Exercices type bac

Systèmes d'exploitation et processus

Exercice 1 *Métropole 2021 (4 points)*

Cet exercice porte sur les systèmes d'exploitation : gestion des processus et des ressources.

Les parties A et B peuvent être traitées indépendamment.

Partie A

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)	demander (imprimante)
demander (modem)	demander (imprimante)	demander (table traçante)
exécution	exécution	exécution
libérer (modem)	libérer (imprimante)	libérer (table traçante)
libérer (table traçante)	libérer (modem)	libérer (imprimante)

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

1. Les processus s'exécutent de manière concurrente. Justifier qu'une situation d'interblocage peut se produire.
2. Modifier l'ordre des instructions du programme 3 pour qu'une telle situation ne puisse pas se produire. Aucune justification n'est attendue.
3. Supposons que le processus p1 demande la table traçante alors qu'elle est en cours d'utilisation par le processus p3. Parmi les états suivants, quel sera l'état du processus p1 tant que la table traçante n'est pas disponible :

   a) élu ;

   b) bloqué ;

   c) prêt ;

   d) terminé.

Partie B

Avec une ligne de commande dans un terminal sous Linux, on obtient l'affichage suivant :

```
UID    PID  PPID C STIME TTY    TIME     CMD
```

```
...
pi    6211   831 8 09:07 ?  00:01:16 chromium-browser ...
pi    6252  6211 0 09:07 ?  00:00:00 chromium-browser --type=zygote ...
pi    6254  6252 0 09:07 ?  00:00:00 chromium-browser --type=zygote ...
pi    6294  6211 4 09:07 ?  00:00:40 chromium-browser --type=gpu-process ...
pi    6300  6211 1 09:07 ?  00:00:16 chromium-browser --type=utility ...
pi    6467  6254 1 09:07 ?  00:00:11 chromium-browser --type=renderer ...
pi   12835   836 0 09:13 ?  00:00:00 oosplash ...
pi   12073 12835 2 09:13 ?  00:00:15 soffice.bin ...
pi   12253   831 1 09:13 ?  00:00:07 sense_emu_gui
pi   20029  6254 56 09:21 ?  00:00:28 chromium-browser --type=renderer ...
pi   20519   676 0 09:22 pts/0  00:00:00 ps -ef
```

La documentation Linux donne la signification des différents champs : UID (identifiant utilisateur), PID (identifiant de processus), PPID (PID du processus parent), C (pourcentage d'utilisation du processeur), STIME (heure de lancement), TTY (terminal de contrôle), TIME (temps d'exécution), CMD (nom de la commande).

1. Parmi les quatre commandes suivantes, laquelle a permis cet affichage ?

   a) `ls -l`

   b) `ps -ef`

   c) `cd ..`

   d) `chmod 741 processus.txt`

2. Quel est l'identifiant du processus parent à l'origine de tous les processus concernant le navigateur Web (chromium-browser) ?

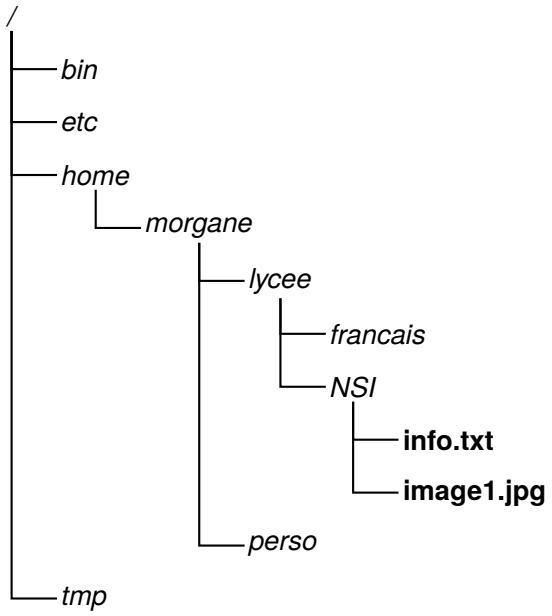
3. Quel est l'identifiant du processus dont le temps d'exécution est le plus long ?

Exercice 2 *Amérique du Nord 2022 (4 points)*

Cet exercice porte sur les systèmes d'exploitation et la gestion des processus par un système d'exploitation.

Cet exercice pourra utiliser des commandes de systèmes d'exploitation de type UNIX telles que `cd`, `ls`, `mkdir`, `rm`, `rmdir`, `mv`, `cat`.

