Voie unique avec tampons intérmédiaires

Ensimag 2A

1 Voie unique

Le problème étudié est celui de l'exclusion mutuelle entre deux classes de processus, les processus d'une même classe ne s'excluant pas mutuellement. Pour cela, on utilise l'analogie suivante : La ligne de chemin de fer Grenoble-Lyon est en travaux. Elle comporte un tronçon à voie unique entre Voiron et Bourgouin (i.e. il y a deux voies entre Grenoble et Voiron, une seule voie entre Voiron et Bourgouin et deux voies entre Bourgouin et Lyon).

La gare de Rives, entre Voiron et Bourgoin, peut stocker deux trains (dans n'importe quel sens) en plus de la voie de circulation.

Les règles de circulation sur la voie unique sont les suivantes :

— Un tronçon à une voie ne doit jamais être emprunté par deux trains allant en sens inverse entre Voiron et Rives et Rives et Bourgouin. Ces deux tronçons sont indépendants.

— Un tronçon à une voie peut-être emprunté par un ou plusieurs trains allant dans le même sens.

— Il n'y a pas de sens de parcours prioritaire.

— La gare de Rives ne peut pas stocker plus de deux trains.

Les voyages des trains des deux classes sont modélisés avec les fonctions suivantes

```
void Train_Lyon_Grenoble() {
  parcours_Lyon_Bourgoin();
  depart_Bourgoin();
  parcours_Bourgoin_Rives();
  attente_BV_a_Rives();
  parcours_Rives_Voiron();
  arrive_Voiron();
  parcours_Voiron_Grenoble();
}
void Train_Grenoble_Lyon() {
  parcours_Grenoble_Voiron();
  depart_Voiron();
  parcours_Voiron_Rives();
  attente_VB_a_Rives();
  parcours_Rives_Bourgoin();
  arrive_Bourgoin();
  parcours_Bourgoin_Lyon();
}
```

Les fonctions de synchronisations pour l'usage des voies sont donc : `depart_Bourgoin()`, `depart_Voiron()`, `arrive_Bourgoin()`, `arrive_Voiron()`, `attente_BV_a_Rives()`, `attente_VB_a_Rives()`.

1. Écrire le code avec des sémaphores sans tenir compte de la gare de Rives et de son stockage possible des trains à Rives. Votre solution pourra être FIFO sur l'ordre d'accès à la voie unique. Vous pouvez aussi compter les trains dans chaque sens.

> **Solution:** classique : un sémaphore pour choisir le sens. Un sas utilisé aux deux bout pour la FIFO. deux mutex différents. deux compteurs dans chaque sens.

```
void depart_Bourgouin() {
  sas.P();
  mB.P()
  if (Sens != BV) {
    if (nbBV == 0) {
      voies.P();
      Sens = BV;
    }

    nbBV++;
  }
  mB.V();
  sas.V()
}
void attente_BV_a_Rives() {}
void arrive_Voiron() {
  mB.P()
  nbBV --;
  if (nbBV == 0) {
    voies.V();
  }
  mB.V()
}
```

2. Écrire le code des fonctions de synchronisations avec des moniteurs. Plusieurs approches sont possibles. Une des approches est la suivante. Vous pouvez compter le nombre de trains dans chaque sens sur chaque tronçon avec les variables `int nbBR, nbRV, nbVR, nbRB`. Vous pouvez choisir un sens courant identique pour les deux tronçons. Vous pouvez compter le nombre de train total dans le bon sens. Si le train au départ n'est pas dans le sens courant, il doit vérifier qu'il a de la place à Rives pour le stocker. Si un train est seul sur les voies, il modifie le sens courant.
   Vous pouvez n'écrire que les 3 fonctions d'un même sens si votre solution est symétrique. Mais il faudra définir toutes les variables.

Solution:

```
int nbDispoRives = 2;
enum Sens = { VB, BV };
int nbSens;
int nbBR, nbRV, nbRB, nbBR;

cond_t cB, cV, c_RBV, c_RVB;
mutex_t m;

void depart_Bourgoin() {
  m.lock()
  if (nbSens == 0)
  Sens = BV;

  while( Sens != BV  && nbDispoRives == 0
         || nbRB != 0 )
  cB. wait(m);

  if (nbSens == 0)
     Sens = BV;
```

```
    if (Sens == BV)
      nbSens ++;
    else
      nbDispoRives --;

    nbBR ++;
    m.unlock();
  }

  void attente_LG_a_Rives() {
    m.lock();
    nbBR --;
    if (nbBR ==0)
      c_RVB.signal();

    while(nbVR != 0)
      c_RBV.wait(m);

    if (Sens != BV)
      nbDispoRives ++;

    nbRV ++;
    m.unlock();
  }

  void arrive_Voiron() {
    m.lock();
    nbRV --;

    if (Sens == BV)
      nbSens --;

    if (NBRV == 0)
      cV.signal();

    m.unlock();
  }
```