TD -Algorithmique, Réseaux, Processus-

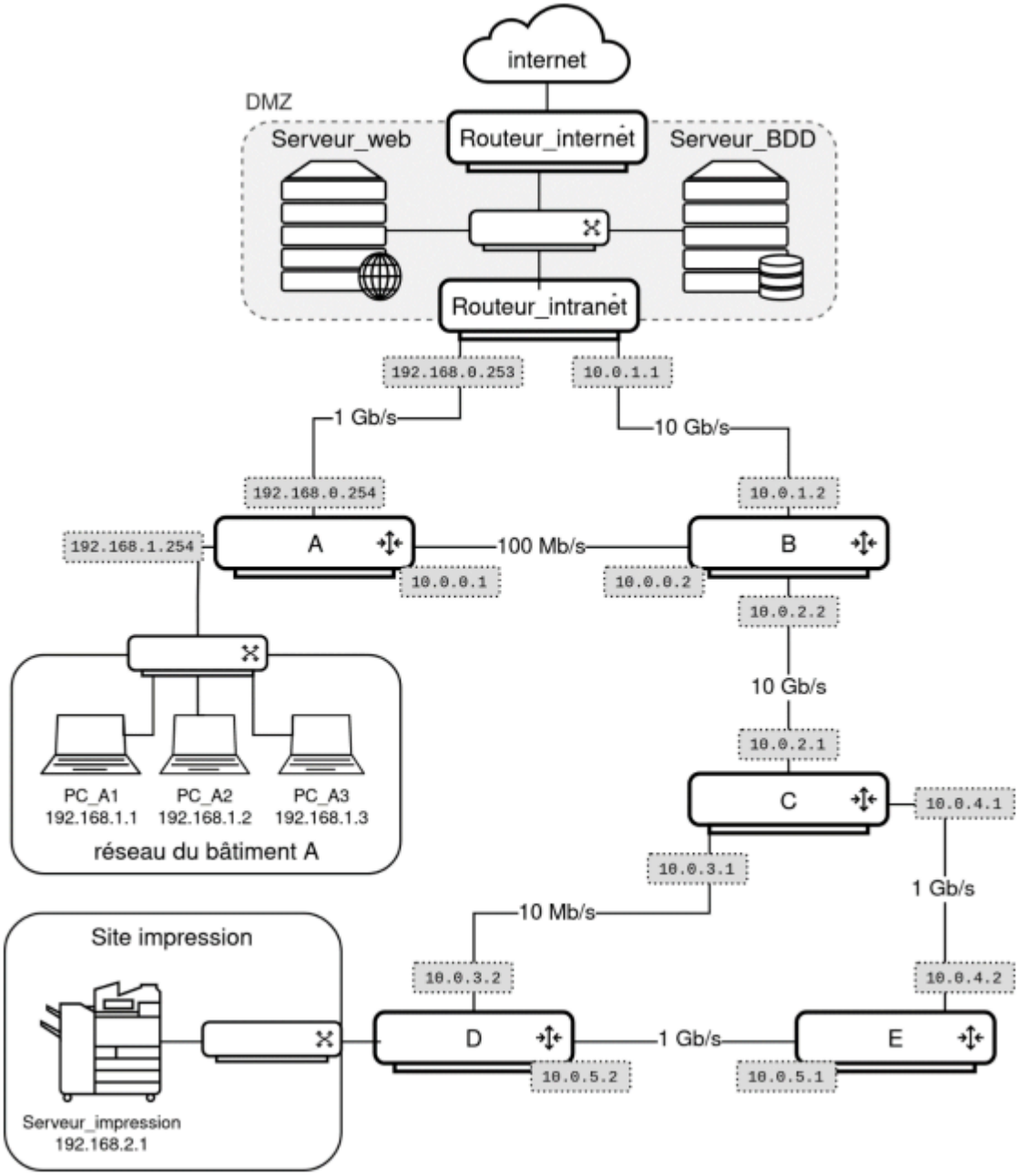
Exercice 1 – Réseaux, DMZ et routage

Cet exercice porte sur les réseaux informatiques, les protocoles réseau, les bases de données relationnelles et les requêtes SQL.

Il est composé de trois parties indépendantes.

La société **LUDOJUV** est spécialisée dans la production et la distribution de magazines ludiques et pédagogiques destinés aux enfants.

L'entreprise étant répartie sur plusieurs bâtiments, son réseau informatique peut être schématisé de la manière suivante :



Partie A – Configuration réseau dans la DMZ

En informatique, on appelle **DMZ** (DeMilitarized Zone) un sous-réseau tampon entre un réseau sécurisé (réseau local) et un réseau non sécurisé (Internet). La DMZ héberge les serveurs qui ont besoin d'accéder à Internet et d'être joignables depuis Internet, et filtre l'accès au réseau local protégé.

