ECOLE CENTRALE DE NANTES

SEC2 / CPP - Le TP sur les Threads

Janvier 2023

L'objectif de ce TP est de travailler sur la concurrence en C++ par l'utilisation des `threads` et des outils de synchronisation de ces processus.
Le travail comporte deux parties (indépendantes).
Des "squelettes de programme" sont fournis pour chaque partie.

1 — LES LECTEURS-ÉCRIVAINS

Le problème des "Lecteurs-Écrivains" a été vu en cours avec une solution de leur synchronisation basée sur les `Sémaphores`. L'objectif de cette partie est d'en faire une implémentation en C++.
Le code fourni est tel que les accès (en lecture ou en écriture) à une ressource partagée ne sont pas synchronisés de telle façon que des lectures de la valeur de cette ressource peuvent être incohérentes.

Q 1.1 Modifier le code des processus `lecteur` et `ecrivain` pour que la synchronisation permette de respecter les règles de cohérence de la valeur de la ressource partagée.

On pourra réaliser ces synchronisations en utilisant un ou des objets de la classe `Semaphore` appelé dans le `main` (le code est fourni ci-dessous).

UN EXEMPLE D'EXÉCUTION DU PROGRAMME DES LECTEURS-ECRIVAINS SANS RESPECTER LES RÈGLES DE COHÉRENCE DE LA RESSOURCE PARTAGÉE.

```
. . .
Ecrivain n° 5 en cours
          Valeur lue = 3
valeur à incrémenter de la ressourcePartagee = 1
Ecrivain n° 2 en cours
valeur à incrémenter de la ressourcePartagee = 2
Ecrivain n° 7 en cours
          Valeur lue = 3
Lecteur n° -2 en cours
valeur à incrémenter de la ressourcePartagee = 2
Ecrivain n° 6 en cours
          Valeur lue = 3
Lecteur n° -3 en cours
Valeur lue = 3
          Valeur lue = 3
valeur à incrémenter de la ressourcePartagee = 2
Ecrivain n° 1 en cours
          Valeur lue = 3
Lecteur n° -4 en cours
valeur à incrémenter de la ressourcePartagee = 3
valeur à incrémenter de la ressourcePartagee = 2
Ecrivain n° 3 en cours
valeur à incrémenter de la ressourcePartagee = 3
valeur à incrémenter de la ressourcePartagee = 3
          Valeur lue = 4                          <<-
valeur à incrémenter de la ressourcePartagee = 3   <<--
valeur à incrémenter de la ressourcePartagee = 2   <<--- valeur à incrémenter inférieure à la valeur à
incrémenter considérée par un précédent écrivain (3)  et même à la valeur lue par un précédent lecteur (4)
Ecrivain n° 2 en cours
valeur à incrémenter de la ressourcePartagee = 3
valeur à incrémenter de la ressourcePartagee = 3
          Valeur lue = 4
          Valeur lue = 4
Lecteur n° -4 en cours
          Valeur lue = 4
valeur à incrémenter de la ressourcePartagee = 3
. . .
```

LE CODE DE SEMAPHORE

```
#include <iostream>
#include <mutex>
#include <condition_variable>

using namespace std;

class Semaphore {
  public:
    Semaphore (int valeur_initiale)
    : val(valeur_initiale)
    {
    }

    inline void V ( int tid ) {
      std::unique_lock<std::mutex> lock(mtx);
      val++;
      cout << "processus " << tid <<  " V" << endl;
      //déblocage d'un processus bloqué (si il en existe)
      cv.notify_one();
    }

    inline void P ( int tid ) {
      std::unique_lock<std::mutex> lock(mtx);
      while(val == 0) {
        cout << "processus " << tid << " P" << endl;
        //attente jusqu'à invocation d'un V() -> blocage du processus appelant
        cv.wait(lock);
        cout << "processus " << tid << " reprend" << endl;
      }
      val--;
    }
  private:
    std::mutex mtx;
    std::condition_variable cv;
    int val=0;
};
```