1. Dans un système d'exploitation de type UNIX, on considère l'arborescence suivante :



On suppose qu'on se trouve actuellement à l'emplacement `/home/morgane`.

a) Parmi les quatre propositions suivantes, donner celle correspondant à l'affichage obtenu lors de l'utilisation de la commande `ls` :

- Proposition 1 : `lycee francais NSI info.txt image1.jpg perso`
- Proposition 2 : `lycee perso`
- Proposition 3 : `morgane`
- Proposition 4 : `bin etc home tmp`

b) Écrire la commande qui permet, à partir de cet emplacement, d'atteindre le répertoire `lycee`.

On suppose maintenant qu'on se trouve dans le répertoire `/home/morgane/lycee/NSI`.

c) Écrire la commande qui permet de créer à cet emplacement un répertoire nommé `algorithmique`.

d) Écrire la commande qui permet, à partir de cet emplacement, de supprimer le fichier `image1.jpg`.

2. On rappelle qu'un processus est une instance d'application. On a exécuté la commande `ps` (avec quelques options). Un extrait du résultat est présenté ci-dessous :

```
UID   PID  PPID  C STIME TTY   TIME     CMD
test   900   739  0 10:51 ?  00:00:00 gvfs-udisks2-vol
test   907   838  0 10:51 ?  00:00:00 gvfsd-trash --sp
test   913   739  0 10:51 ?  00:00:00 gvfsd-metadata
test   923     1  0 10:51 ?  00:00:02 xfce4-terminal
test   927   923  0 10:51 pts/0  00:00:00 bash
test  1036     1  0 11:18 ?  00:00:02 mousepad
test  1058   923  0 11:22 pts/1  00:00:00 bash
test  1153   927  0 11:44 pts/0  00:00:00 vi
test  1154   927  0 11:44 pts/0  00:00:00 python3 prog.py
test  1155  1058  0 11:44 pts/1  00:00:00 ps -aef
```

a) Donner le PID du parent du processus démarré par la commande `vi`.

b) Donner le PID d'un processus enfant du processus démarré par la commande `xfce4-terminal`.

c) Citer le PID de deux processus qui ont le même parent.

d) Parmi tous les processus affichés, citer le PID des deux qui ont consommé le plus de temps du processeur.

3. On considère les trois processus P1, P2 et P3, tous soumis à l'instant 0 dans l'ordre 1, 2, 3 :

Nom du processus	Durée d'exécution	Ordre de soumission
P1	3	1
P2	1	2
P3	4	3

a) On considère que les processus sont exécutés de manière concurrente selon la politique du tourniquet (Round Robin) : le temps est découpé en quantums de temps. Le quantum correspond à une unité de temps. Reproduire le tableau ci-dessous et indiquer dans chacune des cases le processus exécuté par le processeur entre deux unités de temps.

	0–1	1–2	2–3	3–4	4–5	5–6	6–7	7–8
Processus	P1							

b) On considère que les processus sont exécutés en appliquant la politique du « plus court d'abord » (SJF) : les processus sont exécutés complètement dans l'ordre croissant de leurs temps d'exécution. Reproduire et compléter le même tableau.

4. On considère trois ressources R1, R2 et R3 et trois processus P1, P2 et P3 dont les files d'exécution des instructions élémentaires sont indiquées ci-dessous :

Processus P1	Processus P2	Processus P3
Demande R1	Demande R2	Demande R3
Demande R2	Demande R3	Demande R1
Libère R1	Libère R2	Libère R3
Libère R2	Libère R3	Libère R1

a) Rappeler les différents états d'un processus et expliquer pourquoi il y a ici risque d'interblocage, en proposant un ordre d'exécution des instructions élémentaires le provoquant.

b) Proposer un ordre d'exécution des instructions élémentaires sans interblocage.

Exercice 3 *Amérique du Sud 2022 (4 points)*

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.

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. Plus la priorité est grande plus le numéro de priorité est petit. Ainsi le processus P3 est plus prioritaire que P1.

L'ordonnancement est de type **préemptif** : à chaque unité de temps, le processeur choisit d'exécuter le processus ayant le plus petit numéro de priorité. Ceci peut provoquer la suspension d'un autre processus qui reprendra lorsqu'il deviendra le plus prioritaire.

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

1. Reproduire 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).

	0–1	1–2	2–3	3–4	4–5	5–6	6–7	7–8	8–9	9–10	10–11	11–12	12–13	13–14	14–15
Processus	P1														

2. Recopier et compléter les temps de séjour ainsi que les temps d'attente de chacun des processus.

   Rappel : temps de séjour = instant de terminaison − instant d'arrivée ; temps d'attente = temps de séjour − temps d'exécution.

