TP - ASR7 Programmation Concurrente

—

Contrôle continu final

Sylvain Brandel, Yves Caniou, Guillaume Damiand, Meriem Ghali,
Laurent Lefèvre, Thibaut Modrzyk, Grégoire Pichon, Alec Sadler,
Florence Zara, Jerry Lacmou Zeutouo

Printemps 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 (types 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 Préemptif Vs collaboratif (2 points)

Un de vos camarade dit que l'ordonnancement préemptif permet toujours de réduire la somme des temps de réponses par rapport à l'ordonnancement collaboratif.

Q.I.1) - (1 point) Donnez un exemple où l'ordonnancement préemptif permet effectivement de réduire la somme des temps de réponses de deux tâches.

En ordonnancement avec priorité : une tâche de durée 100 qui arrive à t=0, une plus prioritaire de durée 1 qui arrive à t=1. En collaboratif, la somme des temps de réponse est 100+100=200, en préemptif c'est 101 + 1 = 102.

Q.I.2) - (1 point) Donnez un contre exemple c'est à dire un cas où c'est le collaboratif qui donne la meilleure somme de temps de réponse.

Toujours en ordonnancement avec priorité : deux tâches de durée 100, une, la moins prioritaire, arrivée à t=0 et une à t=99. En collaboratif, la somme des temps de réponse est 100+101 = 201, et préemptif c'est 200+100=300.

I.2 Ordonnancement sous Linux (3 points)

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

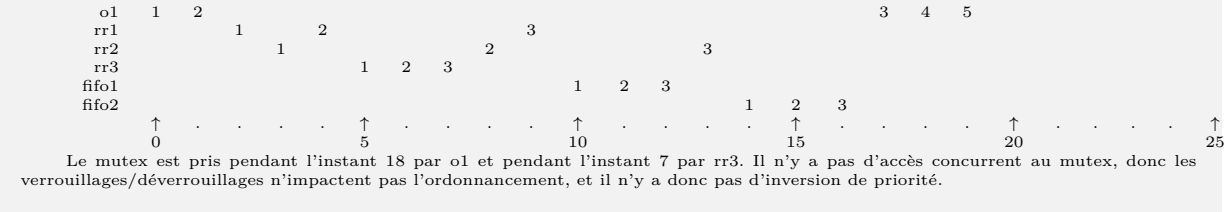
- Le niveau 0 est réservé à `SCHED_OTHER` et les niveaux de priorité 1 à 99 aux politiques `SCHED_FIFO` et `SCHED_RR`.
- Les tâches de priorité 99 sont les tâches de plus forte priorité.
- `SCHED_OTHER` est dédié à l'ordonnanceur temps partagé.
- Le quantum utilisé par la politique `SCHED_RR` est d'une unité 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.
- o1 et rr3 partagent un mutex.

Soit le jeu de tâches apériodiques suivant :

Tâche	Date d'arrivée	Priorité	Durée	Politique	Remarque
o1	0	0	5	SCHED_OTHER	Verrouille le mutex m après 3 unités de temps, et le déverrouille 1 unité de temps plus tard
rr1	2	3	3	SCHED_RR	
rr2	3	3	3	SCHED_RR	
rr3	5	4	3	SCHED_RR	Verrouille le mutex m après 2 unités de temps, et le déverrouille 1 unité de temps plus tard
fifo1	6	3	3	SCHED_FIFO	
fifo2	7	2	3	SCHED_FIFO	

Q.I.3) - (2 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.4) - (1 point) Y a-t-il une inversion de priorité ? Pourquoi ? Si oui, à quel moment ?



Le mutex est pris pendant l'instant 18 par o1 et pendant l'instant 7 par rr3. Il n'y a pas d'accès concurrent au mutex, donc les verrouillages/déverrouillages n'impactent pas l'ordonnancement, et il n'y a donc pas d'inversion de priorité.

II Programmation concurrente : la station de Vélo'v

Nous sommes en l'an 2073, les humains ont été remplacés par des robots contrôlés par un ordinateur central qui exécute un programme multi-thread qui contrôle toute la ville de Lyon et tous ses robots. Une chose n'a pas changée : la ville est toujours parsemée de stations de Vélo'v, vélos en libre-services, qui sont maintenant utilisés par des robots.

