

Théorie et Pratique de la Concurrency

TP n°2 — Concurrency sur une variable partagée

Jean-Baptiste Yunès
Jean-Baptiste.Yunes@u-paris.fr

24 juillet 2026

Attention : il est interdit d'utiliser une intelligence artificielle pour répondre aux questions.

Objectifs

- comprendre comment l'accès concurrent non protégé à une même variable mène à des Data Races et des comportements non déterministes.
- découvrir l'outillage de diagnostic : l'analyseur dynamique de concurrence Go (-race) et la modélisation de la visibilité mémoire en Java.
- appliquer les solutions de base : variables atomiques et verrous d'exclusion mutuelle (Mutex, synchronized).

Go

Écrire un programme en Go permettant de lancer 1000 go routines chacune incrémentant 1000 fois une même variable partagée initialisée à 0 au départ.

Afficher le résultat après la terminaison de toutes les go routines.

Exécuter le code et observer le résultat sur différentes exécutions.

Modifier le programme de sorte à mesurer le temps d'exécution :

- soit en mesurant le temps avant les 1000 incréments, puis après et en faisant la différence avec le package `time`;
- soit en utilisant le *benchmarking* de go.

Lancer l'analyseur de *race condition* avec l'option `-race` pour retrouver où le problème se situe.

Corriger le programme de sorte que le résultat soit le bon :

- avec un entier atomique (package `sync/atomic`);
- avec un mutex (package `sync`).

Comparer les temps.

Java

Même chose en Java. Les Go routines seront donc remplacées par des Threads natifs (de plate-forme).

Pour mesurer les temps on peut utiliser :

- `System.nanoTime`;
- le *flight recorder*.

Pour la correction, on peut utiliser :

- avec un entier atomique (package `java.util.concurrent.atomic`);
- avec un verrou réentrant (package `java.util.concurrent.locks`);
- un bloc synchronisé (`synchronized`).

Pourquoi `volatile` à lui seul ne suffit pas ?