Sumário

Objetivos

- Revisão mutex locks e variáveis de condição
- Mutex locks recursivos
  - conceito
  - implementação

Mutex locks e variáveis de condição

Mutex locks

⇒ Exclusão mútua

- pthread_mutex_lock
- pthread_mutex_unlock

Variáveis de condição

⇒ Sincronização

- pthread_cond_wait
- pthread_cond_signal
- pthread_cond_broadcast
- precisam ser utilizadas em conjunto com mutex_locks

Thread 0 acorda Thread 1

```
int s;                    /* Veja cond_signal.c */
```

Thread 1:

```
mutex_lock(&mutex);
if (preciso_esperar(s))
    cond_wait(&cond, &mutex);
 mutex_unlock(&mutex);
```

Thread 0:

```
mutex_lock(&mutex);
if (devo_acordar_thread_1(s))
    cond_signal(&cond);
mutex_unlock(&mutex);
```

Pelo menos uma thread é acordada

```
int s;                  /* Veja cond_signal_n.c */
```

Thread i:

```
mutex_lock(&mutex);
while (preciso_esperar(s))
    cond_wait(&cond, &mutex);
mutex_unlock(&mutex);
```

Thread 0:

```
mutex_lock(&mutex);
if (devo_acordar_alguma_thread(s))
    cond_signal(&mutex);
mutex_unlock(&mutex);
```

POSIX Programmer's Manual: pthread_cond_wait()

The pthread_cond_signal() function shall unblock at least one of the threads that are blocked on the specified condition variable cond (if any threads are blocked on cond).

Multiple Awakenings by Condition Signal On a multi-processor, it may be impossible for an implementation of pthread_cond_signal() to avoid the unblocking of more than one thread blocked on a condition variable. For example, consider the following partial implementation of pthread_cond_wait() and pthread_cond_signal(), executed by two threads in the order given. One thread is trying to wait on the condition variable, another is concurrently executing pthread_cond_signal(), while a third thread is already waiting.

POSIX Programmer's Manual: pthread_cond_wait()

```
pthread_cond_wait(mutex, cond):
  value = cond->value; /* 1 */
  pthread_mutex_unlock(mutex); /* 2 */
  pthread_mutex_lock(cond->mutex); /* 10 */
  if (value == cond->value) { /* 11 */
    me->next_cond = cond->waiter;
    cond->waiter = me;
    pthread_mutex_unlock(cond->mutex);
    unable_to_run(me);
  } else
     pthread_mutex_unlock(cond->mutex); /* 12 */
  pthread_mutex_lock(mutex); /* 13 */

pthread_cond_signal(cond):
  pthread_mutex_lock(cond->mutex); /* 3 */
  cond->value++; /* 4 */
  if (cond->waiter) { /* 5 */
    sleeper = cond->waiter; /* 6 */
    cond->waiter = sleeper->next_cond; /* 7 */
    able_to_run(sleeper); /* 8 */
  }
  pthread_mutex_unlock(cond->mutex); /* 9 */
```

POSIX Programmer's Manual: pthread_cond_wait()

Com esta alteração garante que acorda apenas uma thread? Veja pthread_cond_wait.c

```
pthread_cond_wait(mutex, cond):
  pthread_mutex_lock(cond->mutex); /* <=== Pega este lock primeiro */
  value = cond->value;
  pthread_mutex_unlock(mutex);
  if (value == cond->value) {
    me->next_cond = cond->waiter;
    cond->waiter = me;
    pthread_mutex_unlock(cond->mutex);
    unable_to_run(me);
  } else
     pthread_mutex_unlock(cond->mutex);
  pthread_mutex_lock(mutex);

pthread_cond_signal(cond):
  pthread_mutex_lock(cond->mutex);
  cond->value++;
  if (cond->waiter) {
    sleeper = cond->waiter;
    cond->waiter = sleeper->next_cond;
    able_to_run(sleeper);
  }
  pthread_mutex_unlock(cond->mutex);
```

Locks simples

Estrutura protegida por um mutex lock

```
typedef struct estrutura {
     mutex_t lock;
     Tipo1 campo1;
     Tipo2 campo2;
     Tipo3 campo3;
} Estrutura;
```

- Como escrever as funções que fazem acesso a estes campos?

Locks simples

Funções atômicas

```
void funcao1(Estrutura *e) {
    mutex_lock(&e->lock);
    /* ... */
    mutex_unlock(&e->lock);
}

void funcao2(Estrutura *e) {
    mutex_lock(&e->lock);
    /* ... */
    mutex_unlock(&e->lock);
}
```

Locks simples

E se funcao2 invocasse funcao1?

```
void funcao2(Estrutura *e) {
    mutex_lock(&e->lock);
    /* ... */
    if (condicao)
        funcao1(e);
    /* ... */
    mutex_unlock(&e->lock);
}
```

Deadlock de uma thread só

```
void f() {
  mutex_lock(&lock);
  mutex_lock(&lock);
}
```

Veja o código: deadlock.c

Locks simples

E se funcao2 invocasse funcao1?

