Compte Rendu : TP2 Programmation Événementielle

Module R309

Programmation événementielle

Taha Adam

Exercice 1 : File d'attente bornée

Objectifs

- Implémenter une file d'attente bornée (FIFO).
- Gérer des accès concurrents avec des threads pour **ajouter** ou **retirer** des objets.
- Suspendre les threads quand la file est pleine ou vide.
- Réaliser des classes `Producteur` et `Consommateur` pour tester la file.

Code Complet

```python
from threading import Thread, Condition
import time
import random

class File:
    def __init__(self, size):
        """Crée une file d'attente bornée de taille sizes"""
        assert isinstance(size, int) and size > 0
        self.size = size
        self.tete = 0
        self.queue = -1
        self.nb_elements = 0
        self.contenu = [None] * size
        self.condition = Condition()

    def vide(self):
        with self.condition:
            return self.nb_elements == 0

    def pleine(self):
        with self.condition:
            return self.nb_elements == self.size

    def ajouter(self, objet):
        with self.condition:
            while self.pleine():
                self.condition.wait()
            self.queue = (self.queue + 1) % self.size
            self.contenu[self.queue] = objet
            self.nb_elements += 1
            print(f"Ajouté : {objet}")
```

```
            self.condition.notify_all()

    def retirer(self):
        with self.condition:
            while self.vide():
                self.condition.wait()
            resultat = self.contenu[self.tete]
            self.tete = (self.tete + 1) % self.size
            self.nb_elements -= 1
            print(f"Retiré : {resultat}")
            self.condition.notify_all()
            return resultat



class Producteur(Thread):
    def __init__(self, file):
        """Crée un producteur qui ajoute des éléments à la file file"""
        assert isinstance(file, File)
        super().__init__()
        self.file = file

    def run(self):
        for _ in range(4):
            valeur = random.randint(1, 100)
            self.file.ajouter(valeur)
            print(f"{self.name} a produit {valeur}")
            time.sleep(random.uniform(1, 5))



class Consommateur(Thread):
    def __init__(self, file, rythme):
        """Crée un consommateur qui retire des éléments de la file file"""
        assert isinstance(file, File)
        assert isinstance(rythme, int) and rythme > 0
        super().__init__()
        self.file = file
        self.rythme = rythme

    def run(self):
        while True:
            valeur = self.file.retirer()
            print(f"{self.name} a consommé {valeur}")
            time.sleep(self.rythme)



if __name__ == "__main__":
    file = File(5)
    producteurs = [Producteur(file) for _ in range(2)]
    consommateurs = [Consommateur(file, rythme=2) for _ in range(2)]


    for producteur in producteurs:
        producteur.start()
```

```
88
89      for consommateur in consommateurs:
90          consommateur.start()
91
92
93      for producteur in producteurs:
94          producteur.join()
95
```

Partie 1 : Initialisation de la classe

```python
class File:
    def __init__(self, size):
        """Crée une file d'attente bornée de taille size"""
        assert isinstance(size, int) and size > 0
        self.size = size
        self.tete = 0
        self.queue = -1
        self.nb_elements = 0
        self.contenu = [None] * size
        self.condition = Condition()
```

- Cette classe **initialise une file d'attente bornée**, permettant de synchroniser les accès grâce à `Condition`.
- Le paramètre `size` définit la **taille maximale** de la file.
- Les variables comme `tete`, `queue`, et `contenu` permettent de gérer la position des éléments dans la file.

Partie 2 : Méthodes de contrôle

```python
    def vide(self):
        with self.condition:
            return self.nb_elements == 0

    def pleine(self):
        with self.condition:
            return self.nb_elements == self.size
```

- La méthode `vide` vérifie si la file ne contient aucun élément.
- La méthode `pleine` vérifie si la file a atteint sa capacité maximale.
- Ces méthodes utilisent un verrou `Condition` pour **éviter les accès concurrents**.

Partie 3 : Ajout d'un élément à la file

```
    def ajouter(self, objet):
        with self.condition:
            while self.pleine():
                self.condition.wait()
            self.queue = (self.queue + 1) % self.size
            self.contenu[self.queue] = objet
            self.nb_elements += 1
            print(f"Ajouté : {objet}")
            self.condition.notify_all()
```

- La méthode `ajouter` permet **d'ajouter un élément** dans la file si elle n'est pas pleine.
- L'appel à `self.condition.wait()` **met en attente** le thread si la file est pleine.
- Une fois l'élément ajouté, `notify_all` réveille les autres threads en attente.

