Master d'informatique | **module PC2R (4I507)** | **Université Pierre et Marie Curie**
Documents autorisés | *Durée 2 heures* | **Année 2014-2015**

Examen réparti du 19 mars 2015

Exercice 1 : Gestion d'imprimantes (fair threads)

On veut modéliser avec des **fairthread** le comportement des clients dans les files d'attente autour des imprimantes. Le dispositif est composé de :

- `NB_IMPRIMANTES` imprimantes (voir dans la suite `ft_scheduler_t sched_imps[NB_IMPRIMANTES];`),
- un technicien pour la maintenance,
- des clients qui font la queue dans les files d'attente de ces imprimantes

Le comportement du technicien est le suivant :

1) Au départ, il est dans la file de l'imprimante 0 en faisant la queue (voir dans la suite la procédure `main()`).
2) Si l'imprimante est libre (`etats[numero_imprimante] == LIBRE`), il occupe l'imprimante pendant n instants pour faire la maintenance, puis la libère. Pendant ces instants de maintenance, les autres clients qui se trouvent dans la file peuvent décider de rester ou de changer de file.
3) Si l'imprimante est occupée, il décide de quitter la file pour aller faire la queue dans la file d'une autre imprimante ayant comme numéro `numero_imprimante = (numero_imprimante + 1) % NB_IMPRIMANTES`.
4) il boucle à l'étape 2).

Le comportement de chaque client est le suivant :

1) Au départ, il est dans la file d'une imprimante au hasard `alea(NB_IMPRIMANTES)` en faisant la queue (voir dans la suite la procédure `main()`).
2) Si l'imprimante est libre (`etats[numero_imprimante] == LIBRE`), il occupe l'imprimante pendant n instants, puis la libère avant de terminer. Pendant ces instants d'impression, les autres clients ou le technicien qui se trouvent dans la file peuvent décider de rester ou de changer de file.
3) Si l'imprimante est occupée, il décide au hasard en utilisant la valeur de (`alea(2) % 2`),
   - si cette valeur est différente de 0, il reste dans la file à attendre au plus m instants. A la fin de ces instants d'attente, si l'imprimante n'est toujours pas libre, il refait l'étape 3).
   - sinon, il quitte la file pour aller vers la file d'une autre imprimante ayant comme numéro `numero_imprimante = (numero_imprimante + 1) % NB_IMPRIMANTES`.
4) Pendant ces m instants d'attente, il est possible que l'imprimante redevienne libre (`etats[numero_imprimante] == LIBRE`).
5) A l'instant où elle est libre, il fait l'étape 2).

Question 1
Si nécessaire, ajouter les outils et compléter la procédure `main()`.

Question 2
Compléter la procédure `technicien()` en évitant les attentes actives (voir le résultat d'une exécution du programme dans la suite).

Question 3
Compléter la procédure `client()` en évitant les attentes actives (voir le résultat d'une exécution du programme dans la suite).

