'exécuter un seul cycle, le processeur exécutera à chaque fois deux cycles du processus choisi. En cas d'égalité de temps restant, l'ordonnanceur départagera toujours en fonction de l'instant d'arrivée.

**c.** Recopier et compléter le schéma ci-dessous donnant le nouvel ordonnancement des quatre processus.



**d.** Calculer la nouvelle moyenne des temps d'exécution des quatre processus et indiquer si cet ordonnancement est plus performant que le précédent.

3. On se propose de programmer l'algorithme du premier ordonnanceur. Chaque processus sera représenté par une liste comportant autant d'éléments que de durées (en nombre de cycles). Pour simuler la date de création de chaque processus, on ajoutera en fin de liste de chaque processus autant de chaînes de caractères vides que la valeur de leur date de création.

```
1 p1 = ['1.8', '1.7', '1.6', '1.5', '1.4', '1.3', '1.2', '1.1']
2 p2 = ['2.6', '2.5', '2.4', '2.3', '2.2', '2.1', '', '']
3 p3 = ['3.2', '3.1', '', '', '']
4 p4 = ['4.2', '4.1', '', '', '', '', '', '']
5 liste_proc = [p1, p2, p3, p4]
```

La fonction `choix_processus` est chargée de sélectionner le processus dont le temps restant d'exécution est le plus court parmi les processus en liste d'attente.

- a. Recopier sans les commentaires et compléter la fonction `choix_processus` ci-dessous. Le code peut contenir plusieurs lignes.

```
1 def choix_processus(liste_attente):
2     """Renvoie l'indice du processus le plus court parmi
3     ceux présents en liste d'attente liste_attente"""
4     if liste_attente != []:
5         mini = len(liste_attente[0])
6         indice = 0
7         ...
8         return indice
```

Une fonction `scrutation` (non étudiée) est chargée de parcourir la liste `liste_proc` de tous les processus et de renvoyer la liste d'attente des processus en fonction de leur arrivée. À chaque exécution de `scrutation`, les processus présents (sans chaînes de caractères vides en fin de liste) sont ajoutés à la liste d'attente. La fonction supprime pour les autres un élément de chaîne de caractères vides.

- b. Recopier et compléter les différentes instructions de la fonction `ordonnancement` pour réaliser le fonctionnement désiré.

```
1 def ordonnancement(liste_proc):
2     """Exécute l'algorithme d'ordonnancement
3     liste_proc -- liste des processus
4     Renvoie la liste d'exécution des processus"""
5     execution = []
6     attente = scrutation(liste_proc, [])
7     while attente != []:
8         indice = choix_processus(attente)
9         ... # A FAIRE (plusieurs lignes de code) ...
10        attente = scrutation(liste_proc, attente)
11    return execution
```