Programmation Concurrente

Janvier 2020 – SI 4ème année

Durée : 2 heure **Documents manuscrits autorisés.**

Problème 1. Allocations de ressources

Plusieurs processus (P_0, P_1, .., Pp) se partagent des ressources communes dont le nombre total N est fixé. Chaque processus P_i connaît le nombre maximum N_i de ressources qu'il va demander. Les demandent se font progressivement à l'aide d'un appel à la procédure allouer(n), puis lorsque le processus n'a plus besoin des ressources, il les libère à l'aide de la procédure libérer(n). Un processus ne demande pas de nouvelles ressources une fois qu'il en a libéré certaines.

Bien évidemment, les ressources acquises par un processus sont indisponibles pour les autres jusqu'à ce qu'elles soient libérées (accès exclusif aux ressources). Si le nombre n de ressources demandées par un processus est supérieur au nombre de ressources disponibles, sa demande n'est pas satisfaite, aucune ressource ne lui est allouée et il se bloque. Il ne pourra continuer son exécution et recevra ses ressources uniquement lorsque l'allocation deviendra possible.

Les constantes sont :

- Nmax : nombre de ressources initialement disponibles

Les fonctions de l'Allocateur sont :

- void allouer(n) // alloue si c'est possible, n ressources au 'processus' qui les a demandé.
- void libérer(n) // libère n ressources.
- entier myId() // renvoie l'id du processus qui exécute la fonction.
- entier ressourceMax(entier id) // renvoie le nombre de ressources max demandé par le processus id.

Les fonctions usuelles liées à la synchronisation :

Sémaphore :

- Sémaphore S = créer Sémaphore(n) // crée un nouveau sémaphore S et l'initialise à n.
- void down(S) // pour le sémaphore S, décrémente de 1 la valeur du compteur et bloque un processus si les conditions sont remplies (cf. le cours)
- void down(S, n) // sémantiquement équivalent à l'exécution en section critique de n fois down(S). n est strictement positif.
- void up(S) // pour le sémaphore S, incrémente de 1 la valeur du compteur et libère un processus si les conditions sont remplies (cf. le cours)
- void up(S, n) // sémantiquement équivalent à l'exécution en section critique de n fois up(S). n est strictement positif.

Moniteur :

- Mutex m = créer Mutex() // crée un verrou réentrant
- Condition c = créer Condition() // crée une variable condition
- attendre(mutex, c) // relâche le verrou mutex et bloque le processus qui exécute la fonction sur la condition c.
- réveil(c) // réveille tous les processus bloqués sur la condition c. Ceux-ci commencent par reprendre le verrou qu'ils avaient libéré avant de poursuivre.

Les algorithmes/le code des fonctions seront données dans un pseudo langage qu'il vous appartient de choisir : proche du C, de Java – en Français, en Anglais, etc. Il est impératif que les algorithmes donnés soient facilement compréhensibles.

Rédigez vos réponses de manière concise.

Préliminaire – 2 points

Question 1. Ecrivez la fonction up(Sémaphore S, entier n) d'un Sémaphore en fonction de la fonction up(S). Si vous jugez que le code de la fonction up doit s'exécuter en section critique, vous appellerez la fonction 'début_section_critique()' pour initier la section critique et 'fin_section_critique()' pour la clore. Les fonctions début_section_critique() et fin_section_critique() sont réentrantes.

Question 2. Ecrivez la fonction down(Sémaphore S, entier n) d'un Sémaphore en fonction de la fonction down(S). Si vous jugez que le code de la fonction up doit s'exécuter en section critique, vous appellerez la fonction 'début_section_critique()' pour initier la section critique et 'fin_section_critique()' pour la clore. La section critique est gérée de manière réentrante.

Sémaphore – 4 points

Voici une proposition de mise en œuvre à l'aide de sémaphores

```
// variables globales partagées
1.    Semaphore mutex = créer Sémaphore(0) ;
2.    Semaphore attente = créer Sémaphore(0) ;
3.    Entier n_libre = N ;  // nombre de ressources disponible
4.    Entier n_att = 0;          // nombre de processus en attente

5.    Procedure demander(n) {
6.        down(mutex) ;
7.        tantque (n > n_libre) {
8.            n_att += 1 ;
9.            down(attente) ;
10.       }
11.   n_libre -= n ;
12.   up(mutex) ;
13.   }

14.   Procedure libérer(n) {
15.        down(mutex) ;
16.       n_libre += n ;
17.       tantque (n_att >0) {
18.            up(attente) ;
19.            n_att -= -1 ;
20.       }
21.       up(mutex) ;
22.   }
```

Question 3. Si vous jugez que le code ci-dessus comporte des erreurs, proposez les nécessaires modification.

Pour cela, je vous demande de remplacer uniquement les lignes qui sont à modifier. Par exemple, si e veux modifier la ligne 8, je mettrais :

```
8.    n_att += 1 ; n_libre -= n ;
```

Cela signifie que je remplace la ligne initiale, par celle-ci.

Moniteurs – 6 points

Question 4. Ecrivez à l'aide des moniteurs les fonctions allouer(n) et libérer(n) de l'Allocateur.

Les demandes faites à allouer(n) et libérer(n) sont supposées correctes (i.e. un processus ne demandera pas plus de ressource que ce qu'il a promis de demander et un processus ne libèrera pas plus de ressources que ce qu'il a acquis précédemment).

Question 5. Est-ce que votre implémentation présente un risque de famine ?
Si oui, expliquez pourquoi et proposez une solution sans famine
Si non, expliquez pourquoi.

Exercice de synchronisation – 4 points

Soient 3 processus exécutant chacun le programme suivant :

```
P1 : Tantque V faire    … ;    A1 ;    … ;    refaire
P2 : Tantque V faire    … ;    A2 ;    … ;    refaire
P3 : Tantque V faire    … ;    A3 ;    … ;    refaire
```

Compléter ces programmes en utilisant des sémaphores, pour réaliser les synchronisations demandées ci-dessous. On définira le plus petit nombre possible de sémaphores, et on précisera la valeur initiale de leur compteur. On pourra, si nécessaire introduire un ou des processus supplémentaires.

Question 6 : Les actions A*i* ne sont jamais simultanées, et se déroulent dans un ordre quelconque.

Question 7 : Les actions A*i* ne sont jamais simultanées, et se déroulent toujours en suivant l'ordre

```
A1 A2 A3 A1 A2 A3 A1 A2 A3 …
```

Question 8 : Les actions A*i* ne sont jamais simultanées, et se déroulent toujours en suivant l'ordre

```
A1 (A2 OU A3) A1 (A2 OU A3) A1 …
```

Question 9 : Les actions A*i* peuvent être simultanées, et se répètent toujours globalement – par exemple :

```
(A2 A1 A3) (A1 A3 A2) (A2 A3 A1) …
```

Question de cours – 4 points

Question 10 : Expliquez le fonctionnement et l'utilisation d'un spinlock

Question 11 : Expliquez le fonctionnement d'une barrière et son utilisation.