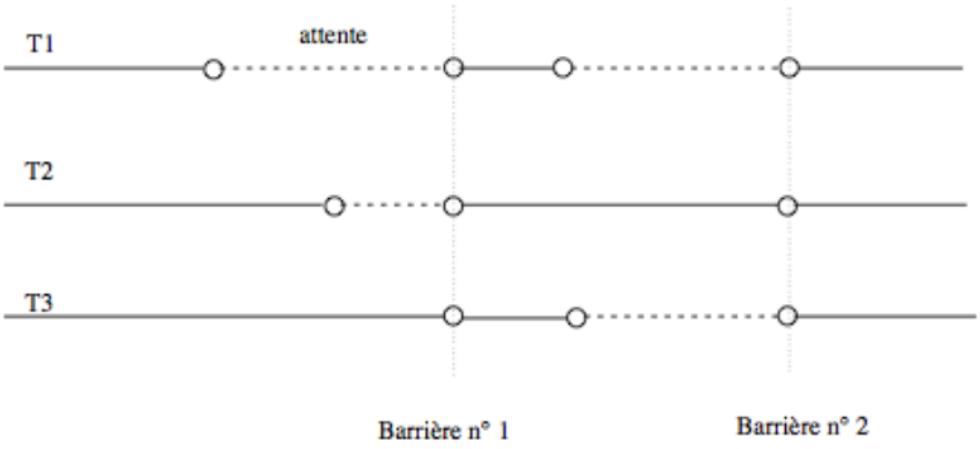
Exercice 1 – Les coureurs

Implémenter en C un système dans lequel plusieurs threads doivent attendre un même instant avant de poursuivre leur travail. Cet instant peut se représenter par une barrière implémentée à l'aide de sémaphores. La figure ci-dessous illustre le cas pour 3 threads et 2 barrières.



1 Compilation

Pour compiler ce projet, j'ai utilisé `cmake`. Le fichier *CMakeLists.txt* permet de générer le *Makefile* avec les options suivantes :

```
[1]  cmake_minimum_required(VERSION 3.10)

[2]  project(Exercice1_Coureurs VERSION 1.0)

[3]  set(CMAKE_C_FLAGS "${CMAKE_C_FLAGS} -Wall -Wpedantic -Wextra")

[4]  include_directories(PUBLIC include)

[5]  set(SOURCE_FILES
         src/main.c
         src/barrier.c)

[6]  find_package(Threads REQUIRED)

[7]  add_executable(race ${SOURCE_FILES})

[8]  target_link_libraries(race PRIVATE Threads::Threads)
```

`[1]` version de `cmake` utilisée

`[2]` nom et version du projet

`[3]` ajout des flags habituels pour l'affichage de tous les warnings

`[4]` le répertoire *include* contient les headers nécessaires à la compilation

`[5]` définition de la variable `SOURCE_FILES` contenant les fichiers sources du projet

`[6]` ajout du package pour l'utilisation des threads

`[7]` la compilation des fichiers sources produit l'exécutable `race`

`[8]` le projet utilise la librairie `Threads` du package `Threads`

Le fichier *CMakeLists.txt* se trouve à la racine de mon projet. Je lance la commande `cmake ..` dans le sous-répertoire *build* pour que les fichiers générés y soient placés :

```
lbin@tr-ubuntu18-server:~/2021_TR/ex1-coureurs/build$ cmake ..
-- The C compiler identification is GNU 7.5.0
-- The CXX compiler identification is GNU 7.5.0
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Looking for pthread.h
-- Looking for pthread.h - found
-- Looking for pthread_create
-- Looking for pthread_create - not found
-- Looking for pthread_create in pthreads
-- Looking for pthread_create in pthreads - not found
-- Looking for pthread_create in pthread
-- Looking for pthread_create in pthread - found
-- Found Threads: TRUE
-- Configuring done
-- Generating done
-- Build files have been written to: /home/lbin/2021_TR/ex1-coureurs/build
lbin@tr-ubuntu18-server:~/2021_TR/ex1-coureurs/build$ ls
CMakeCache.txt  CMakeFiles  cmake_install.cmake  Makefile
```

Par défaut, le *Makefile* est généré en mode debug. Pour le générer en mode release, j'utilise la commande `cmake -DCMAKE_BUILD_TYPE=Release` ***`path`***, où ***`path`*** est l'endroit où se trouve le fichier *CMakeLists.txt*.

Je peux maintenant lancer la compilation à partir du *Makefile* avec la commande `make` qui crée l'exécutable *race* dans le répertoire courant :

```
lbin@tr-ubuntu18-server:~/2021_TR/ex1-coureurs/build$ make
Scanning dependencies of target race
[ 33%] Building C object CMakeFiles/race.dir/src/main.c.o
[ 66%] Building C object CMakeFiles/race.dir/src/barrier.c.o
[100%] Linking C executable race
[100%] Built target race
lbin@tr-ubuntu18-server:~/2021_TR/ex1-coureurs/build$ ls
CMakeCache.txt  CMakeFiles  cmake_install.cmake  Makefile  race
```

2 Tests

Le programme attend en paramètres le nombre de coureurs participant à la course et le nombre d'étapes la composant :

```
lbin@tr-ubuntu18-server:~/2021_TR/ex1-coureurs/build$ ./race
usage: ./race runners stages
```

Les coureurs partent dans l'ordre mais arrivent aux étapes et à l'arrivée dans un ordre aléatoire :