UNIVERSITE PARIS 8 Vincennes - Saint-Denis

RAPPORT DU PROJET : PRODUCTEUR – CONSOMMATEUR

Université Paris 8 – Vincennes Saint Denis UFR MISTIC

MASTER 1 INFORMATIQUE

PROJET EN PROGRAMMATION CONCURRENTE

Membres :

Sarah KADI

Abdelghani AITHAMOU

Aziz BELKACEMI

Dirigé par :

Mr. Ilyes MEHADDI

UNIVERSITE PARIS 8 Vincennes - Saint-Denis

RAPPORT DU PROJET : PRODUCTEUR – CONSOMMATEUR

1. **Description du projet** :

Dans un environnement multi-thread et dans différents contextes de programmation concurrente, le projet représentera un exemple informatique de synchronisation de ressources.

Il s'agit de partager entre deux tâches (producteur et consommateur) une zone de mémoire tampon, représentée dans ce projet par une liste.

Le producteur génère un nombre aléatoire entre 10 et 99 et l'ajoute dans la liste, simultanément, le consommateur affiche cette valeur et la retire à son tour de la liste.

2. **Outils utilisés** :

- Langage utilisé :
  - Python

- Technologies utilisées :
  - Pycharm
  - Github

3. **Implémentation** :

- Les classes utilisées :
  - La classe « Producer » : définie le producteur et contient deux méthodes principales ;
    - randomNumber() : génère un nombre aléatoire entre 10 et 99.

```
def randomNumber(self) :
    return random.randint(10,99)
```

    - wait() : met le thread courant en pause pour une durée de 3 secondes.

```
def wait(self) :
    time.sleep(3)
```

  - La classe « Consumer » : définie le consommateur et contient deux méthodes principales ;
    - displayNumber() : affiche un nombre prit en paramètres.

```
def displayNumber(self):
    return self.number
```

    - wait() : met le thread courant en pause pour une durée de 2 secondes.

```
def wait(self) :
    time.sleep(2)
```

- Le fichier main :

Contient :

- Une classe « TProducer » qui se charge d'initialiser et d'instancier la class « Producer » et qui s'occupe d'ajouter le nombre aléatoire généré dans la liste de données partagées.

- Une classe « TConsumer » qui se charge d'initialiser et d'instancier la class « Consumer » et qui s'occupe d'afficher la donnée lue dans la liste, puis retire cette dernière de la zone mémoire partagée.

Deux cas principaux sont gérés dans cette classe :

- Cas 1 : nombre de producteurs inférieur ou égale au nombre de consommateurs : la gestion est simple, si le dernier producteur produit la donnée, le consommateur courant consomme cette dernière et s'arrête, entrainant ainsi l'arrêt du programme.
- Cas 2 : nombre de producteurs supérieur au nombre de consommateurs : l'ajout d'un consommateur étant impossible, la gestion de ce cas fut de telle sorte que le dernier consommateur ne cesse de consommer la donnée jusqu'à ce que la liste soit vide (que tous les producteurs aient fini de produire les données).

Dans le but d'une bonne gestion de la mémoire partagée, l'ajout de verrous a été primordiale.

Le « main » initialise une liste vide, crée les threads, les lancent, attend la fin de ces derniers et affiche le résultat.

4. **Résultat de l'exécution** :

Après l'exécution du « main » à l'aide de la commande « python main.py » voici un exemple de résultat obtenu :

```
(base) sarahkadi@macbook-air-de-sarah Projet_Programmation_concurrente % python main.py
*Production de la donnée   : 18  | Liste : [18]

*Consommation de la donnée : 18  | Liste : []

*Production de la donnée   : 90  | Liste : [90]

*Consommation de la donnée : 90  | Liste : []

*Production de la donnée   : 37  | Liste : [37]

*Consommation de la donnée : 37  | Liste : []

*Production de la donnée   : 64  | Liste : [64]
*Consommation de la donnée : 64  | Liste : []

*Consommation de la donnée : 71  | Liste : []
*Production de la donnée   : 70  | Liste : [70]

*Production de la donnée   : 77  | Liste : [77]

*Production de la donnée   : 25  | Liste : [25]

*Production de la donnée   : 71  | Liste : [71]


*Consommation de la donnée : 70  | Liste : []

*Consommation de la donnée : 77  | Liste : []

*Consommation de la donnée : 25  | Liste : []


Toutes les données ont été consommées.

Résultat du contenu de la liste après l'execution : []
```