Synchronisation à l'aide des sémaphores et des moniteurs

Samia Bouzefrane
Laboratoire CEDRIC
Conservatoire National des Arts et Métiers
http://cedric.cnam.fr/~bouzefra

Exercice 1 : RDV à n processus

Soient N processus parallèles ayant un point de rendez-vous. Un processus arrivant au point de rendez-vous se met en attente s'il existe au moins un autre processus qui n'y est pas arrivé. Le dernier arrivé réveillera les processus bloqués. La solution suivante résoud ce problème en utilisant des sémaphores.

```
Contexte commun:
        sémaphore mutex = 1, s= 0;
        entier NbArrivés = 0;  /* nbre de processus arrivés au rendez-vous */
Procédure RDV
Début
P(mutex);
    NbArrivés = NbArrivés + 1;
    Si (NBArrivés < N) Alors  /* non tous arrivés */
        V(mutex);      /* on libère mutex et */
        P(s);          /* on se bloque */
    Sinon
        V(mutex);      /* le dernier arrivé libère mutex et */
        Pour i = 1 à N-1 Faire V(s); /* réveille les N-1 bloqués, dans l'ordre d'arrivée */
    Finsi
Fin
```

Question : Traduire cet algorithme en C ensuite en Java en utilisant les sémaphores.

Exercice 2 :

Nous nous intéressons au problème d'allocation de ressources banalisées. Nous supposons disposer dans un système de plusieurs exemplaires d'une même ressource, dont un nombre quelconque peut être demandé par un processus à un moment donné. Pour simplifier, nous considérons un seul type de ressource.

Tout processus actif qui demande des exemplaires de cette ressource, doit solliciter les services d'un allocateur via deux procédures :

- *request*(m) : pour demander l'allocation de *m* exemplaires de la ressource
- *release*(m) : pour libérer les *m* exemplaires de la ressource

L'allocateur maintient une variable critique qui compte le nombre d'exemplaires disponibles afin de satisfaire éventuellement de nouvelles demandes.

Contexte commun : entier NbRessDisponibles :=Max ;

Comportement d'un processus :

```
While (true) {
    request(m) ;
    utiliser les m ressources
    release(m) ;
```

```
        }

Procédure request (entier m) {
        Tant que (NbRessDisponibles<m) bloquer le processus appelant ;
        NbRessDisponibles = NbRessDisponibles –m ;
        // allocation des m ressources
}

Procédure release(entier m) {
        // libération des m ressources
        NbRessDisponibles = NbRessDisponibles +m ;
        Réveiller des processus qui seraient bloqués
}
```

Notons que dans cette solution, plusieurs processus demandeurs peuvent être servis simultanément s'il y a suffisamment de ressources. De plus, la procédure `release` doit réveiller tous les processus en attente pour trouver ceux pour lesquels les demandes peuvent être satisfaites.

Question

Traduire cet algorithme en C ensuite en Java en utilisant les moniteurs.

Notons que sous Unix, il existe une primitive qui permet de réveiller tous les processus à la fois en attente derrière une condition :

```
int pthread_cond_broadcast(pthread_cond_t *condition);
```

Le programme obtenu doit être compilé avec l'option `-lpthread`. `signalAll`() est la méthode équivalente en Java.