Processus	P1	P2	P3	P4
Temps d'exécution	3	4	3	4
Instant d'arrivée	0	2	3	5
Temps de séjour	14 − 0 = 14			
Temps d'attente	14 − 3 = 11			

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

B. Processus et ressources

Dans ce laboratoire, le laborantin utilise simultanément quatre logiciels : un logiciel d'analyse d'échantillons (connecté à l'analyseur), un logiciel d'accès à la base de données des patients (SGBD), un traitement de texte et un tableur.

Le tableau ci-dessous donne l'état à un instant donné des différents processus : M signifie que le processus mobilise la donnée, A qu'il est en attente de la donnée, et − qu'il ne la sollicite pas. Une donnée ne peut être mobilisée que par un seul processus à la fois.

	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.
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.

Exercice 4 *Polynésie 2023 (4 points)*

Cet exercice porte sur la gestion des processus et la programmation orientée objet.

On rappelle qu'un processus est l'instance d'un programme en cours d'exécution. Il est identifié par un numéro unique appelé PID. L'ordonnanceur est la composante du système d'exploitation qui gère l'allocation du processeur entre les différents processus.

Nous allons nous intéresser à l'algorithme d'ordonnancement du **tourniquet** dont le fonctionnement est résumé ci-dessous :

— les processus prêts à être exécutés sont placés dans une file d'attente selon leur ordre d'arrivée ;

— l'ordonnanceur alloue le processeur à chaque processus de la file d'attente un même nombre de cycles CPU, appelé **quantum** ;

— si le processus n'est pas terminé au bout de ce temps, son exécution est suspendue et il est mis à la fin de la file d'attente ;

— si le processus est terminé, il sort définitivement de la file d'attente.

1. On considère trois processus soumis à l'ordonnanceur **au même instant** :

PID	Durée (en cycles CPU)	Ordre d'arrivée
11	4	1
20	2	2
32	3	3

a) Si le quantum du tourniquet est d'**un** cycle CPU, recopier et compléter la suite des PID des processus dans l'ordre de leur exécution : `11, 20, 32, 11, …`

b) Donner la composition de la suite des PID lorsque le quantum du tourniquet est de **deux** cycles CPU.

2. L'objectif est d'implémenter en Python l'algorithme du tourniquet. On utilise la classe `Processus` dont le constructeur est :

```
class Processus:
    def __init__(self, pid, duree):
        self.pid = pid
        self.duree = duree
        self.reste_a_faire = duree
        self.etat = "Prêt"
```

Les états possibles sont : « Prêt », « En cours d'exécution », « Suspendu » et « Terminé ».

a) Recopier et compléter l'instruction Python suivante permettant de créer la liste d'attente initiale :

```
liste_attente = [Processus(..., ...), ..., ...]
```

b) Recopier et compléter les trois méthodes suivantes de la classe `Processus` :

```
def execute_un_cycle(self):
    ...

def change_etat(self, nouvel_etat):
    ...

def est_termine(self):
    ...
```

c) La fonction `tourniquet` ci-dessous implémente l'algorithme. Recopier et compléter le code manquant :

```
def tourniquet(liste_attente, quantum):
    ordre_execution = []
    while liste_attente != []:
        processus = liste_attente.pop(0)
        processus.change_etat("En cours d'exécution")
        compteur_tourniquet = 0
        while ..... and .....:
            ordre_execution.append(.....)
            processus.execute_un_cycle()
            compteur_tourniquet = compteur_tourniquet + 1
        if .....:
            processus.change_etat("Suspendu")
            liste_attente.append(processus)
        else:
            processus.change_etat(.....)
    return ordre_execution
```

Exercice 5 *Métropole 2025, sujet 1 (6 points)*

Cet exercice porte sur l'algorithmique, les structures de données et la gestion de processus.

On cherche à créer une application de type *liste de tâches* à faire pour aider Alice à planifier sa journée. Pour cela Alice saisit les informations concernant chacune des tâches qu'elle doit effectuer : elle indique un nom pour la tâche, ainsi que la durée qu'elle estime nécessaire afin de la réaliser. On représente une tâche saisie par Alice à l'aide d'un objet de type `Tache`, muni de quatre attributs :

— le `numero` de la tâche, saisi par Alice ;

— le `nom` de la tâche, saisi par Alice ;

— la `duree` (un entier exprimé en minutes) nécessaire à la réalisation de la tâche, saisie par Alice ;

— la `duree_restante` (un entier exprimé en minutes) avant la fin de la tâche. Cet attribut sera initialisé avec la durée totale nécessaire à la réalisation de la tâche.