canevas du programme

```c
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include "fthread.h"

#define NB_IMPRIMANTES 3
enum etat { LIBRE, OCCUPE };

typedef struct _INFO_CLIENT {
  int numero_client;
  int numero_imprimante;
} INFO_CLIENT;

typedef struct _INFO_TRACE {
  int numero_imprimante;
  int nb_tours;
} INFO_TRACE;

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

ft_thread_t    traces[NB_IMPRIMANTES];
ft_scheduler_t sched_imps[NB_IMPRIMANTES];
enum etat      etats[NB_IMPRIMANTES];

/********** ?????????? **********/

void traceinstants (void *arg) {
  int numero_imprimante =
    ((INFO_TRACE *)arg)->numero_imprimante;
  int nb_tours          = ((INFO_TRACE *)arg)->nb_tours;
  int i;

  for (i = 0; i < nb_tours; i++) {
    printf(">>>>>>>>>> Imprimante %d - Instant %d :\n",
      numero_imprimante, i);
    fflush(stdout);
    ft_thread_cooperate ();
  }
  printf(">>>>>>>>>> Imprimante %d - Trace last exit\n", ↩
      numero_imprimante);
  fflush(stdout);
}

void client (void *arg) {
  int numero_client, numero_imprimante, nb_instants;
  int fini;
  numero_client     = ((INFO_CLIENT *)arg)->numero_client;
  numero_imprimante =
    ((INFO_CLIENT *)arg)->numero_imprimante;
  fini = 0;

  /********** ?????????? **********/
}
```

```c

void technicien (void *arg) {
  long numero_imprimante = (long)arg;
  int  nb_instants;

  /********** ?????????? **********/
}

int main (void) {
  int          i;
  INFO_CLIENT *info_client;
  INFO_TRACE  *info_trace;
  ft_thread_t  ft_technicien;
  ft_thread_t  ft_clients[10];

  for (i = 0; i < NB_IMPRIMANTES; ++i) {
      sched_imps[i] = ft_scheduler_create ();
      etats[i]          = LIBRE;
      info_trace =
        (INFO_TRACE *)malloc(sizeof(INFO_TRACE));
      info_trace->numero_imprimante = i;
      info_trace->nb_tours          = 1000;
      traces[i] =
        ft_thread_create(sched_imps[i],
          traceinstants, NULL, (void *)info_trace);

      /********** ?????????? **********/
  }
  ft_technicien = ft_thread_create(sched_imps[0],
                    technicien, NULL, (void *)0);

  for (i = 0; i < NB_IMPRIMANTES; ++i) {
    ft_scheduler_start(sched_imps[i]);
  }
  for (i = 0; i < 10; ++i) {
    info_client =
      (INFO_CLIENT *)malloc(sizeof(INFO_CLIENT));
    info_client->numero_client     = i;
    info_client->numero_imprimante = alea(NB_IMPRIMANTES);
    ft_clients[i] =
      ft_thread_create(sched_imps[alea(NB_IMPRIMANTES)],
        client, NULL, (void *)info_client);
  }
  for (i = 0; i < 10; ++i) {
    ft_thread_join(ft_clients[i]);
  }
  ft_scheduler_stop(ft_technicien);
  for (i = 0; i < NB_IMPRIMANTES; ++i) {
      ft_scheduler_stop(traces[i]);
  }
  ft_exit();
  return 0;
}
```

Un extrait d'un exemple d'exécution (1)

```
>>>>>>>>>> Imprimante 0 - Instant 0 :
>>>>>>>>>> Imprimante 1 - Instant 0 :
Debut maintenance sur l'imprimante 0 pour 4 instants.
>>>>>>>>>> Imprimante 2 - Instant 0 :
     .....
>>>>>>>>>> Imprimante 0 - Instant 2 :
Le client 0 fait la queue dans la file de l'imprimante 2.
Le client 0 occupe l'imprimante 2 pour 1 instants.
>>>>>>>>>> Imprimante 0 - Instant 3 :
     .....
>>>>>>>>>> Imprimante 2 - Instant 2 :
Le client 0 libere l'imprimante 2.
Fin maintenance sur l'imprimante 0.
>>>>>>>>>> Imprimante 2 - Instant 3 :
>>>>>>>>>> Imprimante 0 - Instant 5 :
>>>>>>>>>> Imprimante 1 - Instant 3 :
Le client 1 fait la queue dans la file de l'imprimante 2.
Le client 1 occupe l'imprimante 2 pour 2 instants.
>>>>>>>>>> Imprimante 0 - Instant 6 :
     .....
>>>>>>>>>> Imprimante 0 - Instant 8 :
Le client 1 libere l'imprimante 2.
```

```
Debut maintenance sur l'imprimante 1 pour 3 instants.
>>>>>>>>>> Imprimante 0 - Instant 9 :
     .....
>>>>>>>>>> Imprimante 1 - Instant 6 :
Le client 2 fait la queue dans la file de l'imprimante 1.
L'imprimante 1 est occupee. Le client 2 a choisi de ↩
     quitter la file de l'imprimante 1 pour aller vers la ↩
     file de l'imprimante 2.
>>>>>>>>>> Imprimante 0 - Instant 11 :
>>>>>>>>>> Imprimante 1 - Instant 7 :
>>>>>>>>>> Imprimante 2 - Instant 8 :
Fin maintenance sur l'imprimante 1.
>>>>>>>>>> Imprimante 0 - Instant 12 :
     .....
>>>>>>>>>> Imprimante 0 - Instant 13 :
Le client 2 fait la queue dans la file de l'imprimante 2.
Le client 2 occupe l'imprimante 2 pour 3 instants.
>>>>>>>>>> Imprimante 0 - Instant 14 :
     .....
>>>>>>>>>> Imprimante 2 - Instant 10 :
Le client 3 fait la queue dans la file de l'imprimante 1.
Le client 3 occupe l'imprimante 1 pour 4 instants.
```

