MC514

Sistemas Operacionais:

Teoria e Prática

1s2006

Mutex locks e variáveis de condição

Semáforos

- Exclusão mútua
- Sincronização

Exemplo de exclusão mútua

Leitor-Escritor

```
semaforo sem_dado = 1;
```

Leitor:

```
while(true)
    wait(sem_dado);
    le_dado();
    signal(sem_dado);
```

Escritor:

```
while(true)
    wait(sem_dado);
    escreve_dado();
    signal(sem_dado);
```

Exemplo de sincronização

Produtor-Consumidor

```
semaforo cheio = 0, vazio = N;
```

Produtor:

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

Consumidor:

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

Mutex locks

$\Rightarrow$ Exclusão mútua

- pthread_mutex_lock
- pthread_mutex_unlock

Exemplo de exclusão mútua

Leitor-Escritor

```
mutex_lock lock_dado = 1;
```

Leitor:

```
while(true)
    mutex_lock(lock_dado);
    le_dado();
    mutex_unlock(lock_dado);
```

Escritor:

```
while(true)
    mutex_lock(lock_dado);
    escreve_dado();
    mutex_unlock(lock_dado);
```

Variáveis de condição

$\Rightarrow$ 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);
```

Thread 0 acorda alguma thread

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

Thread i:

```
  mutex_lock(&mutex);
  if (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);
```

Thread 0 acorda todas as threads

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

Thread i:

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

Thread 0:

```
  mutex_lock(&mutex);
  if (devo_acordar_uma_ou_mais_threads(s))
     cond_broadcast(&cond);
  mutex_unlock(&mutex);
```

Thread 0 acorda todas as threads

mas algumas delas voltam a dormir

```
int s;                  /* Veja cond_broadcast2.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_uma_ou_mais_threads(s))
     cond_broadcast(&cond);
  mutex_unlock(&mutex);
```

Importância do teste com while

- Thread i vai dormir porque uma condição C é verdadeira
- Thread j acorda thread i porque detectou que C é falsa.
- Antes de thread i voltar a rodar, alguma outra thread tornou C verdade novamente.

- Veja o código `teste_cond_wait.c`

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 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_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;
}
```

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;
}
```

Se são equivalentes, 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

Filósofos (Tanenbaum)

Como reescrever este código com locks e variáveis de condição?

```
semaforo lock;
semaforo filosofo[N] = {0, 0, 0, ..., 0}
int estado[N] = {T, T, T, ...,T}
```

Filósofo i:

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

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

pega_garfos()
  wait(lock);
  estado[i] = H;
  testa_garfos(i);
  signal(lock);
  wait(filosofo[i]);

solta_garfos()
  wait(lock);
  estado[i] = T;
  testa_garfos(fil_esq);
  testa_garfos(fil_dir);
  signal(lock);
```