Exercices - Architecture ordinateur- OS

EXERCICE 1. :

Soit le jeu de tâches ci-dessous qui mélange des tâches périodiques et des tâches ponctuelles.

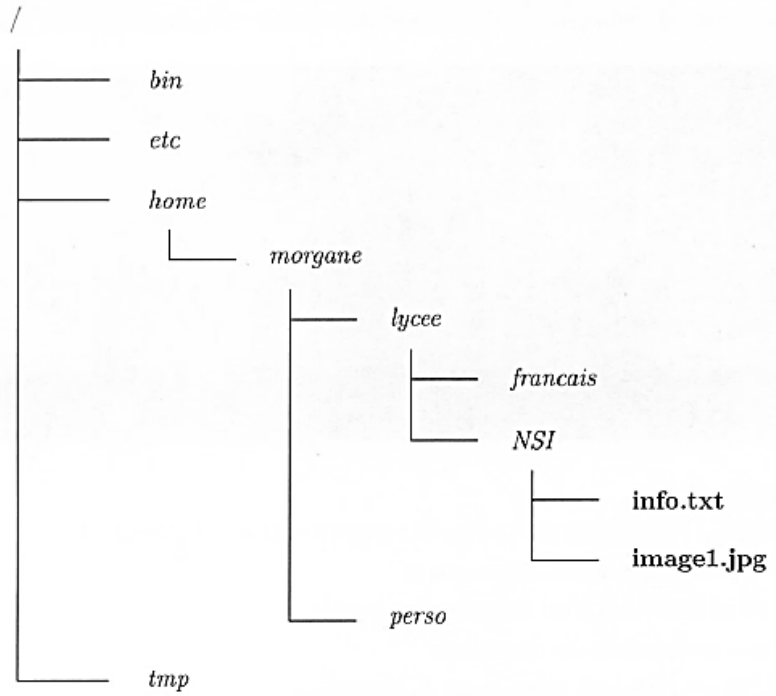
Tâche	Date(s) d'arrivée(s)	Priorité	Durée	Remarque
A	0, 6, 12, 18, 24, 30 ...	10	1	Périodique
B	0, 10, 20, 30 ...	8	4	Périodique
C	21	9	6	Ponctuelle
D	0	1	7	Ponctuelle

1- Faire l'ordonnancement de ces tâches sur 32 unités de temps (ordonnancement préemptif avec priorité)
2- Quel est la durée la plus longue de chaque tâche (Δ entre temps final et temps début prévu) ?

EXERCICE 2. : SUJET BAC 2022

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

1. Dans un système d'exploitation de type UNIX, on considère l'arborescence des fichiers suivante dans laquelle les noms de dossiers sont en italique et ceux des fichiers sont en gras :



On souhaite, grâce à l'utilisation du terminal de commande, explorer et modifier les répertoires et fichiers présents.
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. Un processus peut être démarré par l'utilisateur, par un périphérique ou par un autre processus appelé parent.

La commande UNIX `ps` présente un cliché instantané des processus en cours d'exécution.

On a exécuté la commande `ps` (avec quelques options qu'il n'est pas nécessaire de connaître pour la réussite de cet exercice). Un extrait du résultat de la commande est présenté ci-dessous :

```
UID          PID  PPID  C STIME TTY          TIME CMD
test         900   739  0 10:51 ?        00:00:00 /usr/lib/gvfs/gvfs-udisks2-vol
test         907   838  0 10:51 ?        00:00:00 /usr/lib/gvfs/gvfsd-trash --sp
test         913   739  0 10:51 ?        00:00:00 /usr/lib/gvfs/gvfsd-metadata
test         918   823  0 10:51 ?        00:00:00 /usr/lib/x86_64-linux-gnu/xfce
test         919   823  0 10:51 ?        00:00:00 /usr/lib/x86_64-linux-gnu/xfce
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 /home/test/Documents/
test        1058   923  0 11:22 pts/1    00:00:00 bash
root        1132     2  0 11:37 ?        00:00:00 [kworker/0:0-ata_sff]
root        1134     2  0 11:43 ?        00:00:00 [kworker/0:2-ata_sff]
test        1140   739  0 11:43 ?        00:00:00 /usr/lib/x86_64-linux-gnu/tumb
root        1149     2  0 11:43 ?        00:00:00 [kworker/u2:0-events_unbound]
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
```

On rappelle que :

— l'`UID` est l'identifiant de l'utilisateur propriétaire du processus ;

— le `PID` est l'identifiant du processus ;

— le `PPID` est l'identifiant du processus parent ;

— `C` indique l'utilisation processeur ;

— `STIME` est l'heure de démarrage du processus ;

— `TTY` est le nom du terminal de commande auquel le processus est attaché ;

— `TIME` est la durée d'utilisation du processeur par le processus ;

— `CMD` le nom de commande utilisé pour démarrer le processus.

