Autotest n°5 – Correction

Exercice 1 : (11pt) *Cet exercice traite du thème architecture matérielle, et plus particulièrement des processus et leur ordonnancement.*

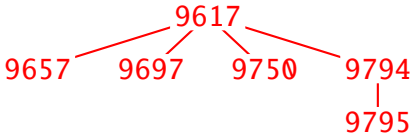
1) Avec la commande `ps -aef` on obtient l'affichage suivant :

```
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/bin/firefox/firefox
9657 9617 17 17:46 pts/0 00:00:04 /usr/bin/firefox/firefox -contentproc -childID
9697 9617  4 17:46 pts/0 00:00:01 /usr/bin/firefox/firefox -contentproc -childID
9750 9617  3 17:46 pts/0 00:00:00 /usr/bin/firefox/firefox -contentproc -childID
9794 9617 11 17:46 pts/0 00:00:00 /usr/bin/firefox/firefox -contentproc -childID
9795 9794  0 17:46 pts/0 00:00:00 /usr/bin/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)

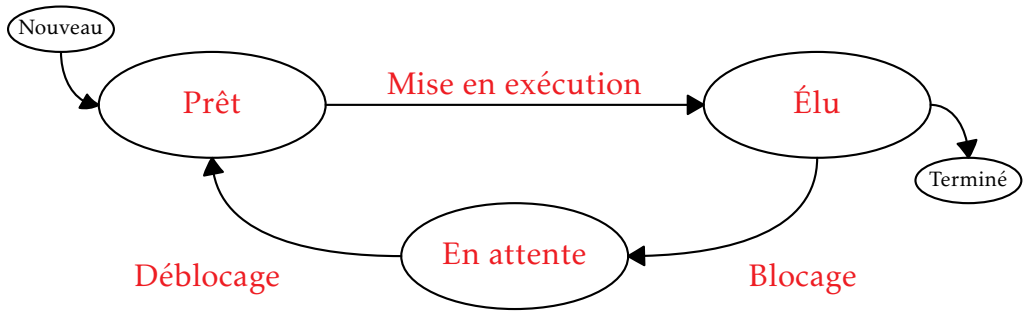
a) Donner sous forme d'un arbre de PID la hiérarchie des processus liés à `firefox`.



b) Indiquer la commande qui correspond au processus parent du premier processus de `firefox`. `bash`

c) La commande `kill` permet de supprimer un processus à l'aide de son PID (par exemple `kill 8600`). Lorsqu'on supprime un processus, tous les sous-processus sont également supprimés.
Indiquer la commande qui permettra de supprimer tous les processus liés à `firefox` et uniquement ces processus. `kill 9617`

2) a) Compléter le schéma ci-dessous avec les termes suivants concernant l'ordonnancement des processus : *Élu*, *En attente*, *Prêt*, *Blocage*, *Déblocage*, *Mise en exécution*.



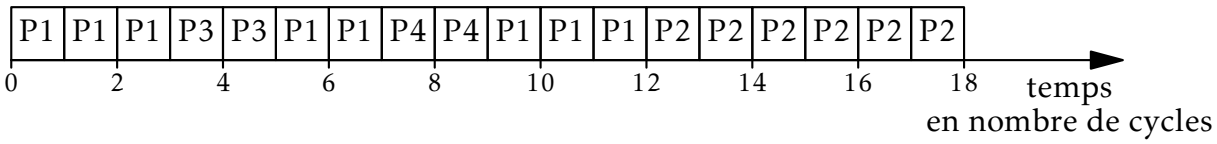
On donne dans le tableau ci-dessous quatre processus qui doivent être exécutés par un processeur. Chaque processus a un instant d'arrivée et une durée, donnés en nombre de cycles du processeur.

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.



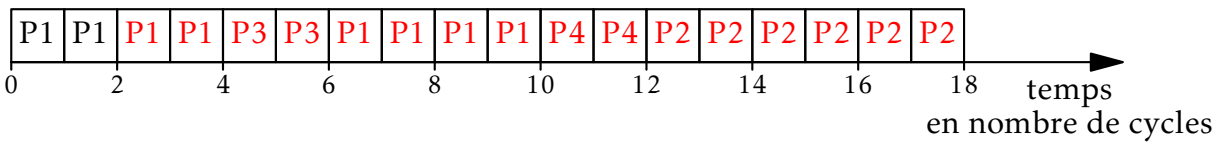
On définit le temps d'exécution d'un processus comme la différence entre son instant de terminaison et son instant d'arrivée.

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

$$\frac{12+2+2+16}{4} = \frac{32}{4} = 8$$

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) 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.

$$\frac{10+3+5+16}{4} = \frac{34}{4} = 8{,}5$$

Cet ordonnancement est moins performant.

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.

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

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. Les processus qui sont terminés ne sont pas remis dans la liste d'attente.

3) a) 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.
Compléter la fonction `choix_processus` ci-dessous.

```
def choix_processus(liste_attente):
    """Renvoie l'indice du processus le plus court parmi
    ceux présents en liste d'attente liste_attente"""
    if liste_attente != []:
        mini = len(liste_attente[0])
        indice = 0
        for i in range(len(liste_attente)):
            if len(liste_attente[i]) < mini:
                indice = i
                mini = liste_attente[i]
        return indice
```

Lors de l'éxecution d'un processus, on supprime son dernier élément, avec la méthode `pop`.

