

Gestion des processus et des ressources

EXERCICES

Dernière mise à jour le : 04/03/2024

■ Exercice 1

Partie A

Q1 : Créez un script Python `infini.py` contenant une boucle infinie.

Q2 : Ouvrez un terminal et tapez la commande `top` pour voir les processus exécutés en temps réel.

Q3 : Ouvrez un *autre* terminal et lancez votre script Python avec la commande `python3 infini.py &` (il faut bien sûr être placé dans le repertoire de ce script).

Q4 : Regardez le premier terminal pour voir le processus correspondant à l'exécution de ce script Python. Observez que ce processus monopolise l'ensemble des ressources du processeur.

Q5 : Repérez le PID de ce processus et stoppez son exécution avec la commande `kill PIDduProcessus`.

i En réalité, ce n'est pas l'ensemble de toutes les ressources du processeur qui ont été mobilisée, comme vous allez le voir dans la suite de l'exercice.

Partie B

Q6 : Utilisez la commande `cat /proc/cpuinfo` et repérez le nombre de "processeurs" disponibles sur votre machine.

i Bien qu'il soit écrit `processor` pour chacun d'eux, cela ne correspond pas exactement à des processeurs. Votre ordinateur possède en réalité un processeur multicoeur (il y en a 2), ce qui lui permet d'exécuter des tâches en parallèle sur chacun de ses coeurs. Ces derniers peuvent eux-mêmes traiter des tâches (ou *threads*) en parallèle (2 également). Votre processeur est donc un "2 coeurs / 4 threads" : cela signifie qu'il peut paralléliser 4 tâches simultanément, réparties sur les 2 coeurs du processeur.

Q7 : Créez un autre script Python appelé `bidon.py` contenant le code suivant :

```
from timeit import timeit

def bidon():
    a = 0
    for i in range(100000):
        a = a**3

# pour lancer 100 fois la fonction bidon et renvoyer le temps d'exécution
temps = timeit(bidon, number=100)
print(temps)
```

Q8 : Exécutez ce script dans un terminal via la commande `python3 bidon.py` et notez le temps d'exécution des 100 appels à la fonction `bidon`.

Q9 : Dans un *autre* terminal, tapez la commande `python3 infini.py &` autant de fois qu'il y a de "processeurs" sur la machine. On va ainsi monopoliser l'ensemble des ressources processeurs de la machine avec des boucles infinies (cette fois-ci toutes les ressources !).

Q10 : Dans le premier terminal, relancez le script `bidon.py`. Comment expliquer le ralentissement observé ?

! **Attention** : N'oubliez pas de stopper les 4 processus infinis en exécutant `kill PID1 PID2 PID3 PID4` où `PIDx` sont les PID des 4 processus correspondants au script `infini.py`.

■ Exercice 2

Lancez Firefox, ouvrez deux ou trois onglets et chargez une page Web dans chacun d'eux.

1. Utilisez la commande `ps -aef` pour repérer les processus correspondant à l'exécution de Firefox et répondez aux questions suivantes :
 - Combien de processus ont été créés ? Vous donnerez leurs PID.
 - Quel processus est le père de tous les autres ? Expliquez.
2. Vérifiez en exécutant la commande `pidof firefox` qui donne le PID du processus père.
3. Essayez la commande `kill PIDduProcessus` sur un des processus fils. Que constatez-vous ? Et sur le processus père ?

■ Exercice 3 : Algorithmes d'ordonnancement

Soient les différents processus suivants :

Processus	Date d'arrivée	Durée de traitement
P_1	0	3
P_2	1	6
P_3	3	4
P_4	6	5
P_5	8	2

Application de plusieurs algorithmes

Q1 : Donnez le diagramme de Gantt pour l'exécution de ces différents processus en utilisant successivement les algorithmes FCFS, RR (quantum = 2 unités de temps) et SRT.