Partie 4 : Retrait d'un élément de la file

```
    def retirer(self):
        with self.condition:
            while self.vide():
                self.condition.wait()
            resultat = self.contenu[self.tete]
            self.tete = (self.tete + 1) % self.size
            self.nb_elements -= 1
            print(f"Retiré : {resultat}")
            self.condition.notify_all()
            return resultat
```

- La méthode `retirer` permet de **retirer un élément** de la file si elle n'est pas vide.
- Elle utilise également `wait` pour attendre si la file est vide.
- Cette méthode met à jour l'indice `tete` pour pointer vers le prochain élément.

Partie 5 : Classe Producteur

```
class Producteur(Thread):
    def __init__(self, file):
        """Crée un producteur qui ajoute des éléments à la file file"""
        assert isinstance(file, File)
        super().__init__()
        self.file = file
```

- La classe `Producteur` hérite de `Thread` pour **exécuter des tâches en parallèle**.
- Le constructeur prend en paramètre une instance de la classe `File` pour y ajouter des éléments.

Partie 6 : Classe Consommateur

```
class Consommateur(Thread):
    def __init__(self, file, rythme):
        """Crée un consommateur qui retire des éléments de la file file"""
        assert isinstance(file, File)
        assert isinstance(rythme, int) and rythme > 0
        super().__init__()
        self.file = file
        self.rythme = rythme
```

- La classe `Consommateur` hérite aussi de `Thread` pour retirer des éléments.
- Le paramètre `rythme` définit **le délai entre deux retraits d'éléments**.

Partie 7 : Tests

```
if __name__ == "__main__":
    file = File(5)
    producteurs = [Producteur(file) for _ in range(2)]
    consommateurs = [Consommateur(file, rythme=2) for _ in range(2)]
```

- Ce bloc **crée une file d'attente** de taille 5.
- Il initialise **2 producteurs** et **2 consommateurs** pour travailler en parallèle.

```
    for producteur in producteurs:
        producteur.start()

    for consommateur in consommateurs:
        consommateur.start()
```

- Les threads des producteurs et consommateurs sont **démarrés simultanément** avec `start`.

```
    for producteur in producteurs:
        producteur.join()
```

- La méthode `join` **attend la fin** des producteurs avant de terminer le programme.

Exercice 2 : Simulation d'un commutateur Ethernet

Objectifs

- **Implémenter une table de communication** pour gérer les adresses MAC et leurs ports associés.
- **Synchroniser les accès** aux ressources partagées grâce aux verrous (`RLock`) et conditions (`Condition`).

- **Gérer la suppression automatique** des adresses MAC via une temporisation (`Tempo`).
- Simuler un switch réseau capable de :
  - Relayer ou diffuser les trames Ethernet.
  - Mettre à jour dynamiquement la table de communication.
- **Manipuler des trames Ethernet** à travers les classes `MAC` et `TrameEthernet`.
- **Gérer des ports en parallèle** pour envoyer et recevoir des trames via la classe `Port`.

