Amérique du Nord – 2024 – sujet1 - Correction

Exercice 1 (6 points)

---

1. Les trois états possibles d'un processus

Dans un système d'exploitation classique, un processus peut être dans trois états :

- **Prêt**
- **En cours d'exécution**
- **Bloqué (en attente)**

👉 *Commentaire :* Un processus bloqué attend une ressource ou un événement.

2. Deux états possibles dans ce contexte simplifié

Ici, on ne gère pas les ressources.

Donc un processus peut être seulement :

- **Prêt**
- **En cours d'exécution**

👉 Il n'y a pas d'état bloqué puisque les ressources ne sont pas modélisées.

---

3. Correction de la méthode `defile`

Problème : Si la file est vide, `pop(0)` provoque une erreur.

Correction :

```
def defile(self):
    if self.est_vide():
        return None
    return self.contenu.pop(0)
```

👉Commentaire : *On teste d'abord si la file est vide pour éviter l'erreur.*

---

4. Chronogramme pour p1, p2, p3, p4

Données :

```
p1 : durée 4, création cycle 0
p3 : durée 5, création cycle 1
p2 : durée 3, création cycle 2
p4 : durée 3, création cycle 3
```

Ordonnancement tourniquet (Round Robin).

Cycle par cycle :

Cycle	Processus exécuté
0	p1
1	p1
2	p3
3	p1
4	p2
5	p3
6	p4
7	p1 (fini)
8	p2
9	p3
10	p4
11	p2 (fini)
12	p3
13	p4 (fini)
14	p3 (fini)

👉 *Le principe :*

- *On ajoute les processus à leur cycle de création.*
- *Chaque processus exécute 1 cycle.*
- *S'il n'est pas fini → retour en file.*

---

5. Classe `Ordonnanceur` (complétion)

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

    def ajoute_processus(self, proc):
        self.file.enfile(proc)

    def tourniquet(self):
        if self.file.est_vide():
            self.temps += 1
            return None

        proc = self.file.defile()
        proc.execute_un_cycle()

        if not proc.est_fini():
            self.file.enfile(proc)

        self.temps += 1
        return proc.nom
```

👉 *Commentaire :*

- *On défiler un processus.*
- *On exécute un cycle.*
- *S'il n'est pas fini → on le remet en file.*
- *On incrémente le temps.*

---

6. Programme principal

```
ordonnanceur = Ordonnanceur()

while True:
    if ordonnanceur.temps in depart_proc:
        ordonnanceur.ajoute_processus(depart_proc[ordonnanceur.temps])

    nom = ordonnanceur.tourniquet()

    if nom is not None:
        print("Cycle", ordonnanceur.temps - 1, ":", nom)

    if ordonnanceur.file.est_vide() and ordonnanceur.temps >
max(depart_proc):
        break
```

👉 *Commentaire :*

- *On vérifie si un processus doit être créé.*
- *On appelle* `tourniquet`*.*
- *On s'arrête quand il n'y a plus de processus.*

---

7. Montrer qu'il y a interblocage

On observe les ressources :

- A acquiert GPU
- B acquiert clavier
- C acquiert port
- D acquiert fichier

Puis :

- A veut libérer GPU
- B veut fichier
- D veut clavier

Situation possible :

- A détient GPU
- B détient clavier
- C détient port
- D détient fichier

Puis :

- B attend fichier (détenu par D)
- D attend clavier (détenu par B)

👉 B attend D
👉 D attend B

C’est un **cycle d’attente circulaire** : B → D → B

Donc interblocage.
