MO806/MC914

Tópicos em Sistemas Operacionais

2s2006

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

Comportamento básico

- 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++;
```

Implementação de semáforos usando mutex locks e variáveis de condição

```
typedef struct {
  int value;     /* Valor atual do semáforo */
  int n_wait;    /* Número de threads esperando */
  mutex_t lock;
  cond_t cond;
} sem_t;
```

sem_init

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

sem_wait

```
int sem_wait(sem_t * sem) {
  mutex_lock(&sem->lock);
  if (sem->value > 0)
    sem->value--;
  else {
    sem->n_wait++;
    cond_wait(&sem->cond, &sem->lock);
  }
  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_post

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

sem_getvalue

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

sem_destroy

```
int sem_destroy(sem_t *sem) {
  if (sem->n_wait)
     return EBUSY;
  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 com bug 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 (bug)

```
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;
}
```

cond_broadcast

```
int cond_broadcast(cond_t *cond) {
  mutex_lock(&cond->lock);
  while (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_broadcast
- 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 ou cond_broadcast bloqueia até todas as threads terem sido acordadas;
- Desempenho ruim, pois estas funções deveriam ser não bloqueantes.
- Veja os códigos: mutex_bloq.c e teste_bloq.c

Implementação de variáveis de condição com bloqueio no cond_signal

```
typedef struct {
  mutex_t lock, lock_aux;
  int n_wait;
  int n_signal;
  sem_t sem;
  sem_t sem_bloq;
} cond_t;
```

cond_signal

```
int cond_signal(cond_t *cond){
  mutex_lock(&cond->lock);
  if (cond->n_wait > 0) {
    cond->n_signal = 1;
    cond->n_wait--;
    sem_post(&cond->sem);
    sem_wait(&cond->sem_bloq);
  }
  mutex_unlock(&cond->lock);
  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);
```

cond_wait

```
mutex_lock(&cond->lock_aux);
cond->n_signal--;
if (cond->n_signal == 0)
  sem_post(&cond->sem_bloq);
mutex_unlock(&cond->lock_aux);
mutex_lock(mutex_externo);
return 0; }
```

cond_broadcast

```
int cond_broadcast(cond_t *cond) {
  mutex_lock(&cond->lock);
  if (cond->n_wait > 0) {
    cond->n_signal = cond->n_wait;
    while (cond->n_wait > 0) {
      cond->n_wait--;
      sem_post(&cond->sem);
    }
    sem_wait(&cond->sem_bloq);
  }
  mutex_unlock(&cond->lock);
  return 0; }
```

Segunda sugestão para solucionar o problema

- Lista ligada para espera.
- Veja o código: mutex_lista.c

Implementação de variáveis de condição com lista ligada

```
typedef struct node_t {
  sem_t sem;
  struct node_t *next;
} node_t;

typedef struct {
  mutex_t lock;
  node_t *first, *last;
} cond_t;
```

Implementação de variáveis de condição com lista ligada

- Cada thread espera em um nó da lista ligada (fila);
- cond_wait coloca um nó ao final da final;
- cond_signal remove o primeiro nó da fila;
- cond_broadcast remove todos os nós.

Terceira sugestão para solucionar o problema

- Estrutura dinâmica para compartilhamento de semáforos
- Veja o código: mutex_comp.c
- Será que funciona?

Compartilhamento de semáforos

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

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

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;
- cond_broadcast limpa os apontadores signal e wait.
- ao final do cond_wait, caso nenhuma outra thread esteja esperando sinal, o semáforo é destruído e o nó é liberado.

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;
    }
    cond->signal->n_signal++;
    sem_post(&cond->signal->sem);
```

cond_signal

```
    /* ... */

    if (cond->signal->n_signal ==
        cond->signal->n_wait)
      cond->signal = NULL;
  }
  mutex_unlock(&cond->lock);
  return 0;
}
```

Como optar?

- Mutex locks e variáveis de condição
  - Separação clara entre sincronização e exclusão mútua
  - Mais fácil de expressar condições complexas para bloqueio
- Semáforos
  - Representação mais compacta para contadores