Le principe de la station est le suivant :

— La station dispose de `NB_PLACES` emplacements (numérotés de 0 à `NB_PLACES`-1). Un emplacement peut être vide, ou plein (c'est à dire contenant un Vélo'v).

— Un robot peut poser un vélo dans la station à l'aide de la fonction `poser_velov()`. Poser un Vélo'v transforme un emplacement vide en emplacement plein.

— Un robot peut prendre un vélo dans la station en appelant la fonction `prendre_velov()`.

II.1 Méthodes bloquantes (7 points)

Dans un premier temps, nous nous intéressons au cas « bloquant », ou quand un robot cherche à prendre ou à poser un Vélo'v alors que ce n'est pas encore possible (pas de place ou de Vélo'v disponible), le robot doit attendre puis réalise l'action.

Q.II.1) - (1 point) Écrire une classe (ou une structure si vous n'êtes pas assez à l'aise avec C++) `StationVelov` qui agira comme un moniteur de Hoare, et déclarer ses champs. Vous aurez au minimum besoin d'instancier un ou plusieurs objets de la bibliothèque de threads C++11, et d'un tableau de booléens pour représenter les emplacements disponibles.

Q.II.2) - (0,5 point) Écrire un constructeur `StationVelov(unsigned nb_velov_dispo)` qui initialise une station avec `nb_velov_dispo` Vélo'v disponibles (et donc `NB_PLACES - nb_velov_dispo` places libres).

Q.II.3) - (1,5 point) Écrire la méthode `int poser_velov()` de `StationVelov` (ou une fonction si vous préférez). Si aucun emplacement n'est disponible, alors la fonction attend qu'un emplacement se libère. Une fois un emplacement disponible, la fonction marque cet emplacement comme occupé et renvoie le numéro de l'emplacement.

Q.II.4) - (1,5 point) Écrire la méthode `int prendre_velov()`. Si aucun Vélo'v n'est disponible (c'est à dire si tous les emplacements sont libres), alors la méthode attend qu'un vélo soit posé. Ensuite, la méthode marque l'emplacement du vélo choisi comme libre et renvoie le numéro de cet emplacement.

Q.II.5) - (1 point) Écrire une fonction `robot()` qui sera exécutée par les threads représentant les robots. Cette fonction doit exécuter `NB_TOURS` (on suppose cette constante déjà définie) fois de suite les opérations `prendre_velov()` puis `poser_velov()`.

Q.II.6) - (1,5 point) Écrire une fonction `main()` qui instancie `NB_ROBOTS` robots et une station `StationVelov`. Faites en sorte que le programme termine proprement quand tous les robots ont terminé leur travail.

II.2 Méthodes non-bloquantes (3 points)

Comme chacun sait, attendre n'est pas forcément la meilleure stratégie quand une action n'est pas possible. Pour prendre un Vélo'v, une autre option quand il n'y en a pas dans la station est de chercher une station voisine qui en a. En appelant la méthode `prendre_velov()`, un robot prendrait donc le risque d'attendre des heures qu'un Vélo'v soit posé, alors qu'une station voisine en aurait à disposition. Nous allons donc écrire des variantes non-bloquantes de ces méthodes (analogues des méthodes `try_lock()` des mutex) : quand l'action est possible, elle est réalisée, mais quand l'action est impossible la méthode termine sans attendre et renvoie un code d'erreur.

On suppose que la classe `StationVelov` dispose d'une méthode `station_voisine()` qui renvoie une station voisine, et on suppose que plusieurs stations sont instanciées (vous n'avez pas à le faire). Pour simplifier le problème, on suppose que le robot essaye toujours de reposer le Vélo'v dans la station de départ.

Q.II.7) - (1 point) Écrire une méthode `essayer_poser_velov()`. Si aucune place n'est disponible, cette méthode renvoie -1. Sinon, la méthode se comporte comme `poser_velov()`.

Q.II.8) - (1 point) Écrire une méthode `essayer_prendre_velov()`. Si aucun Vélo'v n'est disponible, cette méthode renvoie -1. Sinon, la méthode se comporte comme `prendre_velov()`.