La DMZ de l'entreprise est délimitée par les routeurs **Routeur_internet** et **Routeur_intranet**.

Dans cette partie du réseau, l'adressage des machines est construit sur l'adresse de réseau IPv4 **172.16.0.0** et le masque **255.255.255.0**.

1.

Indiquer combien d'octets composent une adresse IPv4.

On attribue les adresses IP suivantes aux routeurs :

- **Routeur_internet** : dernière adresse IP du réseau, soit `172.16.0.254`
- **Routeur_intranet** : avant-dernière adresse IP du réseau, soit `172.16.0.253`

2.

Donner les adresses IP des serveurs de la DMZ en respectant les consignes suivantes :

- **Serveur_web** : première adresse IP du réseau ;
- **Serveur_BDD** : deuxième adresse IP du réseau.

La configuration du poste **PC_A1** est la suivante :

- Adresse IP : `192.168.1.1`
- Masque de sous-réseau : `255.255.255.0`
- Passerelle par défaut : `192.168.0.254`

Ce poste ne parvient pas à accéder à Internet, ni même aux serveurs ou imprimantes de l'entreprise.

L'administrateur réseau effectue les commandes suivantes :

- `ping 192.168.1.2` : la commande réussit ;
- `ping 192.168.0.254` : la commande échoue.

3.

Indiquer ce que permet de tester la commande `ping` .

4.

Donner la nature du problème rencontré et proposer une solution pour y remédier.

Partie B – Routage

Dans l'état actuel du réseau, le routage est configuré manuellement et les tables de routage des routeurs sont les suivantes :

Routeur_intranet	
Destination	Prochain saut
172.16.0.0	-
192.168.0.0	-
192.168.1.0	192.168.0.254
192.168.2.0	192.168.0.254
192.168.3.0	192.168.0.254
10.0.0.0	192.168.0.254
10.0.1.0	-
10.0.2.0	192.168.0.254
10.0.3.0	192.168.0.254
10.0.4.0	192.168.0.254
10.0.5.0	192.168.0.254
0.0.0.0	172.16.0.254

Routeur A	
Destination	Prochain saut
172.16.0.0	192.168.0.253
192.168.0.0	-
192.168.1.0	-
192.168.2.0	10.0.0.2
192.168.3.0	10.0.0.2
10.0.0.0	-
10.0.1.0	10.0.0.2
10.0.2.0	10.0.0.2
10.0.3.0	10.0.0.2
10.0.4.0	10.0.0.2
10.0.5.0	10.0.0.2
0.0.0.0	192.168.0.253

Routeur B	
Destination	Prochain saut
172.16.0.0	10.0.0.1
192.168.0.0	10.0.0.1
192.168.1.0	10.0.0.1
192.168.2.0	10.0.2.1
192.168.3.0	10.0.2.1
10.0.0.0	-
10.0.1.0	-
10.0.2.0	-
10.0.3.0	10.0.2.1
10.0.4.0	10.0.2.1
10.0.5.0	10.0.2.1
0.0.0.0	10.0.1.1

Routeur C	
Destination	Prochain saut
172.16.0.0	10.0.2.2
192.168.0.0	10.0.2.2
192.168.1.0	10.0.2.2
192.168.2.0	10.0.3.2
192.168.3.0	10.0.4.2
10.0.0.0	10.0.2.2
10.0.1.0	10.0.2.2
10.0.2.0	-
10.0.3.0	-
10.0.4.0	-
10.0.5.0	10.0.4.2
0.0.0.0	10.0.2.2

Routeur D	
Destination	Prochain saut
172.16.0.0	10.0.3.1
192.168.0.0	10.0.3.1
192.168.1.0	10.0.3.1
192.168.2.0	-
192.168.3.0	10.0.5.1
10.0.0.0	10.0.3.1
10.0.1.0	10.0.3.1
10.0.2.0	10.0.3.1
10.0.3.0	-
10.0.4.0	10.0.3.1
10.0.5.0	-
0.0.0.0	10.0.3.1

Routeur E	
Destination	Prochain saut
172.16.0.0	10.0.4.1
192.168.0.0	10.0.4.1
192.168.1.0	10.0.4.1
192.168.2.0	10.0.5.2
192.168.3.0	-
10.0.0.0	10.0.4.1
10.0.1.0	10.0.4.1
10.0.2.0	10.0.4.1
10.0.3.0	10.0.4.1
10.0.4.0	-
10.0.5.0	-
0.0.0.0	10.0.4.1

5.