(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.

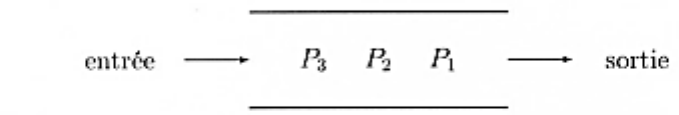
3. On considère les trois processus P1, P2 et P3, tous soumis à l'instant 0 dans l'ordre 1, 2, 3 :

Nom du processus	Durée d'exécution en unité de temps	Ordre de soumission
P_1	3	1
P_2	1	2
P_3	4	3

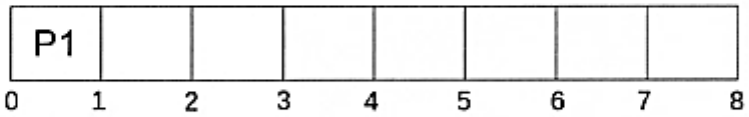
(a) Dans cette question, on considère que les processus sont exécutés de manière concurrente selon la politique du tourniquet : le temps est découpé en tranches nommées *quantums de temps*.

Les processus prêts à être exécutés sont placés dans une file d'attente selon leur ordre de soumission.

Lorsqu'un processus est élu, il s'exécute au plus durant un quantum de temps. Si le processus n'a pas terminé son exécution à l'issue du quantum de temps, il réintègre la file des processus prêts (côté entrée). Un autre processus, désormais en tête de la file (côté sortie) des processus prêts, est alors à son tour élu pour une durée égale à un quantum de temps maximum.

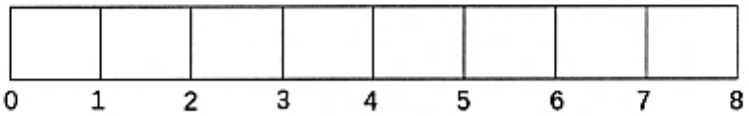


Reproduire le tableau ci-dessous sur la copie et indiquer dans chacune des cases le processus exécuté à chaque cycle. Le quantum correspond à une unité de temps.



(b) Dans cette question, on considère que les processus sont exécutés en appliquant la politique du « plus court d'abord » : les processus sont exécutés complètement dans l'ordre croissant de leurs temps d'exécution, le plus court étant exécuté en premier.

Reproduire le tableau ci-dessous sur la copie et indiquer dans chacune des cases le processus exécuté à chaque cycle.



4. On considère trois ressources R_1, R_2 et R_3 et trois processus P_1, P_2 et P_3 dont les files d'exécution des instructions élémentaires sont indiquées ci-dessous :

Processus P_1
Demande R_1
Demande R_2
Libère R_1
Libère R_2

Processus P_2
Demande R_2
Demande R_3
Libère R_2
Libère R_3

Processus P_3
Demande R_3
Demande R_1
Libère R_3
Libère R_1

(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) Montrer qu'en échangeant les 2 premières lignes des actions du Processus P3, l'interblocage est évité

Processus P_1
Demande R_1
Demande R_2
Libère R_1
Libère R_2

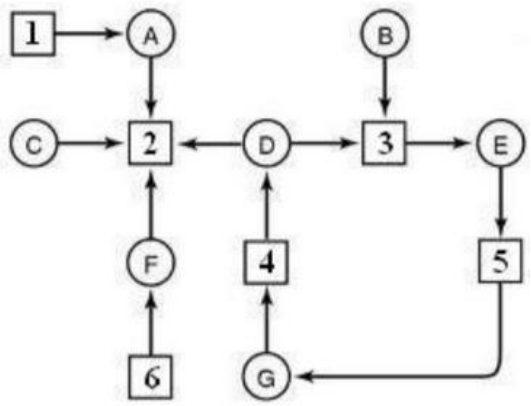
Processus P_2
Demande R_2
Demande R_3
Libère R_2
Libère R_3

Processus P_3
Demande R1
Demande R3
Libère R_3
Libère R_1

EXERCICE 3. :

Le tableau ci-dessous résume les possessions et demandes des processus A et G à un instant t. On donne aussi le graphe correspondant

Processus	Ressources demandées	ressources détenues
A	2	1
B	3	
C	2	
D	2 et 3	4
E	5	3
F	2	6
G	4	5



Question : Y-a-til une situation d'interblocage à cet instant ?

