UNIVERSITÉ DE SHERBROOKE

DÉPARTEMENT D'INFORMATIQUE

IFT 630

Processus concurrents et parallélisme

Examen final

Le lundi 25 avril 2019
de 13 h 30 à 16 h 20

Professeur : Gabriel Girard

Notes :
- Toute documentation est permise.
- Répondez dans les espaces prévus à cet effet.
- Cet examen comporte 10 questions sur 24 pages.
- Le total de l'examen est sur 110. Le résultat sera tronqué sur 100.
- **Justifiez chacune de vos réponses**.

Nom : _____ Prénom : _____

Signature : _____ CIP : _____

| Question | Barème |
|---|---|
| 1 | / 6 |
| 2 | / 12 |
| 3 | / 10 |
| 4 | / 12 |
| 5 | / 8 |
| 6 | / 12 |
| 7 | / 12 |
| 8 | / 14 |
| 9 | / 14 |
| (Bonus) 10 | /10 |
| Total : | /110 |

1. Les serveurs

   (a) Comment un serveur ayant un seul fil d'exécution (*mono-thread*) et utilisant la communication par messages peut-il assurer l'exclusion mutuelle lors de l'accès aux fichiers qu'il contrôle ?

   (b) Systèmes de fichiers distribués.

   Le système de fichiers distribués NFS est un système qui ne maintient aucune connexion (et aucun état) entre le client et le serveur (*stateless*). Expliquez en vos mots ce que signifie «*stateless*» (et ses implications) et donnez un avantage de cette approche.

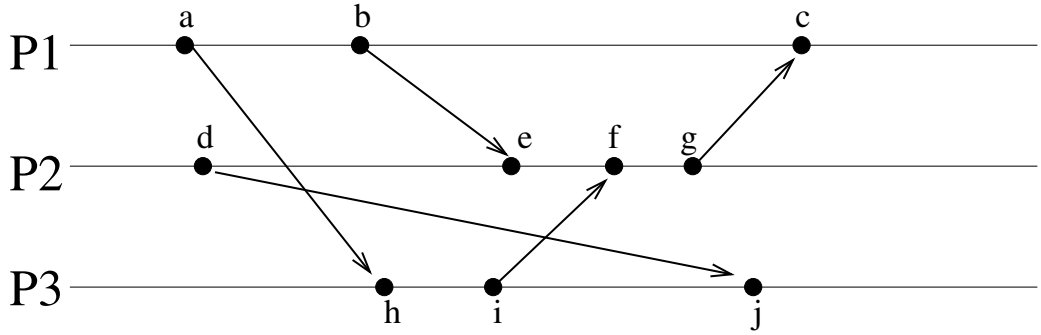
2. Horloges et exclusion mutuelle distribuée

   (a) Expliquez la différence entre «horloge logique» et «horloge physique» et dites pourquoi aucun des deux ne peut fournir un ordonnancement total des événements.

   (b) Quel problème peut-il se produire si lors de la synchronisation de l'horloge, plutôt que d'ajuster l'heure progressivement (en accélérant ou ralentissant la progression de l'horloge), on l'ajuste abruptement en l'avançant ou la reculant directement ?

(c) Horloges logiques (Lamport)

La figure suivante montre un certain nombre d'événements se produisant entre trois processus P1, P2 et P3. En supposant que les horloges sont initialisées à 0 et que les événements sont seulement des envois et des réceptions de messages, quelles sont les valeurs des horloges associées aux événements «a» à «j» ? Servez-vous de la figure pour illustrer les différentes valeurs associées à l'horloge.



