Programmation Concurrente, Réactive et Répartie

Cours N°2

Emmanuel Chailloux

Master d'Informatique
Université Pierre et Marie Curie

année 2010-2011

Informations sur le cours PC2R

Site

`http://www-master.ufr-info-p6.jussieu.fr/site-annuel-courant`
`http://www-apr.lip6.fr/~chaillou/Public/enseignement/2010-2011/pc2r/`

Equipe pédagogique

- cours : **Emmanuel Chailloux**
- TD/TME groupes 1 (mercredi après-midi) : **Philippe Trébuchet**
- TD/TME groupe 2 (mercredi après-midi) : **Philippe Wang, Tong Lieu, Benjamin Canou**
- TD/TME groupe 3 (vendredi après-midi) : **Tong Lieu**

Logiciels du cours

Installation ARI :

- O'Caml 3-11 (préinstallé)
- Java 1.6 (préinstallé)
- GCC 4.1.2 (préinstallé)

Installation dans : `/users/Enseignants/caillou/usr/local/`

- FTthread pour C
- Esterel

Sources dans `/users/Enseignants/caillou/install`

Rappel du cours 1

- parallélisme : perte du déterminisme
- Modèles de parallélisme
  - mémoire partagée :
    synchronisation explicite/communication implicite
  - mémoire répartie :
    synchronisation implicite/communication explicite

  $\Rightarrow$ dualité des deux modèles

Cours 2 : Threads équitables

1. Généralités : coopération *vs* préemption
2. Api Fairthreads en C : scheduler et threads
3. Implantation
4. Evénements
5. Automates

Fair Threads

- Frédéric Boussinot
- projet MIMOSA EMP-CMA / Inria Sophia Antipolis
  sur la programmation réactive :
  `http://www-sop.inria.fr/mimosa/rp`
- Fair Threads :
  `http://www-sop.inria.fr/mimosa/rp/FairThreads/`

Modèle coopératif et préemptif

- ordonnanceur (*scheduler*) : serveur de synchronisation
- 2 types de threads
  - threads liés à un ordonnanceur (modèle coopératif)
  - threads non liés (modèle préemptif)

Caractéristiques

- multiprocesseurs : schedulers et threads non liés;
- déterministe : si tous les threads sont liés à un seul scheduler;
- I/O bloquantes : implantées par threads non liés;

Caractéristiques (suite)

- *instant* : partagé par tous les threads d'un scheduler;
  synchronisation automatique à la fin de chaque instant
- *événement* : diffusion instantanée à tous les threads liés à un même scheduler;
  permet la synchronisation et la communication
- *automate* : pour les petits threads de courte vie;
  implantation légère

Schedulers

- serveur de synchronisation (instants)
- serveur de communication (événements)
- serveur d'exécution (automates)

Ordonnancement coopératif

Durant un instant :

- exécution de chaque thread jusqu'au prochain point de coopération :

Un thread rend la main au scheduler à un point de coopération :

- explicite : fonction `cooperate`
- implicite : attente d'un événement

- pas de priorité entre threads d'un même scheduler

Ordonnancement préemptif

- modèle à mémoire partagée
- perte du déterminisme
- mutuelle exclusion (Mutex)
- attente sur condition (Condition)

prochain cours : Thread en O'Caml

Automates

petit thread ne nécessitant pas une pile propre
contient une liste d'états (code séquentiel)

- s'exécute dans le thread du scheduler
- effectue un changement d'état en un instant
- passage d'un état à un autre :
  - explicite : saut à un état particulier
  - implicite : passage à l'état suivant
- fin de l'automate, à la fin du dernier état
- peut communiquer par événement (état particulier)

Evénements

- création et diffusion d'un événement à tous les trheads
- attente d'un événement à un instant ou au plus sur n instants
- association d'une valeur à un événement pour un instant et récupération de celle-ci
- sélection sur un tableau d'événements

