Université de Vincennes Saint-Denis

PROJET TRAIN

PROGRAMMATOIN TEMPS RÉEL

Réalisé par :

-AOUDJIT Athmane.

-OUALEB Achour.

Enseignant : Aline HUFSCHMITT.

Le 08 Janvier 2018

Table des matières

Remarque : A noter que nous avons fait ce travail ensemble.

Comme nous avons collaboré tous les deux sur chacune des parties de ce projet, il nous est difficile de préciser « qui a fait quoi ».

Introduction

Dans ce projet (l'ensemble des 4 Parties) nos threads représentent les Trains 1,2 et 3. Nous avons utilisé des tableaux de caractères -de type char- pour l'ensemble des voies (ainsi que les gares : une voie A-B est composé de la gare A et B). Ces tableaux sont à une dimension, de ce fait leurs données sont les gares (pour afficher une voie il suffit on utilise un indice i et i+1).

Dans ce projet (sur l'ensemble des Parties) nous avons cinq fonctions : trois fonctions pour les trains, et deux pour les sens. Les premières seront appelées par les threads (trains), tel que en créant un thread, ce dernier appellera une fonction. Ces fonctions vont à leur tour (chacune) appeler les deux fonctions sens() et sensInverse(). Quand les fonctions sens() et sensInverse() sont implémenté par les trains, nous allons gérer la synchronisation de ces derniers par les différents outils demandés.

I. 1ère Partie -Mutex -

Dans cette première partie nous avons choisis le modèle lecteurs/écrivains, mais nous l'avons réadapté à notre situation deux groupes de lecteurs qui partagent (accèdent) à la même ressource. Deux groupes de lecteurs tels que, un représente les trains circulant dans un sens et l'autre les trains qui viendront dans le sens opposé.

Comme dans notre cas nous avons qu'une seule voie entre deux gares, donc les trains allant dans sens -premier groupe de lecteurs- et les trais venant dans le sens inverse -deuxième groupe de lecteurs- doivent partager une même voie (qui représente la ressource).

Comme expliqué ci-dessus nos threads représente les threads à chacun une fonction, qui appelle les fonctions des deux sens avec comme paramètre l'Id du thread (train). Cet Id va nous servir à afficher les messages indiquant qu'un train (1,2 ou 3) veut s'engager (est, sort) d'une voie, en utilisant des conditions.

Nous utilisons trois Mutex : fifo, verrou_sens et verrou_sensInverse, tel que le premier nous sert de file pour gérer les threads (premier arrivé premier à sortir), verrou_sens et verrou_sensInverse qui protégerons la ressource (voie) afin qu'elle soit accessible que par un groupe (train allant dans un sens où sens inverse) à la fois.

Comme dans le modèle lecteurs/écrivains lorsqu'un train voudrais accéder à une voie, nous avons une section critique pour la file des trains avec le mutex_fifo et une section critique avec le mutex_sens (si les trains vont dans sens, ou sensInverse si il vont dans sens Inverse) . Seulement le premier train bloquera les trains venant dans Sens Inverse avec pthread_mutex_lock(sensInverse), ensuite ce train pourra circuler dans sens et en sortant et dans une section critique « sens » il libérera la voie en débloquant pthread_mutex_unlock(sensInverse). De la même manière pour les trains circulant dans l'autre sens ils peuvent bloquer les trains circulant sur sens dans la section critique mutex_fifo (file) et mutex_sensInverse avec pthread_mutex_lock (sens) et les débloqué une fois sorti avec pthread_mutex_unlock(sens).

II. 2ème Partie -Sémaphores –

Dans cette partie comme son nom le montre nous avons utilisé les sémaphores. Trois sémaphores pour les trois trains semtrain1, semtrain2 et semtrain3.

Le principe est de bloquer un thread (train) avec sem_wait(le sémaphore du thread concerné) jusqu'à ce qu'un autre thread lui envoie sem_post(le sémaphore du thread bloqué).

Pour ce faire nous avons déclaré deux tableaux à deux dimensions (Matrice) avec 3 lignes et 2 colonnes comme variable globale. Ces deux tableaux compar1 et compar2 représente nos deux sens (sens et sensInverse) et chaque ligne est réservée à un train.