Code Complet

```
import string
import time
from threading import Thread, RLock, Condition


# Classe MAC
class MAC:
    def __init__(self, adr):
        assert isinstance(adr, str) and len(adr) == 12 and all(c in string.hexdigits for c in adr)
        self.adr = adr

    def __cmp__(self, other):
        assert isinstance(other, MAC)
        if self.adr == other.adr:
            return 0
        else:
            return 1

    def __str__(self):
        return self.adr

    def __setattr__(self, att, val):
        if att == "adr":
            assert isinstance(val, str) and len(val) == 12 and all(c in string.hexdigits for c in val)
        self.__dict__[att] = val

    def __hash__(self):
        return hash(self.adr)


# Classe TrameEthernet
class TrameEthernet:
    def __init__(self, src, dst, data):
        assert isinstance(src, MAC)
        assert isinstance(dst, MAC)
        assert isinstance(data, list)
        self.src = src
        self.dst = dst
        self.data = data

    def getSrc(self):
        return self.src
```

```
43
44      def getDst(self):
45          return self.dst
46
47      def getDate(self):
48          return self.data
49
50      def __str__(self):
51          return "%s:%s" % (self.src, self.dst)
52
53
54  # Classe Tempo
55  class Tempo(Thread):
56      def __init__(self, tab, adrmac, ttl):
57          assert isinstance(tab, ComTable)
58          assert isinstance(adrmac, MAC)
59          assert isinstance(ttl, int) and ttl > 0
60          Thread.__init__(self)
61          self.tab = tab
62          self.adrmac = adrmac
63          self.ttl = ttl
64          self.running = False
65
66      def run(self):
67          self.running = True
68          time.sleep(self.ttl)
69          if self.running:
70              self.tab.remove(self.adrmac)
71
72      def stop(self):
73          self.running = False
74
75
76  # Classe ComTable
77  class ComTable:
78      def __init__(self, ttl):
79          assert isinstance(ttl, int) and ttl > 0
80          self.dict = dict()
81          self.temp = dict()
82          self.ttl = ttl
83          self.lock = RLock()
84
85      def put(self, adrmac, port):
86          assert isinstance(adrmac, MAC) and isinstance(port, Port)
87          self.lock.acquire()
88          if adrmac in self.dict:
89              self.temp[adrmac].stop()
90          self.dict[adrmac] = port
91          self.temp[adrmac] = Tempo(self, adrmac, self.ttl)
92          self.temp[adrmac].start()
93          self.lock.release()
94
95      def keys(self):
96          self.lock.acquire()
97          res = self.dict.keys()
98          self.lock.release()
```

```
        return res

    def remove(self, adrmac):
        self.lock.acquire()
        del self.dict[adrmac]
        self.lock.release()

    def get(self, adrmac):
        self.lock.acquire()
        res = self.dict[adrmac]
        self.lock.release()
        return res

    def __str__(self):
        str_table = ""
        for k in self.dict.keys():
            str_table += k + " : " + str(self.dict[k]) + "\n"
        return str_table


# Classe Port
class Port(Thread):
    def __init__(self, num, sw):
        assert isinstance(sw, Switch)
        assert isinstance(num, int) and num >= 0
        Thread.__init__(self)
        self.num = num
        self.switch = sw
        self.idle = True
        self.cond = Condition()
        self.lock = RLock()

    def run(self):
        while True:
            self.cond.acquire()
            while self.idle:
                self.cond.wait()
            self.cond.notify()
            self.cond.release()
            self.idle = True

    def send(self, trame):
        assert isinstance(trame, TrameEthernet)
        self.idle = False
        self.lock.acquire()
        print("Send out {} on port : {}".format(trame, self.num))
        self.lock.release()

    def receive(self, trame):
        assert isinstance(trame, TrameEthernet)
        self.idle = False
        self.lock.acquire()
        self.switch.commute(trame, self)
        self.lock.release()

    def __str__(self):
```

```
        return str(self.num)


# Classe Switch
class Switch:
    def __init__(self, nbPort, ttl):
        assert isinstance(nbPort, int) and nbPort > 0 and nbPort % 2 == 0
        assert isinstance(ttl, int) and ttl > 0
        self.ports = []
        self.ttl = ttl
        for i in range(nbPort):
            self.ports.append(Port(i, self))
        for port in self.ports:
            port.start()
        self.comTable = ComTable(self.ttl)
        self.lock = RLock()

    def getPort(self, num):
        assert isinstance(num, int) and num >= 0 and num < len(self.ports)
        return self.ports[num]

    def getNbPort(self):
        return len(self.ports)

    def commute(self, trame, inPort):
        assert isinstance(trame, TrameEthernet)
        assert inPort in self.ports
        self.lock.acquire()
        print("\nRéception {} on Port : {}".format(trame, inPort))
        macSrc = trame.getSrc()
        macDst = trame.getDst()

        self.comTable.put(macSrc, inPort)
        if macDst in self.comTable.keys():
            if self.comTable.get(macDst) != inPort:
                self.comTable.get(macDst).send(trame)
        else:
            self.broadcast(trame, inPort)
        self.lock.release()

    def broadcast(self, trame, port):
        assert isinstance(trame, TrameEthernet)
        assert port in self.ports
        for p in self.ports:
            if p != port:
                p.send(trame)


# Programme principal
if __name__ == "__main__":
    a = MAC("AABBCCDD0910")
    b = MAC("AABBCCDD0911")
    t1 = TrameEthernet(a, b, [])
    t2 = TrameEthernet(b, a, [])
    s = Switch(4, 2)
    s.getPort(1).receive(t1)
```

```
    s.getPort(0).receive(t2)
    s.getPort(1).receive(t1)

```

Partie 1 : Classe MAC

```
class MAC:
    def __init__(self, adr):
        assert isinstance(adr, str) and len(adr) == 12 and all(c in
string.hexdigits for c in adr)
        self.adr = adr
```

- La classe **MAC** représente une adresse MAC sous forme hexadécimale.
- Le constructeur vérifie que l'adresse est composée de 12 caractères hexadécimaux.

