UNIVERSITÉ DE SHERBROOKE

DÉPARTEMENT D'INFORMATIQUE

Devoir # 1

IFT 630

Processus concurrents et parallélisme

Devoir à remettre au plus tard le 17 février 2021.

La correction sera basée sur une **démonstration du bon fonctionnement** de vos programmes. Vous devez me soumettre votre programme à l'aide de turnin (dans le projet tp1) avant la date de remise. Une démonstration devra ensuite être organisée pour la correction.

Travail à faire

Vous devez produire :

1. une solution aux problèmes 1, 2 et 3, écrite dans le langage SR (ou JR) [1] utilisant les sémaphores.
2. une solution aux problèmes 1, 2 et 3, écrite dans le langage SR (ou JR) utilisant les moniteurs (option -sw).
3. une solution aux problèmes 1, 2 et 4, écrite dans le langage SR (ou JR) utilisant les messages.

Description des problèmes

1. Le problème des philosophes tel que décrit dans le cours.
2. Le problème de la boutique du barbier.

   Dans une petite ville, on retrouve une boutique de barbier ayant deux portes et quelques chaises. Les clients entrent par une porte et quittent par l'autre porte. À cause de la dimension restreinte de la boutique, un seul client à la fois ou uniquement le barbier peut se déplacer dans la boutique.

1. Les langages SR et JR sont disponibles dans l'environnement Linux des laboratoires du Département d'informatique

Le barbier passe sa vie à servir des clients, un à la fois. Quand il n'y a aucun client dans la boutique, le barbier s'endort. Quand un client arrive et trouve le barbier endormi, il réveille le barbier et s'assoit sur la chaise du barbier pendant que celui-ci lui coupe les cheveux. Si le barbier est occupé, le client s'assoit sur une chaise pour attendre son tour. S'il n'y a pas de chaise, le client s'en va.

Après avoir terminé la coupe de cheveux, le barbier ouvre la porte de sortie pour le client et la ferme lorsqu'il est sorti. S'il y a des clients en attente, le barbier appelle le suivant et attend qu'il soit assis pour le servir. Sinon, le barbier retourne dormir.

Dans votre solution, vous devez utiliser un processus distinct pour chaque client et un pour le barbier. Un client particulier doit se synchroniser avec le barbier pour toutes les étapes de la coupe de cheveux jusqu'à ce qu'il quitte après que la coupe soit terminée.

Vous devez avoir plus de processus « clients » que de chaises disponibles.

Points importants à considérer :

- Le barbier (processus) doit dormir (salle vide) et doit être réveillé lorsque cela est requis (arrivée d'un nouveau client). «Dormir» ici signifie mettre le processus en attente (sur un sémaphore, variable condition ou réception de message).
- Le client attend sur une chaise si le barbier est occupé. Il ne se fait pas couper les cheveux sur cette chaise. Lorsque le barbier devient disponible, il doit se déplacer vers la chaise du barbier.
- Le barbier attend que le client soit assis. Cette situation se produit lorsque client se déplace d'une chaise dans la salle d'attente vers la chaise du barbier. De même, lorsque le barbier se fait réveiller, le client n'est pas nécessairement assis. Il faut donc prévoir un mécanisme avec lequel le barbier attend que le client soit assis.
- Le barbier doit attendre que le client soit sorti pour appeler le prochain client. Il faut donc prévoir un mécanisme qui synchronise la sortie du client avec le barbier.

3. Le problème du parc d'attraction.

   Soit un parc d'attraction possédant des montagnes russes sur lesquelles circule une

voiture pouvant contenir au plus C passagers. Plusieurs personnes (N tel que $N > C$) attendent de façon répétitive pour faire un tour de voiture. La voiture ne part que si elle est pleine et elle fait 5 tours de « piste ». Il est important que, dans vos solutions, les personnes et la voiture soient implantées comme des processus.

Point important à considérer :

→ Il faut que tous les passagers soient débarqués avant de débuter un nouvel embarquement. Vous devez donc prévoir un mécanisme de synchronisation qui signale aux clients en attente que la voiture est vide et qu'ils peuvent débuter l'embarquement.