EXERCICE 4. :

Sept processus Pi sont dans la situation suivante par rapport aux ressources Ri :

- P1 a obtenu R1 et demande R2
- P2 demande R3 et n'a obtenu aucune ressource tout comme P3 qui demande R2
- P4 a obtenu R2 et R4 puis demande R3
- P5 a obtenu R3 et demande R5
- P6 a obtenu R6 et demande R2
- P7 a obtenu R5 et demande R2.

Question : Construire un graphe orienté. Y-a-til une situation d'interblocage à cet instant ?

EXERCICE 5. : SUJET BAC 2021

Les états possibles d'un processus sont : prêt, élu, terminé et bloqué.

1- Expliquer à quoi correspond l'état élu.

2- Proposer un schéma illustrant les passages entre les différents états.

3- On donne ci-contre 2 programmes en pseudo code :

En supposant que les processus correspondant à ces programmes s'exécutent simultanément (exécution concurrente), expliquer le problème qui peut être rencontré.

4- Proposer une modification du programme 2 permettant d'éviter ce problème.

Programme 1	Programme 2
Vérrouiller fichier_1	Vérrouiller fichier_2
Calculs sur fichier_1	Vérrouiller fichier_1
Vérrouiller fichier_2	Calculs sur fichier_1
Calculs sur fichier_1	Calculs sur fichier_2
Calculs sur fichier_2	Dévérrouiller fichier_1
Calculs sur fichier_1	Dévérrouiller fichier_2
Dévérrouiller fichier_2	
Dévérrouiller fichier_1	

EXERCICE 6. : SUJET BAC 2024

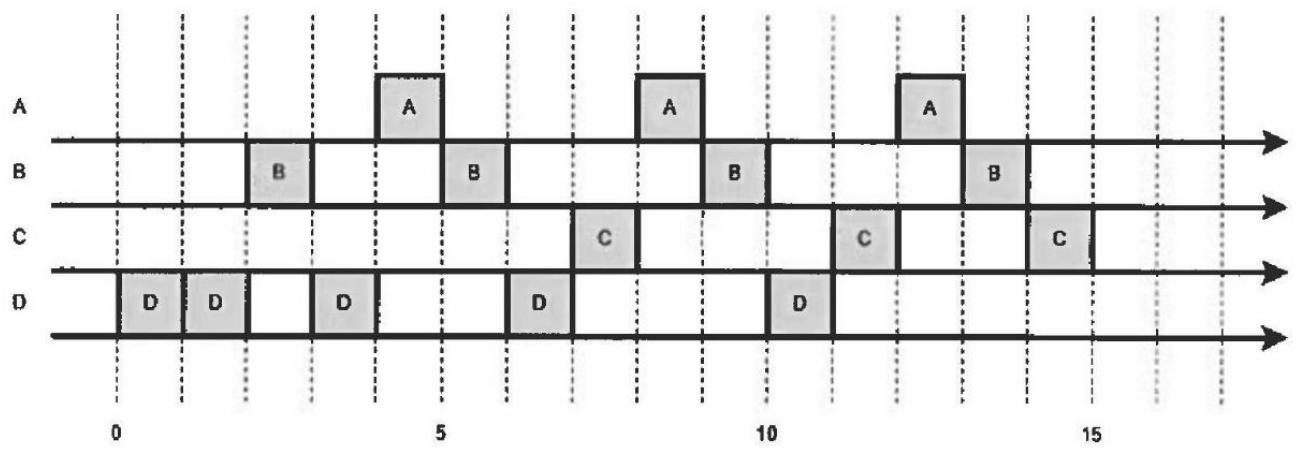
Nous étudions quatre processus A, B, C et D qui utilisent des ressources suivantes :

- un fichier commun aux processus ;
- le clavier de l'ordinateur ;
- le processeur graphique (GPU) ;
- le port 25000 de la connexion Internet.

Voici le détail de ce que fait chaque processus :

A	B	C	D
acquérir le GPU	acquérir le clavier	acquérir le port	acquérir le fichier
faire des calculs	acquérir le fichier	faire des calculs	faire des calculs
libérer le GPU	libérer le clavier	libérer le port	acquérir le clavier
	libérer le fichier	libérer le port	libérer le clavier
			libérer le fichier

On a le chronogramme suivant :



Question : Montrer que l'ordre d'exécution donné aboutit à une situation d'interblocage.