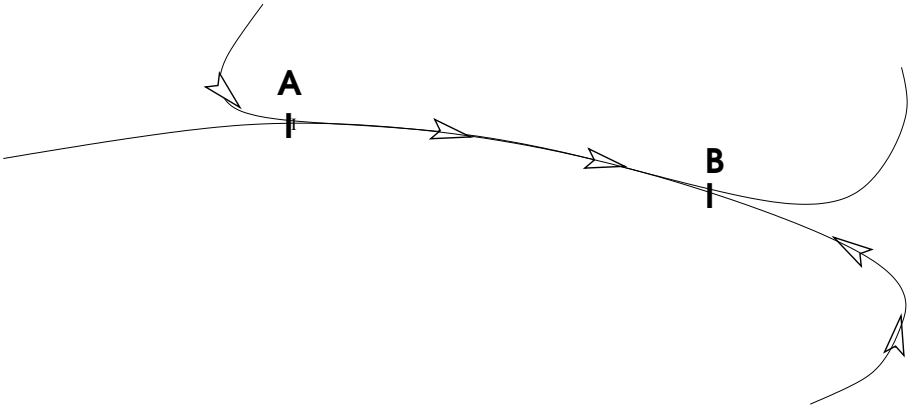
LE SQUELETTE DU MAIN POUR LES LECTEURS-ÉCRIVAINS

```
#include <iostream>
#include <thread>
#include "Semaphore.hpp"

int ressourcePartagee;

void lecteur(int numLecteur){
  for (int i = 0; i < 4; i++){
    std::cout << "Lecteur n° " << numLecteur << " en cours " << endl;
    this_thread::sleep_for(chrono::milliseconds(rand() % 20000) );
    std::cout << "        Valeur lue = " << ressourcePartagee << "  " << endl;
  }
}

void ecrivain(int numEcrivain){
  int x;
  for (int i = 0; i < 4; i++){
    std::cout << "Ecrivain n° " << numEcrivain << " en cours " << endl;
    x = ressourcePartagee;
    this_thread::sleep_for(chrono::milliseconds(rand() % 20000) );
    std::cout << "valeur à incrémenter de la ressourcePartagee = " << x << "  " << endl;
    ressourcePartagee = x+1 ;
  }
}

int main() {
  const int nbr = 8;
  std::thread r[nbr];
  std::thread w[nbr];
  int ressourcePartagee = 0;

  for (int i = 1; i < nbr; i++){
    r[i] = std::thread(lecteur, -i);
    w[i] = std::thread(ecrivain, i);
  }
// Join des threads
  for (int i = 1; i < nbr; i++) {
  r[i].join();
  w[i].join();
  }
  return 0;
}
```

2 — LES TRAINS

Pour cette deuxième partie du TP, la programmation concurrente porte sur un problème de synchronisation de trains qui doivent pouvoir circuler dans les deux sens (en coordination…) sur un tronçon de voie unique.

D'un point de vue pratique, l'objectif sera de vérifier qu'il n'y a pas de *mauvais fonctionnement*[1] de vos programmes au travers de plusieurs jeux d'essais.



NB. En fin de ce sujet de TP, sont fournis des morceaux de code C++ qui pourront être utilisés pour répondre aux questions, ainsi que des exemples de résultats d'exécution conformes à ce qui est attendu (au moins pour les questions 1 et 2).

2.1 PREMIÈRE SYNCHRONISATION SIMPLE

Synchroniser les processus trains pour qu'il n'y ait pas d'accidents (mais que des trains puissent circuler simultanément sur le tronçon partagé lorsqu'ils vont dans le même sens).

Q 2.1 Compléter le programme fourni du processus `Controleur` de façon à ce que le programme fonctionne correctement par rapport à cette première contrainte.

1. le résultat de certaines exécution indique que deux trains sont en circulation simultanément sur le tronçon de voie unique ... en sens inverse.

Le risque de coalition

Le problème de la précédente mise en œuvre est que les trains qui passent dans un sens peuvent faire coalition *et empêcher longtemps (voire indéfiniment...) le passage des trains dans l'autre sens.*

2.2 Deuxième synchronisation

Pour remédier à ce problème du risque de coalition, mettre en oeuvre des synchronisations de façon à ce que *pas plus de deux trains puissent circuler simultanément dans un sens.* Ainsi, lorsque deux trains seront passés dans un sens, les nouvelles demandes de passage seront ré-examinées pour autorisation (dans un sens ou dans l'autre).

Q 2.2 Modifier le programme précédent du processus `Controleur` de façon à ce que le programme fonctionne correctement par rapport à la première contrainte **et** en évitant le risque de coalition.

Cette solution n'est pas équitable

À noter qu'ainsi, il est possible que (par malchance) beaucoup de trains puissent passer dans un sens avant que des trains qui veulent circuler dans l'autre sens y parviennent....

