

Exercice 1 (6 points)

Cet exercice porte sur l'architecture matérielle, les systèmes d'exploitation, et les structures de données linéaires en Python.

Un système d'exploitation permet d'exécuter plusieurs applications à la fois en donnant l'impression qu'elles fonctionnent simultanément. En réalité, le système répartit le temps de calcul du processeur entre les différents processus, de sorte qu'ils s'exécutent chacun à leur tour très rapidement.

Chaque application peut être représentée par un ou plusieurs processus, gérés par le système d'exploitation. Même si un seul processus utilise réellement le processeur à un instant donné, cette alternance rapide donne l'impression que tout s'exécute en même temps. Lorsque le système interrompt un processus pour en exécuter un autre, on parle de *préemption*.

Partie A

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

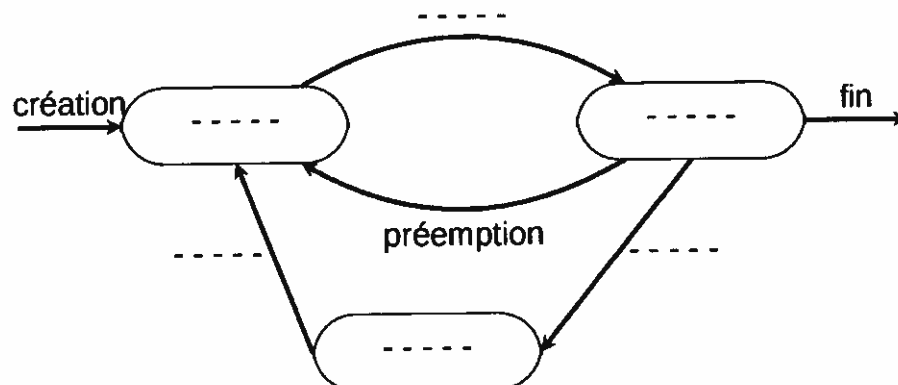


Figure 1. États d'un processus.

On peut imaginer, par exemple, une application de streaming musical qui se compose de trois processus :

- le processus P1 gère l'interface utilisateur et interagit avec les composants visibles : listes de morceaux, boutons de lecture-pause, etc. ;
- le processus P2 assure le téléchargement de la musique et l'écrit dans une mémoire cache locale ;
- le troisième processus P3, qu'on pourrait appeler *lecteur audio*, assure le décodage audio et envoie le flux de données au système pour lecture.

Voici une situation de fonctionnement :

- l'utilisateur décide de mettre l'application en arrière-plan, cachant ainsi l'interface utilisateur ;

- le processus P2 attend un accès à la carte WIFI pour recharger des données ;
 - le processus P3 décode de la musique et l'envoie au système.
2. Indiquer l'état de chacun des trois processus P1, P2 et P3 dans cette situation.
 3. Expliquer, de façon générale, quand est-ce qu'un interblocage de processus peut survenir.

On considère plusieurs ressources dont un processeur graphique (GPU), un microphone (MIC), une caméra (CAM) et un processeur dédié au calcul (CAL). On suppose que chacune de ces ressources ne peut être utilisée que par un seul processus à la fois.

On considère de plus quatre processus nommés P1, P2, P3 et P4. Le tableau suivant récapitule les ressources utilisées par chacun des quatre processus, dans l'ordre où chacun des processus les demande :

P1	P2	P3	P4
demander MIC demander CAL libérer CAL libérer MIC demander CAM demander GPU libérer CAM libérer GPU	demander CAL demander MIC libérer MIC libérer CAL demander CAM libérer CAM	demander CAM libérer CAM demander CAL demander MIC libérer MIC libérer CAL demander GPU libérer GPU	demander GPU demander CAL demander CAM libérer CAM libérer CAL demander MIC libérer GPU libérer MIC

4. Les processus s'exécutent de manière concurrente. Justifier qu'une situation d'interblocage peut se produire.
5. Expliquer l'intérêt d'utiliser une machine équipée de plusieurs processeurs plutôt qu'une machine équipée d'un seul processeur.
6. Décrire un avantage et un inconvénient des systèmes sur puces, tels que ceux utilisés dans les smartphones.

Partie B

On s'intéresse à un ordonnanceur de type tourniquet (round-robin), dans lequel une durée appelée *quantum* est fixée à 2 ms. Chaque processus, lorsqu'il est défilé de la file d'attente, s'exécute sur le processeur pour une durée au plus égale au *quantum* :

- s'il termine son exécution avant ou à la fin du quantum, un autre processus est défilé de la file d'attente, et celui-ci bénéficie d'un nouveau quantum pour s'exécuter.
- s'il n'a pas terminé son exécution, il est mis en queue de la file d'attente.

