Master d'informatique | **module PC2R** | **Université Pierre et Marie Curie**
Documents autorisés | *Durée 2 heures* | **Année 2007-2008**

Examen du 21 janvier 2008

Exercice 1 : Simulation d'embouteillages routiers (fair threads)

Cet exercice vise à simuler un embouteillage routier sur une autoroute à 3 voies. Comme d'habitude chaque voie avance à sa vitesse propre. Une voie contient des voitures qui avancent toutes à la même vitesse. On donne le canevas suivant :

```
#include<stdio.h>
int vitesse[4];
ft_scheduler_t voies[4];
ft_event_t reouverture[4];
ft_thread_t voit[1000];

void voiture(void * arg) {
  int position=0;
  int voie_ici=(int*)arg[1];
  while(position<(int*)arg[0]) {
      ..................
      ..................
      ..................
  }
}

void reouveur(void *arg) {
  for(;;) {
      if(vitesse[(int)arg])
        ft_thread_cooperate();
      else
        ........................
        ........................
  }
}

void modif_vitesse(void *arg) {
  for(;;) {
      ........................
  }
}
```

```
int main() {
  int i,j;
  //Creation des schedulers
  ..............................
  ..............................
  ..............................
  //creation des evemenents
  ..............................
  ..............................
  ..............................
  //creation des threads
  ..............................
  ..............................
  ..............................
  //creation des voiture
  // 250 par voie
  //
  ..............................
  ..............................
  ..............................
  // creation des reouvreur
  // modificateur de vitesse
  //
  ..............................
  ..............................
  ..............................
  //
  //Demarage des schedulers
  //
  ..............................
  ..............................
  ..............................
  ft_exit();
}
```

1. Compléter la fonction `main` du canevas.
2. Compléter les fonctions de modificateur de vitesse et de réouvreur de voie.
   Le modificateur de vitesse effectue une boucle infinie et chaque fois qu'il s'exécute, il tire aléatoirement un nombre entre 0 et 4 (fonction `rand() % 5`) et affecte ce nombre à la vitesse de son scheduler et coopère.
   Le réouvreur tourne aussi une boucle infinie. A chaque fois qu'il s'exécute, il teste si la vitesse de son scheduler est nulle, il réaffecte la vitesse de son scheduler à la valeur 1 et engendre l'évènement réouverture pour tous les threads de son scheduler.

3. Compléter la fonction nommée `voiture` qui sera executée par un fair thread dans un *scheduler*. Cette fonction tournera jusqu'à ce que sa variable `position` atteigne la valeur 2000.

   Chaque fois qu'elle s'exécute, cette fonction ajoute à sa variable `position` le contenu de la variable `vitesse` de son scheduler. Si celle ci vaut zéro cette fonction se met en attente sur l'événement de réouverture de sa voie.

4. Toujours dans la boucle principale de la fonction `voiture`, on demande maintenant qu'au bout de quatre avancées (tours de boucle) il y ait un changement de voie.

Exercice 2 : M-variables avec canaux (en O'Caml)

On définit les `M-variables`, variables synchrones mutables, de la manière suivante :

- une M-variable est soit vide, soit pleine
- il y a 2 opérations de base sur une M-variable :
  - prendre (`mTake`) la valeur d'une M-variable est une opération bloquante tant que la variable n'est pas pleine ;
  - mettre (`mPut`) une valeur dans une M-variable provoque une erreur (exception `Put`) si la M-variable est pleine, sinon la remplit
- à sa création une M-variable est vide.

On définit l'interface suivante :

```
type 'a mvar
val mVar : unit -> 'a mvar
exception Put
val mTake : 'a mvar -> 'a Event.event
val mPut : 'a mvar ->  'a -> unit
```

et l'implantation suivante :

```
type 'a mvar = MV of ('a Event.channel * 'a Event.channel * bool Event.channel);;

let mVar () =
  let takeCh = Event.new_channel ()
  and putCh = Event.new_channel ()
  and ackCh = Event.new_channel () in

  let rec empty () =
    let x = Event.sync (Event.receive putCh) in
      Event.sync (Event.send ackCh true);
      full x
  and full x =
    Event.select
      [Event.wrap (Event.send takeCh x) empty ;
       Event.wrap (Event.receive putCh)
                  (fun _ -> (Event.sync (Event.send  ackCh false); full x))]

  in
    ignore (Thread.create empty ());
    MV (takeCh, putCh, ackCh) ;;

let mTake ( mv : 'a mvar) = match mv with
   MV (takechannel, _, _ ) -> Event.receive takechannel ;;
```

```
exception Put;;
let mPut mv x = match mv with
  MV (takechannel, putchannel, ackchannel) ->
     Event.sync (Event.send putchannel x);
     if (Event.sync( Event.receive ackchannel)) then ()
     else raise Put ;;
```

1. Expliquer la représentation d'une M-variable (type et fonction de création) de cette bibliothèque.
2. Que fait le programme suivant lancé au toplevel ? Justifier votre réponse en détaillant l'exécution et en expliquant le comportement des fonctions de cette bibliothèque.

```
let m1 = Mvar.mVar ();;
let m2 = Mvar.mVar ();;
let main1 () =
  let a1 = Mvar.mPut m1 [4;2;1] in
  let a3 = Mvar.mTake m1 in
  let a5 = Mvar.mTake m1 in a5 ;;

let main2 () =
  try Mvar.mPut m1 [3; 7; 9] with Mvar.Put -> Mvar.mTake m1 ;
  try Mvar.mPut m1 [1; 1; 1] with Mvar.Put -> ();
  ();;

Thread.create main1 ();;
main2();;
```

Exercice 3 : Un système d'arrosage (en Esterel)

Le but de cet exercice est de programmer un système d'arrosage automatique hebdomadaire. Il peut aussi fonctionner en mode manuel.