Implantation des Fair Threads

- en C :
  `http://www-sop.inria.fr/mimosa/rp/FairThreads/FTC/index.html`
- en Java :
  `http://www-sop.inria.fr/mimosa/rp/FairThreads/FTJava/index.html`
- en Scheme :
  `http://www-sop.inria.fr/mimosa/fp/Bigloo/doc/bigloo-8.html`
- en O'Caml (prototype 1) :
  `http://www-apr.lip6.fr/~chaillou/Public/Dev/HirondML/`

Bibliothèque C : scheduler

```
#include <fthread.h>
```

- type `ft_scheduler_t`
- création : `ft_scheduler_t ft_scheduler_create (void)`

  retourne `NULL` si échec de la création
- démarrage :
  `int ft_scheduler_start (ft_scheduler_t sched)`

  code retour 0 ou un code d'erreur $\neq 0$ (`BAD_CREATE`)

Bibliothèque C : scheduler (suite)

Contrôle des threads :

- `int ft_scheduler_stop (ft_thread_t th)`
  force l'arrêt du thread `th`
- `int ft_scheduler_suspend (ft_thread_t th)`
  suspend l'exécution du thread `th` au prochain instant
- `int ft_scheduler_resume (ft_thread_t th)`
  reprend l'exécution du thread `th` au prochain instant

La suspension est prioritaire à la reprise.

Bibliothèque C : thread

- type `ft_thread_t`

```
1 ft_thread_t ft_thread_create (
2   ft_scheduler_t sched,
3   void (*runnable)(void*),
4   void (*cleanup)(void*),
5   void *args
6 )
```

où

- `sched` : scheduler
- `runnable` : fonction de calcul du thread
- `cleanup` : fonction de nettoyage
- `args` : argument des 2 fonctions

Bibliothèque C : thread (suite)

Fin d'un thread :

- fin du calcul de la fonction associée
- appel à `void ft_exit (void)`
- appel à `int ft_scheduler_stop (ft_thread_t th)`

Quand un thread termine la fonction `cleanup` est appelée à l'instant suivant

Bibliothèque C : thread (fin)

Attente de fin d'un thread :

- `int ft_thread_join (ft_thread_t th)`

  attente de la fin du thread `th`
- `int ft_thread_join_n (ft_thread_t th,int n)`

  attente sur au plus n instants

Coopération

- `int ft_thread_cooperate (void)`
  retourne le contrôle au scheduler
- `int ft_thread_cooperate_n (int n)`
  redonne le contrôle pour n instants

Equivalent à :

```
1 for (i=0;i<k;i++) ft_thread_cooperate ();
```

Un premier exemple : Hello World (1)

```
1  #include "fthread.h"
2  #include "stdio.h"
3
4  void h (void *id) {
5    while (1) {
6      fprintf (stderr,"Hello ");
7      ft_thread_cooperate ();
8    }
9  }
10
11 void w (void *id) {
12   while (1) {
13     fprintf (stderr,"World!\n");
14     ft_thread_cooperate ();
15   }
16 }
```

Un premier exemple : Hello World (2)

```
 1
 2  int main(void) {
 3
 4    ft_scheduler_t sched = ft_scheduler_create ();
 5    ft_thread_create (sched,h,NULL,NULL);
 6    ft_thread_create (sched,w,NULL,NULL);
 7    ft_scheduler_start (sched);
 8
 9    ft_exit ();
10    return 0;
11  }
```

Le même en non-déterministe

```
 1  int main (void) {
 2
 3    ft_scheduler_t sched1 = ft_scheduler_create ();
 4    ft_scheduler_t sched2 = ft_scheduler_create ();
 5
 6    ft_thread_create (sched1,h,NULL,NULL);
 7    ft_thread_create (sched2,w,NULL,NULL);
 8
 9    ft_scheduler_start (sched1);
10    ft_scheduler_start (sched2);
11
12    ft_exit ();
13    return 0;
14  }
```

Liaison des threads

- `int ft_thread_unlink (void);`
  délie le thread de son scheduler
- `int ft_thread_link (ft_scheduler_t sched);`
  relie un thread auprès du scheduler `sched`

permet de changer de scheduler.