Un extrait d'un exemple d'exécution (2)

```
>>>>>>>>>> Imprimante 0 - Instant 16 :
Le technicien quitte la file de l'imprimante 2 occupee ↩
     pour aller a la file de l'imprimante 0.
 >>>>>>>>>> Imprimante 1 - Instant 10 :
      .....
>>>>>>>>>> Imprimante 1 - Instant 11 :
Le client 4 fait la queue dans la file de l'imprimante 2.
L'imprimante 2 est occupe, le client 4 veut bien attendre ↩
     au plus 1 instants.
>>>>>>>>>> Imprimante 1 - Instant 12 :
      .....
>>>>>>>>>> Imprimante 0 - Instant 19 :
Le client 3 libere l'imprimante 1.
>>>>>>>>>> Imprimante 2 - Instant 12 :
Debut maintenance sur l'imprimante 0 pour 5 instants.
>>>>>>>>>> Imprimante 1 - Instant 14 :
Le client 2 libere l'imprimante 2.
>>>>>>>>>> Imprimante 0 - Instant 20 :
```

```
>>>>>>>>>> Imprimante 1 - Instant 15 :
Le client 5 fait la queue dans la file de l'imprimante 2.
Le client 5 occupe l'imprimante 2 pour 1 instants.
Le client 4 fait la queue dans la file de l'imprimante 2.
L'imprimante 2 est occupee. Le client 4 a choisi de ↩
     quitter la file de l'imprimante 2 pour aller vers la ↩
     file de l'imprimante 0.
>>>>>>>>>> Imprimante 1 - Instant 16 :
>>>>>>>>>> Imprimante 0 - Instant 21 :
>>>>>>>>>> Imprimante 1 - Instant 17 :
Le client 5 libere l'imprimante 2.
>>>>>>>>>> Imprimante 2 - Instant 13 :
>>>>>>>>>> Imprimante 1 - Instant 18 :
Le client 6 fait la queue dans la file de l'imprimante 0.
L'imprimante 0 est occupe, le client 6 veut bien attendre ↩
     au plus 5 instants.
>>>>>>>>>> Imprimante 1 - Instant 19 :
     .....
```

Exercice 2 : Répartiteur de charge (canaux synchrones)

On veut modéliser [1] avec des **canaux synchrones** le comportement d'une application web avec répartiteur de charge. Le système est composé de clients, de travailleurs, et d'un répartiteur. Chaque agent du système est associé à un canal unique : les clients et les travailleurs ont chacun un canal personnel, le répartiteur de charge est associé au canal de l'application.
Les clients ont tous le même comportement : chaque client désireux de faire une requête commence par envoyer sur le canal de l'application app (associé au répartiteur de charge) son propre canal c et attend de recevoir sur c un nouveau canal de requête/réponse (frais) nc. Il envoie ensuite sa requête sur nc puis recoit une réponse sur nc. Enfin, il recommence (en essayant d'envoyer à nouveau c sur le canal app).
Les travailleurs ont tous le même comportement : chaque travailleur attend sur son propre canal s un nouveau canal de requête/-réponse nc. Il attend ensuite de recevoir une requête sur nc puis envoie une réponse adéquate sur nc. Enfin, il devient de nouveau disponible.
Le répartiteur de charge reçoit sur le canal app des canaux c envoyés par des clients. A chaque réception, il crée un nouveau canal de requête/réponse nc qu'il communique au client en l'envoyant sur c. Il envoie aussi nc au **premier travailleur disponible** sur son canal associé s. Puis il redevient lui-même disponible pour une nouvelle réception sur app.

Question 4
a. Décrire le système par un schéma représentant les agents et les canaux.
b. Distinguer les canaux fixes des canaux créés dynamiquement.
c. Donner le type des canaux.

Question 5
Ecrire le code exécuté par le client. Les requêtes sont, par exemple, des chaînes de caractères.

Question 6
Ecrire le code exécuté par le travailleur. La réponse à une requête est, par exemple, la requête concaténée à un entier (le numéro du travailleur et/ou le numéro de la réponse).

