Maîtrise d'informatique
Module de Programmation parallèle

Interrogation

Philippe Marquet

Mars 2000

Documents de cours et TD autorisés
Durée : 1h30

1 Barrières récursives

On désire implanter une extension des verrous `mutex` de la bibliothèque `pthread` de base. Un de nos `rec_mutex` peut à nouveau être `rec_mutex_lock()` par le thread propriétaire du verrou sans qu'il se bloque. Le propriétaire d'un verrou le relâche par un nombre d'appels à `rec_mutex_unlock()` égal au nombre d'appels à `rec_mutex_lock()` réalisés.

L'interface de la bibliothèque `rec_mutex` est la suivante :

```
typedef struct _rec_mutex_t rec_mutex_t ;

struct _rec_mutex_t { ... } ;

#define REC_MUTEX_INITIALIZER ...

int rec_mutex_init (rec_mutex_t *rm, const pthread_mutexattr_t *attr) ;
int rec_mutex_destroy (rec_mutex_t *rm) ;

int rec_mutex_lock (rec_mutex_t *rm);
int rec_mutex_unlock (rec_mutex_t *rm);
```

Une solution peut être construite à partir de la définition suivante de la structure `_rec_mutex_t` :

```
typedef char bool_t ;

struct _rec_mutex_t {
  pthread_mutex_t      lock ;      /* protège les accès à la structure */
  bool_t               owned ;     /* il y a un propriétaire du verrou */
  pthread_t            owner ;     /* le propriétaire */
  int                  count ;     /* # lock() - # unlock () par owner */
  pthread_cond_t       waiters ;   /* les non-owner se bloquent */
} ;

#define REC_MUTEX_INITIALIZER \
 {PTHREAD_MUTEX_INITIALIZER, FALSE, NULL_TID, 0, PTHREAD_COND_INITIALIZER}
```

Exercice 1
Donner les algorithmes des fonctions `rec_mutex_lock()` et `rec_mutex_unlock()`.

Exercice 2
Donner les codes des fonctions `rec_mutex_lock()` et `rec_mutex_unlock()`.

Id: xam00-interro.tex,v 1.3 2000/03/27 06:11:08 marquet Exp

Exercice 3 (Bonus)
Définir la fonction :

```
int rec_mutex_trylock (rec_mutex_t *rm) ;
```

2 Calcul parallèle de π par intégration

On peut approximer la valeur de

$$\pi = \int_0^1 \frac{4}{1+x^2} \quad dx$$

par

$$w. \sum_{x=\frac{w}{2},1-\frac{w}{2},w} \frac{4}{1+x^2}$$

(Méthode des rectangles ; w est la largeur de chacun des rectangles.) Le programme suivant calcule ainsi une valeur approchée de π :

```
#include <stdio.h>
#include <time.h>

#define f(A) (4.0/(1.0+A*A))

const int n = 1 << 24 ; /* 1024 * 128 * 128 */

int main ()
{
  int i ;
  double w, x, sum, pi ;

  w = 1.0 / n ;
  sum = 0.0 ;
  for (i=1 ; i<=n ; i++) {
    x = w * ((double)i-0.5) ;
    sum += f(x) ;
  }
  pi = w * sum ;

  printf( "computed pi = %24.16g\n", pi ) ;
}
```

Exercice 4
Proposer une parallélisation du programme de calcul de π sur une machine de type MasPar de 128×128 processeurs programmée en MPL.

Exercice 5
Donner l'architecture d'une parallélisation du programme de calcul de π à l'aide de la bibliothèque `pthread` de processus légers.