EXERCICE 7. : MISSION PATH-FINDER SUR LA PLANETRE MARS

La situation :

En 1997, la mission Mars Pathfinder rencontre un problème alors que le robot est déjà sur Mars. Après un certain temps, des données sont systématiquement perdues. Les ingénieurs découvrent alors un bug lié à la synchronisation de plusieurs tâches. Les éléments incriminés étaient les suivants :

- une mémoire partagée, qui était protégée par un mutex (un mutex est un système de verrou du noyau)
- une gestion de bus sur la mémoire partagée, qui avait une priorité haute
- une écriture en mémoire partagée (récupération de données), qui avait la priorité la plus basse
- une troisième routine de communication, avec une priorité moyenne, qui ne touchait pas à la mémoire partagée

Il arrivait parfois que l'écriture (priorité faible) s'approprie le mutex. La gestion du bus (priorité haute) attendait ce mutex. La commutation de tâches laissait alors la routine de communication (priorité moyenne) s'exécuter. Or pendant ce temps, le mutex restait bloqué puisque les ressources étaient allouées à la routine de priorité basse. La gestion de bus ne pouvait donc plus s'exécuter et après un certain temps d'attente (une protection insérée par les ingénieurs via un système dit de chien de garde), le système effectuait un redémarrage. Un tel problème est connu sous le nom d'inversion de priorité.

Le problème n'était pas critique et le code fut corrigé à distance. Toutefois dans d'autres situations, les conséquences auraient pu être catastrophiques. On a ensuite constaté le fait que le problème était déjà survenu lors des essais sans avoir été corrigé.

Simulation d'interblocage : On va considérer un robot qui a trois ressources :

- Des moteurs qui lui permettent de se déplacer
- Une liaison wifi qui lui permet de communiquer
- Une caméra qui filme son environnement

Ce robot a trois processus que l'on notera P1, P2 et P3 :

- P1 est le pilotage manuel qui reçoit les ordres par le wifi et opère les moteurs
- P2 envoie le flux vidéo via la liaison wifi
- P3 est le processus qui fait un autotest matériel, hors liaison wifi

Le robot effectue les 3 taches en parallèle. Cela peut se résumer dans le tableau ci-contre.

P1 : pilotage manuel	P2 : envoi de flux vidéo	P3 : auto-test matériel
Demande R1 (moteurs)	Demande R2 (wifi)	Demande R3 (camera)
Demande R2 (wifi)	Demande R3 (camera)	demande R1 (moteurs)
Libère R1 (moteurs)	Libère R2 (wifi)	Libère R3 (Caméra)
Libère R2(wifi)	Libère R3 (caméra)	Libère R1 (moteurs)

1- Montrer que l'exécution de ces 3 processus avec un ordonnancement de type « *Tourniquet* », conduit à une situation d'interblocage.
2- Proposer une solution qui permette d'éviter ce type de situation.

EXERCICE 8. :

Sur un ordinateur, 4 processus (P1 à P4) se partagent 5 ressources (R1 à R5). Le tableau ci-contre indique à un instant précis quelles ressources sont mobilisées par chacun des processus (lettre M) et lesquelles sont attendues (lettre A).

	R1	R2	R3	R4	R5
P1	M	A			
P2	A				M
P3		M	A	A	
P4			A	M	A

Question : Montrer qu'il y a interblocage.

EXERCICE 9. :

Soient trois processus A, B et C qui utilisent trois ressources R, S et T comme illustré dans le tableau ci-contre :

A	B	C
Demande R	Demande S	Demande T
Demande S	Demande T	Demande R
Libère R	Libère S	Libère T
Libère S	Libère T	Libère R

Si les processus sont exécutés de façon séquentielle : A suivi de B, suivi C, il n'y pas d'interblocage. Supposons maintenant que l'exécution des processus est gérée par un ordonnanceur du type circulaire. Si les instructions sont exécutées dans l'ordre :

1. A demande R
2. B demande S
3. C demande T
4. A demande S
5. B demande T
6. C demande R

Question : a-t-on une situation d'interblocage ?

EXERCICE 10. : SUJET BAC 2024

Cet exercice porte sur la programmation Python, la programmation orientée objet, les structures de données (file), l'ordonnancement et l'interblocage.

On s'intéresse aux processus et à leur ordonnancement au sein d'un système d'exploitation. On considère ici qu'on utilise un monoprocesseur.

1. Citer les trois états dans lesquels un processus peut se trouver.

On veut simuler cet ordonnancement avec des objets. Pour ce faire, on dispose déjà de la classe `Processus` dont voici la documentation :

Classe Processus:

```
p = Processus(nom: str, duree: int)
    Crée un processus de nom <nom> et de durée <duree> (exprimée en
cycles d'ordonnancement)

p.execute_un_cycle()
    Exécute le processus donné pendant un cycle.

p.est_fini()
    Renvoie True si le processus est terminé, False sinon.
```

Pour simplifier, on ne s'intéresse pas aux ressources qu'un processus pourrait acquérir ou libérer.

