1. Analyse de program.cs :
2. **Rôle du Bloc 1 dans la fonction Main :**

Le bloc 1 de la fonction Main est destiné à s'assurer qu'une seule instance de l'application est en cours d'exécution. Cela se fait en utilisant un Mutex nommé, qui est une synchronisation d'objet système qui permet de restreindre l'accès à une ressource ou à un code à un seul processus à la fois.

Voici le rôle détaillé de chaque partie du bloc 1 :

* **Mutex** : Le mutex nommé est initialisé avec le nom de l'application via AppDomain.CurrentDomain.FriendlyName. Cela garantit que deux processus ayant le même nom d'application ne s'exécutent pas en parallèle.
* **Vérification de \_prevInstance** : Le deuxième paramètre de Mutex renvoie une valeur booléenne qui indique si le mutex était déjà possédé par un autre processus. Si une autre instance de l'application est déjà en cours d'exécution, alors \_prevInstance sera false, et l'application actuelle se ferme immédiatement avec return;.

**Améliorations possibles :**

* **Libération du Mutex** : Le mutex est actuellement libéré à la toute fin de la fonction Main par mutex.ReleaseMutex(). Bien que cela semble fonctionner, il serait préférable d'utiliser une structure try-finally pour garantir que le mutex soit toujours libéré, même si une exception se produit pendant l'exécution. Cela évitera les blocages éventuels si une erreur survient.
* **Gestion des ressources** : Utiliser la méthode Dispose sur le mutex garantit une libération appropriée des ressources. Le bloc pourrait être réécrit en utilisant un bloc using pour s'assurer que le mutex est libéré proprement une fois qu'il n'est plus nécessaire.
* **Message d'erreur utilisateur** : Si l'application quitte parce qu'une autre instance est déjà en cours, il serait utile d'afficher un message pour expliquer à l'utilisateur pourquoi le programme s'est fermé.

1. **Rôle du Bloc 2 dans la fonction Main :**

Le bloc 2 de la fonction Main sert à initialiser le nombre de véhicules (\_nb\_cars) qui seront utilisés dans la simulation. Voici les étapes détaillées de ce bloc :

* **Récupération de la valeur passée en argument** : Le programme tente de récupérer un argument args[0] pour déterminer le nombre de véhicules. Si cet argument n'est pas valide (par exemple, si ce n'est pas un entier), la méthode TryParse échoue et la variable \_nb\_cars prend la valeur par défaut de 100.
* **Gestion des exceptions** : Si aucune valeur n'est fournie ou si une erreur se produit (par exemple, absence de l'argument args[0]), le programme entre dans le bloc catch et assigne également la valeur par défaut .

**Améliorations possibles :**

* **Message d'avertissement à l'utilisateur** : Ajouter un message d'avertissement pour informer l'utilisateur qu'une valeur par défaut de 100 est utilisée si l'argument n'est pas fourni ou s'il est incorrect.
* **Réutilisation de la valeur par défaut** : Plutôt que d'écrire deux fois la valeur par défaut (\_nb\_cars = 100;), on peut définir une valeur par défaut en amont pour plus de lisibilité.

1. Le bloc 3 de la fonction Main utilise un événement manuel (ManualResetEvent) pour attendre que tous les threads de voiture aient terminé leur exécution avant de poursuivre. Cela garantit que le programme attend la fin de toutes les opérations avant de se terminer.

**Améliorations** :

* Ajoutez une durée d'attente maximale à l'événement WaitOne pour éviter un blocage potentiel infini.
* Affichez un message d'information indiquant la fin de l'exécution des threads de voiture

1. Analyse de la classe Cl\_server\_socket