Performances des algorithmes d'ordonnancement

On définit les métriques suivantes :

- le **temps de séjour (ou d'exécution)** (ou de rotation) d'un processus : c'est la différence entre la date de fin d'exécution et la date d'arrivée :

$$T_{\text{sej}} = \text{date fin d'exécution} - \text{date d'arrivée}$$

- le **temps d'attente** d'un processus : c'est la différence entre le temps de séjour et la durée du processus :

$$T_{\text{att}} = T_{\text{sej}} - \text{durée du processus}$$

- le **rendement** d'un processus : c'est le quotient entre la durée du processus et le temps de séjour :

$$\text{rendement} = \frac{\text{durée du processus}}{T_{\text{sej}}}$$

Q2 : Pour chacun des trois algorithmes, calculez le temps de séjour, le temps d'attente et le rendement de chaque processus.

Q3 : Quel vous semble être le *meilleur* des trois algorithmes dans notre exemple ? Expliquez.

■ Exercice 4 - Un problème de synchronisation

Voici un script Python dans lequel on crée une variable globale `nombre` qui vaut 0 au départ et à laquelle on ajoute 100 quatre fois successivement grâce à la fonction `ajoute_100`.

```
import time

def ajoute_100():
    global nombre
    for i in range(100):
        time.sleep(0.001) # pour simuler un traitement avec des calculs
        nombre = nombre + 1

if __name__ == '__main__':
    nombre = 0
    for i in range(4):
        ajoute_100()
    print("valeur finale :", nombre)
```

Q1 : Copiez et exécutez le script dans un terminal pour vérifier que la valeur finale de la variable `nombre` est bien égale à 400.

On va maintenant supposer qu'une telle variable est partagée par 4 processus, chacun étant chargé d'ajouter 100 à cette variable.

On a vu dans le cours le programme suivant qui permettait d'illustrer les problèmes de synchronisation dans le cas de ressources partagées (ici une variable partagée) entre plusieurs processus.

```
from multiprocessing import Process, Value
import time

def prendre_un_pion(nombre):
    if nombre.value >= 1:
        time.sleep(0.0005) # pour simuler un traitement avec des calculs
        temp = nombre.value
        nombre.value = temp - 1 # on décrémente le nombre de pions

if __name__ == '__main__':
    # création de la variable partagée initialisée à 1
    nb_pions = Value('i', 1)
    # on crée deux processus
    p1 = Process(target=prendre_un_pion, args=[nb_pions])
    p2 = Process(target=prendre_un_pion, args=[nb_pions])
    # on démarre les deux processus
    p1.start()
    p2.start()
    # on attend la fin des deux processus
    p1.join()
    p2.join()
    print("nombre final de pions :", nb_pions.value)
```

Q2 : Inspirez-vous de ce programme pour créer un script permettant :

- de créer une variable partagée entière appelée `nombre_partage` et initialisée à 0
- de créer 4 processus ayant pour rôle d'exécuter une fonction `ajouter_100` à laquelle on passe `nombre_partage` en argument :
 - la fonction `ajouter_100(nombre)` doit ajouter 100 à la variable partagée `nombre` en utilisant une boucle `for` qui incrémente 100 fois d'une unité la variable
 - on laissera une temporisation pour simuler d'autres calculs
- de démarrer les 4 processus et d'attendre la fin de leur exécution
- d'afficher la valeur finale de la variable `nombre_partage`

Q3 : Exécutez ce script dans un terminal et observez que la valeur finale de la variable `nombre_partage` n'est pas (toujours) égale à 400 comme on pourrait s'y attendre. Comment peut-on expliquer cela ?

Q4 : Ajoutez des affichages dans la fonction `ajouter_100` pour observer ce qu'il se passe. Vous afficherez le numéro du processus en cours d'exécution et la valeur de la variable partagée à la fin de chaque tour de boucle.

Q5 : Utilisez ensuite un verrou pour régler ce problème de synchronisation en protégeant la section critique de la fonction `ajouter_100`. Vérifiez que la valeur finale est toujours égale à 400.

