Master d'informatique | **module PC2R (4I507)** | **Université Pierre et Marie Curie**
Documents autorisés | *Durée 2 heures* | **Année 2017-2018**

Examen du 28 juin 2018

Exercice 1 : Climatisation en Esterel

Le but de cet exercice est de simuler un système de climatisation dans une chambre. Selon la température courante de la chambre et celle sélectionnée, le climatiseur réchauffe (resp. refroidit) en émettant le signal `RC` (resp. `RF`).
Le système fonctionne de la manière suivante : (voir l'exemple de session ci-dessous)

- Le signal valué `TEMP` est émis en continu (ou presque) indiquant la température courante de la chambre.
- On peut sélectionner une nouvelle température en envoyant le signal valué `SET_TEMP`.

En particulier, le climatiseur fonctionne de la manière suivante :

- Il fonctionne en continu.
- Il ignore le signal `START` (voir `climatiseur.canevas`).
- Par défaut, la température sélectionnée et la température courante sont égales à la constante `init_temperature`.
- A tout moment, le signal reçu `SET_TEMP` lui permmet de mettre à jour la température sélectionnée.
- A chaque `tick`, il émet le signal `RC` (resp. `RF`) si la température courante est strictement inférieure (resp. supérieure) à la température sélectionnée.
- Si ces deux températures sont égales, il n'émet rien.
- En l'absence du signal `TEMP`, il garde la température courante précédente.

Question 1. Ecrire le module `climatiseur`.

Le module `chambre` fonctionne de la manière suivante :

- Il fonctionne en continu.
- Un signal `START` démarre le `climatiseur` avec la constante `init_temperature` initialisée à 20.
- Une fois le système démarré et fonctionnant correctement, un nouveau `START` n'a aucun effet (ici, pas de redémarage par un `START`).
- Si le signal `TEMP` est absent pendant 3 `tick` consécutifs, on considère qu'il y a une panne. Le seul signal `ANOMALIE` est émis en continu et aucun autre signal est émis. Seul le signal `START` permet de redémarrer le système.

Question 2. Ecrire le module `chambre`.

```
%%%%%%%%%%%%%%%%%%%%%%%%
% climatiseur.canevas %
%%%%%%%%%%%%%%%%%%%%%%%%

module climatiseur :
  constant init_temperature : integer;
  input    TEMP : integer;      % Pour temperture
  input    SET_TEMP : integer;  % Pour choisir la temperature courante
  output   RC, RF;              % respectivement pour chauffer et refroidir

  % /* ?????????? (2) ?????????? */
end module
--------------------------------------------------------------------------------

%%%%%%%%%%%%%%%%%%%%
% chambre.canevas %
%%%%%%%%%%%%%%%%%%%%

module chambre :

  input  START;
  input  TEMP : integer;        % Pour temperture
  input  SET_TEMP : integer;    % Pour choisir la temperature courante
  output RC, RF;                % respectivement pour chauffer et refroidir
  output ANOMALIE;

  % /* ?????????? (1) ?????????? */
end module
```

```
$ chambre
chambre> ;;;;;
--- Output:
...
--- Output:
--- Output:
chambre> START;; TEMP(23);;TEMP(19);SET_TEMP(30);;
--- Output:
--- Output:
--- Output: RF
--- Output: RF
--- Output: RC
--- Output: RC
--- Output: RC
chambre> ;;;;;TEMP(30);;SET_TEMP(50);;;
--- Output: RC ANOMALIE
--- Output: ANOMALIE
--- Output: ANOMALIE
...
--- Output: ANOMALIE
--- Output: ANOMALIE
chambre> START;TEMP(30);;
--- Output:
--- Output: RF
--- Output: RF
chambre> TEMP(15);;;
--- Output: RC
--- Output: RC
--- Output: RC
chambre>
```

Exercice 2 : Ranger les nombres en FairThreads

Le système fonctionne de la manière suivante :

- Le fairthread `charger` génère un nombre aléatoire (voir `rand()`), met ce dernier dans `source` et met `vide` à `False` pour indiquer qu'il y a une nouvelle valeur.
- Chaque fairthread `deplacer[]` vérifie s'il y a une nouvelle valeur dans `source` (c'est à dire `vide == False`). Si c'est le cas, il déplace ce nombre dans `pair` ou dans `impair` selon sa parité et met `vide` à `True`.
- Ce déplacement peut prendre un certain temps (voir `sleep(alea(5))`.
- On souhaite que pendant ce temps, les autres `deplacer[]` et `changer` continuent, si possible, de travailler normalement.

