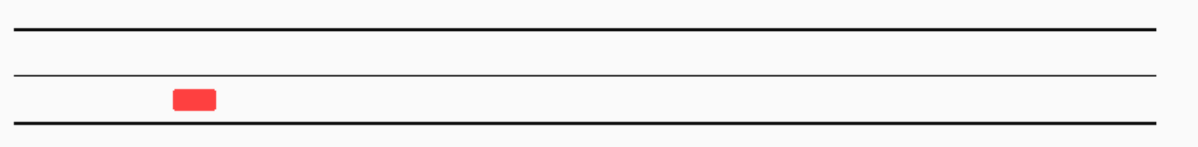
TP 3 - Threads - On the road



Le répertoire de ce TP dans votre dépôt s'appellera "TPThreadRoad".

Comme le nom du TP l'indique, vous allez créer des threads. L'objectif est de faire avancer des voitures le long d'une route, un thread par voiture.

Attendu

Vous avez **deux séances** pour réaliser ce TP.

Fichiers attendus:

- `OS_NomA_NomB`
  - `TPThreadRoad`
    - `.gitignore` (avec tout ce qui n'est pas attendu)
    - `Makefile`
    - `onTheRoad.c`
    - `onTheRoad2.c`
    - `onTheRoad3.c`

Rappel

> Ce qui n'est pas poussé sur votre dépôt GitLab n'est pas corrigé.

Pensez donc bien à effectuer des `commit` et des `push` régulièrement.

Il est souhaitable de créer également un `.gitignore` à la racine du répertoire de vos TPs, OS_NomA_NomB, qui contiendra au moins une ligne: `*.o`

Ce qui est fourni

libroad

Pour ce TP on vous fourni un petit module (bibliothèque) permettant d'afficher une fenêtre dans laquelle est tracée une "route". Ce module permet également d'ajouter des "voitures" (des rectangles) et de faire avancer ces rectangles le long de la route. La description des fonctions disponibles ainsi qu'un programme principal minimal vous sont également fournis (road.h et onTheRoad.c, respectivement).

Vous trouverez également une documentation succincte sur les threads dans le document intitulé : "Unix : quelques fonctions système".

L'objectif du TP est de faire avancer plusieurs voitures dans des threads autonomes.

1. Mise en place - Makefile

Cette étape n'est **pas** triviale et fait partie des exercices du TP.

1.1. Compilation

Récupérez le programme principal `onTheRoad.c` dans le répertoire :

`/users/dut/info/Public/Systeme/onTheRoad/`

ou là.

Pour le faire compiler, créez un `Makefile` qui utilise judicieusement les options `-I`, `-L` et `-l` de gcc. Vous devrez en effet faire en sorte que le programme `onTheRoad.c` soit compilé et linké en utilisant le module road constitué des fichiers `road.h` et `libroad.so` qui se trouvent dans le répertoire `/users/dut/info/Public/Systeme/onTheRoad/`.

Pensez à utiliser des **variables** dans votre `Makefile` pour ne pas avoir à recopier les chemins d'accès et les options de compilation.

Rappel

> Pour inclure une bibliothèque dont le nom de fichier est libroad.so, il faut ajouter le flag `-lroad` à l'édition de lien, **après** le fichier à traiter.

Vous aurez également besoin d'ajouter les bibliothèques `liballegro` et `liballegro_primitives` à la commande d'édition de lien de votre Makefile pour produire l'exécutable.

1.2. Exécution

Pour l'exécution de `onTheRoad`, comme la bibliothèque `libroad.so` est dans un répertoire non standard (et devrait y rester), vous devrez ajouter à votre variable d'environnement `LD_LIBRARY_PATH` le répertoire où elle se trouve pour que bash (en fait ld-linux) soit en mesure de lancer votre programme. La documentation sur cette variable d'environnement se trouve dans le man de ld-linux.

N'hésitez pas à ajouter la commande nécessaire dans votre fichier `.bashrc` afin de ne pas avoir à l'exécuter dans chaque nouveau terminal. Vérifiez que la compilation se passe bien et que le programme s'exécute correctement : une voiture parcourt la route, le programme se termine lorsque vous pressez *esc*.

Si vous souhaitez travailler sur votre ordinateur personnel, pensez à récupérer les fichiers `road.h` et `libroad.so`. Ou à consulter la section [de-chez-vous] .

2. Création d'un thread

> Il est judicieux à ce stade de prendre connaissance du support de cours sur les threads.