Le poste **PC_A1** souhaite accéder au **Serveur_impression**. Donner le chemin suivi par les paquets sous la forme : `élément1 → élément2 → ...`

6.

Le lien entre les routeurs **C** et **D** est coupé. Donner le chemin suivi par les paquets lorsque PC_A1 souhaite accéder au **Serveur_impression** et indiquer s'ils arrivent à destination.

Pour remédier à ce type de problème, l'administrateur décide d'utiliser le protocole **RIP**, un protocole de routage automatique dont la métrique est le nombre minimum de routeurs traversés.

7.

Tous les routeurs sont de nouveau opérationnels. Compléter la table de routage du routeur **C** si les routeurs sont configurés pour utiliser le protocole RIP et lorsque toutes les tables de routage sont stables.

Routeur C		
Destination	Prochain saut	Métrique
172.16.0.0	10.0.2.2	2
192.168.0.0	10.0.2.2	2
192.168.1.0		
192.168.2.0		
192.168.3.0		
10.0.0.0	10.0.2.2	1
10.0.1.0		
10.0.2.0	-	-
10.0.3.0	-	-
10.0.4.0	-	-
10.0.5.0		
0.0.0.0	10.0.2.2	2

8.

Donner le chemin suivi par les paquets lorsque PC_A1 souhaite accéder au **Serveur_impression** en appliquant le protocole RIP.

9.

Expliquer pourquoi ce chemin n’est pas le meilleur choix possible.

10.

Le lien entre les routeurs **C** et **D** est coupé. Indiquer la modification apportée à la table de routage du routeur C et donner le nouveau chemin suivi par les paquets lorsque PC_A1 souhaite accéder au **Serveur_impression**.

Exercice 2 – Programmation Python : piles, POO, tests (mélange faro)

Cet exercice porte sur la programmation Python, la programmation orientée objet, les tests et la structure de données **pile**.

Le **mélange faro** consiste à partager un jeu de cartes en deux moitiés puis à **intercaler** les cartes de ces deux moitiés. On note **n** le nombre de cartes du jeu et on suppose que **n est pair**.

Pour modéliser le jeu de cartes, on utilise une pile `jeu`, instance d'une classe `Pile` dont on suppose l'interface suivante :

- **Constructeur** : `Pile()` crée une pile vide ;
- **empile(x)** : empile `x` sur la pile ;
- **depile()** : dépile et renvoie l'élément au sommet (pile non vide) ;
- **est_vide()** : renvoie `True` si la pile est vide, sinon `False`.

Le jeu de cartes est représenté par une pile contenant `1` au sommet, puis `2` en dessous, et ainsi de suite jusqu'à `n` en bas de pile.

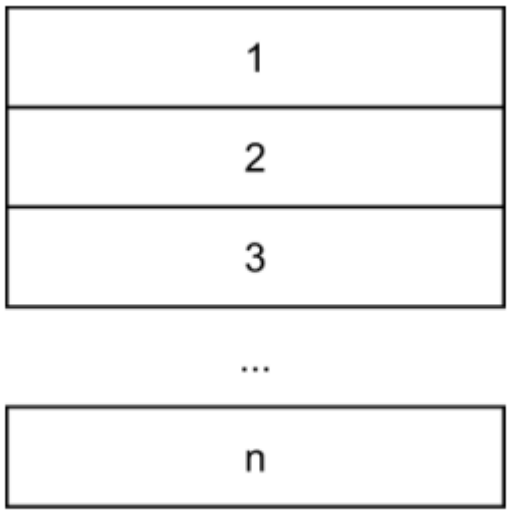


Figure 1. Pile représentant un jeu de cartes

Figure 1. Pile représentant un jeu de cartes

Principe du mélange faro

Le mélange faro est réalisé en trois étapes :

1. On dépile la moitié de `jeu` et on empile chaque élément dépilé dans une pile `moitie1`.
2. On dépile le reste de `jeu` et on empile chaque élément dépilé dans une pile `moitie2`.
3. On reconstitue `jeu` en empilant alternativement un élément de `moitie1` puis un élément de `moitie2`, jusqu'à vider ces deux piles.