On évite toute attente active et toute tâche inutile.

Question 1. Déclarer les variables et/ou autres que vous jugez utiles.

Question 2. Ecrire la procédure `_charger()` pour le fairthread `charger`.

Question 3. Ecrire la procédure `_deplacer()` pour les fairthread `deplacer[]`.

Question 4. Ecrire la procédure principale `main()` pour faire fonctionner le système.

```
/***** A compiler avec

   gcc -o exam280618-ft exam280618-ft.c \
       traceinstantsf.c -I $CHEMIN/include -L $CHEMIN/lib \
       -lfthread -lpthread
*****/

#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>
#include "fthread.h"


typedef enum { True, False } Bool;

int         source, pair, impair;
Bool        vide = True;
ft_thread_t charger, deplacer[5];

/* ?????????? (1) ?????????? */

int alea (int maximum) {
  return (int)((rand() + 1.0) / RAND_MAX * maximum);
}

void _charger (void *arg) {
  int n;

  /* ?????????? (2) ?????????? */
}

void _deplacer (void *arg) {
  int n;

  /* ?????????? (3) ?????????? */
}

int main (void)
{
  long i;

  /* ?????????? (4) ?????????? */

  ft_exit();
  return 0;
}
```

```
$ ./exam280618-ft
charger a genere le nombre 1804289383 et le met dans source.
deplacer[0] prend le nombre 1804289383.
charger a genere le nombre 846930886 et le met dans source.
deplacer[1] prend le nombre 846930886.
charger a genere le nombre 1714636915 et le met dans source.
deplacer[2] prend le nombre 1714636915.
charger a genere le nombre 424238335 et le met dans source.
deplacer[3] prend le nombre 424238335.
charger a genere le nombre 596516649 et le met dans source.
deplacer[4] prend le nombre 596516649.
charger a genere le nombre 1025202362 et le met dans source.
deplacer[2] depose le nombre 1714636915 dans impair.
deplacer[4] depose le nombre 596516649 dans impair.
deplacer[0] depose le nombre 1804289383 dans impair.
deplacer[3] depose le nombre 424238335 dans impair.
deplacer[4] prend le nombre 1025202362.
charger a genere le nombre 1967513926 et le met dans source.
deplacer[1] depose le nombre 846930886 dans pair.
deplacer[2] prend le nombre 1967513926.
charger a genere le nombre 304089172 et le met dans source.
deplacer[0] prend le nombre 304089172.
charger a genere le nombre 35005211 et le met dans source.
deplacer[0] depose le nombre 304089172 dans pair.
deplacer[0] prend le nombre 35005211.
charger a genere le nombre 336465782 et le met dans source.
deplacer[4] depose le nombre 1025202362 dans pair.
deplacer[1] prend le nombre 336465782.
charger a genere le nombre 278722862 et le met dans source.
deplacer[2] depose le nombre 1967513926 dans pair.
deplacer[3] prend le nombre 278722862.
deplacer[1] depose le nombre 336465782 dans pair.
charger a genere le nombre 2145174067 et le met dans source.
deplacer[3] depose le nombre 278722862 dans pair.
deplacer[4] prend le nombre 2145174067.
charger a genere le nombre 635723058 et le met dans source.
deplacer[2] prend le nombre 635723058.
charger a genere le nombre 1125898167 et le met dans source.
```

Exercice 3 : Serveur pour le jeu Pac-Man (C, Java ou OCaml)

La mission dans le jeu bien connu Pacman est d'aider le petit Pacman à manger tous les points qui se trouvent dans le labyrinthe pour avoir un bon score en évitant qu'il soit mangé par ses ennemis qui risquent de croiser son chemin comme dans la figure 1.

