CentraleSupélec | université PARIS-SACLAY

Travaux dirigés - TD
Synchronisation des processus

Exercice 1 : Barrière de synchronisation

On suppose dans un premier temps que l'on ne traite que 2 threads. Ces threads obéissent au modèle suivant :

Thread 1

while $\underline{true}$ **do**
    $calcul\_1()$;
    $barriere\_1()$;
end

Thread 2

while $\underline{true}$ **do**
    $calcul\_2()$;
    $barriere\_2()$;
end

1. En utilisant 2 sémaphores, dont vous préciserez les valeurs initiales, écrivez le pseudo-code des fonctions `barriere_1()` et `barriere_2()`, qui doivent être telles que le **Thread 1** ne sort de `barriere_1()` que lorsque le **Thread 2** a atteint `barriere_2()` (et réciproquement).

Solution

```
Semaphore s1 = new Semaphore(0);
Semaphore s2 = new Semaphore(0);
```

barriere_1()

```
s1.release();
s2.acquire();
```

barriere_2()

```
s2.release();
s1.acquire();
```

2. Généralisez au cas de N threads

Solution

```
    Semaphore s_i = new Semaphore(0);
    // i ∈ 1..n ;
barriere_i()
    for k ← 1 to n − 1 do
    |   s_i.release();
    end
    for k ← 1 to n do
    |   if k ≠ i then
    |   |   s_k.acquire();
    |   end
    end
```

3. Proposez une autre solution, dans laquelle un thread auxiliaire (coordinateur) reçoit un signal quand le **thread i** atteint sa barrière, et débloque les N threads en attente lorsque tous se sont présentés à leur barrière.

Solution

```
    Semaphore s_c = new Semaphore(0);
    Semaphore s_i = new Semaphore(0);
barriere_i()                          coordinateur()
                                          for k ← 1 to n do
                                          |   s_i.acquire();
                                          end
                                          for k ← 1 to n do
    s_i.release();                        |   s_c.release();
    s_c.acquire();                        end
```

Exercice 2 : Gestion de véhicules dans un tunnel

Une route joignant la France à l'Espagne utilise un tunnel à voie unique. En utilisant des sémaphores, vous allez synchroniser les véhicules de manière qu'à tout instant, tous les véhicules circulent dans le même sens.

1. On supposera dans un premier temps que le tunnel peut contenir un nombre quelconque de véhicules, et on tolère les attentes infinies.

Solution

```
Semaphore tunnel = new Semaphore(1);
int nb_fr = 0;
int nb_es = 0;
Semaphore mutex_fr = new Semaphore(1);
Semaphore mutex_es = new Semaphore(1);
```

vers_France()

```
mutex_fr.acquire();
nb_fr++;
if nb_fr = 1 then
    tunnel.acquire();
end
mutex_fr.release();
utiliser_tunnel();
mutex_fr.acquire();
nb_fr–;
if nb_fr = 0 then
    tunnel.release();
end
mutex_fr.release();
```

vers_Espagne()

```
mutex_es.acquire();
nb_es++;
if nb_es = 1 then
    tunnel.acquire();
end
mutex_es.release();
utiliser_tunnel();
mutex_es.acquire();
nb_es–;
if nb_es = 0 then
    tunnel.release();
end
mutex_es.release();
```

2. Modifiez votre solution de manière à garantir que, à un instant donné, au plus 4 véhicules (dans la même direction) sont sur le tunnel.

Solution

```
Semaphore tunnel = new Semaphore(1);
int nb_fr = 0;
int nb_es = 0;
Semaphore mutex_fr = new Semaphore(1);
Semaphore mutex_es = new Semaphore(1);
Semaphore capacite = new Semaphore(4);
```

vers_France()

```
mutex_fr.acquire();
nb_fr++;
if nb_fr = 1 then
    tunnel.acquire();
end
mutex_fr.release();
capacite.acquire();
utiliser_tunnel();
capacite.release();
mutex_fr.acquire();
nb_fr−;
if nb_fr = 0 then
    tunnel.release();
end
mutex_fr.release();
```

vers_Espagne()

```
mutex_es.acquire();
nb_es++;
if nb_es = 1 then
    tunnel.acquire();
end
mutex_es.release();
capacite.acquire();
utiliser_tunnel();
capacite.release();
mutex_es.acquire();
nb_es−;
if nb_es = 0 then
    tunnel.release();
end
mutex_es.release();
```

3. Comment faire en sorte que les véhicules joignant la France (resp. L'Espagne) ne monopolisent pas le tunnel ?

Solution

```
Semaphore tunnel = new Semaphore(1);
int nb_fr = 0;
int nb_es = 0;
Semaphore mutex_fr = new Semaphore(1);
Semaphore mutex_es = new Semaphore(1);
Semaphore capacite = new Semaphore(4);
Semaphore fifo = new Semaphore(1);
```

vers_France()

```
fifo.acquire();
mutex_fr.acquire();
nb_fr++;
if nb_fr = 1 then
    tunnel.acquire();
end
mutex_fr.release();
fifo.release();
capacite.acquire();
utiliser_tunnel();
capacite.release();
mutex_fr.acquire();
nb_fr–;
if nb_fr = 0 then
    tunnel.release();
end
mutex_fr.release();
```

vers_Espagne()

```
fifo.acquire();
mutex_es.acquire();
nb_es++;
if nb_es = 1 then
    tunnel.acquire();
end
mutex_es.release();
fifo.release();
capacite.acquire();
utiliser_tunnel();
capacite.release();
mutex_es.acquire();
nb_es–;
if nb_es = 0 then
    tunnel.release();
end
mutex_es.release();
```

Exercice 3 : Diffusion de messages

On considère un système à 1 producteur et C consommateurs partageant une boîte à lettres pouvant contenir N messages. Le producteur dépose les messages dans la boîte à lettres, et les consommateurs les y récupèrent. Chaque message déposé (une fois) par le producteur doit être lu par les C consommateurs. On impose également que les consommateurs reçoivent les messages dans l'ordre où ils ont été déposés. Il faut noter que les consommateurs ne sont pas synchronisés, et que, par exemple, un consommateur rapide peut avoir lu jusqu'à N messages de plus qu'un consommateur plus lent.

— Développez une solution à ce problème, c'est-à-dire une description du processus producteur et des processus consommateurs qui respectent les spécifications. Cette solution devra utiliser des sémaphores dont le rôle sera explicité.

Remarque : Vous pouvez introduire toute structure de données qui vous paraît utile, sous réserve de la décrire précisément et de la justifier.

Solution

```
int[] bal = new int[N]();
int i_prod = 0;
int[] i_cons = new int[C]();
Semaphore[] place_dispo = new Semaphore[N](C);
// N semaphores initialisés à C ;
Semaphore[] msg_dispo = new Semaphore[N](0);
// N semaphores initialisés à 0 ;
```

producteur()

```
int msg = produire_message();
place_dispo[i_prod].acquire(C);
bal[i_prod] = msg;
msg_dispo[i_prod].release(C);
i_prod = (i_prod +1) mod N;
```

consommateur(int i)

```
msg_dispo[i_cons[i]].acquire();
int msg = bal[i_cons[i]] ;
place_dispo[i_cons[i]].release();
i_cons[i] = (i_cons[i] +1) mod N;
utiliser_message(msg);
```