Compte Rendu : TP1 Programmation Événementielle

Module R309

Programmation événementielle
Taha Adam

Exercice 1 : Compteurs

Développer une classe `Counter` qui hérite de la classe `Thread`. Un compteur doit être capable de :

- Incrémenter ou décrémenter une valeur initiale.
- Créer un pas de comptage, un sens (croissant ou décroissant), un nombre de comptages, et une période entre deux mises à jour.
- Afficher la valeur après chaque mise à jour.
- Lancer deux compteurs différents simultanément.

Code Complet

```python
import threading
import time

class Compteur(threading.Thread):
    """Classe permettant de créer un compteur qui s'incrémente ou se décrémente en fonction des paramètres"""
    def __init__(self, nom, value, pas, sens, nb_comptages, periode):
        """Initialisation de la classe Compteur"""
        threading.Thread.__init__(self)
        self.nom = nom
        self.value = value
        self.pas = pas
        self.sens = sens
        self.nb_comptages = nb_comptages
        self.periode = periode

    def run(self):
        """Méthode permettant de lancer le compteur"""
        for i in range(self.nb_comptages):
            if self.sens == 'montant':
                self.value += self.pas
            else:
                self.value -= self.pas
            print(f"value du {self.nom} : {self.value}")
            time.sleep(self.periode)

compteur1 = Compteur(nom="Compteur1", value=0, pas=1, sens='montant', nb_comptages=10, periode=2)
```

```
compteur2 = Compteur(nom="Compteur2", value=50, pas=1, sens='descendant', nb_comptages=5, periode=1)

compteur1.start()
compteur2.start()

compteur1.join()
compteur2.join()

print("Les deux compteurs ont terminé.")
```

Partie 1 : Initialisation de la classe

```
class Compteur(threading.Thread):
    """Classe permettant de créer un compteur qui s'incrémente ou se décrémente en fonction des paramètres"""
    def __init__(self, nom, value, pas, sens, nb_comptages, periode):
        """Initialisation de la classe Compteur"""
        threading.Thread.__init__(self)
        self.nom = nom
        self.value = value
        self.pas = pas
        self.sens = sens
        self.nb_comptages = nb_comptages
        self.periode = periode
```

- Cette classe hérite de `threading.Thread`, qui permet de gérer plusieurs compteurs en parallèle.
- Les paramètres `nom`, `value`, `pas`, définissent les caractéristiques du compteur.
- L'appel à `threading.Thread.__init__(self)` initialise la classe mère, nécessaire pour activer les fonctionnalités de threading.

Partie 2 : Méthode `run`

```
def run(self):
    """Méthode permettant de lancer le compteur"""
    for i in range(self.nb_comptages):
        if self.sens == 'montant':
            self.value += self.pas
        else:
            self.value -= self.pas
        print(f"value du {self.nom} : {self.value}")
        time.sleep(self.periode)
```

- La méthode est automatiquement appelée lorsqu'on exécute `start()` sur un objet `Compteur`.
- La boucle `for` incrémente ou décrémente la valeur selon le sens spécifié lors de nos tests (`montant` ou `descendant`).

- `time.sleep(self.periode)` introduit une pause entre deux mises à jour de nos compteurs.
- La méthode imprime la valeur actuelle du compteur.

Tests

- **Compteur1** commence à `0` et s'incrémente jusqu'à `10`.
- **Compteur2** commence à `50` et décrémente jusqu'à `45`.
- Le programme utilise donc des threads pour exécuter les compteurs simultanément.

Exercice 2 : Montre Digitale

Créer une montre digitale affichant l'heure au format `hh:mm:ss` avec la possibilité de choisir entre un mode 24 heures ou 12 heures.

Code Complet

```
from threading import Thread
import time

class Montre(Thread):
    """Classe permettant de créer une montre digitale"""
    def __init__(self, h, m, s, mode24=True):
        """Initialisation de la classe Montre"""
        assert isinstance(h, int) and 0 <= h < 24
        assert isinstance(m, int) and 0 <= m < 60
        assert isinstance(s, int) and 0 <= s < 60
        Thread.__init__(self)
        self.h = h
        self.m = m
        self.s = s
        self.mode24 = mode24

    def __str__(self):
        """Méthode permettant d'afficher l'heure"""
        if self.mode24:
            return "%02d:%02d:%02d" % (self.h, self.m, self.s)
        else:
            suffix = "PM" if self.h >= 12 else "AM"
            heure = self.h % 12 or 12
            return "%02d:%02d:%02d %s" % (heure, self.m, self.s, suffix)

    def run(self):
        """Méthode permettant de lancer la montre"""
        while True:
            print(self)
            time.sleep(1)
            self.s += 1
            if self.s == 60:
                self.s = 0
                self.m += 1
                if self.m == 60:
```

```
                    self.m = 0
                    self.h += 1
                    if self.h == 24:
                        self.h = 0

if __name__ == "__main__":
    while True:
        print("\nChoisissez le mode d'affichage :")
        print("1 - Mode 24 heures")
        print("2 - Mode 12 heures")
        choix = input("Entrez votre choix (1 ou 2) : ").strip()
        if choix == "1":
            mode24 = True
            break
        elif choix == "2":
            mode24 = False
            break
        else:
            print("Choix invalide. Essayez encore.")

    h, m, s = 15, 10, 30
    montre = Montre(h, m, s, mode24)
    montre.start()