4. Multiplication de matrices.

   Implanter la multiplication de matrice avec la communication par messages asynchrones. Utiliser le concept de processus travailleurs (workers). Vous avez donc plusieurs processus travailleurs qui font des produits scalaires demandés par un processus maître.

   Votre solution doit utiliser uniquement la communication par messages (aucune variable partagée par les processus travailleurs, aucun sémaphore ni moniteur). Elle devrait pouvoir fonctionner même si les travailleurs sont sur des sites différents.

Questions fréquentes (FAQ)

JR - Comment installer JR ?

Les instructions complètes sont accessibles à partir du site Web du cours. Il n'est pas nécessaire d'installer la partie "multi VM JR programs".

Attention :

JR ne fonctionne qu'avec une vielle version de Java (1.8).

JR - Comment compiler et exécuter un programme en JR

Il suffit de faire «`jr nom_fichier`».

Il est important de ne pas mettre l'extension (.jr).

JR - Comment déclarer un tableau de variables conditions ?

Pour déclarer un tableau de variables conditions dans JR, il faut faire :

```
_condvar cond[5];
```

Et pour l'utiliser :

```
_wait(waiting[i]);
```

SR - Comment installer SR sur Ubuntu ?

Les instructions complètes sont disponibles sur le site Web du cours.

SR - Comment compiler et exécuter un programme ?

Il suffit de faire «`sr -o nom_exécutable nom_fichier.sr`».

Si vous ne mettez pas l'option «-o», l'exécutable se trouvera dans le fichier «a.out».

Attention

Pour une raison bizarre et inconnue, **la compilation échoue si le fichier n'est pas copié localement sur l'ordinateur**. Assurez-vous donc avant de compiler que le fichier est sur le disque local (dans les laboratoires, cela signifie que le fichier ne doit pas se trouver sur le Lecteur_Reseau)

SR - Est-ce normal d'avoir des «Warning» à la fin d'exécution d'un programme ?

Ce «warning» indique qu'il y a des processus qui sont bloqués. Cela est possiblement normal si vous n'avez pas prévu un mécanisme pour terminer tous les processus à la fin de l'exécution du programme. Il n'est pas nécessaire de prévoir un tel mécanisme.

SR - Comment générer des nombres aléatoires ?

Voici quelques exemples de générations de nombres aléatoires en SR :

```
resource main()
       var x : real
       var y : real
       var z : real

process p1
      # genere une valeur aleatoire entre 0 et 1
       x := random()
      write("x =", x)
      # genere une valeur aleatoire entre 0 et 10
       y := random(10)
      write("y =", y)
      # genere une valeur aleatoire entre 4 et 10
       z := random(4,10)
      write("z =", z)
end p1
end main
```

SR - Comment déclarer un tableau de variables conditions ?

Pour déclarer un tableau de variables conditions, appelé cond, il faut faire :

```
_condvar1(cond,0:4)
```

Et pour l'utiliser :

```
_wait(cond[i])
```

SR - Comment concaténer deux chaînes de caractères ?

La concaténation de string se fait avec l'opérateur `||`.

```
var s : string

s :=  "abc" || "def"
```

SR - Comment accéder un caractère particulier dans un chaîne de caractères ?

On peut aussi faire `s[10]` pour accéder le 10^e caractère.

SR - Comment créer des procédures sans exclusion mutuelle dans le moniteurs ?

Oui. Au lieu d'utiliser le mot réservé «`_proc`» pour déclarer une procédure, il suffit d'utiliser le mot réservé «`procedure`».

SR - Comment manipuler les fichiers ?