3. Exclusion mutuelle

   Vous êtes à la tête d'une compagnie qui développe des logiciels pour les systèmes répartis. Un de vos employés vous propose un nouvel algorithme pour implanter l'exclusion mutuelle répartie. Son algorithme suppose l'utilisation d'un jeton, le jeton de verrouillage, qu'un noeud doit détenir pour pouvoir entrer en section critique. Le noeud possédant le jeton est appelé le propriétaire du jeton. L'algorithme suppose aussi la présence d'un noeud centralisé, appelé le gestionnaire, qui conserve la localisation courante du jeton de verrouillage (i.e. l'identification du propriétaire du jeton de verrouillage). Le jeton de verrouillage change de position seulement s'il est réclamé par un autre noeud. Ainsi, le jeton demeure sur le noeud propriétaire tant qu'il est en section critique et, par la suite, tant qu'il n'est pas réclamé par un autre noeud. Initialement le jeton est localisé sur le noeud gestionnaire.

   L'algorithme fonctionne de la façon suivante :

   1. Pour l'acquisition du verrou :
      - 1.1 Le noeud qui veut le verrou envoie une demande au gestionnaire ;
      - 1.2 Le gestionnaire lui retourne la localisation du propriétaire du jeton ;
      - 1.3 Le noeud demandeur contacte le propriétaire du jeton ;
      - 1.4 Le propriétaire maintient une liste locale pour emmagasiner toutes les demandes pour le verrou.
   2. Pour la libération du verrou :
      - 2.1 Le propriétaire choisit le demandeur en tête de liste (prochain propriétaire du verrou) ;
      - 2.2 Il envoie l'identificateur du prochain propriétaire au gestionnaire ;
      - 2.3 Il retire le noeud de la liste ;
      - 2.4 Il envoie le jeton de verrouillage et la liste restante au prochain propriétaire.

   En supposant qu'il n'y ait aucune panne, cet algorithme est-il correct ? Si oui, montrez-le. Si non, donnez un scénario qui montre que la solution est incorrecte.

4. La mémoire partagée distribuée

   Il y a plusieurs modèles de cohérence pour la mémoire partagée distribuée.

   — La cohérence stricte (atomicité)

   > *Toute lecture sur un élément x renvoie une valeur correspondant à l'écriture la plus récente sur x (ordonnancement selon une horloge «absolue»).*

   — La cohérence séquentielle

   > *Le résultat de l'exécution d'un ensemble de processus est identique à celle d'une exécution dans laquelle toutes les opérations ont été exécutées dans un ordre séquentiel S tel que :*
   > — *les opérations exécutées par tout processus p figurent dans S dans le même ordre que dans p ;*
   > — *la cohérence interne des données est respectée dans S (la lecture retourne la dernière valeur écrite dans S, i.e. tous les processus s'accordent sur un ordre global unique).*

   — La cohérence causale

   > *Dans le résultat d'une exécution, les opérations de mémoire qui ont potentiellement un lien causal*[a] *sont vues dans le même ordre par tous les processus du système.*
   >
   > *a.* La causalité est basée sur un lien du type «s'est produit avant» comme pour la définition des horloges logiques.

   — La cohérence PRAM

   > *Les écritures faites par un processus sont vues par tous les autres processus dans l'ordre dans lequel elles ont été émises. Les écritures émises par plusieurs processus peuvent être vues dans des ordres différents par tous les autres processus.*

   Les exemples suivants montrent l'exécution d'opérations sur un axe temporel. Parmi ces exemples d'exécutions, dites pour quels modèles de cohérence elles sont légales. Justifiez chacune de vos réponses.

(a) Exécution 1 :

| | | |
|---|---|---|
| P1 : | W(x,1) | |
| P2 : | | R(x,1) |

(b) Exécution 2 :

| | | | |
|---|---|---|---|
| P1 : | W(x,1) | | |
| P2 : | | R(x,0) | R(x,1) |

(c) Exécution 3 :

| | | |
|---|---|---|
| P1 : | W(x,1) | R(x,2) |
| P2 : | W(x,2) | R(x,1) |

(d) Exécution 4 :

| | | | | | |
|---|---|---|---|---|---|
| P1 : | | W(x,1) | | | |
| P2 : | W(x,2) | | | | |
| P3 : | | | R(x,1) | | R(x,2) |
| P4 : | | | | R(x,1) | R(x,2) |

