TD3 : Utilisation des sémaphores

1 Paralléliser un programme

Nous souhaitons améliorer le temps de réponse du programme ci-dessous en parallélisant son exécution. Nous allons considérer, dans un premier temps, qu'il n'y a pas d'effet de bord entre les quatre fonctions utilisées.

```
begin
    x := bar(fip(init(10)));
    y := gnu(fip(init(20)));
    foo(x * y)
end
```

Pour exprimer l'exécution en parallèle, nous allons utiliser la notation

cobegin `I1; I2; ...; In` **coend**

pour indiquer que les instructions $\texttt{I}_j$ sont indépendantes et peuvent s'exécuter en parallèle. L'instruction **cobegin**/**coend** se termine quand toutes les instructions qui la composent sont terminées.

a) Proposez une version qui va réduire le temps de réponse sans utiliser de sémaphore ni de verrou mais seulement les **cobegin**/**coend**.

b) Si la fonction `fip` utilise un espace global unique, comment garantir le bon fonctionnement de notre programme ? Proposez une nouvelle version basée sur un sémaphore d'exclusion mutuelle (valeur initiale du compteur à 1).

c) Nous avons réglé le problème de `fip` mais nous ne maîtrisons pas l'ordre d'exécution de `fip(init(10))` et `fip(init(20))`. Si le deuxième est beaucoup plus long que le premier, il serait souhaitable de commencer par le premier. Pour ce faire nous allons construire une solution basée sur un sémaphore de droit de passage avec un compteur initialisé à zéro. Cette solution aura la structure suivante :

```
begin
    passage := nouveau sémaphore( 0 );
    co-begin
        begin  /* À FAIRE : calculer x */  end
        begin  /* À FAIRE : calculer y */  end
    co-end
    /* À FINIR */
end
```

d) En ajoutant un deuxième sémaphore de passage, comment faire pour que l'appel de `foo` soit également placé dans une structure **cobegin**/**coend** ?

e) Nous nous apercevons que très souvent les fonctions `gnu` et `bar` renvoient zéro. Comment en tenir compte pour améliorer notre programme ?

2 Synchronisation de l'accès aux ressources

On dispose de n ressources d'une même classe. L'état de ces ressources est donné par le tableau :

```
libre : tableau [ 1 .. n ] de booléens
```

On vous demande d'écrire les trois sections de code ⟨init⟩, ⟨allocation⟩ et ⟨libération⟩ qui respectent les points suivants :

- au début les ressources sont toutes libres,
- le code ⟨allocation⟩ bloque le demandeur si le nombre de ressources libres est égal à zéro ; ou choisit une des ressources libres dans les autres cas ;
- le code ⟨libération⟩ réveille les demandeurs éventuellement endormis.

r := ⟨allocation d'une ressource⟩
⟨utilisation de la ressource de n° r⟩
⟨libération de la ressource de n° r⟩

Bien entendu, il est fortement conseillé d'utiliser un ou plusieurs sémaphores (à compteur) pour programmer cette synchronisation.