Avancer de `n` minutes (`n` entier positif) dans une tâche consiste à diminuer de `n` la durée restante de cette tâche. Une tâche est terminée si la durée restante est négative ou nulle.

Lors de la phase de planification de ses tâches (aucune d'entre elles n'est commencée), Alice liste les tâches suivantes qui doivent être effectuées :

Numéro	Nom	Durée	Durée restante
1	Répondre aux e-mails	45	45
2	Ranger ma chambre	60	60
3	Réviser la NSI	90	90
4	S'entraîner aux échecs	30	30
5	Apprendre le vocabulaire de chinois	30	30
6	Lire *Fondation*	60	60
7	Écrire ma lettre au Père Noël	20	20

On dispose de la classe `Tache` ci-dessous pour représenter les tâches :

```
class Tache:
    def __init__(self, numero, nom, duree):
        self.numero = numero
        self.nom = nom
        self.duree_initiale = duree
        self.duree_restante = duree

    def __repr__(self):
        return '<t' + str(self.numero) + '>'
```

1. Donner le code Python qui permet d'instancier deux variables `tache1` et `tache2` représentant les tâches suivantes :

— tâche numéro 1 : Répondre aux e-mails. Durée estimée : 45 minutes ;

— tâche numéro 2 : Ranger ma chambre. Durée estimée : 60 minutes.

On supposera dans la suite que les variables `tache1`, `tache2`, ..., `tache7` représentent les tâches établies par Alice lors de la phase de planification.

La méthode `__repr__` renvoie une représentation de l'instance sous forme d'une chaîne de caractères. La fonction `print` utilise cette méthode. Ainsi on a :

```
>>> print(tache1)
<t1>
```

2. Recopier et compléter le code de la méthode `avancer` de la classe `Tache` qui permet d'avancer la tâche `self` de `n` minutes.

```
def avancer(self, n):
    ...
```

3. Recopier et compléter le code de la méthode `est_terminee` de la classe `Tache` qui renvoie `True` si la tâche est terminée, ou `False` sinon.

```
def est_terminee(self):
    ...
```

Afin d'aider Alice à planifier sa journée, on lui propose d'associer à chacune des tâches une priorité. La priorité d'une tâche est représentée par un entier de la manière suivante : 1 est la priorité minimale et, plus le nombre est grand, plus la tâche associée est prioritaire.

Pour stocker toutes les tâches à effectuer, on utilise une file, dans laquelle les éléments sont des tuples `(tache, priorite)`. Les éléments stockés dans la file doivent respecter les deux conditions ci-après.

— Condition 1 : les éléments sont rangés par ordre décroissant de priorité. L'élément de priorité maximale se trouve au début de la file, l'élément le moins prioritaire se trouve à la fin de la file.

— Condition 2 : parmi les éléments de même priorité, les éléments sont rangés dans l'ordre dans lequel ils ont été insérés dans la file. Ainsi, le premier élément de priorité p inséré se trouve devant les éléments de même priorité p insérés plus tard.

Par exemple, si la file de tâches `f` est la file :

`[debut] (<t3>, 4) (<t1>, 3) (<t2>, 3) (<t4>, 1) (<t5>, 1) [fin]`

cela signifie que :

— la tâche de priorité maximale est la tâche numéro 3 ;

— les deux tâches à exécuter en priorité après la tâche numéro 3 sont les tâches numéro 1 et numéro 2. La tâche numéro 1 a été ajoutée à la file des tâches à traiter avant la tâche numéro 2 ;

— il n'y a pas de tâche de priorité 2 ;

— les tâches les moins prioritaires de la file sont les tâches numéro 4 et numéro 5. La tâche numéro 4 a été ajoutée avant la tâche numéro 5.

4. Représenter l'état de la file `f` lorsqu'on lui ajoute successivement la tâche numéro 6 avec la priorité 2, puis la tâche numéro 7 avec la priorité 4, en respectant les conditions 1 et 2 décrites ci-dessus.

