CC - ASR7 Programmation Concurrente

—

Contrôle continu

Fabien Rico (et André Franquin)

5 avril 2024

Afin d'obtenir tous les points il vous est demandé de justifier vos résultats. Le barème proposé est susceptible d'être modifié lors de la correction. Il n'est présent que pour vous donner une idée du poids relatif des différentes questions.

Pour toutes les questions, « Implémentez la fonction ... » signifie donner le prototype (type des arguments et de la valeur de retour) et le corps de la fonction. Les détails syntaxiques (point-virgules, arguments exacts des templates, `#include`...) ne sont pas pris en compte dans l'évaluation, mais essayez d'utiliser une syntaxe aussi proche que possible du C++.

Rappels sur C++11 et les threads

Pour vous aider, voici un rappel de la syntaxe C++11 pour les threads :

```
1  // Création et attente de terminaison d'un thread :
2  int main () {
3          // ...
4          std::thread t(f, 42, std::ref(x));
5          // ...
6          t.join();
7  }
8
9  // Déclaration d'un mutex
10 std::mutex m;
11
12 // Verrouillage/déverrouillage d'un mutex :
13 m.lock();
14 // ...
15 m.unlock();
16
17 // Instantiation d'un verrou :
18 {
19         std::unique_lock<std::mutex> l(m);
20         // ...
21 }
22
23 // Opérations sur une variable de condition :
24 std::condition_variable c;
```

```
25      c.wait(l); // l de type verrou (std::unique_lock par exemple)
26      c.notify_one();
27      c.notify_all();
28
29      // Opérations sur une variable de condition :
30      std::condition_variable_any c;
31      c.wait(m); // m de type mutex
32      c.notify_one();
33      c.notify_all();
```

I Ordonnancement

I.1 Ordonnancement sous Linux (4 points)

Dans cet exercice, nous allons utiliser l'implantation POSIX 1003b sur Linux. Il existe 100 niveaux de priorité :

- Le niveau 0 est réservé à `SCHED_OTHER` et les niveaux de priorité 1 à 99 aux politiques `SCHED_FIFO` et `SCHED_RR`.
- Les tâches de priorité 99 sont les tâches de plus forte priorité.
- `SCHED_OTHER` est dédié à l'ordonnanceur temps partagé.
- Le quantum utilisé par la politique `SCHED_RR` est de deux unités de temps.
- Quand une tâche `SCHED_FIFO` ou `SCHED_RR` arrive alors qu'il y a déjà d'autres tâches de même priorité en attente, la nouvelle tâche est insérée en fin de liste.
- L'ordonnancement est préemptif.
- C et D partagent un mutex, `m`.

Soit le jeu de tâches apériodiques suivant :

Tâche	Date d'arrivée	Priorité	Durée	Politique	Remarque
A	0	0	5	SCHED_OTHER	
B	3	6	4	SCHED_FIFO	Après 2 unités de temps de calcul, exécute l'instruction `sleep(6)` qui attend, donc rend la main, pendant 6 unités de temps. La tâche effectue 2 unités de temps de calcul après ce `sleep(6)`
C	9	11	2	SCHED_FIFO	Verrouille le mutex m après 1 unité de temps, et le déverrouille 1 unité de temps plus tard (donc juste avant de terminer)
D	3	5	6	SCHED_RR	Verrouille le mutex m après 1 unité de temps (au début du temps 2), et le déverrouille 4 unités de temps de calcul plus tard (à la fin du temps 5)
E	4	5	4	SCHED_RR	
F	4	5	4	SCHED_RR	

Q.I.1) - (3 points) Dessinez l'ordonnancement généré par l'ordonnanceur (vous pouvez utiliser la feuille de réponse en fin de sujet, n'oubliez pas votre numéro d'anonymat).

Q.I.2) - Calculer le temps de réponse de ces tâches.

II La machine de Gaston pour boucher les bouteilles de vin

Vous devez implémenter une des inventions de Gaston Lagaffe : La machine à boucher les bouteilles de vin. Cette machine est composée de 3 parties :

— une chaine de bouteilles qui doivent être bouchées ;

— une chaine de bouchons ;

— un marteau qui s'abat à chaque fois qu'une bouteille et un bouchon sont en place.

Vous devrez donc lancer 3 threads, un pour chaque partie. La fonction de chaque thread est donnée ici :

```
1  void bouteilles(cMaBlBdV &m, int nb_bouteilles) {
2    int i;
3    for (i=0; i<nb_bouteilles; i++) {
4      m.placeBouteille();
5      cout << "Une bouteille \n";
6    }
7  }
8  void bouchons(cMaBlBdV &m, int nb_bouchons) {
9    int i;
10   for (i=0; i<nb_bouchons; i++) {
11     m.placeBouchon();
12     cout << "Un bouchon \n";
13   }
14 }
15 void marteau(cMaBlBdV &m) {
16   nb_bouchees = 0;
17   while(true) {
18     m.marteauAttend();
19     vlan(); // le marteau s'abat
20     nb_bouchees ++;
21     m.marteauTermine();
22   }
23 }
```

La structure de données `cMaBlBdV` est un moniteur qui s'assure que tout se déroule dans le bon ordre. Il doit faire en sorte que :

— Les fonctions `placeBouchon` et `placeBouteille` attendent qu'il y ait de la place avant de mettre l'objet en position.