2.3 Troisième niveau de contrainte de synchronisation

Toujours pour *remédier au problème du risque de coalition* (cf. paragraphe en fin de section 2.2), mais *de façon plus équitable* pour les trains circulant dans chaque sens, on veut maintenant imposer qu'il n'y ait *pas plus de 3 trains consécutifs dans un sens* (mais ceci seulement dans la situation où des trains attendent pour passer dans l'autre sens).

Q 2.3 Modifier le programme précédent du processus `Controleur` de façon à ce que le programme fonctionne correctement par rapport à la première contrainte et en évitant le risque de coalition **et** en étant équitable.

Le processus Controleur ... à compléter

```
#include <thread>
#include <iostream>
using namespace std;

class Controleur {
  public:
    Controleur (int valeur_initiale) :
    val(valeur_initiale)
    {
    }

    bool controlinEnB(int numero) {
        return true;
    }

    bool controlinEnA(int numero) {
        return true;
    }

    bool controloutEnB(int numero){
      return true ;
    }

    bool controloutEnA(int numero){
      return true;
    }
  private:
    int val;
};
```

LE SQUELETTE DU MAIN POUR LES TRAINS

```
#include <iostream>
#include <thread>
#include <mutex>
#include <condition_variable>
#include "Controleur.hpp"

std::mutex mtx;
std::condition_variable cv;
Controleur ctrl(0);   // LE PROCESSUS CONTROLEUR

// TrainAB
void circuleAB(int numTrain){
  std::cout << "Train n° " << numTrain << " arrive en A vers B " << endl;
  this_thread::sleep_for(chrono::milliseconds(rand() % 1000) );
  std::unique_lock<std::mutex> lck(mtx);
    cv.wait(lck,[numTrain]{return ctrl.controlinEnA(numTrain);}); // ATTENTE D'AUTORISATION DE CIRCULER
    lck.unlock();
  // DEBUT DU PARCOURS A->B
  std::cout << "Train n° "<< numTrain << " circule de A vers B  >>>>>> " << endl;
  this_thread::sleep_for(chrono::milliseconds(rand() % 100) );

  // FIN DU PARCOURS A->B
  std::cout << "Train n° " << numTrain << " quitte le tronçon de voie unique " << endl;
    lck.lock();
  ctrl.controloutEnB(numTrain);  // SIGNAL DE SORTIE AU CONTROLEUR
    lck.unlock();
  cv.notify_all();
}

// TrainBA
void circuleBA(int numTrain){
  std::cout << "Train n° " << numTrain << " arrive en B vers A " << endl;
  this_thread::sleep_for(chrono::milliseconds(rand() % 1000) );
  std::unique_lock<std::mutex> lck(mtx);
  cv.wait(lck,[numTrain]{return ctrl.controlinEnB(numTrain);}); // ATTENTE D'AUTORISATION DE CIRCULER
  lck.unlock();
  // DEBUT DU PARCOURSs B->A
  std::cout << "Train n° "<< numTrain << " circule de B vers A  <<<<<<<<" << endl;
  this_thread::sleep_for(chrono::milliseconds(rand() % 100) );

  // FIN DU PARCOURS B->A
  std::cout << "Train n° " << numTrain << " quitte le tronçon de voie unique " << endl;
   lck.lock();
  ctrl.controloutEnA(numTrain);   // SIGNAL DE SORTIE AU CONTROLEUR
   lck.unlock();
  cv.notify_all();
}

int main(){
  const int nbr = 9;
  std::srand(std::time(nullptr));
  std::thread trainsAB[nbr];
  std::thread trainsBA[nbr];

  // INITIALISE TRAINS ET CONTROLEUR
  for (int i = 1; i < nbr; i++){
    trainsAB[i] = std::thread(circuleAB, i);
    trainsBA[i] = std::thread(circuleBA, -i);
  }
// JOIN DES THREADS
  for (int i = 1; i < nbr; i++) {
    if (trainsAB[i].joinable()) trainsAB[i].join();
    if (trainsBA[i].joinable()) trainsBA[i].join();
  }
  return 0;
}
```

