

MC504 - Sistemas Operacionais

Mutex locks, variáveis de condição e locks recursivos

Islene Calciolari Garcia

Primeiro Semestre de 2014

Sumário

- 1 Revisão Semáforos
- 2 Introdução
- 3 cond_signal
- 4 Locks recursivos
- 5 Implementação reduzida
- 6 Implementação de mutexes
- 7 Futexes
- 8 Bug
- 9 Mutex
- 10 glibc

Semáforos

- Contadores especiais para recursos compartilhados
 - init: inicia o contador com número de recursos disponíveis
 - wait: decrementa, bloqueando o processo se não existirem recursos disponíveis
 - signal ou post: incrementa ou desbloqueia
- Exclusão mútua
- Sincronização

Vários produtores e consumidores

```
semaforo cheio = 0, vazio = N;  
semaforo lock_prod = 1, lock_cons = 1;
```

Produtor:

```
while (true)  
    item = produz();  
    wait(vazio);  
    wait(lock_prod);  
    f = (f + 1) % N;  
    buffer[f] = item;  
    signal(lock_prod);  
    signal(cheio);
```

Consumidor:

```
while (true)  
    wait(cheio);  
    wait(lock_cons);  
    i = (i + 1) % N;  
    item = buffer[i];  
    signal(lock_cons);  
    signal(vazio);  
    consome(item);
```

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);
```

Produtor-Consumidor

```
int c = 0; /* Contador de posições ocupadas */
mutex_t lock_c; /* lock para o contador */

cond_t pos_vazia; /* Para o produtor esperar */
cond_t pos_ocupada; /* Para o consumidor esperar */
```

Produtor-Consumidor

```
int f = 0;
```

Produtor:

```
mutex_lock(&lock_c);
if (c == N)
    cond_wait(&pos_vazia, &lock_c);
f = (f+1)%N;
buffer[f] = produz();
c++;
if (c == 1)
    cond_signal(&pos_ocupada);
mutex_unlock(&lock_c);
```

Produtor-Consumidor

```
int i = 0;
```

Consumidor:

```
mutex_lock(&lock_c);
if (c == 0)
    cond_wait(&pos_ocupada, &lock_c);
i = (i+1)%N;
consome(buffer[i]);
if (c == N-1)
    cond_signal(&pos_vazia);
c--;
mutex_unlock(&lock_c);
```

Produtor-Consumidor

```
cond_t pos_vazia, pos_ocupada; mutex_t lock_v, lock_o;  
int i = 0, f = 0, nv = N, no = 0;
```

Produtor:

```
mutex_lock(&lock_v);  
if (nv == 0) cond_wait(&pos_vazia, &lock_v);  
nv--;  
mutex_unlock(&lock_v);  
f = (f+1)%N;  
buffer[f] = produz();  
mutex_lock(&lock_o);  
no++;  
cond_signal(&pos_ocupada);  
mutex_unlock(&lock_o);
```

Produtor-Consumidor

Consumidor:

```
mutex_lock(&lock_o);
if (no == 0) cond_wait(&pos_ocupada, &lock_o);

no--;
mutex_unlock(&lock_o);
i = (i+1)%N;
consome(buffer[i]);
mutex_lock(&lock_v);
nv++;
cond_signal(&pos_vazia);
mutex_unlock(&lock_v);
```

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?

Mutex lock

Implementação: primeira tentativa

```
int mutex = 0; /* mutex livre */

void pthread_mutex_lock(pthread_mutex_t &mutex) {
    while (mutex != 0);
    mutex = 1;
}

void pthread_mutex_unlock(&mutex) {
    mutex = 0;
}
```

Mutex lock

Implementação: cmpxchg

```
cmpxchg(var, old, new)
```

- var \leftarrow new se var == old
- retorna valor de var antes da operação

```
int mutex = 0; /* mutex livre */
```

```
void pthread_mutex_lock(pthread_mutex_t &mutex) {  
    while (cmpxchg(&mutex, 0, 1) != 0);  
}
```

```
void pthread_mutex_unlock(&mutex) {  
    mutex = 0;  
}
```

Futex: Prototype