Pour manipuler un fichier, il faut créer une variable de type file qui fournira :

- les constantes suivantes :
  - → Mode d'ouverture : READ, WRITE, READWRITE ;
  - → Fin de fichier : EOF (-1).
- les fonctions suivantes :
  - → open, close, remove
  - → read, write
  - → seek, where

Le programme suivant montre comment utiliser les fichiers.

```
var f : file
f := open("nomFichier", READWRITE)
if f = null ->
    write("Erreur a l'ouverture")
fi
read(f,x)
write(f,y)
...
printf(f, format, x1, x2, ...)
scanf(f, format, y1, y2, ...)
....
// x est une chaine de caracteres
get(f, x)  retourne le nbre de caracteres lus
put(f,x)
...
var f:file, e:int
f:= open("foo", READWRITE)
e:= seek(f, EXTEND, 0) // retourne la taille du fichier
                       // EXTEND indique ajouter a la fin du fichier
fa i:=0 to e by 2 ->
      seek(f, ABSOLUTE, i)   // l'autre option du seek est RELATIVE
      put(f, "*")
af
close(f)
```

Voici un second exemple de programme SR qui copie un fichier :

```
resource copie()
   var fichier1, fichier2 : string[64]
   var entree, sortie : file
   var ligne : string[256]

   getarg(1, fichier1)  # 1er paramètre -> nom du fichier à copier
   getarg(2, fichier2)  # 2e paramètre -> nom du fichier en sortie

   entree := open(fichier1, READ)
   if entree = null ->
      write(stderr, "Impossible d'ouvrir ", fichier1, " en lecture"); stop
   fi

   sortie := open(fichier2, WRITE)
   if sortie = null ->
      write(stderr, "Impossible d'ouvrir ", fichier2, "en ecriture"); stop
   fi

   do
      read(entree, ligne) != EOF -> write(sortie, ligne)
   od
end copie
```

SR - Comment accéder des enregistrements précis dans un fichier?

Cela se fait grâce à la commande «seek» qui positionne le pointeur dans le fichier et le retourne sa valeur.

Ainsi, «`e := seek(f, EXTEND, 0)`» positionne le pointeur à la fin du fichier et retourne la longueur du fichier si le pointeur pointait au début du fichier.

La syntaxe exacte est `seek(fichier, stype, offset)`

- `fichier` est l'identificateur du fichier ;
- `stype` est le type de `seek` : ABSOLUTE, RELATIVE, EXTEND
  → avec `ABSOLUTE`, le pointeur est initialisé à «offset»
  → avec `RELATIVE`, le pointeur on additionne «offset» au pointeur courant
  → avec `EXTEND`, le pointeur est placé à la fin du fichier + «offset».

Ainsi, «`e := seek(f, EXTEND, 0)`» positionne le pointeur à la fin du fichier et retourne la longueur du fichier si le pointeur pointait au début du fichier.

SR - Comment précise-t-on les cibles des envois et réceptions de message ?

La communication se fait grâce à des ports de communication en SR. Les ports sont définis grâce à la commande «op». Voici un exemple de code.

```
resource main()
   op pool(index: int)
   const B := 20
   const N := 10
   var vec[1:B]: T  # T est le type fictif du tampon

   fa i := 1 to B -> send pool(i) af

   process p(i := 1 to N)
     ...
     receive pool(x)
     # utilisation de vec[x]
     send pool(x)
     ...
   end
end
```

SR - Comment transmettre une matrice entière ?

Voici une exemple de programme qui transmet des matrices.

```
resource main()
  # Déclaration du port qui transmettra deux entiers et une matrice
  op lien(x:int;y:int; a[1:*,1:*]:int)

  process p1
    # Déclarartion de la matrice 5x5
    var x[1:5,1:5]: int
    fa i:= 1 to 5 ->
      fa j:= 1 to 5 ->
        x[i,j] := i+j;
      af
    af
    send lien(2,3,x)
  end p1

  process p2
    var a,b : int
    var test[1:5,1:5] : int
    receive lien(a, b, test)
    write("a = ", a)
    write("b = ", b)
    fa i:= 1 to 5 ->
      write("\n")
      fa j:= 1 to 5 ->
        printf("%d ", test[i,j])
      af
    af
    write()
  end p2
end main
```

SR - Comment peut-on transmettre une colonne d'une matrice ?

Il est possible d'accéder une ligne ou une colonne d'une matrice en SR grâce à l'opérateur «*». Ainsi, si on veut accéder à la 3^e colonne d'une matrice M, on fait

```
colonne = M[*,3]
```

Un exemple complet est présenté à la figure 1.

Philosophes - Peut-on utiliser un moniteur par fourchettes ?

Non. Toutes les fourchettes doivent être gérées dans un seul moniteur.