Lecture non bloquante (1)

```
gcc -Wall -O3 -D_REENTRANT -I ../include -L../lib \
  nbread.c -lfthread -lpthread
```

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

/*****************************************/
ssize_t ft_thread_read (int fd,void *buf,size_t count) {

   ft_scheduler_t sched = ft_thread_scheduler ();
   ssize_t res;

   ft_thread_unlink ();
   res = read (fd,buf,count);
   ft_thread_link (sched);
   return res;
}
```

Lecture non bloquante (2)

```
1
2  /*****************************************/
3  void reading_behav (void* args) {
4    int max = (int)args;
5    char *buf = (char*)malloc (max+1);
6    ssize_t res;
7    fprintf (stderr,"enter %d characters:\n",max);
8
9    res = ft_thread_read (0,buf,max);
10
11   if (-1 == res) fprintf (stderr,"error\n");
12   buf[res] = 0;
13   fprintf (stderr,"read %d: <%s>\n",res,buf);
14   exit (0);
15 }
```

Lecture non bloquante (3)

```
1
2  int main (void) {
3    ft_scheduler_t sched = ft_scheduler_create ();
4    ft_thread_create (sched,reading_behav,NULL,(void*)5);
5    ft_scheduler_start (sched);
6    ft_exit();
7    return 0;
8  }
```

Implantation

- Utilise les threads POSIX (man pthread)
- environ 1800 lignes de C

Implantation (suite)

```
 1  struct ft_scheduler_t {
 2    ft_thread_t                self;
 3    thread_list_t              thread_table;
 4    thread_list_t              to_run;
 5    thread_list_t              to_stop;
 6    thread_list_t              to_suspend;
 7    thread_list_t              to_resume;
 8    thread_list_t              to_unlink;
 9    broadcast_list_t           to_broadcast;
10    pthread_mutex_t            sleeping;
11    pthread_cond_t             awake;
12    ft_environment_t           environment;
13    int                        well_created;
14  };
```

Implantation (suite)

```
1  struct ft_thread_t {
2     pthread_t                pthread;
3     int                      well_created;
4     pthread_mutex_t          lock;
5     pthread_cond_t           token;
6     int                      has_token;
7
8     ft_executable_t          cleanup;
9     ft_executable_t          run;
10    void                    *args;
11
12    ft_scheduler_t           scheduler;
13 ...};
```

Implantation (suite)

```
1  static void _fire_all_threads (ft_scheduler_t sched) {
2
3     FOR_ALL_THREADS
4     if (_is_fireable (thread)){
5        if (!_is_automaton (thread)) {
6           _transmit_token (sched->self,thread);
7        } else {
8           _run_as_automaton (thread);
9        }
10    }
11    END_FOR_ALL
12 }
```

Evénements

- type `ft_event_t`
- création :
  `ft_event_t ft_event_create (ft_scheduler_t sched);`

A l'instant courant :

- génération : `int ft_thread_generate (ft_event_t evt);`
  engendre l'événement `evt` pour l'instant courant; il aura disparu à l'instant suivant
- ```
  int ft_thread_generate_value (ft_event_t evt,
                                void *val);
  ```
  associe une valeur `val` à la génération de l'événement
  ```

Evénements (suite)

A l'instant suivant

- `int ft_scheduler_broadcast (ft_event_t evt);`
  l'événement `evt` sera engendré au prochain instant
- ```
  int ft_scheduler_broadcast_value (ft_event_t evt,
                                    void *val);
  ```
  `val` est associée à `evt`
  ```

Attente d'un événement

- attente
  - `int ft_thread_await (ft_event_t evt);`
    suspend l'exécution du thread jusqu'à la génération d'`evt`
  - `int ft_thread_await_n (ft_event_t evt,int n);`
    l'attente dure au plus *n* instants.
- récupération d'une valeur :
  - ```
    ft_thread_get_value(ft_event e,
                        int num,
                        void **result)
    ```
    récupère la i-ième valeur associée à l'événement `e` :
    - si elle existe, la valeur est rangée dans `result`, l'appel termine immédiatement
    - sinon, la fonction retourne `NULL` à l'instant suivant
    ```

