Prof. Yann Thoma

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Programmation Concurrente (PCO)
Semestre printemps 2014-2015
Contrôle continu 1 (A)
27.04.2015

Prénom: VALENTIN D.

Nom: MINDER

— Aucune documentation n'est permise, y compris la feuille de vos voisins
— La calculatrice n'est pas autorisée
— Aucune réclamation ne sera acceptée en cas d'utilisation du crayon
— Ne pas utiliser de couleur rouge

La version "B" ne diffère pas pour exos 3 & 4.
Exos 1 & 2: changements négligeables (valeur des variables pour 1, flèches différentes pour 2)

Question	Points	Score
1	10	10
2	10	10
3	15	15
4	15	15
Total:	50	50

Note: 6

Question 1: 10 points

Soit le listing suivant :

```
int x = 0;

void TacheA::run() {
    x += 2;

}

void TacheB::run() {
    x += 5;
}


int main (int argc, char *argv[]) {
    TacheA tA;
    TacheB tB;
    tA.start();
    tB.start();
M:  x=1;
    tA.wait();
    tB.wait();
    std::cout << "Valeur de X: " << x << std::endl;
}
```

Quelles sont les valeurs de x pouvant être observées en fin de programme ?

Une instruction "x += a" n'est pas atomique et peut se décomposer en instruction élémentaires suivantes :

1) reg ← x
2) reg ← reg + a
3) x ← a

reg : registre CPU
x : valeur mémoir

Une (interuption) préemption peut survenir à n'importe quel moment entre ces instructions.

Versions "simples" sans préemption (M = main) avec M, TA, TB

M → TA → TB : x = 8
M → TB → TA : x = 8
TA → M → TB : x = 6
TA → TB → M : x = 1
TB → M → TA : x = 3
TB → TA → M : x = 1

Avec préemption, il y a 7 instructions (TA,TB : 1 à 3, M)
(toutefois l'instr. 2 n'a pas bcp d'importance, donc 5)
donc 5! = 120 déroulements différents

p.ex. (extrait)

M → TA1 → TB1 → TA3 → TB3 += 6
↳ TB3 → TA3 += 3

TA1 → TB1 → M → TA3 → TB3 += 5
↳ TB3 → TA3 += 2

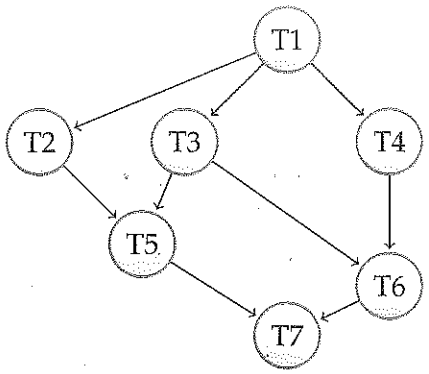
TA1 → TA3 → TB1 → M → TB3 += 7

} nouveau

Les valeurs 1/2/3/5/6/7/8 sont possibles.
(mais pas 0 car au moins 1 des trois M, TA, TB va passer à 1, 2 ou 5)
(et pas 4 car on ne peut l'atteindre avec 0, 1 +2 et +5)

Question 2: 10 points

Soit le graphe d'exécution des tâches suivant :



Les tâches doivent exécuter leur traitement dans l'ordre indiqué par les flèches. Complétez le programme ci-dessous de manière à garantir ce fonctionnement, tout en optimisant le parallélisme potentiel, et ce en utilisant des sémaphores pour la synchronisation.

Rappel sur les sémaphores Qt :

— `QSemaphore::QSemaphore(n)` : correspond à initialiser le sémaphore à `n` (`n` doit être ≥ 0) ;
— `QSemaphore::acquire()` : correspond à `P(sémaphore);`
— `QSemaphore::release()` : correspond à `V(sémaphore).`

```
// Déclarations
QSemaphore q1, q2, q3, q4, q5, q6

// programme principal
int main (int argc, char *argv[]) {
  Task1 t1;
  ...
  Task7 t7;
  q1 QSemaphore (0);
  :
  idem
  q6

  // Toutes les créations de threads
  t1.start();
  ...
  t7.start();

  t1.wait();
  ...
  t7.wait();

}

void Task1::run() {

// Instructions diverses
  q1.release();
  q1.release();
  q1.release();
}

void Task2::run() {
  q1.acquire();
// Instructions diverses
  q2.release()
}

void Task3::run() {
  q1.acquire();
// Instructions diverses
  q3.release()
  q3.release()
}

void Task4::run() {
  q1.acquire();
// Instructions diverses
  q4.release();
}

void Task5::run() {
  q2.acquire();
  q3.acquire();
// Instructions diverses
  q5.release();
}

void Task6::run() {
  q4.acquire();
  q3.acquire()
// Instructions diverses
  q6.release()
}

void Task7::run() {
  q5.acquire()
  q6.acquire()
// Instructions diverses
}
```

attend la fin de 1.

attend la fin de 3 (après l'autre car l'autre n'a qu'un seul prédécesseur)

laisse partir 3 threads attendant sur la fin de 1

Question 3: 15 points

Nous désirons réaliser la modélisation d'un arrêt de bus. Les threads `Personne` devront, lorsqu'ils arrivent à l'arrêt, attendre le bus, sauf s'il y a plus de 30 personnes qui attendent déjà. Dans ce cas, elles continueront leur chemin. L'attente du bus reviendra à suspendre le thread. Les threads `Bus` devront, lorsqu'ils arrivent, embarquer les personnes présentes, mais au maximum 10. L'embarquement reviendra simplement à réveiller les threads personnes concernés.