Barbier - Utilisation des processus dans le problème du barbier (moniteurs) ?

Vous devez avoir un processus par client, un processus pour le barbier et un seul moniteur représentant le salon.

Barbier - Comment gérer l'attente des clients (barbier) si la salle est pleine ?

Dans l'énoncé, il est dit que lorsqu'un client constate qu'il n'y a plus de chaises disponibles (de l'extérieur), il s'en va. Le client (processus) doit alors «aller faire un tour» et éventuellement revenir. Il ne doit pas attendre à l'extérieur si toutes les chaises sont utilisées.

Le concept d'aller faire un tour et revenir n'est pas considéré comme de l'attente active (type d'attente que l'on veut éviter) car cela n'est pas relié à la synchronisation. Ce temps d'attente (ou de passer le temps) peut se faire avec un sleep (nap). Cela remplace une forme de «travail» que le processus fait en attendant d'aller demander la ressource.

Barbier - Accompagnement d'un client par le barbier à la fin de la coupe ?

Le barbier doit reconduire le client à la porte à chaque fois et non seulement lorsqu'il n'y a personne qui attend. Lorsque le barbier reconduit le client, il va ouvrir la porte, attend que le client sorte puis va servir le prochain client.

Parc - Comment gérer les personnes et les voitures dans le problème du parc ?

Vous devez créer un processus par personne et un processus par voitures.

```
# nom du fichier test.sr
# compilation sr -o test test-mat1.sr
# transmission de ligne et de colonne d'une matrice
resource main()
  op lien(x:int;y:int; a[1:*]:int)

  process p1
    var x[1:5,1:5]: int
    #Initialisation de la matrice
    fa i:= 1 to 5 ->
      fa j:= 1 to 5 ->
        x[i,j] := 2+i;
      af
    af
    fa i:= 1 to 5 ->
      write()
      fa j:= 1 to 5 ->
        printf("%d ", x[i,j])
      af
    af
    write("\n")
    #On transmet la ligne 2
    send lien(2,0,x[2,*])
    # On transmet la colonne 3
    send lien(0,3, x[*,3])
  end p1

  process p2
    var a,b : int
    var test[1:5] : int
    # Reception de la colonne 2 et affichage
    receive lien(a, b, test)
    write("a = ", a," , b = ", b)
    fa j:= 1 to 5 ->
      printf("%d ", test[j])
    af
    write("\n")
    # Reception de la colonne 3 et affichage
    receive lien(a, b, test)
    write("a = ", a," , b = ", b)
    fa j:= 1 to 5 ->
      printf("%d ", test[j])
    af
    write()
  end p2
end main
```

Figure 1 – Exemple de manipulation des matrices en SR

Parc - Doit-on avoir une file de clients en attente pour une voiture ?

Oui. Le client ne peut pas simplement aller se promener et revenir plus tard lorsqu'il se rend compte que la voiture n'est pas disponible. Il faut implanter une file d'attente qui permet entre autre de garder une certaine priorité qui va leur garantir une place dans la voiture s'ils étaient les premiers arrivés.

Il ne faut pas oublier qu'un sémaphore fournit une file d'attente.

Matrices - Que signifie «utiliser uniquement la communication par messages» ?

Vous ne devez pas mélanger les sémaphores, les moniteurs et les messages. Lorsque vous utilisez les messages vous devez seulement utiliser les messages. Vous ne devez pas avoir de variables communes (donc pas de sémaphores ou de moniteurs).

Matrices - Doit-on créer les processus dynamiquement lors de la multiplication ?

Pas obligatoire. Il est possible de créer dynamiquement les processus mais il est aussi possible de créer un nombre fixe de processus pour faire les calculs.

Matrices - Comment doit-on obtenir les matrices (lecture, paramètre, fichier, ...) ?

Vous pouvez obtenir la matrice avec la méthode de votre choix. Vous pouvez les définir comme constantes, les lire au clavier ou les lire dans un fichier.

Si vous décidez de les mettre dans un fichier, les fichiers se manipulent de la façon suivante.

```
var i : int  (ou le type désiré)
f := open("nomDuFichier", READ)

r := read(f,i)
if r = EOF ...
```