Master d'informatique | **module PC2R** | **Université Pierre et Marie Curie**
Documents autorisés | *Durée 2 heures* | **Année 2006-2007**

Examen du 18 décembre 2006

Exercice 1 : Jeu coloration mémoire concurrente (C, Java ou O'Caml)

On cherche à simuler un jeu de coloration d'une zone mémoiree en suivant des règles issues des automates cellulaires. Chaque joueur commence la partie avec une zone mémoire de sa couleur. A chaque tour il peut lire la couleur d'une case et colorier une case. Si lors du coloriage d'une case, une zone (une ou plusieurs cases) d'une autre couleur se retrouve entourée de zones de la couleur posée strictement plus grandes, alors toute cette zone change de couleur vers la couleur qui vient d'être utilisée.

Exemple d'une coloration :

```
V V V V R R V R V R R R R B B B
              V                    <-- coup joué
V V V V R R V V V R R R R B B B
          ^ ^                      <-- conséquence du coup joué
V V V V V V V V V R R R R B B B
```

Le monde est rond (le successeur de la dernière case est la première). La zone initiale d'une couleur est de longueur $LMAX$. Les couleurs pourront être représentées par des entiers.

On cherche sur ce petit exemple à tester plusieurs modèles de concurrence. Le langage d'implantation est libre (C, Java, O'Caml). Vous indiquerez les fonctions de la bibliothèque des *fair threads* que vous utilisez, et vous adapterez l'énoncé pour un traitement objet des question si vous choisissez Java.

1. On définit une première fonction qui prenant une couleur et un tableau de couleurs, lit une case du tableau et colorie une case. La fonction peut être très naïve, elle permet d'indiquer le langage choisi pour cet exercice. La couleur représentée par un entier permet aussi de connaître la zone initiale allouée à un joueur (par exemple le joueur 0 va de 0 à $LMAX - 1$, le joueur 2 de $LMAX$ à $2 * LMAX - 1$, le joueur 2 de $2 * LMAX$ à $3 * LMAX - 1$, ...).
2. On cherche à définir l'infrastructure pour des joueurs implantés par des *fair threads*. Tout d'abord on définit les variables globales suivantes : une variable `monde` de type référence sur un tableau d'entiers, un compteur pour le nombre de joueurs (`nbj`) et un répartiteur (`sched`) princicipal qui gérera les joueurs. On définit un client générique par une fonction de lancement (`lance`) d'un *fair thread* qui prend en entrée une fonction de jeu (du type de la fonction précédente) calcule une nouvelle couleur à partir du nombre de joueurs, alloue la zone initiale de ce joueur et attache une fonction de jeu coopérante sur le répartiteur `sched`. Cette fonction utilise une fonction de jeu et la fait coopérer. Ecrire le code de lancement de 3 joueurs.
3. Ecrire une fonction qui effectue des coloriages en décalant d'un certain pas la case coloriée à chaque tour. La case à colorier est lue avant ; si elle est de la bonne couleur on incrémente le pas.
4. On ajoute un *fair thread* pour jouer le rôle d'arbitre. Celui-ci vérifie à la fin du tour que tous les joueurs ont encore au moins une case de leur couleur. Si ce n'est pas le cas, l'arbitre détache le thread du joueur sans couleur du répartiteur `sched`. S'il ne reste plus qu'un joueur, ce dernier est déclaré vainqueur.
5. Modifier le client générique pour répercuter les changements de couleur lors du coloriage d'une case qui entraîne un encerclement d'une zone comme décrit précédemment.
6. Pour diminuer l'avantage de toujours commencer un tour le premier, on cherche maintenant à implanter les différents joueurs par des threads classiques préemptifs. Indiquer comment modifier votre programme en utilisant des threads classiques de manière à ce qu'aucun joueur (thread) n'ait plus qu'un coup d'avance par rapport aux autres et en évitant les attentes actives.

Exercice 2 : Ping/pong bégayant (Esterel)

Le but de cet exercice est de simuler un jeu de PINGPONG avec une boucle sans fin dans laquelle le signal `PING` est répété deux fois dans deux instants (tick) consécutifs suivis de deux `PONG` dans deux instants (tick) consécutifs suivants, etc, ...

Par exemple :

```
;PING;PING;PONG;PONG;PING;PING;PONG,....
```

1. Ecrire le module `M` contenant des signaux `ENTREE` et `SORTIE` et une boucle qui
   – attend à partir de l'instant (tick) courrant deux `ENTREE` consécutifs,
   – à la réception de ces signaux, émet dans deux prochains instants (tick) deux `SORTIE`.
2. En utilisant le module `M` ci-dessus, écrire le module `PINGPONG` permettant de lancer le jeux en émettant automatiquement sans intervention extérieure les signaux suivants

   ```
   PING;PING;PONG;PONG;PING;....
   ```

3. Compléter le module `PINGPONG` permettant d'attendre le départ donné à la main en entrée par l'utilisateur. On acceptera les entrées suivantes
   – `; ; ; UN DEUX ; PARTEZ ; ; ; ;`,
   – `; DEUX UN ; PARTEZ ; ; ;`,
   – `; ; ; UN ; DEUX ; PARTEZ ; ; ; ;`,
   – `; DEUX ; UN ; PARTEZ ; ; ;`,

   Voici un exemple d'exécution

   ```
   PINGPONG> ; PARTEZ; DEUX UN;; UN DEUX; PARTEZ;;;;;
   --- Output:
   --- Output:
   --- Output:
   --- Output:
   --- Output:
   --- Output:
   --- Output: PING
   --- Output: PING
   --- Output: PONG
   --- Output: PONG
   --- Output: PING
   PINGPONG>
   ```

4. Le signal `PARTEZ` toujours donné après les signaux `UN` et `DEUX` ne peut cohabité avec ces deux derniers. Modifier le module `PINGPONG` en conséquence.