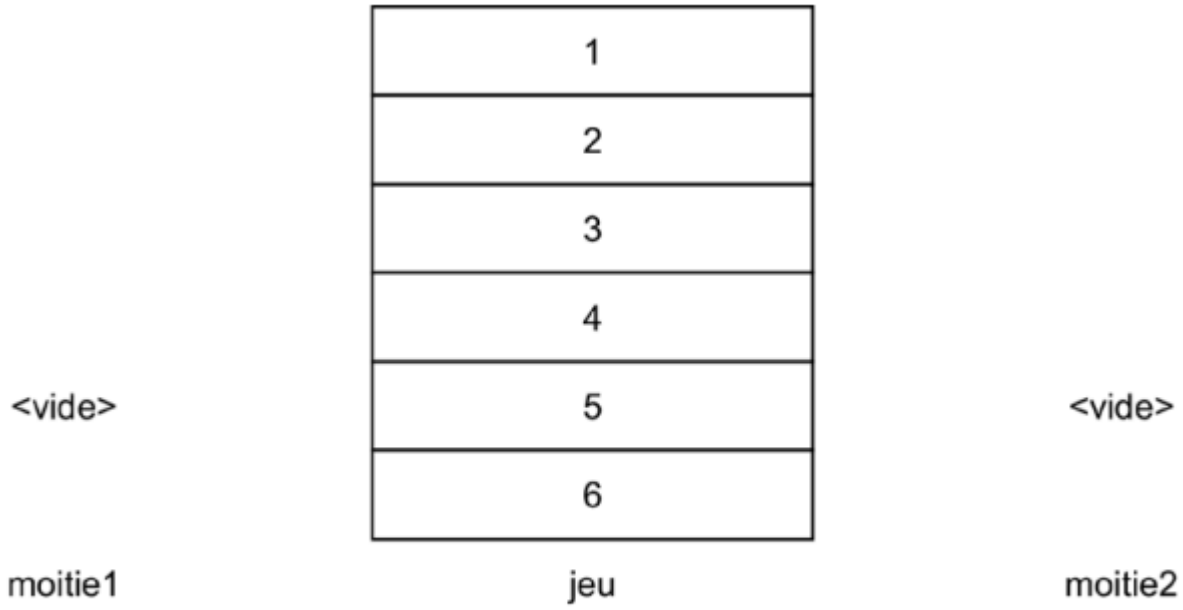


Figure 2. Contenus initiaux des 3 piles

Figure 2. Contenus initiaux des 3 piles

1. Représentation des piles

Représenter sur ta copie le contenu des trois piles `jeu`, `moitie1` et `moitie2` à la fin de chaque étape du mélange faro, pour l'exemple donné.

Fonctions Python

2. Fonction `produire_jeu`

La fonction `produire_jeu(n)` prend en paramètre un entier `n` pair et renvoie une pile représentant le jeu de cartes. Recopier et compléter le code de cette fonction.

```
def produire_jeu(n):
    resultat = Pile()
    for i in range(...):
        resultat.empile(...)
    return resultat
```

3. Fonction `scinder_jeu`

La fonction `scinder_jeu(p, n)` prend en paramètre une pile `p` (le jeu) et l'entier `n` (taille de la pile). Elle renvoie deux piles correspondant aux deux moitiés du jeu. Le code fourni comporte des erreurs : indiquer les lignes à rectifier et les corrections à apporter.

```
def scinder_jeu(p, n):
    m1 = Pile()
    m2 = Pile
    for i in range(n):
        m1.empile(p.depile())
    for i in range(n):
        m2.empile(p.depile())
    return m1, m2
```

4. Fonction `recombiner`

Écrire une fonction `recombiner(m1, m2)` qui renvoie une pile correspondant au jeu obtenu en empilant alternativement, et dans cet ordre, un élément de `m1` puis un élément de `m2` .

5. Fonction `faro`

Écrire une fonction `faro(p, n)` qui prend une pile `p` (le jeu à mélanger) et l'entier `n` , et renvoie une pile correspondant au jeu obtenu après un mélange faro.

Tests et propriété de répétition

Une propriété mathématique assure qu'étant donné un jeu de `n` cartes (avec `n` pair), en répétant suffisamment de fois le mélange faro, on finit par remettre le jeu dans l'ordre initial.

On considère une fonction `identiques(p1, p2)` qui renvoie un booléen indiquant si deux piles contiennent les mêmes éléments, en même nombre et dans le même ordre. La fonction **ne modifie pas** les piles d'entrée.

Pour tester cette fonction, on a commencé par produire le jeu de tests suivant :

```
 p1 = Pile()
p1.empile(1)
p2 = Pile()
assert not identiques(p1, p2)
```

6. Compléter le jeu de tests

Compléter le jeu de tests afin de couvrir au minimum les cas suivants :

- les piles sont différentes mais ont la même taille ;
- les piles sont identiques.