De plus, d'autres processus peuvent être ajoutés à la file d'attente au fur et à mesure qu'ils arrivent.

On considère quatre processus nommés P1, P2, P3 et P4. Le tableau suivant donne les informations temporelles les concernant.

Processus	Instant d'arrivée	Temps d'exécution total
P1	0 ms	6 ms
P2	1 ms	4 ms
P3	3 ms	5 ms
P4	5 ms	3 ms

7. Recopier et compléter le chronogramme suivant en indiquant quel processus utilise le processeur à chaque instant, de 0 ms à la fin de l'exécution de tous les processus. À titre d'exemple, on a déjà indiqué sur le chronogramme que le processus P1 s'exécute entre les instants 0 ms et 2 ms. De plus on a indiqué sous l'axe du temps les instants d'arrivée des quatre processus.

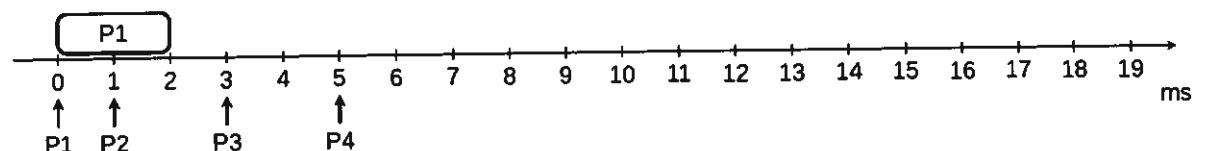


Figure 2. Chronogramme à compléter.

On souhaite modéliser en Python le comportement de l'algorithme du tourniquet. Chaque processus est représenté par un dictionnaire contenant les clés 'nom', 'arrivee' et 'temps', ayant pour valeurs correspondantes le nom (de type string), l'instant d'arrivée (de type int) et le temps d'exécution restant en millisecondes (de type int).

On définit donc ci-dessous quatre variables P1, P2, P3 et P4 représentant les quatre processus considérés ci-avant.

```
1 P1 = {'nom': 'P1', 'arrivee': 0, 'temps': 6}
2 P2 = {'nom': 'P2', 'arrivee': 1, 'temps': 4}
3 P3 = {'nom': 'P3', 'arrivee': 3, 'temps': 5}
4 P4 = {'nom': 'P4', 'arrivee': 5, 'temps': 3}
```

Le quantum est fixé à 2 ms grâce à une variable quantum (de type int).

De plus, on suppose qu'on dispose d'une implémentation de file en Python à travers les fonctions suivantes :

- `creer_file_vide` qui ne prend aucun paramètre et qui renvoie une file vide ;

- `est_vide` qui prend en paramètre une file et qui renvoie un booléen indiquant si cette file est vide ;
 - `enfiler` qui prend en paramètres une file et un élément et qui modifie la file en y ajoutant cet élément en queue (la valeur de retour est `None`) ;
 - `defiler` qui prend en paramètre une file, qui modifie cette file en enlevant son élément en tête et qui renvoie cet élément.
8. Donner les instructions qui permettent de créer une file `fp` contenant les processus `P1`, `P2`, `P3` et `P4`, classés par ordre d'arrivée, le premier arrivé étant en tête de la file.

On souhaite créer une fonction `execute_un_processus` qui réalise une étape de l'algorithme du tourniquet. Plus précisément cette fonction prend en paramètres une file de processus non vide `file_d_attente` et un instant `t` donnant le début de cette étape. Elle extrait de la file le processus à exécuter, puis le remet si besoin dans la file d'attente avec le temps d'exécution restant. De plus cette fonction renvoie l'instant auquel cette étape se termine, qui est soit la fin de l'exécution du processus, soit la fin du quantum.

9. Recopier et compléter le code de la fonction `execute_un_processus` ci-dessous.

```

1 def execute_un_processus (file_d_attente, t):
2     processus = defiler(file_d_attente)
3     if processus['temps'] ... quantum:
4         processus['temps'] = ...
5         ...
6         return t + ...
7     else:
8         return t + ...

```

On se place maintenant dans le cas où tous les processus sont déjà arrivés, autrement dit tous les processus à traiter sont dans la file que l'on prend en entrée. On ne tiendra donc pas compte des temps d'arrivée dans la suite. On souhaite créer une fonction `execute_tous_processus` qui prend en paramètre une file de processus, qui simule le comportement de l'algorithme du tourniquet sur ces processus jusqu'à les avoir tous complètement exécutés, et qui renvoie à quel instant se termine la dernière de ces exécutions. On suppose que l'exécution de ces processus commence à partir de l'instant 0.

10. Recopier et compléter le code de la fonction `execute_tous_processus` ci-dessous.

```
1 def execute_tous_processus (file_d_attente):  
2     t = 0  
3     while ...  
4         t = ...  
5     return t
```