(e) Exécution 5 :

| | | | | | | |
|---|---|---|---|---|---|---|
| P1 : | W(x,1) | | | W(x,3) | | |
| P2 : | | R(x,1) | W(x,2) | | | |
| P3 : | | R(x,1) | | | R(x,3) | R(x,2) |
| P4 : | | R(x,1) | | | R(x,2) | R(x,3) |

(f) Exécution 6 :

| | | | | | |
|---|---|---|---|---|---|
| P1 : | W(x,1) | | | | |
| P2 : | | R(x,1) | W(x,2) | | |
| P3 : | | | | R(x,1) | R(x,2) |
| P4 : | | | | R(x,2) | R(x,1) |

(g) Exécution 7 :

| | | | | |
|---|---|---|---|---|
| P1 : | W(x,1) | | | |
| P2 : | | W(x,2) | | |
| P3 : | | | R(x,1) | R(x,2) |
| P4 : | | | R(x,2) | R(x,1) |

5. Le calcul de Pi en parallèle

   Le programme suivant est un programme séquentiel qui calcule PI.

```
static long nbEtapes = 10000;
double etapes;
int main()
{
   int i;
   double x, pi, somme =0.0;
   etape = 1.0 / (double) nbEtapes;
   for(i=1; i < nbEtapes;i++)
   {
          x = (i-0.5) * etape;
         somme = somme + 4.0/(1.0 + x*x);
   }
   pi = etape * somme;
}
```

   Concevez une version parallèle et dites quelle est l'accélération obtenue. Utilisez la notation que vous désirez.

6. La machine distributrice

   Une machine distributrice distribue des consommations à des clients. Pour simplifier le problème, la machine ne distribue qu'une seule sorte de consommation à un prix unique. La machine contient un nombre fixe de consommations. Lorsqu'il n'y a plus de consommation, la machine signale la situation au responsable de l'inventaire qui se charge de venir remplir la machine. L'achat d'une consommation implique les actions suivantes :

   (a) le client insère l'argent dans la machine ;

   (b) le client demande sa consommation ou annule l'achat ;

   (c) la machine distribue la consommation ou retourne l'argent ;

   (d) la machine avertit le responsable de l'inventaire s'il ne reste plus de consommation.

   Vous devez écrire des programmes qui simulent la machine distributrice, les clients et le responsable. Modélisez la machine comme un processus serveur avec lequel les clients (des processus) communiquent à l'aide de **messages**. Le responsable est aussi un processus qui attend qu'une machine lui signale un manque de consommation pour se mettre à l'action.

7. Fiabilité et performance

   (a) Il y a deux types de reprise : la reprise avant et la reprise arrière. Classez les techniques suivantes selon le type de reprise :

      i. les blocs de reprise ;
      ii. code correcteur d'erreurs ;
      iii. code détecteur d'erreurs et actions conséquentes ;
      iv. les signaux de Unix et les Try/Catch de C++.

   (b) Pourquoi est-ce que l'exactitude (correct ou valide[1]) des différents composants d'un logiciel n'est-elle pas une condition nécessaire à la fiabilité ? Peut-elle quand même être utile ?

1. *Correctness* en anglais.

(c) Performance

Vous devez évaluer la performance d'un serveur dont vous avez la charge. Ce serveur sera dédié à fournir des services sur le Web. Ce service sera alimenté par une base de données. Pour faire votre évaluation, vous devez soumettre votre serveur à une charge extrême. Comme vous ne possédez pas les ressources pour générer des milliers de requêtes, vous décidez d'utiliser l'approche suivante :

— Vous installez et configurez la base de données choisie pour le site et vous générez des requêtes à un rythme continue. Comme le site n'est pas encore fonctionnel et que vous ne possédez pas un historique de fonctionnement (requêtes), vous générez des requêtes aléatoires faisant appel aux différentes fonctionnalités de la base de données.
— Pour simuler une surcharge, vous implantez un programme qui consomme artificiellement des ressources. Ce programme ne fait que des itérations pour consommer du temps du processeur et occuper les unités d'entrées/sorties (sans rien faire d'utile).

