

Devoir Programmation Concurrente

Matthias Coutin 36000206, L3 informatique

13 novembre 2018

Résumé

Dans ce rapport j'expliquerai les étapes de la mise en oeuvre de l'exercice. L'exercice consistant à implémenter un objet File de façon à ce qu'elle puisse être gérée par des Threads, en prenant en compte l'exclusion mutuelle, puis de créer un thread producteur et plusieurs Threads Consommateurs, le tout afficher dans une interface graphique.

1 Conception de l'exercice

Définition des objets clés de l'exercice.

Outils utilisés :

- *Swing* [1] pour l'interface Java
- *Tkinter* [2] Pour l'interface Python

1.1 Conception de la File

Une File est un type de donnée où le stockage et l'accès est effectué par la règle FIFO (First In First Out), qui signifie le premier élément entrant et sera le premier sortant de la file. Pour son implémentation nous avons besoins :

- De la taille de la file, le nombre d'élément qu'elle contient
- D'une fonction enfilé qui ajoute un élément en fin de file
- Et d'une fonction défilé qui retire l'élément en début file
- D'une fonction pour savoir si la file est vide
- D'une fonction pour savoir si la file est pleine

La fonction file vide est nécessaire pour la fonction défilée car si la file est vide on ne peut pas retirer d'éléments, de même pour la fonction file pleine et enfilée, on ne peut pas ajouter un élément quand la file est pleine.

1.2 Conception des Threads

Les Threads qui seront créés sont un thread producteur et un thread consommateur qui se partageront la file comme ressource commune. Le thread Consommateur appellera la fonction défilée et le thread Producteur appellera la fonction empilée, mais pour pouvoir assurer l'exclusion mutuelle, l'utilisation d'un verrou est requis. Le principe était que quand le thread Consommateur accède à la ressource, la ressource est verrouillée pour le thread Producteur et inversement. De plus nous utiliserons les fonctions wait et notifyall pour mettre en attente le travail du thread Consommateur quand la file est vide et mettre en attente le travail du thread Producteur quand la file est pleine, la mise en attente d'un thread Consommateur sera suivie du réveil du thread Producteur et inversement. J'implémenterais aussi l'interruption des Threads. pour leurs arrêts.

1.3 Interface Graphique

L'affichage des différents éléments se fera comme suit :

- Un label (texte) pour l'affichage de la file

- Un label pour afficher le travail du Producteur
- Un label pour afficher le travail du Consommateur
- Un bouton d'arrêt et de reprise des Threads
- Un bouton pour quitter le programme

2 Implementation en Python et en Java

2.1 La Class File

En Python :

```

8
9 class File(Condition):
10     """docstring for File. Prend en parametre une taille max : size"""
11     def __init__(self, size):
12         super(File, self).__init__()
13         self.size = size
14         self.f = list()
15         self.verr = Condition() # Verrou
16
17     """ Méthode enqueue, ajoute un element en tete de file """
18     def enqueue(self, elem):
19         with self.verr: # Gestion du verrou
20             while (len(self.f) >= self.size): # Gestion si file pleine
21                 self.verr.wait() # Attente si file pleine
22                 self.f.insert(0,elem)
23                 self.verr.notifyAll()
24
25     """ Methode dequeue, retire un element en debut file
26     ( l'ajout des element se fait à la suite )"""
27     def dequeue(self):
28         with self.verr: # Gestion du verrou
29             while (len(self.f) <1): # Gestion si file vide
30                 self.verr.wait() # Attente si file vide
31                 a = self.f.pop()
32                 self.verr.notifyAll()
33                 return a
34

```

En Java :

```

5 public class File {
6     int size;
7     ArrayList<Integer> f;
8
9     public File(int size) {           // Constructeur prend une taille max
10                                     // argument size
11         this.f = new ArrayList<Integer>();
12         this.size = size;
13     }
14
15     public synchronized void add(int elem) { /* Gestion de l'ajout d'un element elem
16                                             et gestion du verrou synchronized*/
17         while(f.size() >= this.size) {    // Gestion si la file est pleine
18             try {
19                 wait();
20             } catch (InterruptedException e) {}
21         }
22         f.add((Integer)elem);    // Ajout de l'element à la file
23         notifyAll();
24     }
25
26     public synchronized int remove() { // suppression d'un element dans la file
27         while(f.size()==0) {           // Gestion si file vide
28             try {
29                 wait();
30             } catch (InterruptedException e) {}
31         }
32         int a = f.remove(0);    // Suppression d'un element
33         notifyAll();
34         return a;
35     }
36 > public String displayFile() {=
50
51 }
--