Comme dit ci-dessus chaque train appel sa fonction et chacune (Train1(), Train2() et Train3()), appellera les deux fonctions des sens() et sensInverse(). Et lorsqu'un train veut s'engager sur une voie nous allons affecter à sa ligne le nom de cette voie (pour les deux sens), et avant qu'il s'engage nous allons vérifier (nous utilisons les conditions) si un autre train n'est pas déjà sur la même voie mais dans l'autre sens. Et cela en vérifions les lignes de l'autre tableau, exemple :

Dans la fonction sens :

Le Train 1 veut s'engager sur la voie AB.

compar1 [1][0]=A; et compar1 [1][1]=B;

Au même temps sur sensInverse

Le Train 2 est sur la voie BA.

compar1 [2][0]=A; et compar1 [2][1]=B;

Donc avant que le train un ne puisse s'engager sur AB, nous allons comparer les colonnes la ligne 1 (réservée au train 1) du tableau compar1 (représente sens) avec les colonnes des autres lignes (2, 3) du Tableau compar2 (représente sensInverse). En cas d'égalité nous bloquons ce Train 1 avec sem_wait(semtrain1) –sémaphore du Train 1- jusqu'à ce que le Train 2 qui est sur BA, sort de cette voie et à ce moment il envoie sem_post(semtrain1) pour débloquer le train 1.

De la même manière pour tous les trains et pour les deux sens, sachant que dès qu'un train sorte d'une voie on réinitialise les cases du tableau compar1 ou compar2 (selon le sens sur lequel il est) qui lui sont allouées.

III. 3ème Partie -Verrous –

Comme dans la première partie nous avons utilisé le modèle Lecteurs/Ecrivains, réadapté à notre situation, lorsque les trains roulent dans sens ils représentent les Lecteurs et quand ils vont dans le sens Inverse ils représenteront les Ecrivains. Et comme les précédentes parties nous avons gardé la même structure de données pour les ressources (voies) et trois threads qui représentent nos trains. Ainsi que trois fonctions pour ces trains qui implémenteront à leurs tours les fonctions des deux sens.

Dans cette partie nous utilisons les Verrous (lock), donc pour qu'un train x circulant dans Sens veut entrer sur une voie il demande le verrou de lecture et bloque le verrou avec pthread_rwlock_rdlock(&lock), donc si un autre train voudrais circuler sur cette voie dans le sensInverse il sera en attente du verrou, jusqu'à ce que le premier train (circulant dans Sens) sorte de sa voie. En relâchant pthread_rwlock_unlock(&lock); pour débloquer le thread du sens opposé.

De la même façon un train sur une voie de sensInverse acquière garde le verrou (ce qui laissent le train de Sens en attente) avec pthread_rwlock_wrlock(&lock), jusqu'à ce qu'il quitte cette voie. Il libérera le verrou avec pthread_rwlock_unlock(&lock).

IV. 4ème Partie -Files de Messages –

Nous avons préféré ne pas envoyer les fichiers de cette partie vu que nous n'avons pas fini notre travail. Et nous n'arrivons pas à gérer la synchronisation des trains ni gérer les sens.

V. Les bancs de test & le temps moyens par trajet

Les bancs de test se font d'une façon automatique. Lorsqu'un train implémente sa fonction, dans cette dernière nous appelons les deux fonctions sens et sensInverse 30 fois en utilisant une boucle for.

Pour le calcul du temps moyen que mette un train pour parcourir un trajet, nous calculons la moyenne du temps de tous les trajets. Pour ce faire nous utilisons deux tableaux pour chaque trains (six tableau en tout), tels que sur le premier nous enregistrons les temps de départ (pour chaque trajet) avec la fonction clock(), sur le deuxième nous enregistrons le temps de chaque trajet. Nous calculons le temps de chaque trajet en soustrayant du le temps à la fin d'un trajet le temps de départ avec l'équation

t2a[i]=clock()-t2d[i]

tel que tempsDeTrajet[] = clock() – tempsDeDepart [].

Ensuite nous sommons les temps de trajet que nous divisons par le nombre de trajet.

Pour afficher les messages indiquant l'état des trains sur un fichier et ne laisser que les messages affichant le temps des trajets sur la console. Nous avons utilisé :

Pour les messages de temps de trajet : fprintf(stderr,"le message temps moyen");

Pour les messages indiquant l'état des trains : Printf("état du train ")

Et lors de l'exécution nous utilisons : ./a.out 1> file.txt , pour les afficher sur un fichier.