Attente sur tableau d'événements

permet l'attente sur plusieurs événements.
Le tableau d'événements `array` et le tableau `mask` sont de de longueur *len*.

- ```
  int ft_thread_select(int len,
                       ft_event_t *array,
                       int *mask)
  ```
  suspend l'exécution du thread jusqu'à la génération d'au moins un événement du tableau `array`; le tableau `mask` indique quels sont les événements engendrés.
- ```
  int ft_thread_select_n (int len,ft_event_t *array,
                          int *mask,int timeout);
  ```
  Attente au plus *timeout* instants

Exemple avec événements (1)

```
1  #include "fthread.h"
2  #include <stdio.h>
3  #include <unistd.h>
4
5  ft_event_t e1, e2;
6
7  void behav1 (void *args) {
8
9    ft_thread_generate (e1);
10   fprintf (stdout,"broadcast e1\n");
11
12   fprintf (stdout,"wait e2\n");
13   ft_thread_await (e2);
14   fprintf (stdout,"receive e2\n");
15
16   fprintf (stdout,"end of behav1\n");
17 }
```

Exemple avec événements (2)

```
1
2  void behav2 (void *args) {
3
4    fprintf (stdout,"wait e1\n");
5    ft_thread_await (e1);
6    fprintf (stdout,"receive e1\n");
7
8    ft_thread_generate (e2);
9    fprintf (stdout,"broadcast e2\n");
10
11   fprintf (stdout,"end of behav2\n");
12 }
```

Exemple avec événements (3)

```
 1  int main(void) {
 2
 3     int c, *cell = &c;
 4     ft_thread_t th1, th2;
 5     ft_scheduler_t sched = ft_scheduler_create ();
 6
 7     e1 = ft_event_create (sched);
 8     e2 = ft_event_create (sched);
 9
10     th1 = ft_thread_create (sched,behav1,NULL,NULL);
11     th2 = ft_thread_create (sched,behav2,NULL,NULL);
12
13     ft_scheduler_start (sched);
14
15     pthread_join (ft_pthread (th1),(void**)&cell);
16     pthread_join (ft_pthread (th2),(void**)&cell);
17     fprintf (stdout,"exit\n");
18     exit (0);
19  }
```

Exemple avec événements (4)

```
1  /*
2  broadcast e1
3  wait e2
4  wait e1
5  receive e1
6  broadcast e2
7  end of behav2
8  receive e2
9  end of behav1
10 exit
11 */
```

Tableau d'événements (1)

```
1  /* use of select to await 2 events */
2
3  ft_event_t  a,b;
4
5  void awaiter (void *args)
6  {
7     ft_event_t events [2] = {a,b};
8     int        result [2] = {0,0};
9
10    ft_thread_select (2,events,result);
11    fprintf (stdout, "result: [%d,%d] ",result[0],result↩
          [1]);
12    if (result[0] == 0 || result[1] == 0) {
13       ft_thread_await (result[0]==0 ? events[0] : events↩
             [1]);
14    }
15
16    fprintf (stdout, "both received! ");
17    ft_thread_cooperate ();
18    fprintf (stdout, "exit!\n");
19    exit (0);
20 }
```

Tableau d'événements (2)

```
1  void trace_instant (void *args) {
2
3      int i = 1;
4      while (1) {
5          fprintf (stdout, "\ninstant %d: ",i);
6          i++;
7          ft_thread_cooperate ();
8      }
9  }
```

Tableau d'événements (3)

```
1  void agenerator (void *args) {
2
3     ft_thread_cooperate_n (3);
4     fprintf (stdout, "event a generated! ");
5     ft_thread_generate (a);
6  }
7
8  void bgenerator (void *args) {
9
10    ft_thread_cooperate_n (3);
11    fprintf (stdout, "event b generated! ");
12    ft_thread_generate (b);
13 }
```

