Licence d'Informatique, IUP 3

Devoir de systèmes d'exploitation

À faire en binôme.
Date de remise : lundi 6 décembre 1999.

Feux tricolores

Le but de ce devoir est d'implanter des feux de signalisation tricolores comme des processus autonomes synchronisés.

Dans le document que vous rendrez, vous pourrez rédiger les programmes répondant aux questions dans le langage de votre choix, y compris dans un pseudo-langage tel que celui utilisé en TD. Vous y justifierez l'utilisation des différents sémaphores et variables. Vous y présenterez également les jeux de tests et leurs résultats, que vous commenterez. Les informations concernant le codage sont données à la fin de ce sujet.

Dans toutes les questions, les feux sont supposés être des feux tricolores français usuels, passant du vert au orange, du orange au rouge, puis de nouveau au vert, et ainsi de suite. On négligera les temps de calcul par rapport aux temps d'attente des processus (attente d'un sémaphore, relâche explicite du CPU pour un temps donné, etc.), ce qui est parfaitement réaliste. On ne cherchera pas à utiliser une horloge globale à tous les feux (ce qui est difficilement réalisable sans un système temps-réel) : on ne manipulera le temps qu'à travers la fonction permettant de mettre un processus en sommeil un temps donné.

1. Carrefour simple. On considère dans cette première question la situation la plus simple d'un croisement de deux routes, où les feux fonctionnent de manière symétrique. On impose pour ces feux les conditions suivantes :

- *Sécurité :* Si l'un des deux feux n'est pas rouge, alors l'autre feu est rouge depuis au moins S secondes. Un feu reste orange au moins O secondes.
- *Cohérence :* Si un feu passe du vert au rouge — par l'orange — alors il ne repasse pas au vert avant que l'autre ne soit passé au moins une fois au vert.
- *Progression :* Chaque feu reste au vert au moins V secondes. Un feu ne reste pas rouge plus de R secondes. On suppose $R > V + O + S$.

Justifier la condition portant sur R. Écrire les programmes correspondant à ces deux feux. Si la valeur de R n'intervient pas dans votre code, justifiez-le.

On pourra prendre pour les tests les valeurs $S = 3$ s, $O = 5$ s, $V = 20$ s et $R = 60$ s.

2. Déclenchement. On considère maintenant une situation différente, où une route à grand trafic est intersectée par une route où le trafic est bien plus faible. Sur cette dernière, un mécanisme enterré permet de détecter la présence d'une voiture arrêtée au feu. On souhaite que le feu soit vert autant que possible sur la route à grand trafic. On conserve les conditions *Sécurité* et *Cohérence* mais on modifie la condition *Progression* de la façon suivante :

- *Progression :* Chaque feu reste au vert au moins V secondes. Le feu de la grande route ne reste pas rouge plus de R secondes. Un véhicule arrêté au rouge sur la petite route n'attend pas le feu vert plus de R secondes. On suppose $R \geq V + O + S + 1$.

Écrire les programmes correspondant aux feux de la grande route et de la petite route. Justifiez la manière dont vous garantissez de la condition de progression.

Indication : On pourra utiliser une variable globale pour représenter la présence d'une voiture au dessus du détecteur, et simuler l'arrivée des véhicules à cet endroit par un troisième processus (interrogeant le clavier, ou introduisant des déclenchements de façon aléatoire, par exemple).

3. Avenue. On considère enfin le cas d'une grande avenue, intersectée par N rues transversales. Chaque carrefour est semblable à celui de la première question, et doit vérifier les conditions *Sécurité* et *Cohérence*. En revanche, afin d'améliorer le trafic sur l'avenue, on impose que jamais plus de K rues transversales aient le feu vert (ou orange) simultanément.

Écrire les programmes correspondant aux $2N$ feux. On n'impose pas ici de condition de progression. On fera l'hypothèse que l'implantation des sémaphores garantie que tout processus en attente d'un sémaphore finit toujours par l'obtenir.

Pour les jeux de test, on pourra prendre $N = 5$ et $K = 2$.

Codage

Les processus représentant les feux seront représentés par des *threads* (processus légers). Ces threads pourront partager des informations globales (des sémaphores, bien sûr, mais également d'autres variables globales). Le choix du langage de programmation vous est laissé. Toutefois, il vous est fortement conseillé d'utiliser Java ou Ocaml, pour lesquels nous vous fournissons une implantation des sémaphores.

Java

On crée un thread en invoquant la méthode *start* d'un objet d'une classe héritant de la classe pré-définie *Thread*. Cette méthode *start* appelle la méthode *run*, qui par défaut ne fait rien, et que vous redéfinirez pour y mettre le code de votre processus. Une utilisation typique est donc la suivante :

```
public class MonProcessus extends Thread {
    public void run() {
        < son code ici >
    }
```

avec dans le programme principal :

```
public class MonProgramme {
    public static void main (String[] args) {
        ...
        new MonProcessus().start();
        ...
    }
}
```

La classe *Thread* contient une méthode *sleep* permettant de mettre le thread en sommeil un temps donné. Elle prend en argument un nombre entier de milli-secondes.

La classe *Semaphore* qui vous est fournie a la spécification suivante :

Constructeur	*Semaphore*(int *n*)
Méthodes	void *P*() void *V*()

Informations générales sur Java et ses threads : http ://java.sun.com/docs/books/tutorial/

Ocaml

On crée un thread par (*Thread.create f x*), et ce processus effectue l'application (*f x*) — qui peut bien sûr être une fonction ne retournant pas, constituée d'une boucle infinie. Une utilisation typique est donc la suivante :

```
let processus () =
    < son code ici >

let main () =
    ...
    let p = Thread.create processus () in
    ...
    Thread.join p

let _ = main ()
```

L'appel à (*Thread.join p*) indique au programme principal d'attendre la fin de l'exécution du thread p. Sans cela, le programme terminerait immédiatement. La bibliothèque *Thread* contient une fonction *delay* permettant de mettre en sommeil le thread courant un temps donné. Son argument est un flottant, dont l'unité est la seconde.

Le module *Semaphore* qui vous est fourni a l'interface suivante :

```
type t
val init : int → t
val p : t → unit
val v : t → unit
```

Informations générales sur ocaml et ses threads : http ://caml.inria.fr/ocaml/