Partie 2 : Classe TrameEthernet

```
class TrameEthernet:
    def __init__(self, src, dst, data):
        assert isinstance(src, MAC)
        assert isinstance(dst, MAC)
        assert isinstance(data, list)
        self.src = src
        self.dst = dst
        self.data = data
```

- La classe TrameEthernet représente une trame réseau avec :
  - Une adresse source (`src`) et destination (`dst`).
  - Des données (`data`), sous forme de liste.

Partie 3 : Classe Tempo

```
class Tempo(Thread):
    def __init__(self, tab, adrmac, ttl):
        assert isinstance(tab, ComTable)
        assert isinstance(adrmac, MAC)
        assert isinstance(ttl, int) and ttl > 0
        Thread.__init__(self)
        self.tab = tab
        self.adrmac = adrmac
        self.ttl = ttl
```

- La classe **Tempo** gère une **temporisation** pour supprimer une adresse MAC après expiration du TTL.
- Elle s'exécute en parallèle grâce à l'héritage de `Thread`.

Partie 4 : Classe ComTable

```
class ComTable:
    def __init__(self, ttl):
        assert isinstance(ttl, int) and ttl > 0
        self.dict = dict()
        self.temp = dict()
        self.ttl = ttl
        self.lock = RLock()
```

- La classe **ComTable** implémente une **table de communication** pour associer les adresses MAC à des ports.
- Elle utilise un verrou (`RLock`) pour synchroniser les accès concurrents.

Partie 5 : Ajout d'une adresse dans la table

```
    def put(self, adrmac, port):
        assert isinstance(adrmac, MAC) and isinstance(port, Port)
        self.lock.acquire()
        if adrmac in self.dict:
            self.temp[adrmac].stop()
        self.dict[adrmac] = port
        self.temp[adrmac] = Tempo(self, adrmac, self.ttl)
        self.temp[adrmac].start()
        self.lock.release()
```

- La méthode `put` ajoute une adresse MAC dans table et lance un **timer** (`Tempo`) pour supprimer l'entrée après le TTL.
- Si l'adresse existe déjà, l'ancienne temporisation est arrêtée.

Partie 6 : Classe Port

```
class Port(Thread):
    def __init__(self, num, sw):
        assert isinstance(sw, Switch)
        assert isinstance(num, int) and num >= 0
        Thread.__init__(self)
        self.num = num
        self.switch = sw
        self.idle = True
        self.cond = Condition()
        self.lock = RLock()
```

- La classe **Port** représente un port du switch.
- Elle permet d'envoyer et de recevoir des trames réseau.
- Chaque port fonctionne en **parallèle** grâce à l'héritage de `Thread`.

Partie 7 : Classe Switch

```
class Switch:
    def __init__(self, nbPort, ttl):
        assert isinstance(nbPort, int) and nbPort > 0 and nbPort % 2 == 0
        assert isinstance(ttl, int) and ttl > 0
        self.ports = []
        self.ttl = ttl
        for i in range(nbPort):
            self.ports.append(Port(i, self))
        for port in self.ports:
            port.start()
        self.comTable = ComTable(self.ttl)
        self.lock = RLock()
```

- La classe **Switch** représente un commutateur réseau avec plusieurs ports.
- Elle initialise les ports et une table de communication (`ComTable`).
- Chaque port est exécuté en parallèle grâce à des threads.

Partie 8 : Méthode commute (commutation)

```
    def commute(self, trame, inPort):
        assert isinstance(trame, TrameEthernet)
        assert inPort in self.ports
        self.lock.acquire()
        print("\nRéception {} on Port : {}".format(trame, inPort))
        macSrc = trame.getSrc()
        macDst = trame.getDst()

        self.comTable.put(macSrc, inPort)
        if macDst in self.comTable.keys():
            if self.comTable.get(macDst) != inPort:
                self.comTable.get(macDst).send(trame)
        else:
            self.broadcast(trame, inPort)
        self.lock.release()
```

- La méthode commute gère la commutation des trames :
  - Ajout de l'adresse source à la table de communication.
  - Si l'adresse de destination est connue, la trame est envoyée vers le port correspondant.
  - Sinon, la trame est **diffusée** à tous les ports.

Partie 9 : Tests

```
if __name__ == "__main__":
    a = MAC("AABBCCDD0910")
    b = MAC("AABBCCDD0911")
    t1 = TrameEthernet(a, b, [])
    t2 = TrameEthernet(b, a, [])
    s = Switch(4, 2)
    s.getPort(1).receive(t1)
    s.getPort(0).receive(t2)
    s.getPort(1).receive(t1)
```

- Le test créé :
    - Crée deux adresses MAC et des trames Ethernet associées.
    - Initialise un **switch** avec 4 ports et un TTL de 2 secondes.
    - Simule la **réception** de trames sur les ports du switch.