POSIX Threads (pthreads)

Programação paralela multithreaded

Júlio Cesar Torelli

Rodrigo D. Malara

USP São Carlos

POSIX “Portable Operating System Interface”

- Interface de Sistema Operacional Portável
- O sistema UNIX se desenvolveu como um software de código aberto
- Desenvolvimento de sistema UNIX incompatíveis: *System V* e *BSD*, além de outras implementações
- IEEE propõe em 1988 o padrão **IEEE Std. 1003.1**
- O nome **POSIX** foi inicialmente utilizado para referenciar o padrão 1003.1
- POSIX 1003.1c define uma interface de programação para o desenvolvimento de aplicações *multithread*.
- Uma implementação desta interface é chamada **Posix Threads** ou simplesmente **Pthreads**

POSIX Threads (Pthreads)

- API composta por aproximadamente 60 funções
- Definida apenas para linguagem C/C++
- Programador é responsável (explicitamente) pela criação e sincronização dos threads.

- Basicamente, disponibiliza três grupos de funções:
  - Manipulação de *threads* (criação, destruição, definição de atributos, prioridades, etc.)
  - MUTEX (*Mutual Exclusion locks*)
  - Variáveis de condição

POSIX Threads: Um exemplo (1/3)

- Um exemplo

```
#include <pthread.h>
#include <stdio.h>
#define NUM_THREADS 10

void *PrintHello(void *numFor)
{
    printf("\tthread %d: Hello World!\n",
numFor);
    pthread_exit(NULL);
}
```

POSIX Threads: Um exemplo (2/3)

- Um exemplo (continuação …)

```
int main (int argc, char *argv[])
 {
   pthread_t threads[NUM_THREADS];
   int rc, t;
   for(t = 0; t < NUM_THREADS; t++){
      printf("Criando thread %d\n", t);
      rc = pthread_create(&threads[t], NULL, PrintHello, (void *)t);
      if (rc){
         printf("ERRO; cód. retorno de pthre                    );
         exit(-1);
      }
   }
   pthread_exit(NULL);
 }

//Programa baseado em (LLNC, 2003)
```

