Programmation concurente

Laboratoire 3 - Gestion d’un aéroport

HE-Arc

Pedro Costa et Killian Castella

# Rappel des objectifs

Le but de ce travail était de simuler la gestion d’un aéroport comportant un nombre défini de pistes d’atterrissage, de décollage et de places au sol à l’aide de mécanisme de concurrence Java.

Un autre objectif est d’implémenter ce problème de deux manières différentes :

* Avec un tampon et des méthodes Synchronized
* Avec une Blocking Queue

Et d’ensuite comparer les différences de vitesse entre les deux façons de faire.

Et il est également demandé de tester l’influence du nombre d’élément dans l’aéroport.

# Descriptif de l’implémentation

## Méthode Blocking queue

La méthode de la Blocking Queue utilise l’interface BlockingQueue<E> pour créer des collections d’objet dont nous limitons le nombre. Cette méthode correspond bien à notre problème de gestion d’aéroport pour plusieurs raisons, tout d’abord elle permet de limiter le nombre d’objet dans la collections ce qui va tout à fait pour simuler un nombre de piste d’atterrissage limité ou tout autre élément limité. Egalement le fait que dans les file la logique « premier arrivé, premier servi » s’applique permet de gérer facilement l’ordre d’arrivée des avions.

Chacun des éléments limités en nombre est donc une Blocking Queue dans le programme. Dès qu’un thread se libère, le premier élément de la file d’attente est traité ce qui libère une place pour un nouvel avion.

## Méthode tampon circulaire

L’autre manière de faire était d’utiliser un tampon (buffer en anglais) circulaire, dans notre cas ce tampon est un simple tableau dont nous limitons l’accès à N cases en faisant un simple module %N sur l’indice souhaité qui est incrémenté à chaque nouvelle demande.

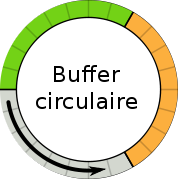


Figure Représentation d'un tampon circulaire

Par exemple imaginons un tampon limité à trois places

0%3 = 0

1%3 = 1

2%3 = 2

3%3 = 0

1%3 = 1

2%3 = 2

Etc. …

Ainsi cela nous permet de limiter artificiellement la taille de notre collection pour restreindre la taille des infrastructures de l’aéroport.

Lorsqu’un thread se libère, nous allons chercher l’élément suivant dans notre tampon circulaire pour l’éxecuter.

## Suivi durant le projet

Pour développer ce projet, nous avons commencé par implémenter le tout sans la gestion des threads. Puis nous avons ajouté cette notion de threads une fois que la base était fonctionnelle. A partir de là, il a fallu rendre le programme thread-Safe. Nous avons donc tout au long du développement fait de nombreux tests pour éviter le maximum de problème de concurrence.

# Tests de performance et comparatif

Une fois cela réalisé, nous avons effectué des tests de rapidité pour comparer l’efficacité en termes de vitesse des deux manières de faire.

Tous les tests ont été effectués sur deux ordinateurs différent pour une plus grande fiabilité des résultats.

Spécification des PCs :

|  |  |  |
| --- | --- | --- |
| Nom du PC | Nombre de cœurs physiques | Nombre de cœurs logique |
| PC 1 | 4 | 8 |
| PC 2 | 2 | 4 |

Et voici les résultats que nous avons obtenus :

PC 1 :

Temps d'exécution de la Blocking Queue : 30 269 ms

Temps d'exécution du tampon circulaire : 28 637 ms

PC 2 :

Temps d'exécution de la Blocking Queue : 29 781 ms

Temps d'exécution du tampon circulaire : 31 267 ms

On voit donc très clairement que les deux manières de faire sont très proches. En faisant plusieurs essais, il y a des inversions du meilleur temps sur les deux machines. Cela a surement été dû à des programmes tiers sur les PCs.

