Leçon 21 : Problèmes et stratégies de cohérence et de synchronisation

I- Introduction

I-1) Motivation

Définition 1 : Un processus est un programme en cours d'exécution sur un ordinateur. Il dispose d'une zone mémoire en RAM (pour stocker la pile, les données de travail, etc.). Le système d'exploitation identifie les processus grâce à un numéro unique appelé PID

Définition 2 : Un fil d'exécution (ou thread) est un sous processus qui partage la mémoire avec les autres fils du processus.

Syntaxe 3 : En C, un programme peut lancer des fils d'exécution de type pthread_t et que l'on lance avec la fonction pthread_create dans la bibliothèque <pthread.h>

Exemple 4 : Somme d'un tableau parallélisé d'un tableau T de taille 2m

```
res = 0
```

F1 :

```
1  pour i allant de 0 à m-1
2      s = res + T[2i]
3      res = s
```

F2 :

```
1. pour i allant de 0 à m-1
2      s = res + T[2*i+1]
3      res = s
```

On peut donc aller deux fois plus vite.

Question : Que se passe-t-il sur l'exécution $F_1:2 \to F_2:2 \to F_1:3 \to F_2:3$?

I-2) Définitions générale

Définition 5 : On appelle appel bloquant un appel à une fonction qui attendra que les conditions soient réunies avant de terminer.

Exemple 6 : L'appel à recv en C attend que quelque chose arrive pour terminer.

Définition 7 : On dit qu'il y a absence de famine (ou vivacité) si aucun fil n'attendra indéfiniment.

Définition 8 : On dit qu'il y a équité quand tous les fils sont traités de la même manière.

II- Exclusion mutuelle et verrous

II-1) Définition

Définition 9 : Une section critique est un ensemble de morceau de code qui ne doit être exécuté que par un seul fil à la fois

Le problème de l'exclusion mutuelle est le problème consistant à garantir qu'on aura toujours au plus un fil dans la section critique.

Exemple 10 : Les lignes 2 et 3 de l'exemple 4 doivent être une section critique pour chaque fil.

Remarque 11 : Dans une section critique, les morceaux de code ne sont pas forcément les mêmes pour chaque fil (si un fil ne fait que lire quand l'autre ne fait que écrire dans une case partagée, leur code incompatibles n'est pas le même).

Solution 12 : Les verrous.

Définition 13 : Un verrou (ou mutex) est une structure de données permettant deux opérations :

- prendre() : appel bloquant qui demande l'accès au verrou
- rendre() : appel qui libère le verrou.

Propriété 14 : Une implémentation efficace des verrous doit avoir les propriétés suivantes :

1) Exclusion mutuelle : Un fil à la fois à accès au verrou
2) Absence de famine : Si un fil demande le verrou, il finira par l'obtenir
3) Équité : Aucun fil n'est favorisé.

II-2) Implementation des verrous pour 2 fils

Définition 15 : Une opération est dite atomique si elle ne peut pas être interrompue. Dans notre cas, une opération atomique correspond à une instruction en langage machine. On a notamment les opérations :

- lire une case mémoire
- écrire dans une case mémoire
- effectuer une opération arithmétique / logique

Remarque 16 : Attention ! Une opération dans un langage de programmation n'est pas atomique en général.

Sujet : 21

Propriété 17: Si deux fils écrivent la même case mémoire simultanément, la case mémoire contiendra soit la valeur du premier fils, soit celle du second. De même, si un fil lis dans la case quand un autre écrit, la valeur sera écrite et le premier fil lira la valeur précédente ou actualisée.

Algorithme 18: Algorithme de Peterson pour un verrou à deux fils

```
tour = -1                                   } variables globales
veut_entrer = [false, false]

rendre (i):
    veut_entrer[i] = true
    tour = 1 - i
    while (tour == 1-i && veut_entrer[1-i]) {}

prendre (i):
    veut_entrer[i] = false
```

Développement 1: Présentation de l'algorithme de Peterson et preuve de fonctionnement.

II-3) Généralisation à n fils

Algorithme 19: Boulangerie de Lamport

```
prendre (i):
    Acq[i] = 1;                                   } Obtenir un numéro
    int t = 0;
    for (int j = 0; j < n; j++)
        t = MAX(t, num[j]);
    num[i] = t+1
    Acq[i] = 0
    for (int j = 0; j < n; j++){                  } Attendre son tour pour prendre le verrou.
        while (Acq[j] == 1);
        while (num[j] > 0 && (num[j] < num[i]
            || (num[j] == num[i] && j < i)));
    }

rendre (i):        num[i] = 0
```

Analogie 20: On prend un ticket dans une boulangerie, et on attend que ce soit notre tour.