2. Citer les deux seuls états possibles pour un processus dans ce contexte.

Pour mettre en place l'ordonnancement, on décide d'utiliser une file, instance de la classe `File` ci-dessous.

Classe File

```
1  class File:
2      def __init__(self):
3          """ Crée une file vide """
4          self.contenu = []
5
6      def enfile(self, element):
7          """ Enfile element dans la file """
8          self.contenu.append(element)
9
10     def defile(self):
11         """ Renvoie le premier élément de la file et l'enlève de
la file """
12         return self.contenu.pop(0)
13
14     def est_vide(self):

15         """ Renvoie True si la file est vide, False sinon """
16         return self.contenu == []
```

Lors de la phase de tests, on se rend compte que le code suivant produit une erreur :

```
1 f = File()
2 print(f.defile())
```

3. Rectifier sur votre copie le code de la classe `File` pour que la function `defile` renvoie `None` lorsque la file est vide.

On se propose d'ordonnancer les processus avec une méthode du type *tourniquet* telle qu'à chaque cycle :

- si un nouveau processus est créé, il est mis dans la file d'attente ;
- ensuite, on défile un processus de la file d'attente et on l'exécute pendant un cycle ;
- si le processus exécuté n'est pas terminé, on le replace dans la file.

Par exemple, avec les processus suivants

Liste des processus		
processus	cycle de création	durée en cycles
A	2	3
B	1	4
C	4	3
D	0	5

On obtient le chronogramme ci-dessous :

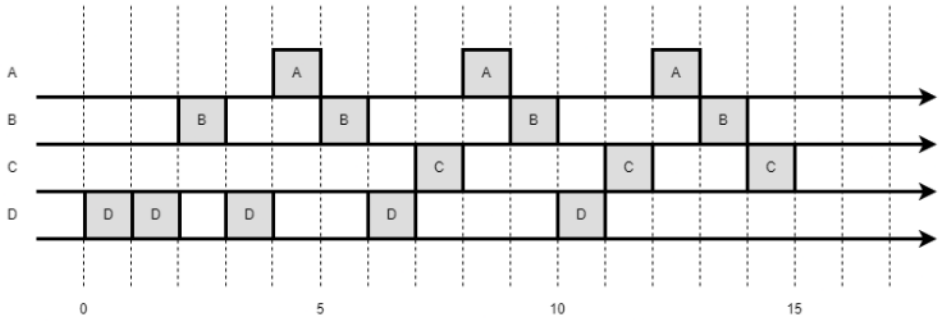


Figure 1. Chronogramme pour les processus A, B, C et D

Pour décrire les processus et le moment de leur création, on utilise le code suivant, dans lequel `depart_proc` associe à un cycle donné le processus qui sera créé à ce moment :

$[0-1]$	$[5-6]$	$[10-11]$
$[1-2]$	$[6-7]$	$[11-12]$
$[2-3]$	$[7-8]$	$[12-13]$
$[3-4]$	$[8-9]$	$[13-14]$
$[4-5]$	$[9-10]$	$[14-15]$

```
p1 = Processus("p1", 4)
p2 = Processus("p2", 3)
p3 = Processus("p3", 5)
p4 = Processus("p4", 3)
depart_proc = {0: p1, 1: p3, 2: p2, 3: p4}
```

Il s'agit d'une modélisation de la situation précédente où un seul processus peut être créé lors d'un cycle donné.