Question 7
Ecrire le code du répartiteur de charge. Ce dernier utilisera une primitive de **sélection** pour envoyer le canal nc au premier travailleur disponible.

Question 8
Enfin, écrire le code de la simulation qui crée et exécute les différents threads.

Question 9
Expliquer comment modifier le code pour implémenter *une politique de répartition de charge* : l'application contient une file des travailleurs disponibles, chaque travailleur se place en queue de file quand il a fini de traiter une requete et le répartiteur de charge transmet le canal nc au premier travailleur de la file (qui se retire de la file).

1. Il ne s'agit pas ici d'écrire un système client/serveur comme dans l'Exercice 3.

Exercice 3 : Dictionnaires (clients-serveur)

On cherche à implanter un service de dictionnaire sous la forme d'un serveur qui répond à des requêtes textuelles de programmes clients. Un dictionnaire associe une définition à un mot. On en simplifiera l'usage en acceptant au maximum une seule définition par mot.

Le serveur est en attente de connexion. Quand une connexion survient, il construit une prise de communication avec le client ayant effectué la connexion tant que celui-ci ne rompera pas la communication soit via une commande du protocole soit en fermant la socket.

On définit le protocole suivant :

- pour la fermeture de la socket
  `DISCONNECT/user/`
  `(C->S) demande la fermeture de la socket de communication du client "user"`
- pour la consultation du dictionnaire :
  `GET/word/`
  `(C->S) demande de la définition du mot "word" par un client`
  `DEF/word/definition/`
  `(S->C) retourne au client la définition "definition" du mot "word" demandé si elle existe`
  `NODEF/word/`
  `(S->C) indique au client que le mot "word" demandé n'a pas de définition sur le serveur`
- pour la modification du dictionnaire :
  `PUT/word/definition/`
  `(C->S) un client indique la définition "definition" d'un mot "word"`
  `OLDDEF/word/definition/`
  `(S->C) retourne au client l'ancienne définition "definition" du mot "word" défini`
  `NOOLDDEF/word/`
  `(S->C) indique au client que le mot "word" défini ne l'était pas avant`

Dans tout l'exercice vous pouvez reprendre sans les ré-écrire des parties de code du polycopié en les citant. Vous pourrez répondre dans votre langage préféré pour le serveur (OCaml, Java, C) et pour le client (OCaml, Java, C) sachant que le langage choisi par le client doit être différent de celui du serveur. Vous pouvez supposer connues des fonctions de construction et d'extraction des arguments des requêtes et réponses du protocole. Indiquez tout de même leur signature et ce qu'elles font.

Question 10

Ecrire l'organisation générale du serveur, ainsi que la gestion de l'ouverture et la fermeture d'une socket pour un client.

Question 11

Ecrire la partie traitement des requêtes de consultation du serveur. Précisez bien la structure de données que vous utilisez pour le dictionnaire. Que se passe-t-il si plusieurs requêtes de consultation, issues de différents clients, arrivent en même temps sur le serveur ?

Question 12

Ecrire la partie traitement des requêtes de modification du serveur. Faites en sorte que s'il y a une reqête `GET` ou `PUT` en cours de traitement, celui-ci se termine avant que tout autre traitement sur le même mot ne soit lancé. Indiquez votre mécanisme de synchronisation avant de l'implanter.

Question 13

Ecrire un client dans un autre langage que le serveur précédent qui prend une liste de serveurs dictionnaire (noms + ports) et un mot, ouvre une connexion sur chacun d'eux, et lance une recherche séquentielle de la définition du mot sur les serveurs. La recherche s'arrête dès que la définition est trouvée sur un serveur, elle est alors affichée. Si ce n'est pas le cas la recherche passe au serveur suivant. Si le dernier serveur consulté n'a toujours pas la définition, le client affiche "NODEF" à l'écran. Précisez avant de l'implanter si vous fermez la connextion à un serveur dès la fin de la recherche sur celui-ci ou bien à la fin de la recherche séquentielle sur l'ensemble des serveurs.

Question 14

Proposez (et implantez) une modification de ce client pour effectuer cette recherche en parallèle. Tous les serveurs sont alors consultés en parallèle, sachant que la première déifnition retournée est celle qui sera affichée par le client. Précisez votre mécanisme de synchronisation avant de l'implanter.