Tableau d'événements (4)

```
 1  int main (void) {
 2
 3     ft_scheduler_t sched = ft_scheduler_create ();
 4
 5     a = ft_event_create (sched);
 6     b = ft_event_create (sched);
 7     ft_thread_create (sched,trace_instant,NULL,NULL);
 8
 9     ft_thread_create (sched,agenerator,NULL,NULL);
10     ft_thread_create (sched,awaiter,NULL,NULL);
11     ft_thread_create (sched,bgenerator,NULL,NULL);
12
13     ft_scheduler_start (sched);
14
15     ft_exit ();
16     return 0;
17  }
```

Tableau d'événements (5)

```
1
2  /* result
3
4  instant 1:
5  instant 2:
6  instant 3:
7  instant 4: event a generated! result: [1,0] event b ↩
       generated! both received!
8  instant 5: exit!
9  end result */
```

Automates

Ensemble de macros permettant de décrire les états d'un automate et les attentes sur événements.

Création :

```
1 ft_thread_t ft_automaton_create (ft_scheduler_t sched,
2                                  ft_automaton_t automaton,
3                                  ft_executable_t cleanup,
4                                  void *args)
```

attente sur événement avec automate (1)

```
1  #include "fthread.h"
2  #include <stdio.h>
3
4  /* simultaneous events */
5
6  ft_event_t event1,event2;
7
8  DEFINE_AUTOMATON (autom) {
9
10    BEGIN_AUTOMATON
11       STATE_AWAIT (0,event1);
12       STATE_AWAIT (1,event2)
13       {
14          fprintf (stdout, "both events are received! ");
15       }
16    END_AUTOMATON
17 }
```

attente sur événement avec automate (2)

```
 1  void generator (void *args) {
 2
 3     ft_thread_cooperate_n (4);
 4     fprintf (stdout, "event1 generated! ");
 5     ft_thread_generate (event1);
 6
 7     ft_thread_cooperate_n (4);
 8     fprintf (stdout, "event1 and event2 are generated! ");
 9     ft_thread_generate (event1);
10     ft_thread_generate (event2);
11
12     ft_thread_cooperate ();
13     fprintf (stdout, "exit\n");
14     exit (0);
15  }
```

attente sur événement avec automate (3)

```
1  void traceInstants (void *args) {
2
3     int i = 0;
4     for (i=0;i<10;i++) {
5        fprintf(stdout,"\n>>>>>>>>>>>> instant %d: ",i);
6        ft_thread_cooperate ();
7     }
8     fprintf (stdout, "exit!\n");
9     exit (0);
10 }
```

attente sur événement avec automate (4)

```
1  int main () {
2
3     ft_scheduler_t sched = ft_scheduler_create ();
4
5     event1 = ft_event_create (sched);
6     event2 = ft_event_create (sched);
7
8     ft_thread_create (sched,traceInstants,NULL,NULL);
9
10    if (NULL == ft_automaton_create (sched,autom,NULL,NULL)↩
          ) {
11       fprintf (stdout, "cannot create automaton!!!\n");
12    }
13    ft_thread_create (sched,generator,NULL,NULL);
14
15    ft_scheduler_start (sched);
16
17    ft_exit ();
18    return 0;
19 }
```

attente sur événement avec automate (5)

```
1  /* result
2
3  >>>>>>>>>>>> instant 0:
4  >>>>>>>>>>>> instant 1:
5  >>>>>>>>>>>> instant 2:
6  >>>>>>>>>>>> instant 3:
7  >>>>>>>>>>>> instant 4: event1 generated!
8  >>>>>>>>>>>> instant 5:
9  >>>>>>>>>>>> instant 6:
10 >>>>>>>>>>>> instant 7:
11 >>>>>>>>>>>> instant 8: event1 and event2 are generated! ↩
       both events are received!
12 >>>>>>>>>>>> instant 9: exit
13 end result */
```

API FT pour O'Caml

- TER puis projet migration de threads
  - HirondML :
    `http://www-apr.lip6.fr/~chaillou/Public/Dev/HirondML/`
  - sémantique différente sur :
    - l'envoi/réception d'événements
  - pas de préemptif (sauf thread Caml)
  - mais bibliothèque spéciale pour les I/O
- Master STL :
  - HirondML 2: respect de la sémantique des FT