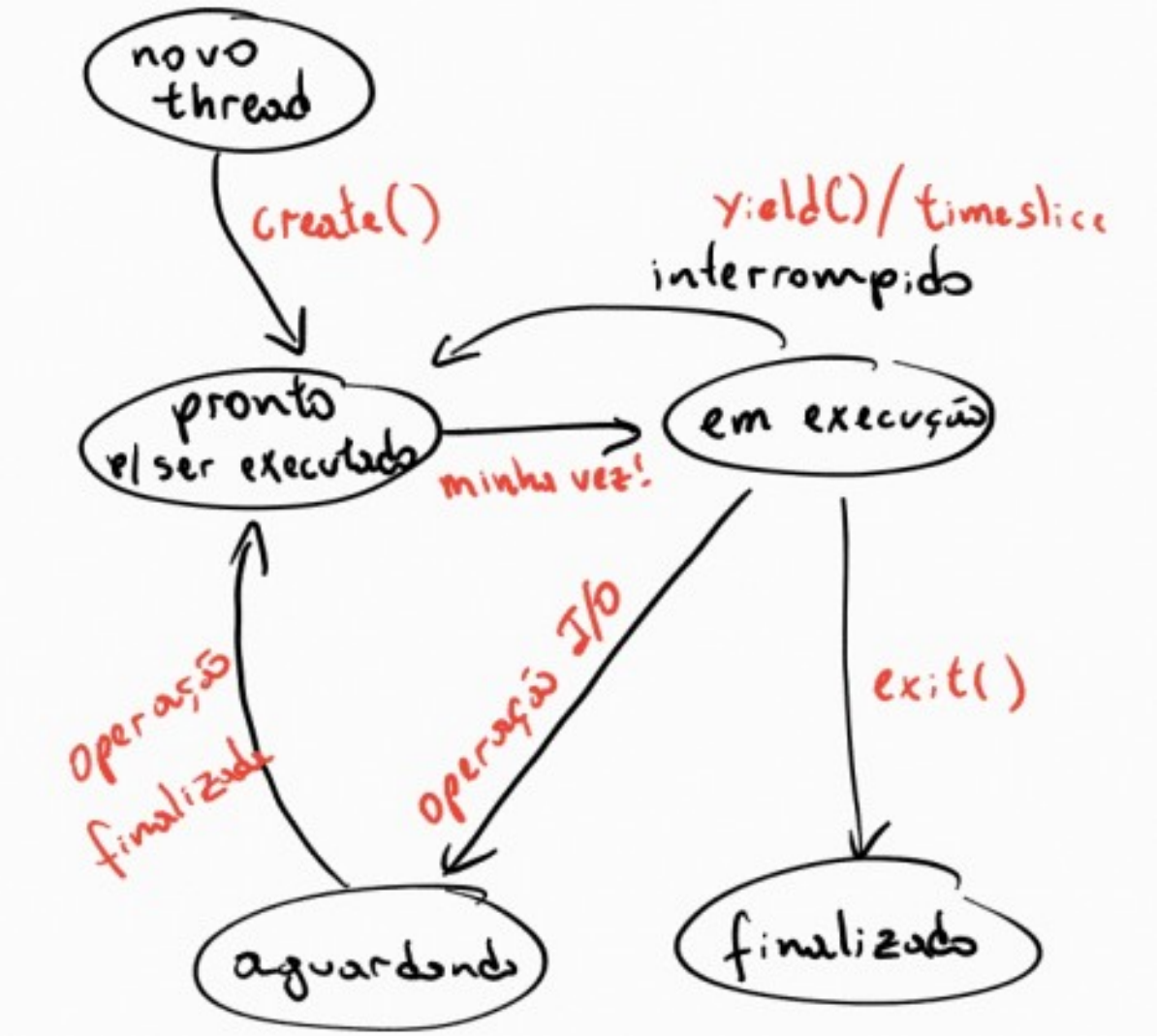
Um único parâmetro por vez

POSIX Threads: Um exemplo (3/3)

- Saída (uma possibilidade)

```
thread 0: Hello World!;
thread 1: Hello World!;
thread 5: Hello World!;
thread 8: Hello World!;
thread 4: Hello World!;
thread 2: Hello World!;
thread 6: Hello World!;
thread 7: Hello World!;
thread 9: Hello World!;
thread 3: Hello World!;
```

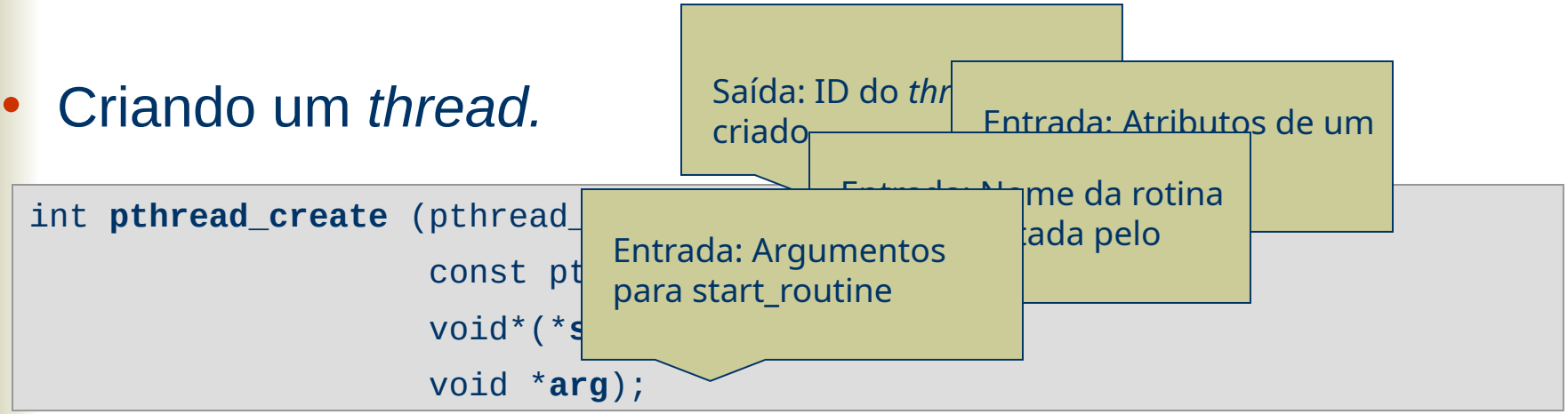
POSIX Threads: Estados dos Threads



POSIX Threads: Criando um *thread*

- Todos os programas devem incluir o arquivo **`pthreads.h`**

- Criando um *thread.*



Retorno:

SE SUCESSO

Retorna 0 (zero)

SENAO

EAGAIN: O limite do sistema foi atingido.

EINVAL: O valor especificado em attr é inválido.

POSIX Threads: Terminando um *thread*

- O thread é automaticamente destruído quando ele executa o procedimento que lhe foi designado. Mas isto pode ser feito explicitamente através da função **`pthread_exit()`**

```
void pthread_exit (void *status);
```

Também é útil para aguardar o término dos threads criados pelo thread que estiver invocando pthread_exit()

Argumento:

O parametro **status** pode ser utilizado para um thread que esteja aguardando (join) o término deste

POSIX Threads: Maior Controle na Execução (1/3)

- É possível que um thread abra mão do processamento antes de terminar ou de ser preemptada.
- Isto permite um controle mais afinado sobre o comportamento dos threads durante a execução.
- Usa-se a função sched_yield

```
int sched_yield ();
```

Retorno:

SE SUCESSO

Retorna 0 (zero)

SENAO

ENOSYS : não suportado pela implementação.

POSIX Threads: Sincronização com MUTEX (1/3)

"O *mutex* funciona como uma trava parecida com as encontradas em armários públicos em aeroportos ou alguns bancos. Se a porta estiver aberta, é só usar (e trancar). Se estiver fechada você deve esperar a sua vez" (GUBITOSO, 2003).