On suppose déjà définies les méthodes suivantes pour la classe `File` :

— `File()` : crée et renvoie un objet de type `File`, vide ;

— `enfiler(self, e)` : ajoute l'élément `e` à la fin de la file ;

— `defiler(self)` : renvoie, en le supprimant de la file, le premier élément de la file si cela est possible ;

— `examiner(self)` : renvoie, sans le supprimer de la file, le premier élément de la file si cela est possible ;

— `est_vide(self)` : renvoie `True` si la file est vide, ou `False` sinon.

5. En repartant de la file `f` suivante :

`[debut] (<t3>, 4) (<t1>, 3) (<t2>, 3) (<t4>, 1) (<t5>, 1) [fin]`

donner la valeur de `f.defiler()[0]`, et représenter le contenu de la file `f` après l'exécution de cette instruction.

6. En repartant de la même file `f`, donner la valeur de `f.examiner()[1]`, et représenter le contenu de la file `f` après l'exécution de cette instruction.

On souhaite écrire une fonction `ajouter_file_prio` qui prend en paramètres :

— une file `f` dont les éléments sont des tuples `(tache, priorite)` respectant les deux conditions de l'énoncé ;

— une tâche `t` ;

— la priorité `p` de la tâche `t` ;

et qui ajoute le tuple `(t, p)` à la bonne position dans la file `f`.

On utilise une file auxiliaire `f_aux` que l'on remplit en défilant les éléments en début de file `f` tant que la priorité du premier élément de la file est supérieure ou égale à `p`. Puis on enfile l'élément `(t, p)` dans la file auxiliaire. On défile ensuite tous les éléments restants de `f` dans `f_aux` et enfin on enfile dans `f` tous les éléments de `f_aux`.

7. Recopier et compléter le code de la fonction `ajouter_file_prio`.

```
1 def ajouter_file_prio(f, t, p):
2     f_aux = File()
3     while ...:
4         ...
```

```
    ...enfiler(...)
    while not ...:
        ...
    while not ...:
        ...
```

8. Donner le coût d'exécution temporel dans le pire des cas de la fonction `ajouter_file_prio`, en fonction du nombre `m` d'éléments de la file `f`.

Une fois qu'Alice a entré les tâches qu'elle doit effectuer, leur durée estimée, ainsi que la priorité à laquelle elle doit les effectuer, l'application lui propose un planning en utilisant la technique dite Pomodoro :

— la tâche à effectuer est la tâche qui se trouve en tête de file ;

— on défile cette tâche de la file des tâches à effectuer ;

— on avance cette tâche de 25 minutes ;

— si cette tâche n'est pas terminée, on rajoute cette tâche dans la file des tâches à effectuer, avec la même priorité qu'initialement (en utilisant la fonction `ajouter_file_prio`) ;

— si cette tâche se termine au cours des 25 minutes, alors Alice attend la fin des 25 minutes en se reposant ;

— on continue ces étapes tant que la file des tâches à effectuer n'est pas vide.

On rappelle les tâches à effectuer ci-dessous, classées par ordre de priorité. On considérera que les tâches sont ajoutées à la file de priorité dans l'ordre du tableau ci-dessous :

Numéro	Nom	Durée	Priorité
3	Réviser la NSI	90	4
7	Écrire ma lettre au Père Noël	20	4
1	Répondre aux e-mails	45	3
2	Ranger ma chambre	60	3
6	Lire *Fondation*	60	2
4	S'entraîner aux échecs	30	1
5	Apprendre le vocabulaire de chinois	30	1

9. Indiquer pour chaque bloc de 25 minutes la tâche qui avance, en suivant le modèle proposé, jusqu'à la fin de toutes les tâches. On fera particulièrement attention au cas où la tâche n'est pas terminée : celle-ci est rajoutée à la file des tâches à effectuer (dont elle avait été supprimée) avec la même priorité qu'initialement, en respectant les conditions 1 et 2 de l'énoncé.

10. Écrire le code d'une fonction `planning` qui prend en paramètre une file de priorité `f` dont les éléments sont des tuples `(tache, prio)`, et qui renvoie une liste de tâches, dans l'ordre dans lequel elles vont être effectuées par tranche de 25 minutes avec la méthode Pomodoro. Par exemple, si `tache1`, `tache2` et `tache3` sont les tâches numéro 1, numéro 2 et numéro 3, alors le programme suivant :

```
file = File()
for t, p in [(tache1, 3), (tache2, 3), (tache3, 4)]:
    ajouter_file_prio(file, t, p)
print(planning(file))
```

produit l'affichage : `[<t3>, <t3>, <t3>, <t3>, <t1>, <t2>, <t1>, <t2>, <t2>]`