7. Fonction `ordre_faro`

Écrire une fonction `ordre_faro(n)` qui prend un entier `n` pair et renvoie le plus petit nombre de répétitions du mélange faro nécessaire pour que le jeu revienne dans l'ordre initial.

Exercice 3 – Processus et ordonnancement

1. Lecture d'un extrait de la commande `ps`

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 **processus parent**.

La commande UNIX `ps` affiche un instantané des processus en cours d'exécution. Un extrait du résultat est donné 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 :

- **UID** : identifiant de l'utilisateur propriétaire du processus ;
- **PID** : identifiant du processus ;
- **PPID** : identifiant du processus parent ;
- **C** : utilisation processeur ;
- **STIME** : heure de démarrage du processus ;
- **TTY** : terminal auquel le processus est attaché ;
- **TIME** : temps processeur consommé ;
- **CMD** : commande ayant démarré le processus.

Questions

1. Donner le PID du parent du processus démarré par la commande `vi` .
2. Donner le PID d'un processus enfant du processus démarré par la commande `xfce4-terminal` .
3. Citer le PID de deux processus qui ont le même parent.
4. Parmi tous les processus affichés, citer le PID des deux qui ont consommé le plus de temps processeur.

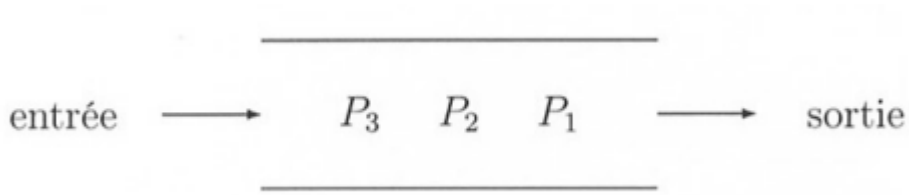
2. Ordonnancement

On considère trois processus **P1**, **P2** et **P3**, tous soumis à l'instant 0 dans l'ordre 1, 2, 3. Leur durée d'exécution (en unités de temps) est donnée :

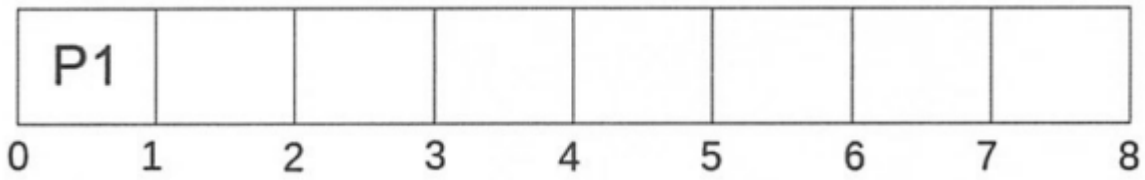
Nom du processus	Durée d'exécution en unité de temps	Ordre de soumission
P_1	3	1
P_2	1	2
P_3	4	3

(a) Politique du tourniquet (Round Robin)

Le temps est découpé en tranches appelées **quantums** (ici : 1 unité 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. S'il n'a pas terminé, il réintègre la file (côté entrée).

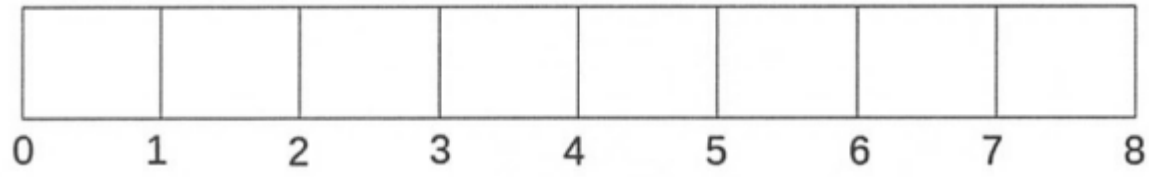


Reproduire le tableau ci-dessous et indiquer dans chaque case le processus exécuté à chaque cycle.



(b) Politique du plus court d'abord

Les processus sont exécutés complètement dans l'ordre croissant de leurs durées d'exécution : le plus court est exécuté en premier. Reproduire le tableau ci-dessous et indiquer dans chaque case le processus exécuté à chaque cycle.



3. Interblocage (deadlock)

On considère trois ressources **R1**, **R2** et **R3** et trois processus **P1**, **P2** et **P3**. Chaque processus exécute une suite d'instructions élémentaires de type : demander une ressource / libérer une ressource.

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

Questions

1. Rappeler les différents états d'un processus et expliquer pourquoi il y a ici un risque d'interblocage, en proposant un ordre d'exécution qui le provoque.
2. Proposer un ordre d'exécution des instructions élémentaires permettant d'éviter l'interblocage.