— La fonction `marteauAttend` se bloque tant qu'il n'y a pas à la fois une bouteille et un bouchon.

— La fonction `marteauTermine` libère la place occupée par la bouteille et son bouchon.

II.1 Premier travail

Dans un premier temps vous ne vous occupez pas de la fin des threads (le thread marteau boucle indéfiniement).

Q.II.1) - Faire une fonction principale qui lance les 3 threads (le nombre de bouchons et de bouteilles est identique).

Q.II.2) - Faire le moniteur `cMaBlBdV`, c'est à dire :

2(a) - la déclaration, les initialisations et destructions de ressources,

2(b) - les 4 fonctions `placeBouchon`, `placeBouteille`, `marteauAttend` et `marteauTermine`.

La fonction `vlan()` actionne le marteau, son code n'est pas à fournir.

II.2 Fin des threads

Dans le code donné, vous pouvez remarquer que le thread `marteau` ne s'arrête jamais. Il faut changer cela.

Q.II.3) - Faites en sorte de modifier les codes pour que le thread `marteau` s'arrete. Attention, il ne doit pas oublier de bouteille ou de bouchon et il ne doit pas y avoir de risque qu'il se bloque indéfiniment.

Q.II.4) - Modifier les codes de manière à récupérer le nombre de bouteilles bouchées dans la fonction principale. Ce nombre est calculé par le thread `marteau` dans la variable `nb_bouchees`. La fonction principale doit attendre tous les threads, et afficher le nombre de bouteilles bouchés à la fin.

II.3 Gaston

Gaston Lagaffe a commis une erreur de conception pour sa machine. En effet, s'il s'approche trop près du marteau, il est assommé.

Supposez qu'il y a un thread `gaston` qui s'approche aléatoirement du marteau. C'est à dire qu'il modifie une variable globale `gaston_est_trop_pres`. Vous devez le protéger afin qu'il ne soit pas blessé.

Q.II.5) - Faites le code d'un autre thread : `fantasio` qui surveille cette variable et :

- lorsque la variable `gaston_est_trop_pres` est *vraie*, il bloque le marteau
- lorsqu la variable `gaston_est_trop_pres` est *fausse*, il libère le marteau.

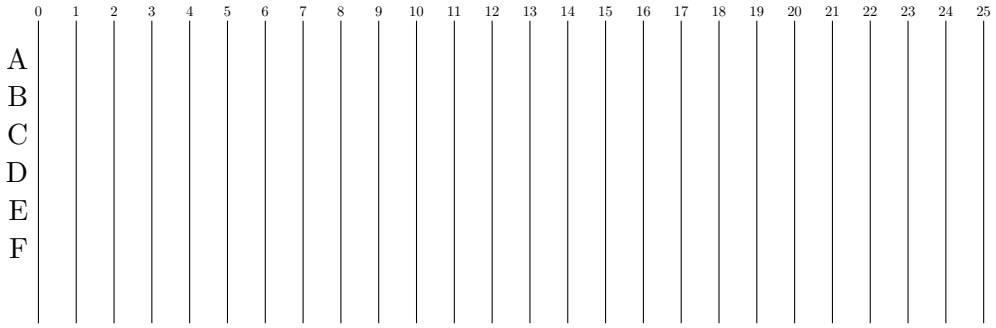
Attention pour cette partie, vous pouvez modifier les codes déjà donnés, mais comme vous n'avez pas accès à celui du thread `gaston`, il ne sera pas possible d'assurer la protection du garçon de bureau dans toutes les situations. Contentez-vous de faire un code qui bloque et débloque le marteau lorsque le thread `fantasio` lit la variable.

III Feuille de réponse

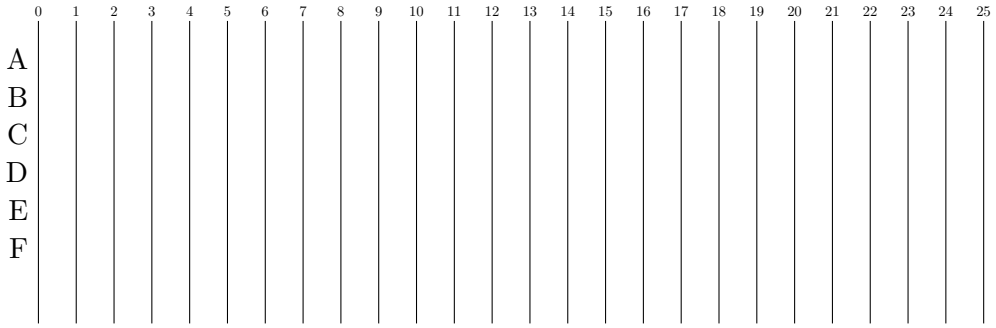
Numéro d'anonymat : _ _ _ _ _ _ _ _ _ _ _

Un seul des tableaux doit être rempli, les autres peuvent servir de brouillons ou de secours en cas d'erreur. **Un seul sera corrigé**, barrez distinctement les brouillons.

Brouillon :



Brouillon :



Question I.1 :