Q.II.9) - (1 point) Écrire une fonction `robot_v2()` similaire à `robot()` mais qui applique le principe suivant (toujours en boucle, répétée `NB_TOURS` fois) :

— Essayer de prendre un Vélo'v

— Si la prise de Vélo'v échoue, alors prendre (en attendant si besoin) un Vélo'v dans la station voisine.

— Essayer de reposer le Vélo'v dans la station de départ

— Si le dépôt échoue, alors poser le Vélo'v dans une station voisine (en attendant si nécessaire).

III Administration système

III.1 Installation d'Apache HTTP server (3 points)

Le but de cet exercice est d'installer et lancer le serveur web Apache, en version 2, sur une distribution Ubuntu.

Q.III.1) - (0,5 point) Quelle commande utiliser pour trouver le nom exact du paquet à utiliser avec les informations ci-dessus ?

apt search apache

Q.III.2) - (0,5 point) Quelle commande utiliser pour installer le paquet en question (qui s'appelle `apache2`) ?

apt install apache2

Q.III.3) - (0,5 point) L'installation terminée, vous lancez une commande qui vous répond :

```
* apache2.service - LSB: Apache2 web server
   Loaded: loaded (/etc/init.d/apache2; bad; vendor preset: enabled)
  Drop-In: /lib/systemd/system/apache2.service.d
           ‘-apache2-systemd.conf
   Active: inactive (dead) since Tue 2018-04-03 14:45:44 UTC; 5 days ago
     Docs: man:systemd-sysv-generator(8)
  Process: 6725 ExecStop=/etc/init.d/apache2 stop (code=exited, status=0/SUCCESS)
  Process: 6707 ExecStart=/etc/init.d/apache2 start (code=exited, status=0/SUCCESS)

Apr 03 14:45:44 machine apache2[6707]: (98)Address already in use: AH00072:
                                       make_sock: could not bind to address [::]:80
Apr 03 14:45:44 machine apache2[6707]: (98)Address already in use: AH00072:
                                       make_sock: could not bind to address 0.0.0.0:80
Apr 03 14:45:44 machine apache2[6707]: no listening sockets available, shutting down
Apr 03 14:45:44 machine apache2[6707]: AH00015: Unable to open logs
Apr 03 14:45:44 machine apache2[6707]: Action 'start' failed.
Apr 03 14:45:44 machine apache2[6707]: The Apache error log may have more information.
Apr 03 14:45:44 machine apache2[6707]:  *
Apr 03 14:45:44 machine apache2[6725]:  * Stopping Apache httpd web server apache2
Apr 03 14:45:44 machine apache2[6725]:  *
Apr 03 14:45:44 machine systemd[1]: Started LSB: Apache2 web server.
```

Quelle était cette commande ?

systemctl status apache2.service

Q.III.4) - (1 point) Que pouvez-vous en déduire ?

Le serveur n'est pas lancé (Active : inactive), à cause d'un problème de conflit sur le port 80 (Address already in use). Un autre serveur tourne probablement.

Q.III.5) - (0,5 point) Comment résoudre la situation ci-dessus ?

En désinstallant l'autre serveur web qui est vraissemblablement installé sur notre machine.

III.2 Mises à jour (1 point)

Q.III.6) - (0,5 point) Votre voisin de bureau tente de faire les mises à jour sur son serveur qui tourne sous Ubuntu. Il lance la commande `apt upgrade` et est surpris de ne voir aucune mise à jour à faire. Qu'a-t-il oublié ?

`apt update`, qui télécharge la liste des paquets disponibles dans la dernière version.

Q.III.7) - (0,5 point) Une fois l'oubli ci-dessus réparé, que va faire précisément la commande `apt upgrade` (citez au moins deux étapes) ?

Identifier les paquets qui ont besoin d'une mise à jour, télécharger les nouvelles versions de tous ces paquets, puis les installer (et accessoirement poser des questions à l'utilisateur).