Quel type de charge de travail de test venez-vous de créer :

— une charge de tests réelle ;
— une charge de tests synthétique naturelle (banc d'essai) ;
— une charge de tests synthétique hybride ;
— une charge de tests artificielle (timings, mélange d'instructions, programmes synthétiques, programmes noyaux, ...).

Justifiez votre réponse.

8. Le parc jurassique contient un musée de dinosaures et un parc pour des expéditions. Dans le parc, il y a $m$ visiteurs et $n$ voitures d'expédition pouvant contenir un seul passager. Les visiteurs se promènent dans le musée pendant un certain temps puis se mettent en ligne pour se promener en voiture. Lorsqu'une voiture est disponible, elle prend un passager et part pour le tour du parc qui dure un temps non déterminé. Si les $n$ voitures sont occupées, les visiteurs attendent. Si une voiture est prête et qu'il n'y a aucun passager, elle attend. Implantez un programme qui utilise les moniteurs pour implanter ce système. Modélisez les voitures et les passagers comme des processus.

9. Les molécules d'eau

   Soit des atomes d'hydrogène et d'oxygène qui se promènent dans l'espace en essayant de se regrouper en molécules d'eau. Ce regroupement requiert que deux atomes d'hydrogène et un atome d'oxygène se synchronisent. Soit une simulation où les atomes d'hydrogène et les atomes d'oxygène sont représentés par des processus. Vous devez implanter à l'aide des sémaphores les fonctions «HReady» et «OReady» qui sont appelées respectivement par les atomes d'hydrogène et d'oxygène quand ils sont prêts à former une molécule d'eau.

10. (**Bonus**) Exclusion mutuelle (10 points)

Vous avez écrit le programme suivant en C++ qui démarre 10 fils tentant chacun d'incrémenter une variable globale «x».

```
#include <iostream>
#include <thread>
static const int max = 10;
int x = 0;

void pcs()
{
   if (x<3) { x=x+1;}
}

int main() {
    std::thread t[max];
    for (int i = 0; i < max; ++i) { t[i] = std::thread(pcs);}
    for (int i = 0; i < max; ++i) { t[i].join(); }
     std::cout << " x = " << x << std::endl;
    return 0;
}
```

Malheureusement, vous constatez à l'exécution que la valeur affichée de «x» dépasse parfois 3. Vous consultez vos collègues et deux d'entre eux vous proposent les solutions suivantes.

(a) Déclarer la variable «x» comme étant atomique [2]. Le programme devient :

```
#include <iostream>
#include <thread>
#include <atomic>
static const int max = 10;

std::atomic<int> x(0);

void pcs()
{
   if (x<3)  { x=x+1;}
}

int main() {
    std::thread t[max];
    for (int i = 0; i < max; ++i) { t[i] = std::thread(pcs);}
    for (int i = 0; i < max; ++i) { t[i].join(); }
    std::cout << " x = " << x << std::endl;
    return 0;
}
```

Cette solution est-elle fonctionnelle ?

2. Une variable atomique est protégée contre les conditions de course, i.e. deux opérations simultanées par deux fils distincts (lecture et écriture par exemple) sur cet objet auront un comportement bien défini. En gros cela signifie souvent ne pas mettre la variable en cache.

**Voir la solution (b) sur la page suivante $\longrightarrow$**

(b) Votre second collègue vous propose la solution suivante qui utilise un `mutex` :

```
#include <iostream>
#include <thread>
#include <atomic>
#include <mutex>
static const int max = 10;
std::mutex verrou;
std::atomic<int> x(0);

void pcs()
{
   verrou.lock();
   if (x<3)  {  x=x+1;
                verrou.unlock();
             }
}
int main() {
    std::thread t[max];
    for (int i = 0; i < max; ++i) { t[i] = std::thread(pcs); }
    for (int i = 0; i < max; ++i) { t[i].join(); }
    std::cout << " x = " << x << std::endl;
    return 0;
}
```

Cette solution fonctionne-t-elle ?

Fin de l'examen

Bonne chance !