Rapport CCTP

Grégory GUICHARD, L3 informatique

13 novembre 2018

Résumé

Durant l'année de L3 informatique, les étudiants participent à une UE de programmation concurrente. La programmation concurrente est un paradigme de programmation tenant compte, dans un programme, de l'existence de plusieurs piles sémantiques qui peuvent être appelées threads, processus ou tâches. Elles sont matérialisées en machine par une pile d'exécution et un ensemble de données privées. A la fin de cette UE, les étudiants doivent maîtriser ce concept grâce aux différents travaux pratiques effectués.

1 INTRODUCTION

Pour ce TP il faut écrire un programme qui crée une file d'entiers de capacité maximale 20 ainsi qu'un thread producteur et plusieurs threads consommateurs se partageant cette file et répétant indéfiniment les actions suivantes.

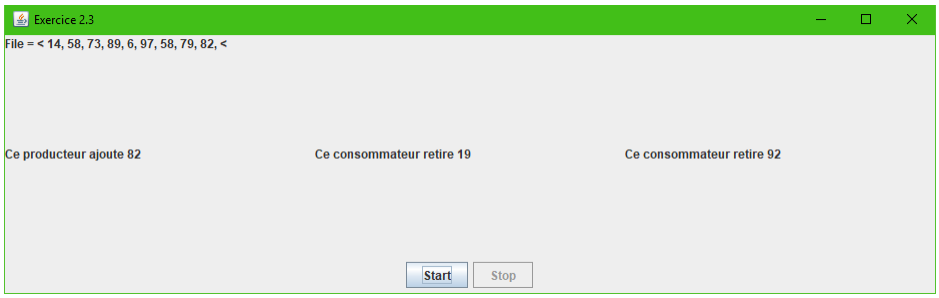
- **[Producteur]** Ajout d'un entier choisi aléatoirement entre 1 et 100 puis repos. Si la file est pleine, ce thread est mis en attente jusqu'à ce qu'une place soit disponible.
- **[Consommateurs]** Retrait d'un entier, affichage de cet entier puis repos. Si la file est vide, le thread est mis en attente jusqu'à ce qu'un entier soit disponible.

Les temps de repos sont fixés à la création des threads. Le programme est écrit en Java et en Python avec une interface graphique.

2 Le programme en Java

2.1 Interface graphique

Pour la programmation en Java, l'interface graphique a été réalisé grâce à Swing[3]. Voici l'aperçu de la fenêtre lorsque le programme est en cours :



La fenetre est divisé en 3 parties principales :

- **La zone nord** Dans cette zone est affiché, la file après une modification.
- **La zone centrale** Elle permet d'afficher les modifications apportées par les producteurs et les consommateurs.
- **La zone sud** Cette zone comprends deux boutons. L'un pour commencer ou reprendre le programme, l'autre pour le mettre en pause.

```
//La zone nord
container.setLayout(new BorderLayout());
container.add(afffile, BorderLayout.NORTH);

//La zone centrale
zoneModif.setLayout(new GridLayout());
zoneModif.add(modProd);
zoneModif.add(modCons1);
zoneModif.add(modCons2);
container.add(zoneModif, BorderLayout.CENTER);

//la zone sud
zoneBouton.add(boutonStart);
zoneBouton.add(boutonStop);
container.add(zoneBouton, BorderLayout.SOUTH);
```

2.2 La variable File

Le point important de ce programme est la variable File. Elle doit être accessible à tous les producteurs et les consommateurs mais ils ne peuvent pas la modifier simultanément. Pour cela, j'utilise une variable de type LinkedList[1]. Cette liste contient uniquement des entiers.

```
private LinkedList<Integer> laFile;
```