```

2.2 La Class Producteur et Consommateur

En Python :

```

61 class Prod(Thread):
62     """docstring for Prod. Prend en parametre une file n, un temps attente att
63     son label lab et le label de la file file"""
64     def __init__(self, n,att, lab,file):
65         Thread.__init__(self)
66         self.interrupt = True
67         self.n = n
68         self.att = att
69         self.daemon = True # Arret du Thread quand le programme principal et arreter
70         self.lab = lab
71         self.file = file
72
73     def run(self):
74         self.interrupt = True # Gestion l'interruption
75         while self.interrupt:
76             a = random.randint(0,100)
77             self.n.enqueue(a)
78             self.lab.config(text="Producteur : (+)" + str(a)) # Affichage du Travail
79                                                         # du producteur
80             self.file.config(text = str(self.n.f)) # Affichage de la file
81             time.sleep(self.att/1000)
82
83     def stop(self): # Fonction qui gère l'arret du Thread
84         self.interrupt = not self.interrupt

```



```

36 class Conso(Thread):
37     """docstring for Conso. Prend en parametre une file n, un temps attente att
38     son label lab et le label de la file file"""
39     def __init__(self, n,att, lab,file):
40         Thread.__init__(self)
41         self.interrupt = True
42         self.n = n
43         self.att = att
44         self.daemon = True # Arret du Thread quand le programme principal et arreter
45         self.lab = lab
46         self.file = file
47
48     def run(self):
49         self.interrupt = True # Gestion l'interruption
50         while self.interrupt:
51             e = self.n.dequeue() # retire l'élément dans la file
52             self.lab.config(text="Consomateur : (-) " + str(e)) # Affichage du Travail
53                                                         # du consommateur
54             self.file.config(text = str(self.n.f)) # Affichage de la file
55             time.sleep(self.att/1000)
56
57     def stop(self): # Fonction qui gère l'arret du Thread
58         self.interrupt = not self.interrupt
59

```

En Java :

```

5 public class Prod extends Thread {
6     private File f;
7     private int t;
8     private JLabel prod;
9     private JLabel file;
10
11     public Prod(File f, int t, JLabel prod, JLabel file) {
12         /*Prend en parametre une file f, un temps attente t
13         son label prod et le label de la file file*/
14         this.f = f;
15         this.t = t;
16         this.prod = prod;
17         this.file = file;
18         setDaemon(true); // arret du Thread si le programme principal s'arrete
19     }
20
21     public void run(){
22         try {
23             while(!interrupted()) { // gestion de interruption
24                 int a = (int)(Math.random()*100);
25                 this.prod.setText("Producteur : " + "(+) "+a); // Affichage de l'element à ajouter
26                 this.file.setText(this.f.displayFile()); // Affichage de la file
27                 f.add(a); // ajout de l'element dans la file
28                 sleep(t);
29             }
30
31         } catch(InterruptedException e) {}
32     }
33 }
34

```

```

5 public class Conso extends Thread {
6     private File f;
7     private int t;
8     private JLabel cons;
9     private JLabel file;
10
11     public Conso(File f, int t, JLabel cons, JLabel file) {
12         /*Prend en parametre une file f, un temps attente t
13         son label cons et le label de la file file*/
14         this.f = f;
15         this.t = t;
16         this.cons = cons;
17         this.file = file;
18         setDaemon(true); // arret du Thread si le programme principal s'arrete
19     }
20
21     public void run(){
22         try {
23
24             while(!interrupted()) { // gestion de interruption
25
26                 int a = f.remove(); // retire un element de la file
27                 this.cons.setText("Consomateur : " + "(-) "+a); // affichage de l'element retiré
28                 this.file.setText(this.f.displayFile()); // Affichage de la file
29
30                 sleep(t);
31             }
32         } catch (InterruptedException e) {}
33     }
34 }
35

```

3 Résultat

Affichage graphique des fenêtres, des programmes en Python et Java :
En Python :



En Java :



Références

- [1] Swing. <https://docs.oracle.com/javase/tutorial/uiswing/index.html>.
- [2] Tkinter. <https://docs.python.org/2/library/tkinter.html>.