4. Recopier et compléter sur votre copie le chronogramme ci-dessous pour les processus `p1`, `p2`, `p3` et `p4`.

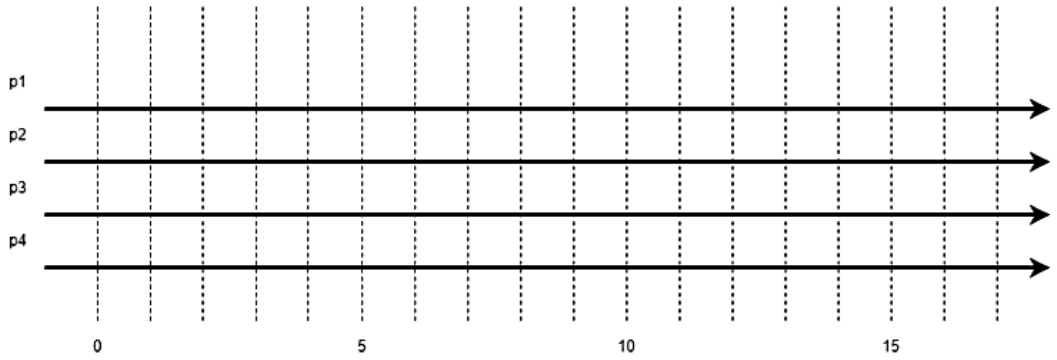


Figure 2. Chronogramme pour les processus p1, p2, p3 et p4

Pour mettre en place l'ordonnancement suivant cette méthode, on écrit la classe `Ordonnanceur` dont voici un code incomplet (l'attribut `temps` correspond au cycle en cours) :

$[0-1]$	$[5-6]$	$[10-11]$
$[1-2]$	$[6-7]$	$[11-12]$
$[2-3]$	$[7-8]$	$[12-13]$
$[3-4]$	$[8-9]$	$[13-14]$
$[4-5]$	$[9-10]$	$[14-15]$

```
class Ordonnanceur:

    def __init__(self):
        self.temps = 0
        self.file = File()

    def ajoute_nouveau_processus(self, proc):
        '''Ajoute un nouveau processus dans la file de
          l'ordonnanceur. '''
        ...

     def tourniquet(self):
        '''Effectue une étape d'ordonnancement et renvoie le nom
          du processus élu.'''
        self.temps += 1
        if not self.file.est_vide():
            proc = ...
            ...
            if not proc.est_fini():
                ...
            return proc.nom
        else:
            return None
```

5. Compléter le code ci-dessus.

À chaque appel de la méthode `tourniquet`, celle-ci renvoie soit le nom du processus qui a été élu, soit `None` si elle n'a pas trouvé de processus en cours.

6. Écrire un programme qui :

- utilise les variables `p1`, `p2`, `p3`, `p4` et `depart_proc` définies précédemment ;
- crée un ordonnanceur ;
- ajoute un nouveau processus à l'ordonnanceur lorsque c'est le moment ;
- affiche le processus choisi par l'ordonnanceur ;
- s'arrête lorsqu'il n'y a plus de processus à exécuter.

Dans la situation donnée en exemple (voir Figure 1), il s'avère qu'en fait les processus utilisent des ressources comme :

- un fichier commun aux processus ;
- le clavier de l'ordinateur ;
- le processeur graphique (GPU) ;
- le port 25000 de la connexion Internet.

Voici le détail de ce que fait chaque processus :

Liste des processus			
A	B	C	D
acquérir le GPU	acquérir le clavier	acquérir le port	acquérir le fichier
faire des calculs	acquérir le fichier	faire des calculs	faire des calculs
libérer le GPU	libérer le clavier	libérer le port	acquérir le clavier
	libérer le fichier		libérer le clavier
			libérer le fichier

7. Montrer que l'ordre d'exécution donné en exemple aboutit à une situation d'interblocage.

EXERCICE 11. : SUJET BAC 2024

1. Expliquer succinctement les différences entre les logiciels libres et les logiciels propriétaires.
2. Expliquer le rôle d'un système d'exploitation.

On donne ci-dessous une arborescence de fichiers sur un système GNU/Linux (les noms encadrés représentent des répertoires, les noms non encadrés représentent des fichiers, `/` correspond à la racine du système de fichiers) :

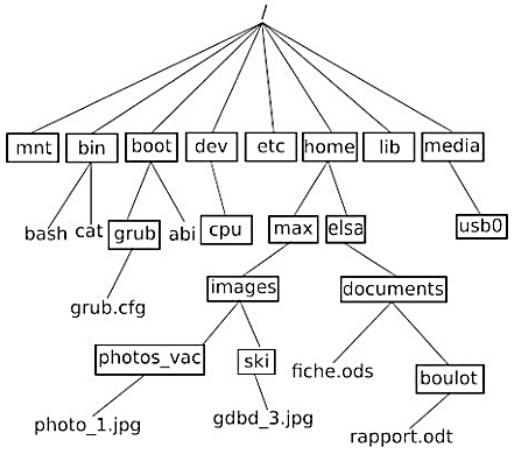


Figure 1. Arborescence de fichiers

3. Indiquer le chemin absolu du fichier `rapport.odt`.

On suppose que le répertoire courant est le répertoire `elsa`.

