Compte rendu TP POSIX

Nechi Aziz & Ghaith Nabli

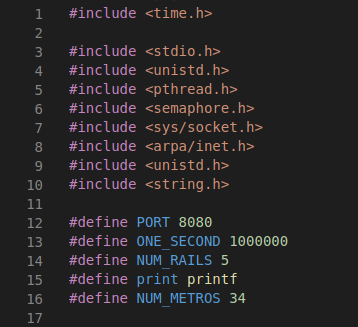
Tout le code, ainsi qu’une petite démo, de notre travail est disponible sur ce lien github: *https://github.com/yNeshy/posixTP*

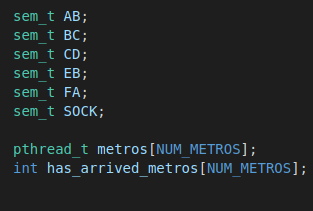
Question 1:

* Chaque métro sera modélisé par un thread.
* Chaque ligne de métro aura un identifiant: AB pour le segment reliant les stations A et B.
* Chaque ligne de métro sera protégée par une sémaphore.

code des Threads en C :

#### Déclaration des variables:



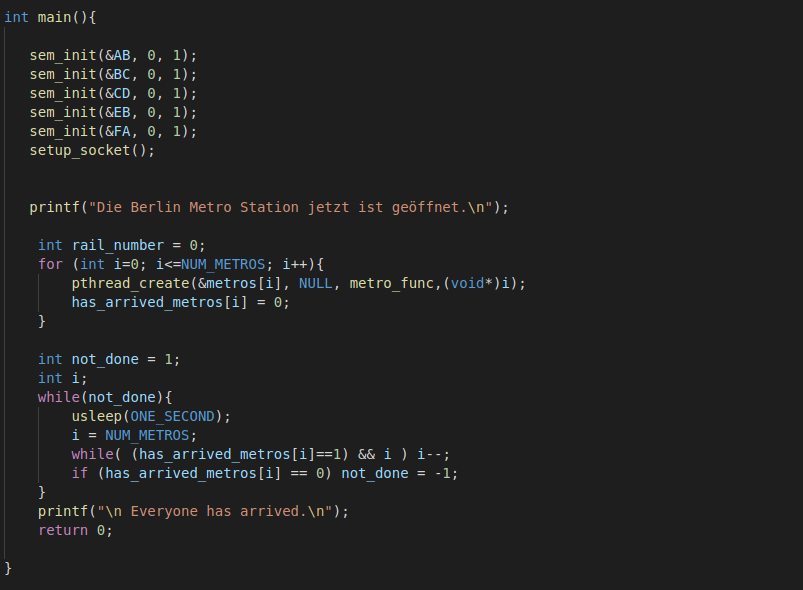


#### 

En C, comme illustré dans le Tutoriel Posix, les threads sont supportés nativement par l’OS.

Dans notre cas Ubuntu, crée un *worker* pour chaque thread, indépendant programme principale le lançant.

#### Initialisations :



Les metros possèdent deux attribut:

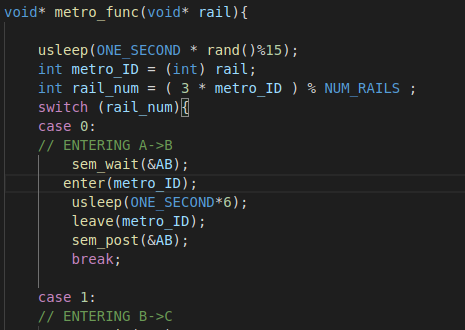
* Un ID: Identifiant unique, par le biais duquel il sera identifié à l’affichage
* Un rail de métro, qui, dans le but de simplifier les fonctions et limiter l’interdépendance, est déductible à partir de l’ID: rail\_num = ( 3 \* metro\_ID ) % NUM\_RAILS

#### Fonction de chaque metro:

Chaque thread (métro) est lancé avec pour seul argument son ID.

On en déduit le segment de rail et y lance le métro.

Notez bien qu' avant d’entrer, chaque métro demande l'accès, en demandant à acquérir la sémaphore du rail.



enter et leave feront l’objet de la simulation (question 2).

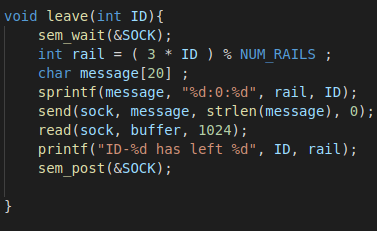
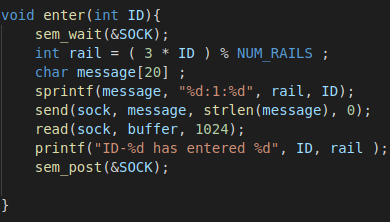
Question 2:

#### Principe de la simulation:

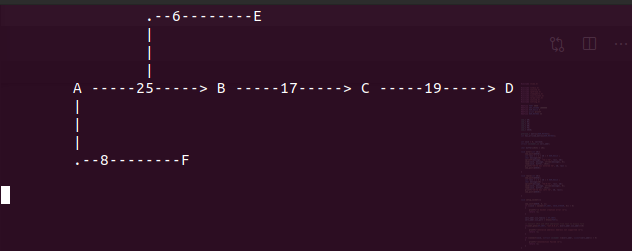
Nous avons fait la simulation de l’activité du métro en Command Line Interface:

Le programme principale implémente une WebSocket, connectée à un serveur WebSocketServer implémenté en Python.

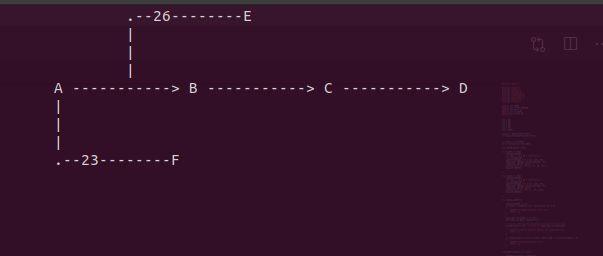
La Socket est protégée elle aussi par une sémaphore.

Chaque thread peut demander l’accès à la socket pour envoyer au serveur des informations quant à son activité (entrer ou sortie).

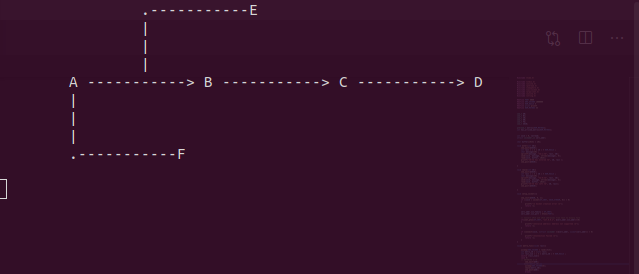
#### Aperçu :



Chaque metro est identifié par son ID.



Les threads C ne se chevauchent jamais.



État final: les threads ont tous été exécutés.