Pour un fonctionnement automatique, l'utilisateur indique le jour, l'heure, la minute et la durée de l'arrosage en envoyant les signaux suivants :

```
;AUTOMATIQUE AUTO_JOUR(1) AUTO_HEURE(16) AUTO_MINUTE(30) AUTO_DUREE(20);
```

Les jours varient de 0 (Dimanche) à 6 (Samedi), les heures de 0 à 23 et les minutes de 0 à 59. La durée est expremiée en seconde.
Dans l'exemple ci-dessus, l'utilisateur a programmé le début de l'arrosage fixé à un lundi à 16h30 pour une durée de 20 secondes.
Pendant que l'arrosage automatique est en attente ou en cours, l'utilisateur a toujours la possibilité de lancer un arrosage manuel en envoyant par exemple les signaux suivants :

```
;MANUEL MANUEL_DUREE(20);
```

Ils permettent de lancer instantanément un arrosage manuel indépendemment de l'arrosage programmé.

Pour les outils concernant le temps (voir dans l'annexe), la procédure `temps()` donne le jour, l'heure et la minute du moment de l'appel et le fonction `TU()` renvoie l'heure actuelle du moment de l'appel sous forme du nombre de secondes écoulées depuis le `1er Janvier 1970 à 00h00m00s GMT`.

1. module `chronos` : Avec l'aide de la procédure `temps()`, écrire le module `chronos` qui émet à chaque tick les signaux valués JOUR, HEURE et MINUTE accompagnés de la valeur respective du jour, de l'heure et de la minute du moment.

2. module `commande_manuel` : Avec l'aide de la fonction `TU()`, écrire le module `commande_manuel` qui émet instantanément le signal `ARROSER` à chaque `tick` pendant toute la durée indiquée en seconde par la valeur associée au signal `MANUEL_DUREE`.
3. module `commande_automatique` : Avec l'aide de la fonction `TU()` et les signaux émis par le module `chronos` à chaque `tick`, écrire le module `commande_automatique` qui, au jour, à l'heure et à la minute indiquée (voir les signaux valués `AUTO_JOUR`, `AUTO_HEURE` et `AUTO_MINUTE`) émettra le signal `ARROSER` à chaque `tick` pendant toute la durée indiquée en seconde par la valeur associée au signal `AUTO_DUREE`. Bien sûr, une fois le temps écoulé, il s'arrête pour recommencer au même jour, heure et minute de la semaine qui suit, etc, ...
4. module `controle_pluie` : On souhaite ajouter à ce dispositif un module `controle_pluie` qui à chaque réception du signal valué `PLUIE` vérifie sa valeur associée. Si elle est supérieure ou égale à 20, le module émet à chaque `tick` le signal `ARRET` jusqu'au prochain signal `PLUIE` avec une valeur associée inférieure à 20. Le signal valuée `PLUIE` est donnée ici par l'utilisateur.Ecrire le module `controle_pluie`.
5. module `systeme` : On souhaite que tout le système d'arrosage soit suspendu tant que le signal `ARRET` est présent. On souhaite aussi que l'utilisateur puisse réinitialiser pour recommencer tout le système en tapant le signal `STOP`. Ecrire le module `systeme` pour un système d'arrosage indiqué ci-dessus.

Exercice 4 : Concours en ligne (serveur)

On cherche à implanter un service réseau permettant de départager n joueurs en classant les réponses en fonction d'une moyenne calculée sur leurs réponses. Un fois inscrits, les joueurs peuvent participer au prochain tour de collecte de réponses. Celui-ci peut démarrer dès qu'il y a assez de joueurs ou bien à intervalle régulier s'il y a suffisamment de personnes prêtes à concourir.

On implantera dans un langage quelconque (O'Caml, Java ou C) un tel serveur en utilisant un protocole texte. Le classement s'effectuera en fonction de la différence du nombre proposé (entre 1 et 1000) par rapport à la moyenne de l'ensemble des réponses des joueurs.

Dans un premier temps, les inscriptions à la prochaine collecte s'effectueront à la fin du tour de collecte précédent.

1. Ecrire la partie du serveur qui fait les inscriptions des joueurs pour un tour de collecte. A partir de 10 inscriptions le tour de collecte peut commencer. Ce début est signaler aux joueurs en attente.
2. Ecrire la partie du serveur correspondant à un tour de collecte, et qui calcule le classement des joueurs. Si un joueur se déconnecte entre l'inscription et le tour de collecte, ou même pendant le tour de collecte, cela ne doit pas perturber les autres joueurs.
3. Modifier le serveur pour pouvoir effectuer les inscriptions des joueurs à tout instant, y compris pendant un tour de collecte ; dans ce cas les inscriptions sont valables pour le prochain tour de collecte.
4. Modifier le serveur pour qu'il puisse lancer le tour de collecte à intervalle régulier s'il y a assez d'inscrits (plus de 5). Les insctiptions restent actives pendant le tour de collecte.
5. Proposer sans l'implanter une adaptation de votre serveur en utilisant les objets distants Java RMI en indiquant les interactions (appels de méthodes distantes) entre le serveur et les clients.