Les threads `Personne` appeleront la méthode `peopleArrives()` alors que les bus appeleront la méthode `busArrives()`.

En utilisant uniquement des sémaphores Qt, écrivez la classe `BusStop` pour répondre à cette question.

Rappel sur les sémaphores Qt :

— `QSemaphore::QSemaphore(n)` : correspond à initialiser le sémaphore à `n` (`n` doit être ≥ 0) ;
— `QSemaphore::acquire()` : correspond à `P(sémaphore)` ;
— `QSemaphore::release()` : correspond à `V(sémaphore)`.

Le code suivant illustre un exemple d'utilisation :

```
BusStop *b;

// exemple de tâche
void Personne::run() {
    b->peopleArrives();
    ...
}

// exemple de tâche
void Bus::run() {
  while (1) {
      b->busArrives();
        ...
      }
}

// programme principal
int main (int argc, char *argv[]) {
    b = new BusStop();
    // création de threads
    ...
}
```

```
#include <QSemaphore>

class BusStop {
  BusStop();
  ~BusStop();

  void peopleArrives();
  void busArrives();
};
```

```
#define BUS_MAX 10
#define PERS_MAX 30
class BusStop {
    QSemaphore * mutex;      // protège nbWaiting
    QSemaphore * waiting;    // pour l'attente des personnes
    unsigned int nbWaiting;  // nb de personnes en attente
                             (à libérer et/ou pour passer tout droit...)
public:
    BusStop() {
        mutex = new QSemaphore(1);
        waiting = new QSemaphore(0);
        nbWaiting = 0;
    }
    ~BusStop() {
        delete mutex;
        delete waiting;
    }
    void peopleArrives() {
        mutex->acquire();
        if (nbWaiting > PERSMAX) {
            mutex->release();   // la personne continue son chemin s'il y a plus de PERSMAX personnes.
            return;
        } else {
            nbWaiting++;
            mutex->release();   // restitution mutex (pour autre personne ou bus)
            waiting->acquire(); // mise en attente.
        }
    }
    void busArrives() {
        mutex->acquire();
        int toLiberate = BUS_MAX;
        while (toLiberate > 0 && nbWaiting > 0) {
            toLiberate--;
            nbWaiting--;
            waiting->release()
        }
        mutex->release();
    }
};  // fin de class BusStop
```

on peut être préempté entre mais pas grave: 1→30 vont attendre et suivants passer tout droit.

// on libère des personnes jusqu'à ce qu'il n'y en ait plus ou jusqu'à ce que le bus soit plein.

Question 4: 15 points

Nous désirons contrôler l'accès à un pont à deux voies, et ce pour des voitures et des camions. Les voitures pèsent 1 tonne alors que les camions pèsent 10 tonnes. La première voie ne peut être empruntée que par des voitures, car elle est plus étroite que la deuxième, alors que la deuxième peut l'être par des camions et des voitures. Chacune des voies peut supporter au maximum 100 tonnes.

La méthode `carAccess()` retourne un entier qui vaut 1 si la voiture peut passer par la première voie, et 2 si elle doit passer par la deuxième. La méthode `truckAccess()` ne retourne rien car les camions sont obligés de passer par la deuxième voie.

Les deux méthodes sont potentiellement bloquantes si le véhicule doit attendre la libération d'espace sur le pont, et les voitures passeront sur la première voie si elle est libre et sinon sur la deuxième. Si aucune des voies n'est libre, alors les voitures attendent et passeront par la première voie.

A vous d'implémenter la class `DoubleBridge` pour répondre à ce cahier des charges.

```
class DoubleBridge
{
  DoubleBridge();
  ~DoubleBridge();

  int accessCar();
  void accessTruck();
};
```

en brun: rajouté après coup pour éviter famine camion sur la voie 2 (voir commentaire 1.10)

ajout: void carLeave(int voie);
void truckLeave();
};

```
#define CAR_WEIGHT 1
#define TRUCK_WEIGHT 10
#define MAX_WEIGHT 100

class DoubleBridge {
    QSemaphore* mutex;
    unsigned int load1;
    unsigned int load2;
    QSemaphore* waiting1;
    QSemaphore* waiting2;
    unsigned int nbWaiting2;

DoubleBridge() {
    mutex = new QSemaphore(1);
    waiting1 = new QSemaphore(0);
    waiting2 = new QSemaphore(0);
nbWaiting2 = load1 = load2 = 0;
}
~DoubleBridge() {
    delete mutex;
    delete waiting1;
    delete waiting2;
}
```