III.3 Droits et utilisateurs (4 points)

Votre login Unix est `chaprot`. Vous travaillez dans un environnement multi-utilisateur où les utilisateurs (comme vous) n'ont pas les droits ni le mot de passe root. Vous allez voir votre administrateur système avec une question précise en tête. Votre administrateur entre les commandes suivantes :

```
addgroup tp-asr7
adduser lurong tp-asr7
adduser chaprot tp-asr7
```

Il vous demande ensuite d'exécuter sur votre compte (`chaprot`) :

```
mkdir tp-asr7
chgrp tp-asr7 tp-asr7
chmod g+rwx tp-asr7
```

Pour vous aider, voici un extrait des documentations des commandes :

```
$ adduser --help
adduser --group [--gid ID] GROUP
addgroup [--gid ID] GROUP
   Add a user group
```

```
addgroup --system [--gid ID] GROUP
   Add a system group

adduser USER GROUP
   Add an existing user to an existing group
$ chgrp --help
Usage: chgrp [OPTION]... GROUP FILE...
  or:  chgrp [OPTION]... --reference=RFILE FILE...
Change the group of each FILE to GROUP.
```

Dans la précipitation, vous avez fini par oublier la question que vous aviez posé à votre administrateur ...

Q.III.8) - (0,5 point) Quel était le but de la manœuvre ?

Q.III.9) - (1,5 point) Détaillez le rôle de chacune des 6 commandes.

Q.III.10) - (0,5 point) Que donne la commande `ls -ld tp-asr7` après l'exécution des commandes ? (`ls -d rep` liste le répertoire lui-même, pas son contenu)

Q.III.11) - (0,5 point) Sachant que vous êtes un utilisateur avancé d'Unix, pourquoi avez-vous eu besoin d'aller voir votre administrateur ?

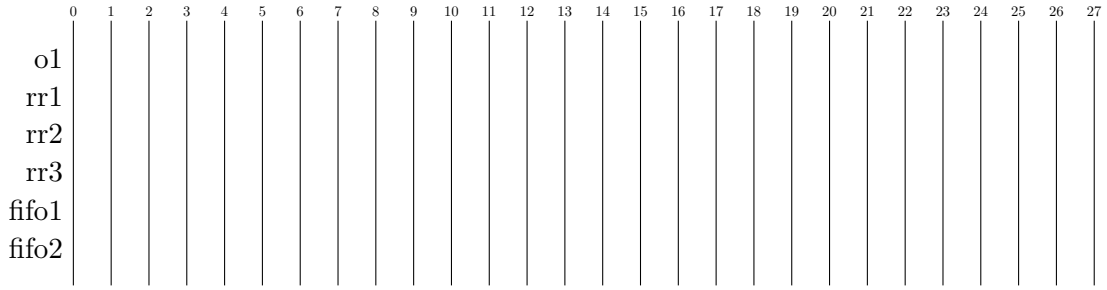
Q.III.12) - (1 point) Bonus (non-vu en cours) : vous retournez voir votre administrateur pour le remercier, mais vous ajoutez : « au fait, il manquait le `chmod g+s tp-asr7` ». Pourquoi ?

IV Feuille de réponse

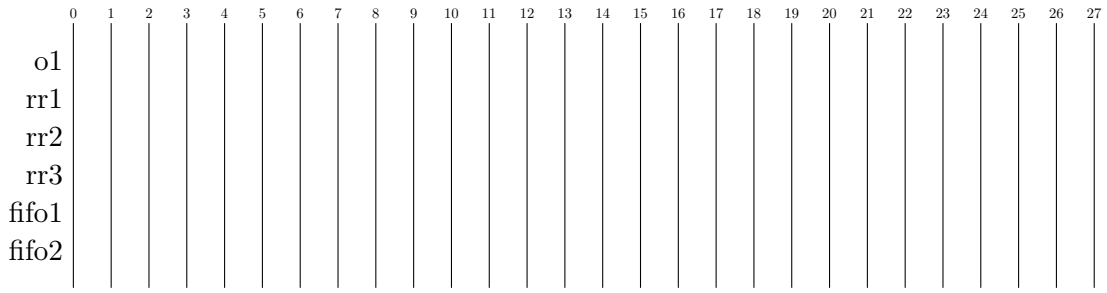
Numéro d'anonymat : _ _ _ _ _ _

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.3 :