TP 5 Synchronisation

ABDULLAH HASIM Mohd Thaqif
NGUYEN Minh Trung

QUESTION 1

Oui . Au départ, la fonction de synchronisation ne fait rien donc on a pas de mutex pour donner un accès exclusif à la donnée. Donc, il est possible un rédacteur modifie la donnée lue par un lecteur qui est en train de dormir. Quand il a fini de faire « dodo », il vérifie que la donné lue a été modifié donc l'affichage « lecture incohérente » se passe.

QUESTION 2

- La solution proposée n'est pas suffisant pour la lecture. En verrouillant le mutex au début de la lecture et le déverrouille à la fin de la lecture ne permet pas d'une lecture en parallèle .

QUESTION 3

```
Mutex LR;
initialiser_lecteur_redacteur(Mutex M) { initialiser(M): }
debut_lecture(Mutex M, Mutex LR) {
        lock(LR) ;
        Lecteur++ ;
        Si Lect == 1
                lock(M);
        unlock(LR) ;
}
fin_lecture(Mutex M, Mutex LR) {
        lock(LR) ;
        Lecteur-- ;
        Si Lecteur == 0
                unlock(M);
        unlock(LR) ;
}
debut_ecriture(Mutex M) {lock(M);}
fin_ecriture(Mutex M) {unlock(M);}
```

- Mutex M sert à protéger l'accès à la donnée.
- Mutex LR sert à protéger le compteur Lecteur qui compte le nombre de la lecture.
- Cette solution permet plusieurs threads de lecteurs lire la donnée protégée en parallèle.

QUESTION 4

Les sémaphores utilisés dans le programme sont actuellement de type mutex car ils sont tous initialisés avec un seul jeton. L'intérêt d'utiliser un sémaphore est pour pouvoir libérer la donnée qui est bloquée par l'autre thread.

1. **Lecteurs prioritaires :**

3 sémaphores :

1. lecteur
2. rédacteur
3. donnée

- On utilise un sémaphore **lecteur** pour protéger le compteur **Lecteur**. Il est initialisé à 1.
- Sémaphore **donnée** pour protéger l'accès à la donnée (lire/écrire).
- Sémaphore **rédacteur** sert à donner une priorité à la lecture. On veux qu'à la fin de l'écriture, le rédacteur signale à une lecture en attente (s'il y a) . Ensuite, on signale un rédacteur en attente grâce au sémaphore **rédacteur**.

```
debut_lecture(){
        P(lecteur) ;
        Compteur_lecteur++ ;
        if(Compeur_lecteur == 1)
                P(donnee) ;
        V(lecteur) ;
}

fin_lecture(){
        P(lecture) ;
        Compteur_lecteur-- ;
        if(Compteur_lecteur == 0)
                V(donnee)
        V(lecture) ;
}

debut_redaction(){
        P(redacteur) ;
        P(donnee) ;
}

fin_redaction(){
        V(donnee) ;
        V(redacteur) ;
}
```

2. **<u>Rédacteurs prioritaires :</u>**

- On a 4 sémaphores.

1. debut_lecteur
2. lecteur
3. donnee
4. redacteur

- On modifie un peu le principe utilisé pour les lecteurs prioritaires. On indique la ressource du sémaphore **debut_lecteur** indisponible tant qu'il y a des rédacteurs arrivent pour modifier la donnée.
- Le dernière rédacteur signale à une lecteur qu'il peut commencer à lire (Compteur_redacteur == 0).

```
debut_lecture(){
        P(debut_lecteur) ;
        P(lecteur) ;
        Compteur_lecteur++ ;
        if(Compeur_lecteur == 1)
                P(donnee) ;
        V(lecteur) ;
        V(debut_lecteur) ;
}

fin_lecture(){
        P(lecture) ;
        Compteur_lecteur-- ;
        if(Compteur_lecteur == 0)
                V(donnee)
        V(lecture) ;
}

debut_redaction(){
        P(redacteur) ;
        Compteur_redacteur++ ;
        if(Compteur_redacteur == 1)
                P(debut_lecteur) ;
        P(redacteur) ;
        P(donnee) ;
}

fin_redaction(){
        V(donnee) ;
        P(redacteur) ;
        Compteur_redacteur-- ;
        if(Compteur_redacteur == 0)
                V(debut_lecteur) ;
        V(redacteur) ;
}
```

3. **<u>Ordre d'arriver :</u>**

- Notre programme peut donner une priorité selon l'ordre d'arriver par contre il ne peut pas donner aux lecteurs un accès parallèle à la donnée.

- Le principe :
  - Un thread qui arrive doit d’abord vérifier que la file d’attente n’est pas vide. Si elle est vide, le thread peut commencer à verrouiller le mutex. Si le mutex est bloqué, il doit attendre dans la file. Sinon, il peut commencer à modifier l’état de sémaphore .
  - En sortir du sémaphore, le thread doit signaler un thread dans la file d’attente s’il existe.