Gestion de concurrence et exclusion mutuelle

Gestion de la concurrence - Sémaphore

Théorie des Sémaphore

Les sémaphore est un théorème mathématique.
Supposons que nous avons un pool de N ressources (N imprimantes, N fichiers, N variables) identique et interchangeable. De plus, des processus souhaite accéder à ces ressources.
Un sémaphore permet de protéger un pool de ressource de la concurrence d'accès de deux tâches.
Un sémaphore est une variable, qui contient comme données le nombre de ressources du pool disponible à l'instant t.
Tout sémaphore est accompagné de trois opérande, l'addition, la soustraction et l'initialisation.

- Initialisation
  $S = n$ tel que $n \in [0; N]$
- Soustraction (sub)
  SUB(p) : $-p$, demande à prendre p ressources de mon sémaphore.
  Si $S >= p$ alors
  $S = S - p$
  SINON
  mise en attente de la tâche
- Addition
  ADD(q) : $+q$, je libère q ressources
  $S = S + q$
  réveiller la 1er tâche en attente pouvant être satisfait.

Les opérations d'addition et de soustraction sont ATOMIQUE.

Exemple d'algorithme :
Tâche A

```
SUB(2)
// Accès à deux ressources
ADD(2)
```

Tâche B

```
SUB(3)
// Accès à trois ressources
ADD(3)
```

Programmation des Sémaphore en C

Les sémaphore avec les threads.
Lorsque une opération est faite elle ne se fait que de un en un. Récupération d'une ressource ou libération d'une ressource.

- Déclaration
  `sem_t mySem;`
- Initialisation
  `sem_init(&mySem, 0, n);` où n est la valeur initiale
- Soustraction
  `sem_wait(&mySem);` => -1 sur le sémaphore ou si sémaphore nul alors attente du thread.
- Addition
  `sem_post(&mySem);` => +1 sur le sémaphore et éventuellement réveille du me premier thread en attente.

Exclusion mutuelle - Mutex

C'est par exemple, l'accès simultané par deux thread à une donnée commune.
Pour palier au problème de manipulation d'une variable on vas utiliser un type de variable : `pthread_mutex_t mymutex = PTHREAD_MUTEX_INITIALIZER` (vérifier dans la doc pour la sémantique)
Pour protéger une instruction de l'accès simultané, il faut l'entourer des fonction de blocage:

```
pthread_mutex_lock(&mymutex);
// Instructions protégées
pthread_mutex_unlock(&mymutex);
```

Si la variable mutex est déjà bloqué alors la fonction `pthread_mutex_lock` est bloquant, donc met en attente le thread.

> **Remarque**
>
> Utilisez le mécanisme de Mutex seulement lors de problème de concurrence sur un petit groupe d'instructions.

Application sur serveur multithread à modèle statique

Un serveur de type multithreading à modèle statique est composer d'une temporisation dans le worker tant que celui-ci n'est pas activé. Actuellement l'attente est faire à l'aide d'une fonction bloquante pendant x microseconde. Ce type d'attente est dit active.

Une attente dit active, prend du temps machine, et le temps de réaction n'est pas instantané.
Cette attente active peut être remplacer par l'utilisation d'une attente sur sémaphore. Pour ne pas utiliser la machine.