

MC514–Sistemas Operacionais: Teoria e Prática
1s2008

**Semáforos ==
Mutex locks e variáveis de condição?**

Poder computacional equivalente

É possível implementar...

- semáforos utilizando mutex locks e variáveis de condição?
- mutex locks e variáveis de condição utilizando semáforos?

Semáforos

- `sem_init(s, 5)`

- `wait(s)`

```
if (s == 0) bloqueia_processo();  
else s--;
```

- `signal(s)`

```
if (s == 0 && existe_processo_bloqueado)  
    acorda_processo();  
else s++;
```

- Vamos completar o arquivo `mutex2sem.c`

Implementação de semáforos usando mutex locks e variáveis de condição

```
typedef struct {  
    int value;      /* Valor atual do semáforo */  
    mutex_t lock;  
    cond_t cond;  
} sem_t;
```

sem_init

```
int sem_init(sem_t *sem, int pshared,  
             unsigned int value) {  
    sem->value = value;  
    mutex_init(&sem->lock, NULL);  
    cond_init(&sem->cond, NULL);  
    return 0;  
}
```

sem_wait

```
int sem_wait(sem_t * sem) {  
    mutex_lock(&sem->lock);  
    while (sem->value == 0)  
        cond_wait(&sem->cond, &sem->lock);  
    sem->value--;  
    mutex_unlock(&sem->lock);  
    return 0;  
}
```

sem_post

```
int sem_post(sem_t * sem) {  
    mutex_lock(&sem->lock);  
    sem->value++;  
    cond_signal(&sem->cond);  
    mutex_unlock(&sem->lock);  
    return 0;  
}
```

sem_trywait

```
int sem_trywait(sem_t * sem) {  
    int r;  
    mutex_lock(&sem->lock);  
    if (sem->value > 0) {  
        sem->value--;  
        r = 0;  
    } else  
        r = EAGAIN;  
    mutex_unlock(&sem->lock);  
    return r;  
}
```


sem_getvalue e sem_destroy

```
int sem_getvalue(sem_t *sem, int *sval) {  
    *sval = sem->value;  
    return 0;  
}
```

```
int sem_destroy(sem_t *sem) {  
    mutex_destroy(&sem->lock);  
    cond_destroy(&sem->cond);  
    return 0;  
}
```

Implementação de mutex locks utilizando semáforos Sem verificação de erros

```
typedef struct {  
    sem_t sem;  
} mutex_t;
```

mutex_init e mutex_destroy

```
int mutex_init(mutex_t *lock, mutex_attr* attr) {  
    return sem_init(&lock->sem, 0, 1);  
}
```

```
int mutex_destroy(mutex_t *lock) {  
    return sem_destroy(&lock->sem);  
}
```

mutex_lock e mutex_unlock

```
int mutex_lock(mutex_t *lock) {  
    return sem_wait(&lock->sem);  
}
```

```
int mutex_unlock(mutex_t *lock) {  
    return sem_post(&lock->sem);  
}
```

Implementação de variáveis de condição usando locks e semáforos

- Mutex locks podem ser implementados com semáforos.
- Uso de locks para coerência com a interface do `cond_wait`:

```
int cond_wait(cond_t *cond,  
              mutex_t *mutex_externo);
```

Implementação com bug de variáveis de condição usando locks e semáforos

```
typedef struct {  
    mutex_t lock;  
    sem_t sem;  
    int n_wait;  
} cond_t;
```

cond_init

```
int cond_init(cond_t *cond) {  
    mutex_init(&cond->lock, NULL);  
    sem_init(&cond->sem, 0, 0);  
    n_wait = 0;  
    return 0;  
}
```

cond_wait

```
int cond_wait(cond_t *cond,  
              mutex_t *mutex_externo) {  
    mutex_lock(&cond->lock);  
    cond->n_wait++;  
    mutex_unlock(&cond->lock);  
    mutex_unlock(mutex_externo);  
    sem_wait(&cond->sem);  
    mutex_lock(mutex_externo);  
    return 0;  
}
```


cond_signal

```
int cond_signal(cond_t *cond) {  
    mutex_lock(&cond->lock);  
    if (cond->n_wait > 0) {  
        cond->n_wait--;  
        sem_post(&cond->sem);  
    }  
    mutex_unlock(&cond->lock);  
    return 0;  
}
```

Bug!

