Asie – 2025 – sujet2 - Correction

Exercice 2 (6 points)

---

1) Nom du programme

Il s'agit d'un **ordonnanceur** (scheduler).

👉*Commentaire : Un ordonnanceur est un programme du système d'exploitation chargé de déterminer quel processus doit être exécuté par le processeur.*

---

2) États possibles d'un processus

Les principaux états sont :

- **Prêt** (en attente du processeur)
- **En cours d'exécution**
- **Bloqué** (attente d'un événement)
- **Terminé**

👉*Commentaire : On peut aussi trouver les états « nouveau » ou « suspendu » selon les systèmes.*

---

3) Structure la plus adaptée

✔ **Réponse : Proposition 2 — File**

👉*Commentaire : Une file fonctionne en **FIFO (First In, First Out)**.*
Cela garantit qu'un processus arrivé en premier sera traité en premier à priorité égale.
Une liste simple ne garantit pas automatiquement le comportement FIFO.
Une pile (LIFO) serait incorrecte ici.

---

4) Constructeur de la classe Processus

```
class Processus:
    def __init__(self, PID, priorite, temps_CPU):
        self.PID = PID
        self.priorite = priorite
        self.temps_utilisation = 0
        self.temps_CPU = temps_CPU
```

👉*Commentaire :*

- `temps_utilisation` *démarre à 0*
- `temps_CPU` *correspond au temps total nécessaire*

---

5) Simulation

On part de :

```
liste_files = [[P3, P2, P1], [], []]
```

On suppose qu'on enlève à chaque fois le processus en tête.

Cycle 1

CPU = P1
`liste_files = [[P3, P2], [], []]`

Cycle 2

P1 est rétrogradé en priorité 1
CPU = P2
`liste_files = [[P3], [P1], []]`

Cycle 3

P2 est rétrogradé en priorité 1
CPU = P3
`liste_files = [[], [P2, P1], []]`

Cycle 4

P3 est rétrogradé en priorité 1

CPU = P3 (continue si aucune priorité supérieure)
`liste_files = [[], [P2, P1], []]`

Cycle 5

CPU = P1
`liste_files = [[], [P2], [P3]]`

👉*Commentaire:*

On vérifie à chaque cycle :

- *si processus terminé*
- *sinon augmentation du temps d'utilisation*
- *rétrogradation de priorité*

---

6) Pourquoi le long processus peut ne jamais terminer?

Le processus long :

- démarre priorité 0
- est rétrogradé à 1
- puis 2
- puis 3
- etc.

Le processus 1000 va avoir une priorité qui va donc diminuer de 1 après chaque exécution. Pendant ce temps, de nouveaux petits processus arrivent en priorité 0.

👉 Il se retrouve donc avec une priorité de plus en plus faible (valeur numérique grande). Il n'est plus jamais sélectionné.

C'est un phénomène appelé **famine (starvation)**.

---

7) Pourquoi le nouveau système règle le problème?

Avec `temps_d_attente` :

- Si un processus attend trop longtemps,
- Sa priorité augmente (valeur numérique diminue),
- Il redevient prioritaire.

👉 Cela empêche la famine. On parle de mécanisme de **vieillissement (aging)**.

---

8) Fonction meilleur_priorite

```
def meilleur_priorite(liste_files):
    for i in range(len(liste_files)):
        if liste_files[i] != []:
            return i
    return None
```

👉*Commentaire : On parcourt dans l'ordre croissant car la priorité la plus forte est la plus petite valeur.*

---

9) Fonction prioritaire

```
def prioritaire(liste_files):
    for i in range(len(liste_files)):
        if liste_files[i] != []:
            return liste_files[i].pop(0)
    return None
```

👉*Commentaire : `pop(0)` retire le premier élément (FIFO) et supprime le processus de la file*

---

10) Fonction gerer

```python
def gerer(p, liste_files):

    # Cas 1 : aucun processus en cours
    if p is None:
        return prioritaire(liste_files)

    # Cas 2 : processus terminé
    if p.temps_utilisation >= p.temps_CPU:
        return prioritaire(liste_files)

    # Sinon on continue
    p.temps_utilisation += 1

    # Vérifie si une priorité supérieure ou égale existe
    meilleure = meilleur_priorite(liste_files)

    if meilleure is not None and meilleure <= p.priorite:
        # On rétrograde le processus courant
        p.priorite += 1
        liste_files[p.priorite].append(p)
        return prioritaire(liste_files)

    else:
        p.priorite += 1
        return p
```