DEUX RÉSULTATS D'EXÉCUTION

On donne maintenant deux exemples d'exécution de programmes correspondant à la résolution des contraintes de synchronisation de la question 2.

```
> ./main
Train n° 1 arrive en A vers B
Train n° -3 arrive en B vers A
Train n° 4 arrive en A vers B
Train n° -4 arrive en B vers A
Train n° 6 arrive en A vers B
Train n° 2 arrive en A vers B
Train n° 3 arrive en A vers B
Train n° -2 arrive en B vers A
Train n° 5 arrive en A vers B
Train n° -5 arrive en B vers A
Train n° -6 arrive en B vers A
Train n° 7 arrive en A vers B
Train n° -7 arrive en B vers A
Train n° -1 arrive en B vers A
Train n° 8 arrive en A vers B
Train n° -8 arrive en B vers A
Train n° 1 circule de A vers B  >>>>>>
Train n° 6 circule de A vers B  >>>>>>
Train n° 1 quitte le tronçon de voie unique
Train n° 7 circule de A vers B  >>>>>>
Train n° 6 quitte le tronçon de voie unique
Train n° 7 quitte le tronçon de voie unique
Train n° -7 circule de B vers A  <<<<<<<<
Train n° -7 quitte le tronçon de voie unique
Train n° 2 circule de A vers B  >>>>>>
Train n° 5 circule de A vers B  >>>>>>
Train n° 2 quitte le tronçon de voie unique
Train n° 5 quitte le tronçon de voie unique
Train n° 8 circule de A vers B  >>>>>>
Train n° 8 quitte le tronçon de voie unique
Train n° -4 circule de B vers A  <<<<<<<<
Train n° -4 quitte le tronçon de voie unique
Train n° -3 circule de B vers A  <<<<<<<<
Train n° -6 circule de B vers A  <<<<<<<<
Train n° -3 quitte le tronçon de voie unique
Train n° -6 quitte le tronçon de voie unique
Train n° 3 circule de A vers B  >>>>>>
Train n° 3 quitte le tronçon de voie unique
Train n° -8 circule de B vers A  <<<<<<<<
Train n° -8 quitte le tronçon de voie unique
Train n° -5 circule de B vers A  <<<<<<<<
Train n° -2 circule de B vers A  <<<<<<<<
Train n° -2 quitte le tronçon de voie unique
Train n° -5 quitte le tronçon de voie unique
Train n° 4 circule de A vers B  >>>>>>
Train n° 4 quitte le tronçon de voie unique
Train n° -1 circule de B vers A  <<<<<<<<
Train n° -1 quitte le tronçon de voie unique
>
```

```
> ./main
Train n° 1 arrive en A vers B Train n° 2 arrive en A vers B
Train n° -3 arrive en B vers A
Train n° -4 arrive en B vers A
Train n° -1 arrive en B vers A Train n°
Train n° 4 arrive en A vers B
Train n° 6 arrive en A vers B
Train n° Train n° 3 arrive en A vers B
5 arrive en A vers B
-2 arrive en B vers A
Train n° -5 arrive en B vers A
Train n° Train n° 7 arrive en A vers B
Train n° Train n° -7 arrive en B vers A

-6 arrive en B vers A
Train n° -8 arrive en B vers A
8 arrive en A vers B
Train n° -3 circule de B vers A  <<<<<<<<
Train n° -4 circule de B vers A  <<<<<<<<
Train n° -3 quitte le tronçon de voie unique
Train n° -4 quitte le tronçon de voie unique
Train n° -1 circule de B vers A  <<<<<<<<
Train n° -1 quitte le tronçon de voie unique
Train n° 1 circule de A vers B  >>>>>>
Train n° 5 circule de A vers B  >>>>>>
Train n° 1 quitte le tronçon de voie unique Train n°
5 quitte le tronçon de voie unique
Train n° -5 circule de B vers A  <<<<<<<<
Train n° -7 circule de B vers A  <<<<<<<<
Train n° -5 quitte le tronçon de voie unique
Train n° -7 quitte le tronçon de voie unique
Train n° -6 circule de B vers A  <<<<<<<<
Train n° -8 circule de B vers A  <<<<<<<<
Train n° -6 quitte le tronçon de voie unique
Train n° -8 quitte le tronçon de voie unique
Train n° 3 circule de A vers B  >>>>>>
Train n° 4 circule de A vers B  >>>>>>
Train n° 3 quitte le tronçon de voie unique
Train n° 4 quitte le tronçon de voie unique
Train n° 7 circule de A vers B  >>>>>>
Train n° 8 circule de A vers B  >>>>>>
Train n° 7 quitte le tronçon de voie unique
Train n° 8 quitte le tronçon de voie unique
Train n° 2 circule de A vers B  >>>>>>
Train n° 6 circule de A vers B  >>>>>>
Train n° 6 quitte le tronçon de voie unique Train n°
2 quitte le tronçon de voie unique
Train n° -2 circule de B vers A  <<<<<<<<
Train n° -2 quitte le tronçon de voie unique
>
```