- Thread 0 executa `cond_wait`
- Thread 1 executa `cond_signal`
- Thread 2 executa `cond_wait` e não fica bloqueada
- Thread 0 continua esperando...
- Veja o código: `mutex_bug.c` e `bug.c`

Primeira sugestão para solucionar o problema

- Thread que executou `cond_signal` fica bloqueada até a outra thread ter sido acordada;
- Qual é o ponto fraco desta abordagem?
- Veja se a implementação sugerida funciona...
- Como seria uma implementação incluindo `cond_broadcast`?

Implementação de variáveis de condição com bloqueio no cond_signal

```
typedef struct {  
    mutex_t lock;  
    sem_t sem;  
    int n_wait;  
    sem_t block_signal;  
} cond_t;
```

cond_init

```
int cond_init(cond_t *cond) {  
    mutex_init(&cond->lock, NULL);  
    sem_init(&cond->sem, 0, 0);  
    sem_init(&cond->block_signal, 0, 0);  
    n_wait = 0;  
    return 0;  
}
```

cond_wait

```
int cond_wait(cond_t *cond,  
              mutex_t *mutex_externo) {  
    mutex_lock(&cond->lock);  
    cond->n_wait++;  
    mutex_unlock(&cond->lock);  
    mutex_unlock(mutex_externo);  
    sem_wait(&cond->sem);  
    sem_post(&cond->block_signal);  
    mutex_lock(mutex_externo);  
    return 0;  
}
```

cond_signal

```
int cond_signal(cond_t *cond) {  
    mutex_lock(&cond->lock);  
    if (cond->n_wait > 0) {  
        cond->n_wait--;  
        sem_post(&cond->sem);  
        sem_wait(&cond->block_signal);  
    }  
    mutex_unlock(&cond->lock);  
    return 0;  
}
```

Segunda sugestão para solucionar o problema

- Lista ligada (fila) de semáforos;
- `cond_wait` coloca um nó ao final da fila;
- `cond_signal` remove o primeiro nó da fila;
- `cond_broadcast` remove todos os nós.
- Veja o código: `mutex_lista.c`

Terceira sugestão para solucionar o problema

- Compartilhamento de semáforos
- `cond_wait` é executado na estrutura apontada pelo campo `wait`
- `cond_signal` é executado na estrutura apontada pelo campo `signal`
 - caso `signal` seja nulo, a estrutura apontada por `wait` é movida para `signal`;

Compartilhamento de semáforos

```
typedef struct node_t {  
    int n_wait;  
    sem_t sem;  
} node_t;
```

```
typedef struct {  
    mutex_t lock;  
    node_t *signal;  
    node_t *wait;  
} cond_t;
```

cond_wait

```
int cond_wait(cond_t *cond, mutex_t *ext) {  
    mutex_lock(&cond->lock);  
    if (!cond->wait)  
        cond->wait = novo_cond_wait();  
    node_t n = cond->wait;  
    n->n_wait++;  
    mutex_unlock(&cond->lock);  
    mutex_unlock(&ext);  
    sem_wait(&n->sem);  
    /* ... */  
}
```

cond_wait

```
int cond_wait(cond_t *cond, mutex_t *ext) {  
    /* ... */  
    mutex_lock(&cond->lock);  
    n->n_wait--;  
    if (n->n_wait == 0)  
        destroi(n);  
    mutex_unlock(&cond->lock);  
    return 0;  
}
```

cond_signal

```
int cond_signal(cond_t *cond) {  
    mutex_lock(&cond->lock);  
    if (cond->signal || cond->wait) {  
        if (!cond->signal) {  
            cond->signal = cond->wait;  
            cond->wait = NULL;  
        }  
        sem_post(&cond->signal->sem);  
    }  
    mutex_unlock(&cond->lock);  
    return 0;  
}
```

Outra abordagem...

- Garantir que **pelo menos uma** thread será acordada no `cond_signal`
- Veja man `pthread_cond_signal`
- Veja a implementação `pthread_cond_wait.c`