Possíveis soluções:

- Replicação de código
- Função auxiliar não atômica

```
void funcao1(Estrutura *e) {
     mutex_lock(&e->lock);
     aux_funcao1(e);
     mutex_unlock(&e->lock);
  }
```

Locks recursivos

```
void f() {
  mutex_lock(&lock);
  /* faz alguma coisa */
  mutex_unlock(&lock);
}

void g() {
  mutex_lock(&lock);
  f();
  /* faz outra coisa */
  mutex_unlock(&lock);
}
```

Locks recursivos
e variáveis de condição

```
typedef struct {
  pthread_t thr;
  cond_t cond;
  mutex_t lock;
  int c;
} rec_mutex_t;
```

rec_mutex_lock()

```
int rec_mutex_lock(rec_mutex_t *rec_m) {
  pthread_mutex_lock(&rec_m->lock);
  if (rec_m->c == 0) {  /* Lock livre */
    rec_m->c = 1;
    rec_m->thr = pthread_self();
  } else  /* Mesma thread */
    if (pthread_equal(rec_m->thr,
                      pthread_self()))
        rec_m->c++;
    else {
     /* Thread deve esperar */
```

rec_mutex_lock()

```
    else {
        /* Thread deve esperar */
        while (rec_m->c != 0)
          pthread_cond_wait(&rec_m->cond,
                            &rec_m->lock);
        rec_m->thr = pthread_self();
        rec_m->c = 1;
    }
    pthread_mutex_unlock(&rec_m->lock);
    return 0;
}
```

rec_mutex_unlock()

```
int rec_mutex_unlock(rec_mutex_t *rec_m) {
  pthread_mutex_lock(&rec_m->lock);
  rec_m->c--;
  if (rec_m->c == 0)
    pthread_cond_signal(&rec_m->cond);
  pthread_mutex_unlock(&rec_m->lock);
  return 0;
}
```

Verificação de erros

rec_mutex_unlock()

```
int rec_mutex_unlock(rec_mutex_t *rec_m) {
  pthread_mutex_lock(&rec_m->lock);
  if (rec_m->c == 0 ||
      !pthread_equal(rec_m->thr,
                     pthread_self())) {
      pthread_mutex_unlock(&rec_m->lock_var);
      return ERROR;
  }
  else
    /* ... */
```

Locks recursivos
Implementação reduzida

```
typedef struct {
  pthread_t thr;
  mutex_t lock;
  int c;
} rec_mutex_t;
```

rec_mutex_lock()

```
int rec_mutex_lock(rec_mutex_t *rec_m) {
  if (!pthread_equal(rec_m->thr,
                     pthread_self())) {
    pthread_mutex_lock(&rec_m->lock);
    rec_m->thr = pthread_self();
    rec_m->c = 1;
  }
  else
    rec_m->c++;
  return 0;
}
```

rec_mutex_unlock()

```
int rec_mutex_unlock(rec_mutex_t *rec_m) {
  if (!pthread_equal(rec_m->thr, pthread_self())
      || rec_m->c == 0)
    return ERROR;
  rec_m->c--;
  if (rec_m->c == 0)
    rec_m->thr = 0;
    pthread_mutex_unlock(&rec_m->lock);
  return 0;
}
```

- A implementação reduzida tem comportamento equivalente à primeira?
- Veja o código pthread_mutex_lock.c
- Quando que as variáveis de condição são mais apropriadas?

Filósofos Famintos

Codificação com mutexes locks e variáveis de condição

```
mutex lock;
cond_var filosofo[N];
int estado[N] = {T, T, T, ...,T}
```

Filósofo i:

```
while (true)
    pensa();
    pega_garfos();
    come();
    solta_garfos();
```

Filósofos famintos

Codificação com mutexes locks e variáveis de condição

Observe a codificação a seguir e diga qual é a diferença com relação à implementação com semáforos da solução do livro do Tanenbaum.

```
pega_garfos()
  mutex_lock(&lock);
  estado[i] = H;
  while (estado[fil_esq] != E &&
         estado[fil_dir] != E)
     cond_wait(filosofo[i], lock);
    estado[i] = E;
  mutex_unlock(&lock);
```

Filósofos famintos

Codificação com mutexes locks e variáveis de condição

```
testa_garfos(int i)
  if (estado[i] == H && estado[fil_esq] != E &&
      estado[fil_dir] != E)
      estado[i] = E;
      cond_signal(filosofo[i]);

solta_garfos()
  mutex_lock(lock);
  estado[i] = T;
  testa_garfos(fil_esq);
  testa_garfos(fil_dir);
  mutex_unlock(lock);
```