```
long sys_futex (
    void *addr1,
    int op,
    int val1,
    struct timespec *timeout,
    void *addr2,
    int val3);

int syscall(SYS_futex, addr1, FUTEX_XXXX,
           val1, timeout, addr2, val3);
```

FUTEX_WAIT

```
/* Retorna -1 se o futex não bloqueou e
   0 caso contrário */
int futex_wait(void *addr, int val1) {
    return syscall(SYS_futex, addr, FUTEX_WAIT,
                  val1, NULL, NULL, 0);}
```

- Bloqueio até notificação
- Não há bloqueio se *addr1 != val1
- Veja o código ex0.c

FUTEX_WAKE

```
/* Retorna o número de threads acordadas */
int futex_wake(void *addr, int n) {
    return syscall(SYS_futex, addr, FUTEX_WAKE,
                  n, NULL, NULL, 0);}
```

- Quantas threads acordar?
 - 1
 - 5
 - INT_MAX (todas)
- Veja os códigos ex1.c e ex2.c

Mutex lock

Implementação: cmpxchg e futex

```
int mutex = 0; /* mutex livre */

void pthread_mutex_lock(pthread_mutex_t &mutex) {
    while (cmpxchg(&mutex, 0, 1) != 0)
        futex_wait(&mutex, 1);
}

void pthread_mutex_unlock(&mutex) {
    mutex = 0;
    futex_wake(&mutex, 1); /* Como evitar esta */
                           /* chamada se ninguém */
                           /* estiver esperando? */
}
```

Mutex lock

Implementação: tentativa de evitar futex_wake desnecessários

```
int mutex = 0; /* mutex livre */
int nw = 0;      /* threads esperando

void pthread_mutex_lock(pthread_mutex_t &mutex) {
    atomic_inc(&nw);
    while (cmpxchg(&mutex, 0, 1) != 0)
        futex_wait(&mutex, 1);
    atomic_dec(&nw);
}

void pthread_mutex_unlock(&mutex) {
    mutex = 0;
    if (nw > 0)
        futex_wake(&mutex, 1);
}
```

Mutex: proposta com bug

```
class mutex {  
public:  
    mutex () : val (0) { }  
    void lock () {  
        int c;  
        while ((c = atomic_inc (val)) != 0)  
            futex_wait (&val, c + 1); }  
    void unlock () {  
        val = 0; futex_wake (&val, 1); }  
private:  
    int val;  
};
```

Mutex: proposta com bug

- atomic_inc como descrito no artigo:
 - Incrementa atomicamente val
 - Retorna valor anterior
- Garante exclusão mútua
- Se a fila não estiver vazia, uma thread irá conseguir pegar o lock após um unlock.
- Se não há espera, a última chamada de sistema é desnecessária
- Livelock
- Overflow (2^{32})

Mutex: segunda proposta

- Significado para val
 - 0: unlocked
 - 1: locked, sem espera
 - 2: locked, com espera
- `cmpxchg(var, old, new)`
 - $\text{var} \leftarrow \text{new}$ se $\text{var} == \text{old}$
 - retorna valor de var antes da operação
- by Ulrich Drepper

Mutex: segunda proposta

```
class mutex {  
public:  
    mutex () : val (0) { }  
    void lock () {  
        int c;  
        if ((c = cmpxchg (val, 0, 1)) != 0)  
            do {  
                if (c == 2 || cmpxchg (val, 1, 2) != 0)  
                    futex_wait (&val, 2);  
            } while ((c = cmpxchg (val, 0, 2))!= 0);  
    }  
}
```

Mutex: segunda proposta

```
void unlock () {
    if (atomic_dec (val) != 1) {
        val = 0;
        futex_wake (&val, 1);
    }
}
private:
    int val;
};
```

Implementação da glibc

- Tipos de lock
 - Fast
 - Com verificação de erros
 - Recursivos
 - Robustos
 - Adaptativos
 - Com controle de prioridade
- Elisão de locks
 - Veja Lock Elision Guide
 - Hardware Transaction Memory

glibc: possíveis contribuições

glibc-2.18/nptl/TODO:

- a new attribute for mutexes: number of times we spin before calling sys_futex
- for adaptive mutexes: when releasing, determine whether somebody spins. If yes, for a short time release lock. If someone else locks no wakeup syscall needed.