4. Indiquer le chemin relatif du fichier `photo_1.jpg`.

L'utilisatrice Elsa ouvre une console (aussi parfois appelée terminal), le répertoire courant étant toujours le répertoire `elsa`. On fournit ci-dessous un extrait du manuel de la commande UNIX `cp` :

```
NOM
    cp - copie un fichier

UTILISATION
    cp fichier_source fichier_destination
```

5. Déterminer le contenu du répertoire `documents` et du répertoire `boulot` après avoir exécuté la commande suivante dans la console :

```
cp documents/fiche.ods documents/boulot
```

Dans la suite de l'exercice, on s'intéresse aux processus. On considère qu'un monoprocesseur est utilisé. On rappelle qu'un processus est un programme en cours d'exécution. Un processus est soit élu, soit bloqué, soit prêt.

6. Recopier et compléter le schéma ci-dessous avec les termes suivants :

   élu, *bloqué*, *prêt*, *élection*, *blocage*, *déblocage*.

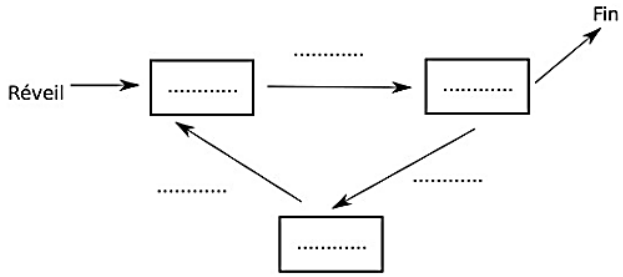


Figure 2. Schéma processus

7. Donner l'exemple d'une situation qui contraint un processus à passer de l'état élu à l'état bloqué.

L'ordonnanceur peut utiliser plusieurs types d'algorithmes pour gérer les processus.

L'algorithme d'ordonnancement par "ordre de soumission" est un algorithme de type FIFO (First In First Out), il utilise donc une file.

8. Nommer une structure de données linéaire de type LIFO (Last In First Out).

À chaque processus, on associe un instant d'arrivée (instant où le processus demande l'accès au processeur pour la première fois) et une durée d'exécution (durée d'accès au processeur nécessaire pour que le processus s'exécute entièrement).

Par exemple, l'exécution d'un processus P4 qui a un instant d'arrivée égal à 7 et une durée d'exécution égale à 2 peut être représentée par le schéma suivant :

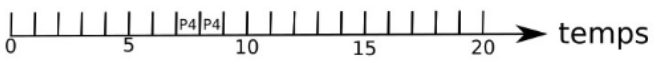


Figure 3. Utilisation du processeur

L'ordonnanceur place les processus qui ont besoin d'un accès au processeur dans une file, en respectant leur ordre d'arrivée (le premier arrivé étant placé en tête de file). Dès qu'un processus a terminé son exécution, l'ordonnanceur donne l'accès au processus suivant dans la file.

Le tableau suivant présente les instants d'arrivées et les durées d'exécution de cinq processus :

5 processus		
Processus	instant d'arrivée	durée d'exécution
P1	0	3
P2	1	6
P3	4	4
P4	6	2
P5	7	1

9. Recopier et compléter le schéma ci-dessous avec les processus P1 à P5 en utilisant les informations présentes dans le tableau ci-dessus et l'algorithme d'ordonnancement "par ordre de soumission".

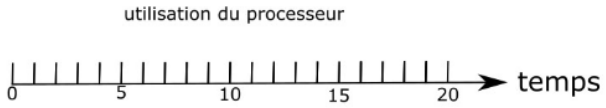


Figure 4. Utilisation du processeur

On utilise maintenant un autre algorithme d'ordonnancement : l'algorithme d'ordonnancement "par tourniquet". Dans cet algorithme, la durée d'exécution d'un processus ne peut pas dépasser une durée Q appelée quantum et fixée à l'avance. Si ce processus a besoin de plus de temps pour terminer son exécution, il doit retourner dans la file et attendre son tour pour poursuivre son exécution.

Par exemple, si un processus P1 a une durée d'exécution de 3 et que la valeur de Q a été fixée à 2, P1 s'exécutera pendant deux unités de temps avant de retourner à la fin de la file pour attendre son tour ; une fois à nouveau élu, il pourra terminer de s'exécuter pendant sa troisième et dernière unité de temps d'exécution.

10. Recopier et compléter le schéma ci-dessous, en utilisant l'algorithme d'ordonnancement "par tourniquet" et les mêmes données que pour la question 9, en supposant que le quantum Q est fixé 2.

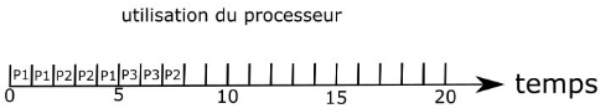


Figure 5. Utilisation du processeur

On considère deux processus P1 et P2, et deux ressources R1 et R2.