Comparaison du temps moyen de trajet selon la méthode :

Méthode Mutex:

Le temps moyen de trajet du train 3 : 1723.533325

Le temps total de parcours du train 3 : 51706.000000

Le temps moyen de trajet du train 2 : 1815.633301

Le temps total des parcours du train 2 :54469.000000

Le temps moyen de trajet du train 1 : 1826.766724

le temps total de parcours du train 1: 54803.000000

Méthode Sémaphores :

Le temps moyen de trajet du train 3 : 2071.966553

Le temps total de parcours du train 3 : 62159.000000

Le temps moyen de trajet du train 1 : 2327.100098

Le temps total de parcours du train 1 : 69813.000000

Le temps moyen de trajet du train 2 : 2525.000000

Le temps total de parcours du train 2 : 75750.000000

Méthode Verrous :

le temps moyen de parcours du train 1 : 1699.666626

le temps totaux de parcours du train 1: 50990.000000

le temps moyen de parcours du train 3 : 1751.333374

le temps total de parcours du train 3 : 52540.000000

le temps moyen de parcours du train 2 : 1776.900024

le temps total des parcours du train 2 :53307.000000

D'après ces résultats, et sans tenir compte de la méthode utilisée, nous remarquerons que le temps moyen de trajet pour les trois trains est entre ~1700 et 2500. Nous pouvons voir aussi qu'à chaque fois qui est le train qui finit le premier (varie d'une exécution à l'autre).

Dans cet exemple, nous remarquons que le train 3 fini premier dans la méthode Mutex et Sémaphores, avec le plus temps de trajet de : 1723.533325 (Mutex) et 2071.966553 (Sémaphore), et fini 2eme dans la méthode des Verrous avec 1751.333374 s.

Le train 1 fini premier avec 1699.666626 (Verrous) deuxième 2327.100098 (Sémapores) et troisième1826.766724 (Mutex).

Le train 2 fini deuxième avec 1815.633301 (Mutex) troisième 2525.000000 (Sémapores) et troisième1776.900024 (Verrous).

VI. Problèmes rencontrés :

Durant la réalisation de notre projet, nous avons rencontré quelques difficultés, d'abord nous n'avons mis du temps à bien comprendre la problématique, ensuite pour choisir les structures de données adapté et ce que ces dernières vont représenter (gare, voie, sens…).

Nous avons aussi trouvé des difficultés à gérer les sens, sans pour cela empêcher les trains allant sur une même voie d'y accéder à la fois.

En fin la quatrième partie réalisation avec les files de messages, nous avons essayé de résoudre le problème avec mais nous n'arrivant pas gérer nos trains ni les sens, bien que nous avons compris le principe mais faute de pratique avec cette méthode et faute de temps mis sur les premières hélas nous n'avons pas réussis à la faire.

Conclusion

Maintenant que nous avons réussis plus au moins à réaliser la majorité du travail qui nous est demandé. Bien qu'il nous manque deux ou trois choses à réaliser, faute de temps et le fait d'en avoir beaucoup mis afin de mettre en œuvre les autres parties. Et malgré les difficultés rencontrées depuis le début de ce travail, nous pensons que ce travail nous a énormément aidés, tant sur le plan personnel et en apports en connaissances.

D'un autre côté, ce projet nous a permet d'abord sur un plan personnel de nous dépasser de ne pas s'arrêter à la première difficulté rencontrée, et connaitre la nécessité du travail en groupe et d'être polyvalent. Et ensuite ce travail nous a permet d'être autonome en cherchant de la documentation de suivre des tutoriels afin de trouver des solutions ou de comprendre une notion quelconque.

D'un autre côté, ce travail nous beaucoup apporté comme connaissance. Ce projet représente une sorte de rattrapage, et nous a offert une seconde chance afin de comprendre ce qui nous a échappé lors des séances cours et de mieux maitriser les notions déjà acquises.

Grâce à ce projet, nous avons monté en compétence dans la programmation C d'une manière générale, et plus particulièrement la programmation Temps Réel, la synchronisation des threads avec les différentes méthodes : Mutex, Sémaphores, les Verrous et les Files de Messages. Ainsi que les différents modèles (Lecteurs/Ecrivains en particulier).