// protège load1 et 2, nbWaiting2
// charge actuelle sur chacune des voies
// semaphores d'attente pour chacune des voies.
// nombre de camions en attente sur la file 2

```
void accessTruck (){
    mutex->acquire(); // camion en attente
    nbWaitings2++;
    while (load2 + TRUCK_WEIGHT > MAX_WEIGHT){
(#)     mutex->release();
        waitings2->acquire();
        mutex->acquire();
    }
    load2 += TRUCK_WEIGHT;
    nbWaitings2--;
    mutex->release(); // camion passe
}
void truckLeave (){
    mutex->acquire();
    load2 -= TRUCK_WEIGHT;
    waitings2->release();   // libération d'un char sur la file 2.
    mutex->release();
}
int accessCar (){
    mutex->aquire()
    if (load1 + CAR_WEIGHT <= MAX_WEIGHT)
        load1 += CAR_WEIGHT;
        mutex->release();
        return 1;                                   } 1 essai sur la piste 1
(*) } if (nbWaitings2 == 0) &&
    if (load2 + CAR_WEIGHT <= MAX_WEIGHT) {
        load2 += CAR_WEIGHT;
        mutex->release();
        return 2;                                   } 1 essai sur la piste 2
    }
    while (load1 + CAR_WEIGHT > MAX_WEIGHT){
        mutex->release();
        waitings1->acquire();
        mutex->aquire()                             } attente sur la piste 1
    }
    load1 += CAR_WEIGHT;
    mutex->release();
    return 1;
}
```

```
void carLeave (int voie ) {
    mutex -> acquire();
    if (voie == 1){
        load1 -= CAR_WEIGHT;
        waiting1->release();

    }else if (voie == 2){
        load2 -= CAR_WEIGHT;
        waiting2 ->release();
    } else {
        // error ?!

    }
    mutex -> release();
}

}; // fin de classe.
```

La solution en bleu ci-dessus garanti bien le protocole et l'exclusion mutuelle. Toutefois on peut observer une famine des camions s'il y a beaucoup de voiture en continue et passent sur la voie 2 dès qu'1 tonne est libre quand elle arrivent (et les camions passent pas car ils doivent attendre 10 tonnes). Pour éviter cela voir les ajouts en brun: une variable nbWaiting2 qui mémorise combien de vhc sont en attente sur la voie 2. S'il n'est pas zéro (aka des camions attendent) aucune voiture ne pourra passer devant grâce à (*).

Seul défaut: lorsqu'une voiture sort de la voie 2, elle (#) peut réveiller un camion qui devra directement réattendre car il n'y a pas assez de tonnes disponibles. (mais au moins une voiture ne lui passera pas devant)

TE PCO connection

```
class DoubleBridge {
    QSemaphore * mutex;
    QSemaphore * attente1;
    QSemaphore * attente2;
    int nbAttente1;
    int nbAttente2;
    int poids1;
    int poids2;

DoubleBridge () {
    mutex    = new QSemaphore(1);
    attente1 = new QSemaphore(0);
    attente2 = new QSemaphore(0);
    nbAttente1 = nbAttente2 = poids1 = poids2 = 0;
}

~DoubleBridge(){
    delete mutex;
    delete attente1;
    delete attente2;
}

int carAccess () {
    mutex -> acquire ();
    if (poids1 + CAR_WEIGHT <= MAX_LOAD) {
        poids1 += CAR_WEIGHT;
        mutex -> release ();
        return 1;
    }
    if (poids2 + CAR_WEIGHT <= MAX_LOAD) {
        poids2 += CAR_WEIGHT
        mutex -> release();
        return 2;
    }
    nbAttente1 ++;
    mutex -> release ();
    attente1 -> acquire ();
    poids1 += CAR_WEIGHT;
    mutex -> release()
    return 1

void truckAccess () {
    mutex -> acquire()
    if (poids2 + TRUCK_WEIGHT > MAX_LOAD){
        nbAttente2 ++;
        mutex -> release();
        attente2 -> acquire();
    }
    poids2 += TRUCK_WEIGHT;
    mutex -> release();
}
```

Sk (nbWaitins2 == 0)

```
void carLeave(int voie) {
    mutex->acquire()
    if (voie == 1) {
        poids1 -= CAR_WEIGHT
        if (nbAttente1 > 0) {
            nbAttente1--;
            attente1->release()   // transmission du mutex
            return;
        } else {
            mutex->release();
        }
    } else {
        poids2 -= CAR_WEIGHT
        if ((nbAttente2 > 0) && (poids2 + TRUCK_W <= MAX_LOAD)) {
            nbAttente2--;
            attente2->release();   // transmission du mutex
            return;
        } else {
            mutex->release()
        }
    }
}
void truckLeave() {
    mutex->acquire();
    poids2 -= TRUCK_WEIGHT
    if (
        nbAttente--;
        attente2->release();
        return;
    } else {
        mutex->release();
    }
}

// version avec des while possible...

// attention famine des camions possible
```