Une implémentation pour OCaml

- Première implémentation : Une surcouche des threads OCaml
  - Un scheduler est un jeton
  - Synchronisation à base de conditions
  - Détachement de fair thread « facile »
- Les problèmes
  - Implémentation simple mais peu efficace
  - Surcouche des threads OCaml pour l'implémentation des threads détachés

les threads détachés sont surtout utilisés pour des IO bloquantes

Idée : une séparation des tâches

- Une librairie pour effectuer des actions bloquantes
- Simuler la concurrence des schedulers
- Nouvelle donne
  - La réactivité d'un scheduler n'est plus assurée, mais on peut mesurer les schedulers « trop lent » facilement par une trace
  - On perd la couche POSIX

C'est l'implémentation actuelle.

FT : Exemple1

```
1  let sched=Fthread.create_scheduler();;
2
3  let rec fth x=
4    Printf.printf "je suis le ft %d\n" x;
5    Fthread.cooperate();
6    fth x
7  ;;
8
9  Fthread.create sched fth 1;
10 Fthread.create sched fth 2;
11 Fthread.start_scheduler sched;
12 Fthread.exit();;
13
14 (* affichage de fth 1 | fth2 ----> *)
15 (* je suis le ft 1
16    je suis le ft 2
17    je suis le ft 1
18    ... *)
```

FT : Compilation exemple 1

sur ari-31-312-01

```
$ ocamlopt -c -I ../lib exfthread.ml
$ ocamlopt -I ../lib unix.cmxa fthread.cmxa \
    exfthread.cmx -o exfthread.exe
$ ./exfthread.exe
je suis le ft 1
je suis le ft 2
je suis le ft 1
je suis le ft 2
je suis le ft 1
je suis le ft 2
...
```

où ../lib correspond à :
/users/Enseignants/chaillou/install/migrate-0.3/lib

FT : Exemple 2 (1)

```
1  let table1=Fthread.create_scheduler();;
2  let table2=Fthread.create_scheduler();;
3  let finale=Fthread.create_scheduler();;
4
5  let nbr_vainqueur=ref 0;;
6  let finale_commencee=ref false;;
7  let passe=Fthread.create_event();;
8  ...
9  for i=1 to 4 do
10   Fthread.create table1 joueur (i,[1;0;2;4;0;3]);
11   Fthread.create table2 joueur ((i+5),[2;1;0;4;3])
12 done;
13
14 Fthread.create table1 joueur (5,[1;2;8;4;4;3;2;8]);
15 Fthread.create table2 joueur (10,[1;8;3;3;2;8]);
16 Fthread.start_scheduler table1;
17 Fthread.start_scheduler table2;
18 Fthread.exit();;
```

FT : Exemple 2 (2)

```
1  let rec joueur (id,cartes)=
2    match cartes with
3        c :: rc ->
4          if c=0 then (... Fthread.awaitn 1 passe;  joueur (↩
               id,rc))
5          else if c>7 then (...     incr nbr_vainqueur; ↩
               Fthread.link_to finale;
6                                joueur (id, rc))
7          else (... Fthread.cooperate(); joueur (id,rc))
8     | _ -> (... if !nbr_vainqueur = 2 then (
9               nbr_vainqueur:=0; Fthread.start_scheduler ↩
                    finale)
```

Bibliographie

- Boussinot, F. – Java Fair Threads – Inria research report, RR-4139, 2001.
- Boussinot, F. – FairThreads: mixing cooperative and preemptive threads in C – Inria research report, RR-5039, December, 2003.
- Serrano, M. et Boussinot, F. et Serpette, B. – Scheme Fair Threads – 6th sigplan International Conference on Principles and Practice of Declarative Programming (PPDP), Verona, Italy, Aug, 2004, pp. 203–214.
- Chailloux, E. et Ravet, V. et Verlaguet, J. — HirondML: Fair Threads Migrations for Objective Caml — Parallel Processing Letters, volume=18-1, 2008.