11. Décrire une situation qui conduit les deux processus P1 et P2 en situation d'interblocage.

$[0-1]$	$[5-6]$	$[10-11]$
$[1-2]$	$[6-7]$	$[11-12]$
$[2-3]$	$[7-8]$	$[12-13]$
$[3-4]$	$[8-9]$	$[13-14]$
$[4-5]$	$[9-10]$	$[14-15]$

EXERCICE 12. : SUJET BAC 2023

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*)

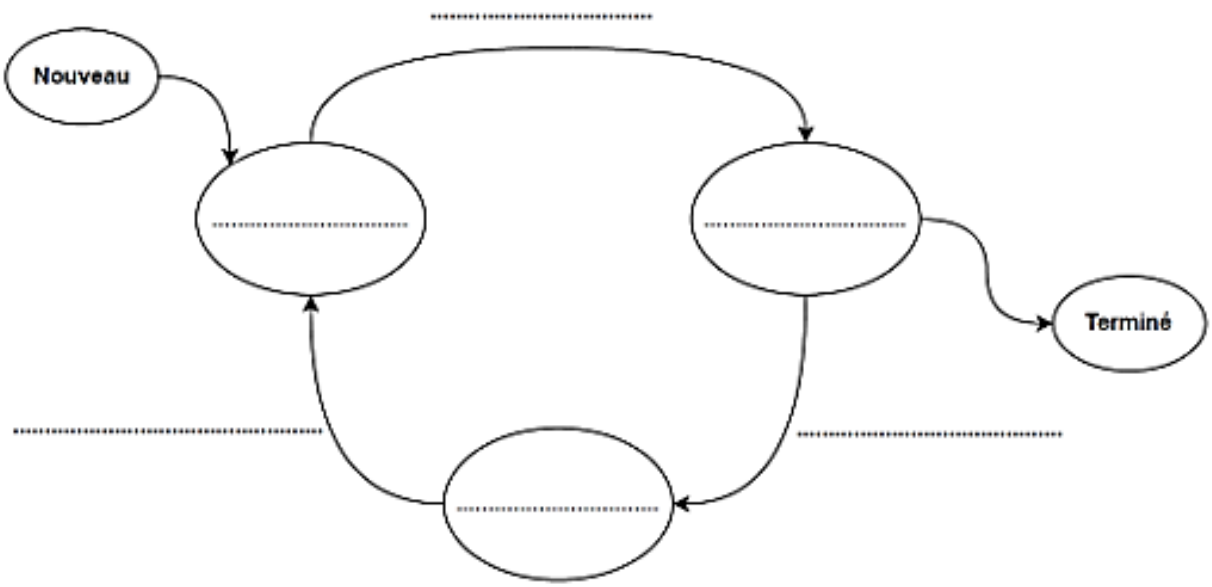
a. Donner sous forme d'un arbre de PID la hiérarchie des processus liés à `firefox`.

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.

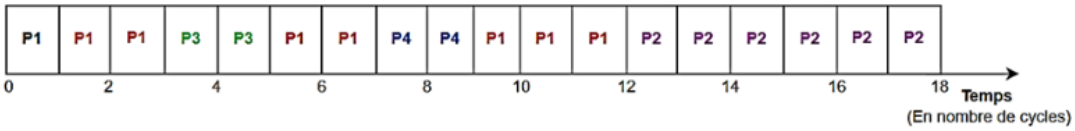
Processus	P1	P2	P3	P4
Instant d'arrivée	0	2	3	7
Durée	8	6	2	2

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.

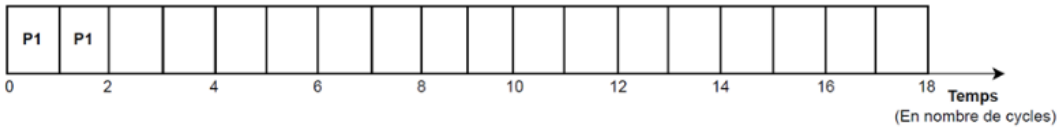


On définit le temps d'exécution d'un processus comme la différence entre son instant de terminaison et son instant d'arrivée.

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.

$[0-1]$	$[5-6]$	$[10-11]$
$[1-2]$	$[6-7]$	$[11-12]$
$[2-3]$	$[7-8]$	$[12-13]$
$[3-4]$	$[8-9]$	$[13-14]$
$[4-5]$	$[9-10]$	$[14-15]$