Par la suite nous avons testé de varier le nombre d’infrastructure de l’aéroport et de comparé à nouveau les temps. Ici les tests ont été uniquement effectués avec la méthode de la BlockingQueue

|  |  |  |
| --- | --- | --- |
| **Nombre de pistes d’atterrissage** | **Temps PC 1** | **Temps PC 2** |
| 1 | 43917 ms | 45094 ms |
| 2 | 30855 ms | 29267 ms |
| 3 | 29094 ms | 28899 ms |
| 4 | 27733 ms | 29455 ms |
| 5 | 27258 ms | 29658 ms |
| 6 | 26828 ms | 27549 ms |
| 7 | 28053 ms | 28002 ms |
| 8 | 28215 ms | 27690 ms |
| 9 | 27388 ms | 29279 ms |
| 10 | 28190 ms | 30125 ms |
| 11 | 27469 ms | 29332 ms |
| 12 | 26872 ms | 26788 ms |
| 13 | 28046 ms | 27887 ms |
| 14 | 29822 ms | 28516 ms |
| 15 | 27475 ms | 27288 ms |
| 16 | 29734 ms | 27668 ms |
| 17 | 29319 ms | 26261 ms |
| 18 | 26150 ms | 28254 ms |
| 19 | 27645 ms | 28237 ms |
| 20 | 29039 ms | 28764 ms |

|  |  |  |
| --- | --- | --- |
| **Nombre de pistes de décollage** | **Temps PC 1** | **Temps PC 2** |
| 1 | 44136 ms | 43412 ms |
| 2 | 30257 ms | 26707 ms |
| 3 | 25982 ms | 26069 ms |
| 4 | 25434 ms | 23720 ms |
| 5 | 24718 ms | 23570 ms |
| 6 | 22691 ms | 26509 ms |
| 7 | 22084 ms | 23218 ms |
| 8 | 25923 ms | 22861 ms |
| 9 | 27154 ms | 25276 ms |
| 10 | 21951 ms | 24630 ms |
| 11 | 27671 ms | 25125 ms |
| 12 | 25225 ms | 25855 ms |
| 13 | 24041 ms | 25398 ms |
| 14 | 24489 ms | 24761 ms |
| 15 | 27772 ms | 24189 ms |
| 16 | 24901 ms | 27069 ms |
| 17 | 27021 ms | 23659 ms |
| 18 | 25171 ms | 29307 ms |
| 19 | 25946 ms | 24787 ms |
| 20 | 26441 ms | 24607 ms |

|  |  |  |
| --- | --- | --- |
| **Nombre de places dans le parking** | **Temps PC 1** | **Temps PC 2** |
| 1 | 46626 ms | 47634 ms |
| 2 | 26860 ms | 24470 ms |
| 3 | 19487 ms | 19184 ms |
| 4 | 15057 ms | 15744 ms |
| 5 | 14488 ms | 12808 ms |
| 6 | 12045 ms | 14017 ms |
| 7 | 11921 ms | 12934 ms |
| 8 | 11428 ms | 11178 ms |
| 9 | 10908 ms | 11397 ms |
| 10 | 11443 ms | 9946 ms |
| 11 | 10344 ms | 10882 ms |
| 12 | 11130 ms | 10503 ms |
| 13 | 10329 ms | 9889 ms |
| 14 | 10447 ms | 9693 ms |
| 15 | 10422 ms | 10485 ms |
| 16 | 10858 ms | 9468 ms |
| 17 | 9577 ms | 9339 ms |
| 18 | 10219 ms | 10142 ms |
| 19 | 9989 ms | 10514 ms |
| 20 | 10085 ms | 9304 ms |

On constate donc assez logiquement que plus l’aéroport a d’infrastructure à disposition, plus la gestion d’un nombre donné d’avions se fait rapidement.

Ce qui est plus étonnant c’est que le nombre de cœurs du processeur n’a pas l’air d’avoir d’impact sur la vitesse d’exécution des threads.

# Conclusion

Arrivé au terme du projet, nous avons réussi à implémenter la gestion de l’aéroport selon les deux manières de faire demandées, ce qui a permis d’approfondir nos connaissances en programmation concurrente.

Néanmoins nous nous attendions à de plus grande différence d’efficacité entre les deux méthodes utilisées. Et nous avons également été surpris de voir que la puissance du processeur n’affectait pas la vitesse d’exécution, il est possible que la puissance alloué à la JVM soit limitée.