```
>>> proc = ['1.8', '1.7', '1.6', '1.5', '1.4']
>>> proc.pop()
'1.4'
>>> proc
['1.8', '1.7', '1.6', '1.5']
```

b) Compléter la fonction `ordonnancement` pour réaliser le fonctionnement désiré.

```
def ordonancement(liste_proc):
    """Exécute l'algorithme d'ordonnancement
    liste_proc -- liste des processus
    Renvoie la liste d'exécution des processus"""
    execution = []
    attente = scrutation(liste_proc, [])
    while attente != []:
        indice = choix_processus(attente)
        processus = attente[indice]
        execution.append(processus.pop())
        attente = scrutation(liste_proc, attente)
    return execution
```

Exercice 2 : *Cet exercice porte sur la gestion des processus et des ressources par un système d'exploitation.*

Les parties A et B peuvent être traitées indépendamment.

Partie A : Ordonnancement des processus

Dans le laboratoire d'analyse médicale d'un hôpital, plusieurs processus peuvent demander l'allocation du processeur simultanément.

Le tableau ci-dessous donne les demandes d'exécution de quatre processus et indique :

- le temps d'exécution du processus (en unité de temps) ;
- l'instant d'arrivée du processus sur le processeur (en unité de temps) ;
- le numéro de priorité du processus (classé de 1 à 10).

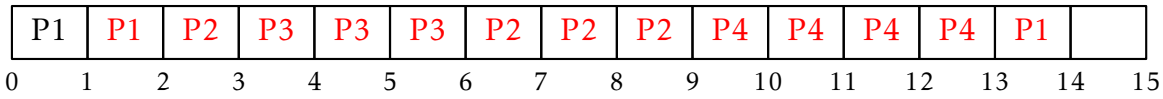
Plus la priorité est grande plus le numéro de priorité est petit.

Ainsi le processus P3, du tableau ci-dessous, est plus prioritaire que le processus P1.

L'ordonnancement est de type préemptif, ce qui signifie qu'à chaque unité de temps, le processeur choisit d'exécuter le processus ayant le plus petit numéro de priorité (un seul processus à la fois). Ceci peut provoquer la suspension d'un autre processus qui reprendra lorsqu'il deviendra le plus prioritaire dans la file d'attente.

Processus	Temps d'exécution	Instant d'arrivée	Numéro de priorité
P1	3	0	4
P2	4	2	2
P3	3	3	1
P4	4	5	3

1) Compléter le diagramme ci-dessous et indiquer dans chacune des cases le processus exécuté par le processeur entre deux unités de temps (il peut y avoir des cases vides).



2) Compléter les temps de séjour ainsi que les temps d'attente de chacun des processus (toujours en unités de temps).

Temps de séjour = instant de terminaison – instant d'arrivée

Temps d'attente = temps de séjour – temps d'exécution

Processus	Temps d'exécution	Instant d'arrivée	Numéro de priorité	Temps de séjour	Temps d'attente
P1	3	0	4	14 – 0 = 14	14 – 3 = 11
P2	4	2	2	9 – 2 = 7	7 – 4 = 3
P3	3	3	1	6 – 3 = 3	3 – 3 = 0
P4	4	5	3	13 – 5 = 8	8 – 4 = 4

3) À quelles conditions le temps d'attente d'un processus peut-il être nul ?

Solution : Il faut que ce soit le processus le plus prioritaire pendant toute son exécution.

Partie B : Processus et ressources

Dans ce laboratoire d'analyse médicale de l'hôpital, le laborantin en charge du traitement des différents prélèvements (sanguins, urinaires et biopsiques) utilise simultanément quatre logiciels :

- Logiciel d'analyse d'échantillons (connecté à l'analyseur)
- Logiciel d'accès à la base de données des patients (SGBD)
- Traitement de texte
- Tableur

Le tableau ci-dessous donne l'état à un instant donné des différents processus (instances des programmes) qui peuvent soit mobiliser (M) des données (D1, D2, D3, D4 et D5), soit être en attente des données (A) ou ne pas les solliciter (-).
Une donnée ne peut être mobilisée que par un seul processus à la fois. Si un autre processus demande une donnée déjà mobilisée, il passe en attente.
Exemple : le SGBD mobilise la donnée D4 et est en attente de la donnée D5

	D1	D2	D3	D4	D5
Analyseur échantillon	M	-	-	A	-
SGBD	-	-	-	M	A
Traitement de texte	-	M	A	-	-
Tableur	A	-	M	-	M

1) À partir du tableau ci-dessus, démontrer que, à cet instant, les processus s'attendent mutuellement.
   Solution : Toutes les ressources sont mobilisées et tous les processus sont en attente d'une autre ressource. Aucun de ces processus ne peut avancer.
2) Comment s'appelle cette situation ?
   Solution : C'est une situation d'interblocage.
3) On suppose que l'analyseur d'échantillon libère la ressource D1. Donner un ordre possible d'exécution des processus.
   Solution :
   - Le tableur peut terminer son exécution et libérer D1, D3 et D5.
   - Le traitement de texte peut terminer son exécution et libérer D2 et D3.
   - Le SGBD peut terminer son exécution et libérer D4 et D5.
   - L'analyseur d'échantillon peut terminer son exécution.