

Annexe K

Travaux Pratiques 6

L'objectif de ce TP est de mettre en pratique les concepts présentés dans le chapitre Synchronisation à savoir, la notion de synchronisation entre activités.

Nous vous demandons de télécharger depuis le serveur SVN (voir chapitre 5) le projet TP6_Eleves et de nommer l'importation TP6 (dernier écran d'importation).

Le déroulement du TP est subdivisé en trois parties. La première consiste à valider les acquis du TP5. On vous demande de trouver un bug dans le code fourni en relation avec la gestion des activités.

La seconde partie se consacre à la manipulation puis l'implémentation de mécanismes de synchronisation. La dernière partie se focalisera sur la gestion des interblocages.

Question 1 - Durée estimée : 30 minutes

Avant d'exploiter la classe Thread introduit à la fin de la séance dernière, nous allons analyser à nouveau les techniques exploitées dans le cadre de l'ordonnancement des activités sur un mode preemptif. Deux activités sont créées lors de la question 1 et doivent afficher un message : 'thread 1 schedulé', 'thread 2 schedulé'. Les messages ne s'affichent pas, pourquoi ?

Question 2 - Durée estimée : 30 minutes

Dans cette question nous vous demandons de synchroniser un producteur/consommateur via une sémaphore. Un producteur produit des données à lire. Un consommateur lit les données. Dans notre cas, la donnée produite par notre consommateur est un simple caractère. Ce caractère est écrit dans un tableau à l'emplacement n (un tampon). La valeur de n est incrémentée par le producteur après chaque donnée produite. Le producteur ne produit que 70 données. Un consommateur ne peut lire que les données produites. Utilisez une seule sémaphore pour garantir ce comportement.

Le code du producteur et du consommateur sont dans le répertoire `Applications/ProdCons`. Le code de la Sémaphore dans `sextant/Synchronisation/Semaphore`.

Question 3 - Durée estimée 30 minutes

Nous voulons maintenant que le producteur puis produire une infinité de données. Pour cela nous allons implémenter un tampon circulaire. Arrivé à la 70 dixième donnée, le Producteur recommence à écrire en début de tableau. Cependant, il ne peut écrire dans une case du tableau que si la donnée présente à été lue par le consommateur. Proposez une solution sur papier à ce nouveau problème (ne pas l'implémenter), un deuxième sémaphore vous sera probablement nécessaire.

Question 4 - Durée estimée 30 minutes

Nous vous fournissons dans le répertoire **Application** une classe **Hello** affichant à l'écran une chaîne de caractère. Comme pour le TP4, nous souhaitons pouvoir afficher à l'écran la chaîne **Hello World**. Cependant cet affichage est produit par entrelacement des exécutions de deux activités. Au dernier TP, nous avons montré que système d'ordonnancement préemptif ne permettait plus de synchroniser nos activités.

Utilisez deux sémaphores (`sem1` et `sem2`) pour coordonner l'affichage des deux activités. Vous devez ici modifier le code de `Hello.cpp` et celui du `main`.

Ce que nous vous demandons d'implémenter s'appelle un rendezvous.

Question 6 - Durée estimée 15 minutes

Observez le fonctionnement du `spinlock`. Nous rappelons que le `spinlock` ou verrou tournant est un mécanisme simple de synchronisation basé sur l'attente active.

Pourquoi notre code boucle-t'il indéfiniment sur l'exécution d'une seule Activité ?. Rajouter le code nécessaire à la classe sémaphore permettant de garantir un fonctionnement correct. Indice : protégez l'affectation à la variable `value` dans les méthodes `P()` et `V()` en se servant de la variable de classe.

Question 7 - Durée estimée 45 minutes

Développez un `Mutex` dans le répertoire associé. Un `mutex` peut être vu comme une sémaphore spécialisée (héritage). En effet, un `mutex` est une sémaphore initialisée à 1. La classe `mutex` doit disposer de deux méthodes (au moins) `lock()` (prise du `mutex`) et `unlock()` (libération du `mutex`). Un `mutex` ne peut être libéré que par l'activité qui l'a pris.

Servez vous entre autres de la fonction `thread_get_current()`

Question 8 - Durée estimée 30 minutes

Comme vous pouvez le constater, les codes du driver clavier sont programmés sur la base d'une attente active. Revisitez ces codes pour ne plus utiliser d'attente active en se basant sur la notion de sémaphore.

Question 9 - Durée estimée 30 minutes

Comme constaté, lorsque au sein du main vous appelez `clavier->getchar()`, votre système provoque une erreur. Comment l'expliquez vous ? Indice : Attente passive bloquante, Ordonnanceur.

Question X - Durée estimée 60 minutes

Un salon de coiffure est composé d'une salle d'attente de n chaises et d'une salle où le coiffeur s'exécute. S'il n'y pas de client, le coiffeur va dormir sur sa chaise. Si un client arrive dans le salon de coiffure et que les n chaises de la salle d'attente sont occupés, il retourne chez lui. Si un client arrive alors que le coiffeur est occupé, le client va s'asseoir dans la salle d'attente. Et si le coiffeur dort lors de l'arrivée d'un client, ce client le réveille.

On vous demande de synchroniser un coiffeur et un certain nombre de clients à l'aide de sémaphores. On vous donne un pseudo code représentant l'activité du coiffeur et celui d'un client. Typiquement, le client et le coiffeur doivent se synchroniser pour arriver à l'état "coupe les cheveux" en même temps.

```
2 coiffeur :
3   repeat

5       "coupe_les_cheveux"

7   until true

9 Client :
10  repeat
11      Si compteur = n+1 then exit()
12      FinSi

14      compteur = compteur +1

16      "coupe_les_cheveux"

18      compteur=compteur-1

20 until true
```

Listing K.1 – Pseudo code du coiffeur

Proposez une solution sur papier validée par votre professeur avant de commencer l'implémentation.