# Programmation répartie

# SDR 2022 / Labo 2

*Temps à disposition : 8 périodes.*
*Travail à réaliser avec le même groupe d'étudiants que le labo 1, ou en parler avec l'enseignant.*
*La présence aux labos est obligatoire.*
*En cas de copie manifeste entre les rendus de deux labos, tous les étudiants des deux groupes se verront affecter la note de 1.*

*Labo distribué le vendredi 21 octobre 2022 à 14h55. Vous réutiliserez le repo github **privé** du labo 1. Vous devrez pousser votre travail en l'état sur ce repo au moins une fois par semaine, en début de chaque séance de labo.*

***A rendre le vendredi 25 novembre 2022 à 14h55.** Un readme indiquant comment utiliser votre programme et précisant ce qui a été réalisé, ce qui fonctionne et ce qui reste à faire, devra être déposé dans votre repository. Un email rappelant vos noms et repository confirmera votre rendu qui sera accessible dans la branche master (ou main).*

## Objectifs

- Comprendre le fonctionnement d'un algorithme d'exclusion mutuelle réparti.
- Utiliser l'algorithme de Lamport pour maintenir la cohérence de variables réparties sur un ensemble de processus.
- Réaliser des communications TCP-IP en mode Socket en GO.
- Gérer la concurrence d'accès aux variables avec des goroutines et des channels en GO.

## Introduction et limites

Partager des variables parmi un ensemble de processus est un problème qui peut se résoudre par le biais d'un algorithme d'exclusion mutuelle. Dans ce laboratoire, nous allons utiliser **l'algorithme de Lamport** comme algorithme d'exclusion mutuelle pour gérer l'accès à la section critique.

Il s'agit de reprendre la petite application permettant la répartition de bénévoles pour l'organisation de manifestations développée dans le 1er laboratoire et de la faire fonctionner avec plusieurs serveurs, sans serveur central. Chaque client détient une copie des variables partagées et doit obtenir l'accès à des sections critiques pour y apporter des modifications et diffuser celles-ci sur les autres clients.

*Les communications se feront en mode client-serveur connecté (TCP). On supposera que les processus et le réseau qui les interconnecte sont entièrement fiables et que tous les processus lancés au démarrage restent actifs. On ne gèrera pas les déconnections.*

## Contraintes et qualité du code source

Les consignes du labo1 sur ce point restent les mêmes pour ce labo.

***Pour rappel, le non-respect des consignes ci-dessous ou un code de mauvaise qualité seront sanctionnés par des points négatifs dans l'évaluation.***

## Tests (10 points)

Les tests d'intégration automatisés en mode client-serveur développés dans le premier labo, le programme de test se comportant comme un seul client, doivent toujours fonctionner**. Si ceux développés dans le premier labo étaient insuffisants, vous devez les compléter ou les corriger.**

De la même façon également que dans le premier labo, on doit pouvoir vérifier qu'il ne présente pas de risques quant à la concurrence d'accès aux variables, en particulier en cas d'inscriptions simultanées de

bénévoles au même poste. Les serveurs doivent pouvoir être lancé en mode « debug », ce qui aura pour effet de les ralentir artificiellement afin de les mettre en situation d'accès concurrent. On activera alors 2 ou 3 clients selon un mode d'emploi documenté dans le readme et des traces d'exécution nous permettrons de vérifier la bonne sérialisation des accès concurrents. L'idée est de mettre en valeur les différentes situations possibles d'évolution de l'algorithme de Lamport. **Ces scénarios devront donc être plus élaborés que ceux fournis dans le premier labo.**

Par ailleurs, l'application doit passer le test du « go race » en multi-utilisation.

## Aspects réseau (15 points)

En suivant les directives de votre readme, on doit pouvoir cloner votre repo et y trouver un fichier de configuration qui contiendra la description du réseau (nombre de processus serveur, leurs numéros, adresses et numéros de port). Il permettra également de sélectionner l'option « debug » de ralentissement et trace pour vérifier le bon fonctionnement de l'exclusion mutuelle. On supposera le fichier de configuration sans erreurs.

Chaque serveur devra pouvoir être démarré avec son numéro en paramètre. On suppose qu'on les démarrera tous, chacun avec un numéro différent en paramètre. Lorsqu'un processus démarre, il établit des connexions avec tous les autres serveurs démarrés auparavant. Vous devez attendre que l'ensemble des serveurs soient prêt avant d'accepter la connexion des clients.

Il sera alors possible de lancer plusieurs clients en précisant leur nom, qui sera supposé unique. Un client se connectera soit sur un serveur de numéro passé explicitement en paramètre, soit aléatoirement sur un des serveurs définis dans le fichier de configuration.

Les déconnexions ou abort des clients seront correctement gérés par les serveurs. Il n'est pas demandé de gérer la panne d'un serveur.

Votre readme devra décrire clairement comment utiliser votre application et ce qui fonctionne ou non.

L'application doit fonctionner dans les environnements Windows, Mac et Linux.

## Exclusion mutuelle selon l'algorithme de Lamport (25 points)

Toutes les fonctionnalités développées dans le cadre du labo 1 devront être reprises à l'identique dans cet environnement de serveurs répartis.

Pour cela, les données des différents serveurs devront être synchronisées entre elles et leur accès devra être protégé par un mécanisme d'exclusion mutuelle. On mettra en œuvre pour cela l'algorithme de Lamport. Il est explicitement demandé d'implémenter l'algorithme de Lamport et **non** un algorithme conçu à partir de celui-ci (Ricart-Agrawala, Carvalho-Roucairol).