   Un autre exemple d'exécution

   ```
   PINGPONG> UN DEUX PARTEZ;;
   *** Error: exclusion violated: UN # PARTEZ
   *** Error: exclusion violated: DEUX # PARTEZ
   --- Output:
   PINGPONG>
   ```

Exercice 3 : Bataille navale à n joueurs (O'Caml ou autre)

Le but de cet exercice est de définir le cadre pour des jeux en réseau à plusieurs joueurs. On se servira de ce cadre pour définir un serveur d'un jeu classique : la bataille navale. Les règles du jeu sont les suivantes : chaque joueur place ses bateaux sur une grille tenue secrête et tente de repérer les bateaux des autres joueurs. A chaque tour chaque joueur envoie une sonde à une position donnée et reçoit les résultats de cette sonde pour toutes les grilles. Quand un joueur a toutes les cases de tous ses bateaux repérées, il sort du jeu. Chaque joueur joue à son tour. Quand il ne reste plus qu'un joueur, celui-ci est déclaré vainqueur. Les grilles seront de taille $10*10$. Chaque joueur possédera 3 bateaux : 1 de taille 1 case, 1 de taille 2 cases et 1 de taille 3 cases.

Le serveur doit donc gérer les tâches suivantes :
- Attendre que tous les joueurs (pour un nombre fixé) soient connectés (et que chacun d'eux ait indiqué la position de ses bateaux) et leur indiquer le début de partie ;
- Faire tant que le jeu n'est pas fini :
  - Attendre les coups de tous les joueurs ;
  - Indiquer les coups joués et les résultats à tous les joueurs ;
  - Modifier la grille de chaque joueur ;

Voici quelques indications pour réaliser les questions du serveur : chaque client est géré par le serveur par la création (côté serveur) d'un nouveau *thread*. Chaque client a été identifié (avec un numéro unique) à sa première connexion sur le serveur. Tous les *thread*s gérant des clients partagent les grilles de chaque joueur mises à jour par les coups joués. Le *thread* joueur attend que tous les coups soient joués pour envoyer à son client les résultats de son coup.

On définit le protocole de communication suivant :
- du client vers le serveur :
  - "BATEAU" suivi sur une ligne de sa taille n puis sur $2*n$ lignes les coordonnées du bateau ;
  - "COUP" suivi sur deux lignes des coordonnées X et Y du coup ;
- du serveur vers le client :
  - "JOUEUR" suivi sur une autre ligne du numéro du joueur ;
  - "DEBUT" pour le démarrage de la partie ;
  - "FIN" pour la fin de partie ;
  - NOMBRE suivi de ce nombre de lignes contenant le résultat obtenu
    - "ALEAU", pour une case vide
    - "VU", pour une case d'un élément de bateau
    - "EXPOSE" si la dernière case d'un bateau est vue

    de cette sonde sur toutes les grilles ; la ième ligne correspond à la ième grille. Pour simplifier le traitement on peut aussi envoyer les informations de l'effet de la sonde sur soi-même.

Vous pouvez enrichir ou modifier ce protocole si besoin est.

On conseille d'écrire ce programme en O'Caml. Vous pouvez vous inspirer des différents exemples O'Caml donnés en cours et en TD. Dans ce cas là vous indiquerez les morceaux de programme que vous utilisez en faissant référence aux documents utilisés. Voici un exemple de squelette de serveur :

```
let establish_server client_fun addr port num =
   let sockaddr = Unix.ADDR_INET(addr, port) in
   let sock = Unix.socket Unix.PF_INET Unix.SOCK_STREAM 0 in
     Unix.bind sock sockaddr ;
```

```
Unix.listen sock num;
while true do
  let (s, caller) = Unix.accept sock
  in
    Thread.create client_fun  s
done ;;
```

On supposera connues les fonctions `input_line` de type `Unix.file_descr -> string` et `output_line` de type `Unix.file_descr -> string -> unit` vont respectivement lire une ligne et écrire une ligne sur un canal (`file_descr`) d'entrée/sortie.

Si vous préférez écrire le serveur en Java ou en C, il vous est alors demandé de fournir une description complète du serveur, mais vous pouvez supposer connues les fonctions de base d'entrées/sorties en indiquant leurs signatures. De même vous pouvez aussi vous servir de fonctions ou de classes utilisées dans les polycopiés, en y faisant référence explicitement.

1. Donnez les différentes communications pour une partie qui attend 3 joueurs qui jouent ensuitent pendant deux tours.
2. Donnez l'architecture générale de votre serveur (fonctions ou hiérarchie de classes) en indiquant les variables partagées par les différents processus. On supposera qu'à chaque nouvelle connexion, un processus léger est lancé pour traiter ce nouveau client.
3. Ecrivez la partie attente de connexion de tous les joueurs du serveur. A chaque nouvelle connexion, le nouveau joueur est identifié par un numéro unique.
4. Ecrivez la partie traitement des réponses d'un joueur en suivant le protocole indiqué.
5. Ecrivez la détection de la sortie d'un joueur quand tous ses bateaux sont détectés ainsi que la fin du jeu.
6. Ecrivez le lancement de votre serveur sur la machine `exam.jussieu.fr` sur le port 1288 pour des parties de 5 joueurs en dix coups.
7. Expliquez comment tester le serveur avec `telnet`. Que se passe-t-il en fonction de votre code quand la partie s'arrête ?
8. Décrivez, sans les implantez, les modifications à apporter à votre architecture logicielle pour autoriser plusieurs parties simultanées ?