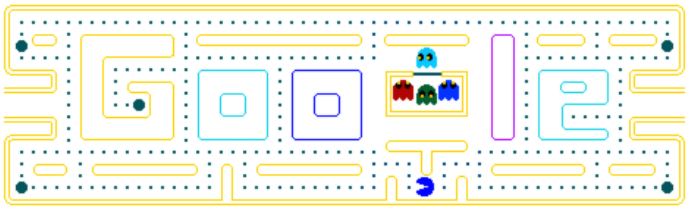

Fig. 1: Configuration d'un tableau de partie

Pour cet exercice on cherchera à implanter une version réseau simplifiée de ce jeu tout d'abord avec une version mono-joueur puis une version multi-joueurs où plusieurs petits Pacmen seront des programmes indépendants qui interagiront en communiquant sur le même serveur. Dans les deux versions chaque ennemi est implanté par un thread différent géré par le serveur. On ne s'intéressera qu'à la partie serveur et au protocole de communications. Le langage d'implantation est à choisir entre C, Java et OCaml.

Le serveur attend une connexion d'un joueur et lance un thread particulier pour communiquer avec ce joueur. Ce thread gère le tableau de la partie, les différents éléments qui en font partie : le Pacman que manipule le client, les différents ennemis qui interviennent dans le jeu, les cases spéciales, le score du joueur et la fin de la partie. Le tableau d'une partie est un labyrinthe rectangulaire possédant des murs et des points à ramasser pour les cases sans mur. La partie s'arrête quand tous les points sont ramassés ou quand le Pacman se fait manger par un ennemi. On utilisera que deux types d'ennemis : le fixe qui occupe une case et mange ceux qui y passent, et l'aléatoire qui se déplace de manière aléatoire et peut manger le Pacman s'il le rencontre.

A la connexion du joueur, le serveur envoie la description du tableau de la partie. Le joueur indique le début de partie et peut ensuite changer la direction de déplacement du Pacman dans les 4 directions possibles. Le personnage avance selon la direction donnée si cela est possible mais peut être bloqué par un obstacle. Quand un ennemi apparait, le serveur indique sa nature et sa position au client joueur. L'ennemi fixe ne bouge pas, par contre l'ennemi aléatoire oui ; les deux indiquent à chaque instant du jeu leur position au joueur. Il peut avoir plusieurs ennemis (plus que deux) en même temps. Les ennemis peuvent se croiser. A chaque instant (tour de jeu), le serveur regarde si le client change de direction, le fait bouger si cela est possible et compte s'il y a des points pris, fait bouger les différents ennemis, et vérifie que la partie est finie, à ce moment là il envoie le score au client.

Question 1. Donner la représentation de ces différentes informations (tableau de la partie, score, joueur, ennemis) et préciser le protocole textuel que vous utiliserez.
Question 2. En réutilisant une structure de serveur comme celle présentée dans le polycopié, écrire la gestion d'une partie où il n'y aurait pas d'ennemi, mais seulement un joueur qui se déplace dans le labyrinthe. Le serveur doit tenir compte de l'interaction.
Question 3. Ajouter ensuite l'apparition d'un ennemi tous les n instants pour une durée de vie de p instants (tours de jeu). La détection de la fin de partie doit alors prendre en compte le fait qu'un ennemi puisse manger le joueur. Indiquer aussi comment vous détectez le fait que le Pacman puisse être mangé. Chaque ennemi est implanté par un thread. Le thread s'arrête quand l'ennemi disparait.

On passe maintenant à une version multi-joueurs. La partie démarre lorsque les n joueurs sont connectés. Chacun sera initialement à une position différente. Seul un joueur peut être sur une case, et donc gagner le point qui y est. Les joueurs ne peuvent donc pas se croiser. La partie est finie quand tous les points sont gagnés ou quand tous les joueurs sont mangés.

Question 4. Indiquer les modifications à apporter à votre serveur pour tenir compte de l'attente de n joueurs avant le démarrage de la partie.
Question 5. Indiquer ensuite les modifications à apporter au protocole de la partie pour tenir compte de l'information reçue par les joueurs et de celles envoyées à tous les joueurs par le serveur.
Question 6. Indiquer les changements à apporter à votre serveur pour tenir compte des modifications effectuées au protocole de communication et pour gérer le déplacement de chaque joueur. Si un joueur est mangé, il reçoit toujours les informations de déplacement des autres joueurs. A la fin de la partie les scores de tous les joueurs sont envoyés.
Question 7. Implanter les changements indiqués précédemment.