■ Exercice 5

d'après le sujet du bac NSI 2021

1) La commande `ps` suivie éventuellement de diverses options permet de lister les processus actifs ou en attente sur une machine. Sur une machine équipée du système d'exploitation GNU/Linux, la commande "`ps -aef`" permet d'obtenir la sortie suivante (extrait) :

UID	PID	PPID	C	STIME	TTY	TIME	CMD
root	1	0	0	10:01	?	00:00:02	/sbin/init splash
root	4	2	0	10:01	?	00:00:00	[kworker/0:0H]
root	6	2	0	10:01	?	00:00:00	[mm_percpu_wq]
....
bob	3383	1	0	10:25	?	00:00:00	sh -c /usr/bin/google-chrome-sta

a) Quelle est la particularité de l'utilisateur "root" ?

b) Quel est le processus parent du processus ayant pour PID 3383

Dans un bureau d'architectes, on dispose de certaines ressources qui ne peuvent être utilisées simultanément par plus d'un processus, comme l'imprimante, la table traçante, le modem. Chaque programme, lorsqu'il s'exécute, demande l'allocation des ressources qui lui sont nécessaires. Lorsqu'il a fini de s'exécuter, il libère ses ressources.

Programme 1	Programme 2	Programme 3
demander(table traçante) demander (modem) exécution libérer (modem) libérer (table traçante)	demander (modem) demander (imprimante) exécution libérer (imprimante) libérer (modem)	demander (imprimante) demander (table traçante) exécution libérer (table traçante) libérer (imprimante)

2) On appelle p1, p2 et p3 les processus associés respectivement aux programmes 1, 2 et 3.

- Justifier qu'une situation d'interblocage peut se produire.
- Modifier l'ordre des instructions du programme 3 pour qu'une telle situation ne puisse pas se produire.

■ Exercice 6

Cet exercice est issu du sujet 2021 du bac NSI

Partie A

Cette partie est un questionnaire à choix multiples (QCM). Pour chacune des questions, une seule des quatre réponses est exacte. Le candidat indiquera sur sa copie le numéro de la question et la lettre correspondant à la réponse exacte. Aucune justification n'est demandée. Une réponse fautive ou une absence de réponse n'enlève aucun point.

1) Parmi les commandes ci-dessous, laquelle permet d'afficher les processus en cours d'exécution ?

- dir
- ps
- man
- ls

2) Quelle abréviation désigne l'identifiant d'un processus dans un système d'exploitation de type UNIX ?

- PIX
- SIG
- PID
- SID

3) Comment s'appelle la gestion du partage du processeur entre différents processus ?

- L'interblocage
- L'ordonnancement
- La planification
- La priorisation

4) Quelle commande permet d'interrompre un processus dans un système d'exploitation de type UNIX ?

- stop
- interrupt
- end
- kill

Partie B

1) Un processeur choisit à chaque cycle d'exécution le processus qui doit être exécuté. Le tableau ci-dessous donne pour trois processus P1, P2, P3 :

- la durée d'exécution (en nombre de cycles),
- l'instant d'arrivée sur le processeur (exprimé en nombre de cycles à partir de 0),
- le numéro de priorité.

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	Temps de séjour	Temps d'attente
P1	3	0	$14 - 0 = 14$	$14 - 3 = 11$
P2	4	2		
P3	3	3		
P4	4	5		

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

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 d'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.

2. Comment s'appelle cette situation ?

3. On suppose que l'analyseur d'échantillon libère la ressource D1. Donner un ordre possible d'exécution des processus.

Références :

- Equipe éducative DIU EIL, cours et exercices sur le *Partage des ressources et virtualisation*, Audrey Queudet, Université de Nantes.
- Cours d'Olivier Lecluse sur la [Gestion des ressources](#)
- Documentation officielle de la bibliothèque [multiprocessing](#) de Python.
- Dépôt GitHub de David Roche pour les [énoncés](#) des exercices 5 et 6 au format Markdown (extrait d'exercices de bac).

Germain BECKER, Lycée Mounier, ANGERS

Ressource éducative libre distribuée sous [Licence Creative Commons Attribution - Partage dans les Mêmes Conditions 4.0 International](#)



Voir en ligne : info-mounier.fr/terminale_nsi/archi_se_reseaux/gestion-processus-ressources-exercices