Mon choix s'est porté sur ce type de variable car l'ajout et la suppression des valeurs sont simplifés. Pour l'ajout, j'utilise la méthode add() et pour la suppression, la méthode remove(). Avec cette méthode, lors de la suppression, les indices sont automatiquement gérés et les nombres décalés. Je n'ai donc pas à le faire manuellement. Pour empêcher une utilisation simultanée par différents producteurs ou consommateurs les méthodes sont verrouillées grâce au mot clé synchronized.

```
public synchronized void empiler(int nombre) throws InterruptedException {
    //Si le nombre est négatif on sort de la méthode
    if (nombre <= 0) return;
    //Tant que la file est supérieur à 20, on attends pour rajouter un nombre dans la file
    while (this.laFile.size() >= 20) { wait(); }
    //On ajoute le nombre a la file
    laFile.add(nombre);
    notifyAll();
}

public synchronized int depiler() throws InterruptedException {
    //Si la file est vide, on attends pour enlever en nombre
    while (this.laFile.size() == 0) { wait(); }
    //On enleve le premier nombre de la file
    int PremierNombre = laFile.get(0);
    laFile.remove();
    notifyAll();
    return PremierNombre;
}
```

Les méthodes sont respectivement misent en attente si un producteur souhaite empiler un nombre sur une file pleine ou si un consommateur souhaite depiler un nombre sur une file vide grâce à la méthode wait(). Ils sont ensuite invités à réessayer grâce à la méthode notifyAll() qui permet de mettre fin à la pause.

3 Le programme en Python[2]

3.1 Interface graphique

L'interface graphique est réalisée avec Tkinter[4]. Comparée à celle de Java, elle possède une zone pour afficher la file lors des modifications et une autre zone pour afficher les modifications faites par les consommateurs et les producteurs.


```
self.btnStart = Button(self)
self.btnStart["text"] = "Start"
self.btnStart["command"] = self.start
self.btnStart.pack(side="bottom")

self.afficheFile = Label(self, text="File")
self.afficheFile.pack(side="top")

self.affProd = Label(self, text="Producteur")
self.affProd.pack()
self.affCons1 = Label(self, text="Consommateur 1")
self.affCons1.pack()
self.affCons2 = Label(self, text="Consommateur 2")
self.affCons2.pack()
```

3.2 La variable File

En python, j'ai utilisé un tableau pour la variable file. En effet, le langage permet facilement de gérer ce type de variable. Il est inutile de gérer les indices lors de l'ajout ou de la suppression d'une valeur. Pour empêcher une utilisation simultanée par différents producteurs ou consommateurs les méthodes sont verrouillées grâce à un verrou créé avec la classe Condition().

```
class File():
    def __init__(self):
        self.tab = []
        self.cond = Condition()

    def add(self,nb):
        with self.cond:
```

```
            while len(self.tab) >= 20:
                self.cond.wait()
            self.tab = self.tab + [nb]
            print("++ Ajout de " + str(nb))
            self.cond.notifyAll()

    def pop(self):
        with self.cond:
            while len(self.tab) == 0:
                self.cond.wait()
            print("-- Retrait de " + str(self.tab.pop(0)))
            self.cond.notifyAll()
```

Ici aussi, les méthodes sont respectivement mise en attente grâce à la méthode wait() et réveillées grâce à la méthode notifyAll().

4 Conclusion

Ce TP est l'un de mes premiers pas dans le langage Python. il m'a fallu un temps d'adaptation en ce qui concerne la syntaxe. En ce qui me concerne, le programme en Python n'est pas terminé. En effet, l'interface graphique ne permet pas d'afficher la file ni les modifications apportées par les producteurs et les consommateurs. J'ai sans cesse rencontré la même erreur : *TabError : inconsistent use of tabs and spaces in indentation.* Cette erreur est dûe à la syntaxe de Python qui n'utilise pas d'accolades mais l'indentation. Même après avoir recommencé l'écriture du code à plusieurs reprises, l'erreur revenait à des moments différents ce qui m'a bloqué pour terminer le programme.

Références

[1] Linkedlist. https://docs.oracle.com/javase/7/docs/api/java/util/LinkedList.html#LinkedList().

[2] Python. https://docs.python.org/3/contents.html.

[3] Swing. https://docs.oracle.com/javase/tutorial/uiswing/index.html.

[4] Tkinter. https://docs.python.org/2/library/tkinter.html.