Remarque 21: L'attente ici (II-2 et II-3) est active (on fait des tours quand on attend)

Syntaxe 22: En C, les mutex sont disponibles dans la bibliothèque pthread
type : pthread_mutex_t initialisation : pthread_mutex_init
prendre : pthread_mutex_lock rendre : pthread_mutex_unlock

III - Synchronisation et sémaphores

Syntaxe 23: Pour synchroniser des fils, la commande pthread_join de la bibliothèque pthread.h permet à un fil d'attendre la fin et de récupérer la valeur de retour d'un des fil fils.

Remarque 24: Néanmoins, on peut souhaiter plus de synchronisation entre les fils.

III-1) Sémaphore et synchronisation

Analogie 25: Tableau des clés d'un Hotel.

Définition 26: Un sémaphore est un compteur qui propose les opérations suivantes:
- Initialiser à une valeur entière
- décrémenter : appel bloquant, décrémente le compteur si il est positif; attend qu'il le soit sinon
- incrémenter : augmente le compteur. Si il devient positif cela libère un fil si un attendait

Application 27: Limiter l'accès à n ressources

Remarque 28: Un sémaphore initialisé à 1 implémente un verrou.

Syntaxe 29: En C, les sémaphores sont disponibles dans la bibliothèque <semaphore.h>
type : sem_t initialisation : sem_init
décrémenter : sem_wait incrémenter : sem_post

Propriété 30: Un sémaphore est implémenté sans attente active (normalement).

Problème 31: Problème du rendez-vous
On a n fils qui doivent se synchroniser. Chacun travaillent en deux phases. Dans la première phase, tous les fils sont indépendants et peuvent s'exécuter

simultanément. La deuxième phase d'un fil ne peut débuter que si tous les fils ont terminé la première.

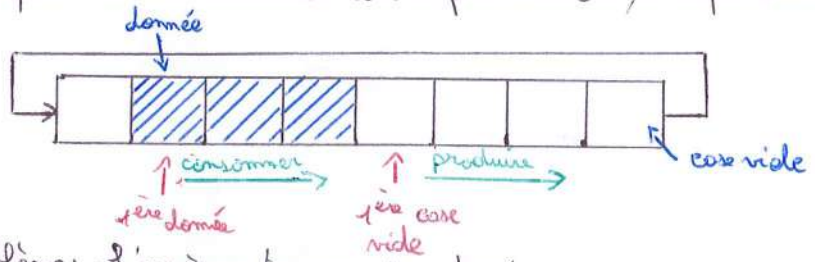
Développement 1 : Problème du Rendez-vous

III-2) Cas classique d'utilisation

Problème 32 : Producteur / consommateur

On dispose d'un tampon de taille N, et on a deux types de fils :
- des producteurs qui produisent des ressources et les stockent dans le tampon
- des consommateurs qui consomment les données produites (consommer une donnée libère son emplacement)

Pour que la donnée consommée soit la plus vieille, le tampon est circulaire



Problème : - L'accès au tampon est partagé
- les consommateurs doivent attendre qu'une donnée soit produite
- les producteurs doivent attendre que de la place soit libérée dans le tampon.

Solution :

```
sem_t verrou, vide, plein;
sem_init(&verrou, 0, 1);
sem_init(&vide, 0, N);
sem_init(&plein, 0, 0);
```

Producteur :

```
int item;
while(1){
    item = produire_item();
    // On attend une place
    sem_wait(&vide);
    sem_wait(&verrou);
    inserer_item(item);
    // on la remplie
    sem_post(&plein);
    sem_post(&verrou);
}
```

Consommateur :

```
int item;
while(1){
    // On attend une place
    sem_wait(&plein);
    sem_wait(&verrou);
    item = enlever_item();
    // On libère une place
    sem_post(&vide);
    sem_post(&mutex);
    consommer_item();
}
```

Remarque 33 : On peut implémenter un sémaphore par de l'attente active et des verrous. Donc naturellement, ce qu'apporte souvent les sémaphores, par rapport aux verrous, c'est moins d'attente active.

IV - Interblocage

Définition 34 : On dit qu'il y a interblocage quand tous les fils sont bloqués et ne seront jamais libérés

Exemple 35 :

t1 :	t2 :
prendre(m1)	prendre(m2)
prendre(m2)	prendre(m1)

Problème 36 : Dîner des philosophes

On a cinq philosophes autour d'une table ronde et une fourchette entre chaque. Chaque philosophe alterne moment de faim et de réflexion. Quand il a faim, il attend de prendre ses deux fourchettes, puis il mange, les repose, etc.

Solution naïve : Un verrou par fourchette, chaque philosophe attend de prendre sa fourchette gauche, puis sa droite.

Problème : Interblocage

TD 37 : L'implémenter pour observer cet interblocage.

Solution :

1) Avec des verrous : chaque philosophe essaye d'abord de prendre sa fourchette de plus petit indice

2) Avec des sémaphores : On n'autorise seulement 4 philosophes à essayer de prendre des fourchettes. Le sémaphore indique le nombre restant de philosophes qui peuvent essayer (initialisé à 4 donc) et quand ils ont fini de manger, ils incrémentent le sémaphore.