```

Partie 1 : Méthode `__str__`

```
def __str__(self):
    """Méthode permettant d'afficher l'heure"""
    if self.mode24:
        return "%02d:%02d:%02d" % (self.h, self.m, self.s)
    else:
        suffix = "PM" if self.h >= 12 else "AM"
        heure = self.h % 12 or 12
        return "%02d:%02d:%02d %s" % (heure, self.m, self.s, suffix)
```

- La méthode renvoie l'heure sous forme de chaîne formatée en fonction du choix de l'utilisateur.
- Si `mode24` est activé, l'heure est affichée en format 24 heures. Sinon, elle est convertie au format 12 heures avec `AM` ou `PM`.
- La logique pour `self.h % 12 or 12` assure qu'on affiche `12` au lieu de `0` en mode 12 heures.

Partie 2 : Méthode `run`

```
def run(self):
    """Méthode permettant de lancer la montre"""
    while True:
        print(self)
        time.sleep(1)
```

```
        self.s += 1
        if self.s == 60:
            self.s = 0
            self.m += 1
            if self.m == 60:
                self.m = 0
                self.h += 1
                if self.h == 24:
                    self.h = 0
```

- La méthode simule une horloge en incrémentant les secondes chaque seconde grâce à `time.sleep(1)`.
- Lorsque les secondes atteignent 60, elles sont réinitialisées à 0, et les minutes sont incrémentées. Le même principe s'applique pour les minutes et les heures.

Tests

- L'horloge affiche en temps réel, l'heure en fonction de l'affichage demandée à l'utilisateur

---

Exercice 3 : Associations Volatiles

Créer une table DNS avec une durée de vie limitée pour chaque association. Les entrées expirent automatiquement après un temps donné.

Code Complet

```
import threading
import time

class TableDns:
    """Classe permettant de gérer une table DNS"""
    def __init__(self, ttl):
        """Initialisation de la classe TableDns"""
        self.ttl = ttl
        self.table = {}
        self.verrou = threading.Lock()
        self.thread_nettoyage = threading.Thread(target=self.nettoyer_entrees_expirees)
        self.thread_nettoyage.daemon = True
        self.thread_nettoyage.start()

    def ajouter_entree(self, addr_sym, addr_ip):
        """Méthode permettant d'ajouter une entrée dans la table DNS"""
        with self.verrou:
            tps_expiration = time.time() + self.ttl
            self.table[addr_sym] = (addr_ip, tps_expiration)
            print(f"Ajouté : {addr_sym} -> {addr_ip} (expire dans {self.ttl} secondes)")

    def obtenir_ip(self, addr_sym):
        """Méthode permettant d'obtenir l'adresse IP associée à une adresse symbolique"""
        with self.verrou:
```

```
            entree = self.table.get(addr_sym)
            if entree and entree[1] > time.time():
                return entree[0]
            else:
                return None

    def nettoyer_entrees_expirees(self):
        """Méthode permettant de nettoyer les entrées expirées"""
        while True:
            with self.verrou:
                temps_actuel = time.time()
                cles_expires = [cle for cle, (_, temps_exp) in
self.table.items() if temps_exp <= temps_actuel]
                for cle in cles_expires:
                    del self.table[cle]
                    print(f"Entrée expirée supprimée : {cle}")
            time.sleep(1)

table_dns = TableDns(ttl=5)
table_dns.ajouter_entree("takagii.jp", "192.168.16.1")

time.sleep(3)
print("IP pour takagii.jp :", table_dns.obtenir_ip("takagii.jp"))

time.sleep(3)
print("IP pour takagii.jp après expiration :",
table_dns.obtenir_ip("takagii.jp"))

```

Partie 1 : Ajout d'une entrée

```
def ajouter_entree(self, addr_sym, addr_ip):
    """Méthode permettant d'ajouter une entrée dans la table DNS"""
    with self.verrou:
        tps_expiration = time.time() + self.ttl
        self.table[addr_sym] = (addr_ip, tps_expiration)
        print(f"Ajouté : {addr_sym} -> {addr_ip} (expire dans {self.ttl}
secondes)")
```

- La méthode utilise un verrou (`self.verrou`) pour protéger les données partagées dans un contexte à plusieurs thread.
- Chaque entrée ajoutée à la table est associée à une durée de vie (`ttl`).
- Le `time.time()` calcule le moment exact où l'entrée doit expirer.

Partie 2 : Nettoyage des entrées expirées

```python
def nettoyer_entrees_expirees(self):
    """Méthode permettant de nettoyer les entrées expirées"""
    while True:
        with self.verrou:
            temps_actuel = time.time()
            cles_expires = [cle for cle, (_, temps_exp) in
self.table.items() if temps_exp <= temps_actuel]
            for cle in cles_expires:
                del self.table[cle]
                print(f"Entrée expirée supprimée : {cle}")
        time.sleep(1)
```

- Cette méthode fonctionne en arrière-plan comme un daemon donc en (arrière plan) grâce au thread `self.thread_nettoyage` (démarré au moment de l'initialisation).
- Elle scanne la table DNS pour trouver et supprimer les entrées expirées.
- `time.sleep(1)` garantit que le nettoyage est effectué à intervalles réguliers, sans surcharger le CPU.

Tests

- Les entrées expirées sont supprimées automatiquement après le délai défini.
