

EXERCICE

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

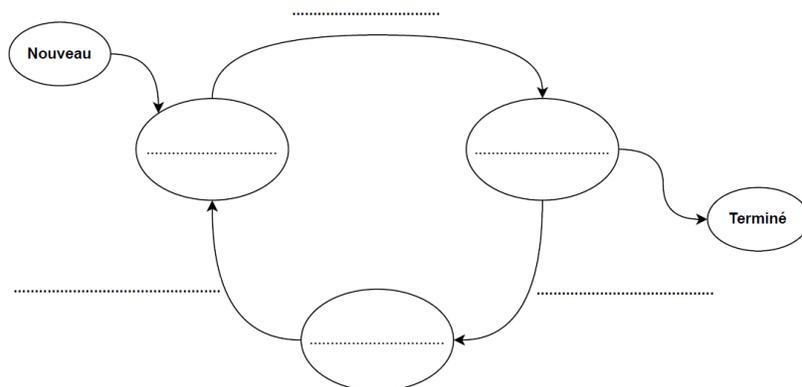
On rappelle que : *PID* = Identifiant d'un processus (*Process Identification*)

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

- a. Donner sous forme d'un arbre de PID la hiérarchie des processus liés à `firefox`.
- b. Indiquer la commande qui a lancé le premier processus de `firefox`.
- c. 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.

- a. 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.

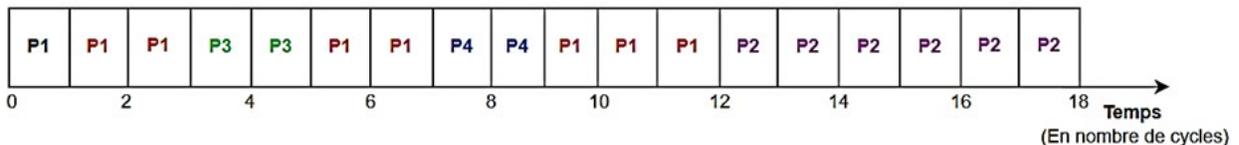
Processus	P1	P2	P3	P4
Instant d'arrivée	0	2	3	7
Durée	8	6	2	2

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.

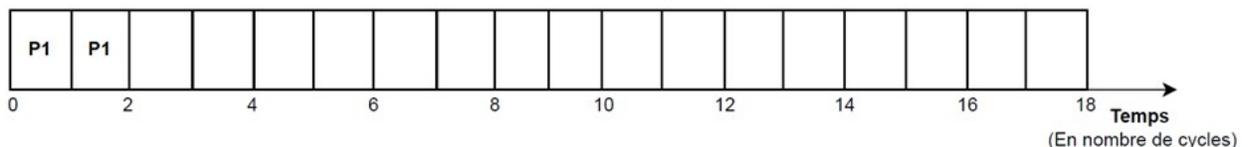


On définit le temps d'exécution d'un processus comme la différence entre son instant de terminaison et son instant d'arrivée.

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
```