**Ressources partagées**

* Une ressource est une entité utilisée par une tâche (périphérique, variable, buffer, structure de données)
* Une ressource partagée est une entité utilisée par plusieurs tâches
* Certaines ressources peuvent être corrompues s’ils sont utilisés par plusieurs tâches simultanément
* Il faut obtenir l'accès exclusif à la ressource avant de l'utiliser. Si l’accès exclusif n'est pas obtenu, l’ordre d’exécution des tâches pourrait altérer les résultats

**Les mécanismes d’exclusion mutuelle**

* L’exclusion mutuelle est le mécanisme d’accès exclusif à une ressource partagée ou à une section de code critique. Les méthodes pour assurer l’exclusion mutuelle sont :
* désactiver/activer les interruptions :
* affecte le temps de réponse du système
* seule solution lors de variables partagées avec une ISR
* désactiver/activer l’ordonnanceur :
* inefficace lors de variables partagées avec une ISR
* le système a alors un comportement non préemptif
* opération de TestAndSet:
* test atomique un variable global à 1 ou 0
* si le test est software il faut désactiver les interruptions
* sinon le test doit être implémenté matériellement
* utiliser les mutex et les sémaphores:
* les mutex et sémaphores binaires : 0 ou 1
* les sémaphores compteurs : 0 à max (255, 65535, …)

**Sections critiques**

Une section de code critique est une région de code indivisible. Elle doit être exécutée sans interruptions.

**Sémaphores**

Un sémaphore est une variable protégée (ou un type de donnée abstrait) et constitue la méthode utilisée couramment pour restreindre l'accès à des ressources partagées dans un environnement de programmation concurrente.

Il a été inventé par Edsger Dijkstra et utilisé pour la première fois dans le système d'exploitation THEOS.

Les sémaphores fournissent la solution la plus courante pour les problèmes ressources partagées mais il faut d'éviter les interblocages (deadlocks).

**Mutexes**

Un mutex est similaire au sémaphore booléen dont le jeton, une fois pris, ne peut être relâché que par celui qui l'avait pris !

Une tâche qui prend deux fois le même mutex (sans l'avoir relâché) ne sera pas bloquée.

Une tâche prend le mutex, utilise la ressource partagée et libère le mutex. Quand un mutex est pris, aucune autre tâche ne peut prendre ce mutex.

Ceci garantit qu’une seule tâche à la fois peut accéder à la ressource partagée.

**Les Sémaphores binaires**

Un sémaphore peut être utilisé par une tâche pour envoyer un signal à une autre tâche.

Il peut être pris par une tâche et libéré par une autre. Deux opérations sont possibles sur les sémaphores :

► prendre Ressource, P « Proberen »

► rendre Ressource, V « Verhogen »

Un sémaphore dispose d'une variable compteur, valant 0 à la création du sémaphore, qui sera modifiée par les opérations P et V. Les structures des deux fonctions sont les suivantes :

P :

* compteur = compteur - 1;
* si compteur < 0 alors bloquer la tâche active;

V :

* compteur = compteur + 1;
* si compteur <= 0 alors débloquer la tâche la plus prioritaire;

Quand une tâche souhaite accéder à une ressource elle appelle la fonction P. Si aucune autre tâche n'a pris le contrôle de la ressource le compteur du sémaphore passe à -1 et la tâche utilise la ressource. Dans le cas contraire, le compteur du sémaphore est décrémenté et la tâche se bloque jusqu'à ce que la ressource soit libre.

Lorsque la tâche a terminé d'utiliser la ressource elle doit appeler la fonction V de libérer la ressource et débloquer une autre tâche. Il existe un danger lors de l'utilisation des sémaphores d'exclusion mutuelle.

Ce danger est décrit dans l'exemple suivant :

La tâche 1 prend une ressource et se met en attente d'un événement provenant de la tâche 2.

Lorsqu'elle prend la main, la tâche 2 attend la libération de la ressource et se bloque sur le sémaphore mutex. Elle ne pourra donc jamais envoyer l'événement à la tâche 1.

Ce phénomène s'appelle interblocage (deadlock)

**Signalisation par sémaphore**

Les sémaphores permettant l'accès aux ressources partagés peuvent également être vus comme un moyen de signalisation entre deux tâches.

Ainsi, pour signaler à une tâche la présence d'une donnée valide, il est possible d’effectuer l'opération V sur un sémaphore associé à cette donnée. La tâche destinataire effectuera, lorsqu'elle le souhaitera, l'opération P sur ce même sémaphore. Elle pourra alors utiliser la donnée.

Si l'ordre des opérations est inversé, par exemple si la tâche destinataire est prête avant la tâche émettrice, il n'y a aucun problème : elle se bloquera sur le sémaphore jusqu'à ce que la tâche émettrice effectue l'opération V. Outre la signalisation d'événement d'une tâche à une autre, un sémaphore peut être utilisé, dans une certaine limite, comme un moyen de rendez-vous.

Dans ce cas, il faut que la tâche effectuant l'opération P sur un sémaphore soit exécutée en premier.

Elle se bloque alors sur le sémaphore jusqu'à ce qu'une autre tâche effectue le V. Les deux tâches pourront alors reprendre chacune leur exécution.

**spin lock**

Un cas simple de protection d’un ressource : par scrutation de flag / variable

Peut être fait :

* avec une opération de test et de MAJ du flag indivisible
* avec un simple masquage des interruption