Autres mécanismes de synchronisation de threads

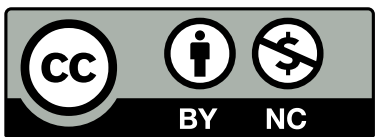
Pierre David
pda@unistra.fr

Université de Strasbourg – Licence d'informatique

2017 – 2018

Plan

Licence d'utilisation



Plan

Introduction

Mécanismes de synchronisation de threads déjà rencontrés :

1. *mutex* : mécanisme de base
2. *spin* : analogue au *mutex*, mais avec attente active
3. sémaphore POSIX (non spécifique aux threads)

Les threads offrent deux autres mécanismes :

4. conditions (*condition variables*)
5. verrous lecteur/écrivain

Plan

Conditions

Problème de synchronisation fréquent :

- un thread attend un événement ou une condition
- la condition est matérialisée par une variable
- un autre thread signale que l'événement est arrivé ou que la condition est remplie

Conditions

Exemple de solution :

- le premier thread attend que la condition soit réalisée :

  cond $\leftarrow$ 0
  tant que cond = 0
  ;

- lorsque la condition est réalisée, le deuxième thread fait :

  cond $\leftarrow$ 1

$\Rightarrow$ cette solution utilise l'attente active !

Exercice : implémenter cela avec des *mutex* POSIX pour réaliser une attente passive...

Conditions

Solution simple : utiliser les conditions POSIX

```
pthread_cond_t  c ;
pthread_mutex_t m ;
int var ;

void attendre (void) {
    pthread_mutex_lock (&m) ;
    var = 0 ;
    while (var == 0)
        pthread_cond_wait (&c, &m) ;
    pthread_mutex_unlock (&m) ;
}

void reveiller (void) {
    pthread_mutex_lock (&m) ;
    var = 1 ;   /* section critique si non atomique */
    pthread_mutex_unlock (&m) ;
    pthread_cond_signal (&c) ;
}
```

Conditions

- une condition est forcément associée à un *mutex*
  $\Rightarrow$ l'accès à la variable est protégé par le *mutex*
- le *mutex* n'est pas *explicitement* déverrouillé lors de l'attente
  - déverrouillage implicite pendant `pthread_cond_wait`
  - re-verrouillage implicite à la sortie de `pthread_cond_wait`
  - $\Rightarrow$ évite les sections critiques
- le réveil peut être provoqué par un autre événément
  $\Rightarrow$ toujours vérifier la raison du réveil
  $\Rightarrow$ cf `while (var == 0)` dans l'exemple

Conditions

- `int pthread_cond_init (pthread_cond_t *, pthread_condattr_t *)`
  ou utilisation de `PTHREAD_COND_INITIALIZER`
- `int pthread_cond_destroy (pthread_cond_t *)`
- `int pthread_cond_wait (pthread_cond_t *, pthread_mutex_t *)`
- `int pthread_cond_timedwait (pthread_cond_t *, pthread_mutex_t *, struct timespec *habs)`
  Avec `struct timespec { time_t tv_sec; long tv_nsec; }`
- `int pthread_cond_signal (pthread_cond_t *)`
- `int pthread_cond_broadcast (pthread_cond_t *)`

Plan

Lecteurs/écrivains

Problème classique de synchronisation lecteurs/écrivains :
$\Rightarrow$ solution simple avec les threads POSIX

```
pthread_rwlock_t verrou ;

int lecteur (void) {
    pthread_rwlock_rdlock (&verrou) ;
    int v = /* lecture */ ;
    pthread_rwlock_unlock (&verrou) ;
    return v ;
}

void ecrivain (int v) {
    pthread_rwlock_wrlock (&verrou) ;
    /* ecriture */ = v ;
    pthread_rwlock_unlock (&verrou) ;
}
```

- implémentation : priorité aux écrivains

Verrous lecteurs/écrivains

- `int pthread_rwlock_init (pthread_rwlock_t *, pthread_rwlockattr_t *)`
- `int pthread_rwlock_destroy (pthread_rwlock_t *)`
- `int pthread_rwlock_rdlock (pthread_rwlock_t *, pthread_mutex_t *)`
- `int pthread_rwlock_wrlock (pthread_rwlock_t *, pthread_mutex_t *)`
- `int pthread_rwlock_tryrdlock (pthread_rwlock_t *, pthread_mutex_t *)`
- `int pthread_rwlock_trywrlock (pthread_rwlock_t *, pthread_mutex_t *)`
- versions avec `timedrdlock` et `timedwrlock`
- `int pthread_rwlock_unlock (pthread_rwlock_t *)`