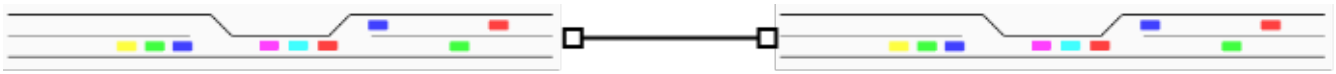

Le répertoire de ce TP dans votre **dépôt GIT** s'appellera **"TPRoadSocket".**

L'objectif de ce TP est de faire passer des voitures d'une machine à l'autre via des sockets. Idéalement, une voiture devra pouvoir traverser toutes les machines de la salle de TP à la fin de la séance. Pour cela vous allez devoir synchroniser des instances du programme qui fait passer des voitures sur la route.

Attendu

Vous avez **trois séances** pour ce TP.

Fichiers attendus:

- `OS_Nom1_Nom2/`
  - `TPRoadSocket/`
    - `.gitignore` (avec tout ce qui n'est pas attendu)
    - `Makefile`
    - `RoadSocketIn.c`
    - `RoadSocketInOut.c`
    - `RoadSocketArgs.c`
    - `RoadSocketWAN.c`

Mise en place

- Récupérez le répertorie `Reseau` situé dans le répertoire `/users/but/info/Public/Systeme/`. Ce répertoire contient un module Reseau (constitué des deux fichiers Reseau.c et Reseau.h) contenant des fonctions qui simplifient l'ouverture de connexions réseaux en mode client ou en mode serveur. **La documentation de ces fonctions est dans les □chiers Reseau.h et Reseau.c.**
- Pour démarrer, récupérez un programme fonctionnel du TP "TPThreadRoad" ou "TPSynchronisation" que vous avez réalisé plus tôt dans l'année. Si votre TP n'était pas fonctionnel, vous pouvez récupérer le programme d'un autre binôme pour démarrer sur une bonne base.

Le étapes pour mettre en place la compilation et l'exécution sont les mêmes que pour le TP précédent.

1. Création d'une voiture sur message socket

Vous répondrez à cette question dans un fichier ***RoadSocketIn.c***.

L'objectif ici est de créer une nouvelle voiture lors de la réception d'un message sur un socket.

1.1. Création d'un serveur socket

Ajouter au programme de base qui fait passer des voitures la création d'un serveur qui écoute sur un port prédéfini.

Toutes les informations nécessaires sont dans le fichier Reseau.h que vous devez avoir copier dans votre répertoire de TP.

1.2. Tests

Pour tester la réception d'un message par votre programme, utilisez la commande `"nc"` (pour *netcat*).

Par exemple, pour vous connecter en tant que client à un serveur socket écoutant sur le port 4321, vous pouvez lancer la commande :

```
$ nc localhost 4321
```

Tout ce que vous tapez au clavier ensuite est envoyé sur la socket.

Vérifiez que vous pouvez créer des voitures à l'aide de `nc` une fois que votre programme est lancé.

L'avancement des voitures doit rester dans un thread.

2. Envoi d'un message sur sortie d'une voiture

Vous répondrez à cette question dans un fichier ***RoadSocketOut.c***.

2.1. Création d'un client socket

Ajoutez au code précédent la création d'un client socket et sa connection à un serveur sur un port prédéfini.

2.2. Envoi du message sur sortie voiture

Faites en sorte que lorsqu'une voiture à terminé de traverser la route, un message soit envoyé sur la socket cliente que vous venez d'ajouter à votre programme.

2.3. Tests

Là encore vous pouvez utiliser `nc` pour tester la nouvelle fonctionnalité. Par exemple, pour lancer

un serveur écoutant sur le port 4242, vous pouvez lancer la commande :

```
$ nc -l localhost 4242
```

Une fois cette commande lancée et un client connecté, vous verrez le texte reçu sur la socket apparaître à l'écran.

Vérifiez que vous recevez bien un message lorsqu'une voiture termine sa traversée.

> La difficulté de cette question est de ne pas bloquer tout le programme sur l'attente d'une connexion entrante ou sortante. Il vous faudra donc mettre ces attentes dans des threads.

Il est fortement recommandé de faire des schémas à ce stade pour clarifier les choses.

3. Prise en compte des arguments de la ligne de commande

Vous répondrez à cette question dans un fichier ***RoadSocketArgs.c***.

Vous ajouterez la prise en compte des arguments de la ligne de commande afin de pouvoir préciser :

- le numéro de port d'écoute pour le serveur,
- le numéro de port de connexion pour le client.

4. Intégration en local

Vous répondrez à cette question dans un fichier ***RoadSocketInOut.c***.

4.1. Deux programmes en local

Lancez deux instances de votre programme (dans deux terminaux différents) avec des ports compatibles : le client de la première instance doit se connecter au serveur de la deuxième.

Vérifiez qu'une voiture se créé dans la première instance sur réception d'un message socket puis qu'elle continue dans la deuxième instance une fois la première traversée terminée.

4.2. Boucle

Testez une boucle : faites en sorte que votre programme crée le serveur et attende les connexions dans un thread puis se connecte en tant que client à ce même serveur. Démarrez la boucle en envoyant un message via `nc` à votre serveur. Vous devriez voir une voiture "boucler".

Il est possible de connecter plusieurs clients à un même serveur.

5. Passage d’une machine à l’autre - Boucle dans la salle

Vous répondrez à cette question dans un fichier ***RoadSocketWAN.c***.

Ajoutez la gestion d’un argument supplémentaire : le nom de la machine à laquelle le client doit se connecter.

Testez le passage d’une voiture d’une machine à une autre.

L’objectif collectif final est de réussir à faire faire le tour de la salle de TP à une voiture.