Dans l'état actuel du programme, la voiture avance à chaque passage dans la boucle principale grâce à la fonction `road_stepCar`. Pour découpler l'avancée de la voiture de cette boucle principale, vous allez créer un thread pour faire avancer la voiture.

2.1. Avancer

Pour faire avancer la voiture dans un thread, commencez par créer une fonction "avancer" qui appelle la fonction `road_stepCar`. Comme cette fonction devra être lancée dans un thread, elle aura la signature imposée par `pthread_create`:

```
void* avancer(void* parametre);
```

Dans un premier temps, vous pouvez ne rien passer en paramètre et fixer en dur le paramètre de `road_stepCar`.

Validez que vous arrivez bien à faire avancer la voiture à partir d'un nouveau thread créé avec `pthread_create` qui se lance dans la fonction `avancer`.

Il faudra notamment s'assurer que `road_stepCar` est appelée plusieurs fois. Veillez également à utiliser judicieusement `usleep` pour pouvoir la voir passer.

> Pour pouvoir utiliser les threads, il vous faudra utiliser le flag *quivabien*® à l'édition de lien.

2.2. Terminer "avancer" proprement

Avant de réaliser cette étape, votre voiture devra traverser la fenêtre jusqu'à disparaître.

Vous allez maintenant faire en sorte que le thread qui fait avancer la voiture se termine proprement c'est-à-dire que la fonction `avancer` doit se terminer lorsque la voiture arrive au bout de la route. Pour cela vous aurez besoin de prendre en compte la valeur de retour de `road_stepCar` dont vous trouverez la description dans road.h.

3. Tester la fin d'un thread

Vous passerez à un **nouveau fichier** intitulé `onTheRoad2.c` pour cette étape, après avoir **poussé** votre *onTheRoad.c* courant.

On aimerait que le programme sorte lorsqu'il n'y a plus de voiture sur la route. Faites en sorte que la boucle principale teste si le thread qui fait avancer votre voiture a terminé, en plus du test sur la touche *esc*. Pour cela, vous aurez besoin de la fonction `pthread_tryjoin_np`.

Notez qu'un thread se termine lorsque la fonction qu'il exécute se termine.

4. Multiplication des voitures

Vous passerez à un **nouveau fichier** intitulé `onTheRoad3.c` pour cette étape, après avoir **poussé** votre *onTheRoad2.c* courant.

Le but ici est de multiplier les voitures sur la route. Pour cela, vous allez créer une fonction qui ajoute un certain nombre de voitures.

Cette fonction `creerVoitures` devra:

- créer des voitures;
- lancer un thread exécutant la fonction `avancer` pour chacune des voitures créés.

Comme le thread principal est occupé à mettre à jour l'affichage, il faudra que la fonction `creerVoitures` soit, elle aussi, exécutée dans un thread à part.

Remarque: Si ce n'était pas le cas, vous devrez passer en paramètre un **identifiant de voiture** aux "sous-threads" qui exécutent `avancer`. Faute de quoi, tous vos threads feront avancer la même voiture...

Notez que si vous lancez deux voitures simultanément, leur affichage sera superposé et une "collision" sera détectée et affichée en rouge.

5. Questions "maîtrise"

Dans un nouveau fichier, `onTheRoadMaitrise.c`, vous traiterez les points suivants:

- Pour obtenir des résultats visuellement intéressants, faites varier l'intervalle de lancement des voitures de manière aléatoire ainsi que le sens de circulation à l'aide de la fonction `rand`.
- Offrez à l'utilisateur la possibilité de créer une voiture lorsqu'il appuie sur "V"
- et de faire une pause lorsqu'il appuie sur la "P".

Vous aurez besoin des fonctions:

- `al_get_keyboard_state` et
- `al_key_down`

ainsi que de la liste des codes de touche

Le `#include` qui va bien :

```
#include <allegro5/allegro.h>
```

Rerappel

> Ce qui n'est pas poussé sur votre dépôt GitLab n'est pas corrigé.

6. De chez vous

Si vous souhaitez avancer sur ce TP depuis chez vous, en plus d'installer une version compatible de la bibliothèque Allegro5, il vous faudra soit :

- Récupérer les fichiers `road.h` et `libroad.so`. Ceci ne fonctionnera que sous une version récente de linux.
- Compiler la bibliothèque `libroad` à partir des sources que vous trouverez dans ce dépôt.