POSIX Threads: Sincronização com MUTEX (2/3)

- MUTEX “Mutual Exclusion”: provê exclusão mútua no acesso a recursos compartilhados. Em Pthreads um *mutex* é uma variável do tipo **`pthread_mutex_t`** e deve ser inicializada antes do uso (e destruída após isto)

```
int pthread_mutex_init (pthread_mutex_t *mp,
                        const pthread_mutexattr_t *mattr);
```

```
int pthread_mutex_destroy(pthread_mutex_t *mp);
```

POSIX Threads: Sincronização com MUTEX (3/3)

- O mutex deve ser adquirido por um thread antes de entrar em uma região crítica. Se neste momento outro *thread* tiver o *lock (mutex)* o thread corrente é blo

```
int pthr
```

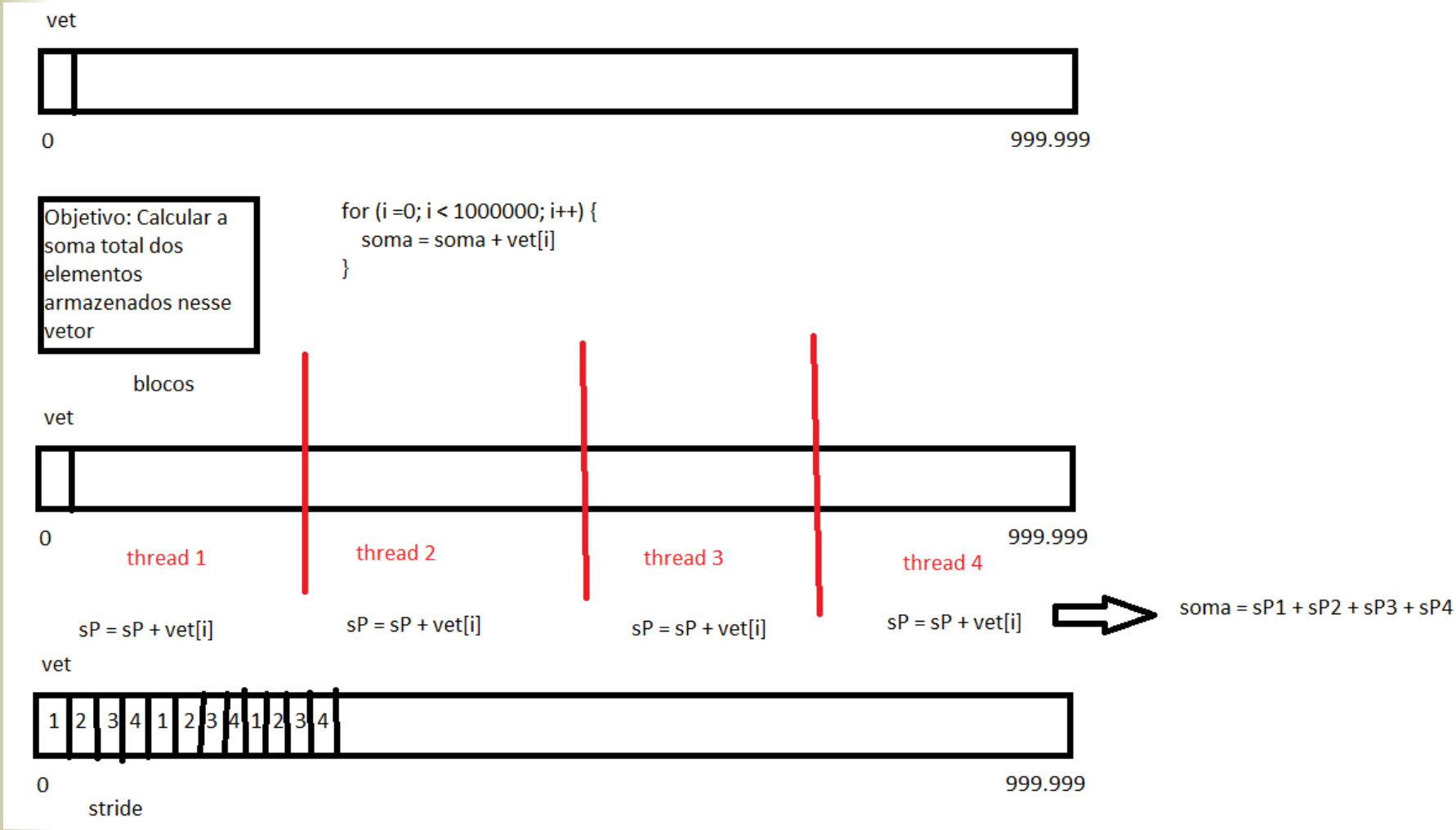
Com mutexes é possível garantir que apenas um *thread* esteja em uma seção crítica por vez. Tem-se então a exclusão mútua e a garantia da consistência do(s) dado(s) compartilhados.

- Após
  do m
  possivelmente bloqueados, seja desbloqueados e possam executar suas seções críticas.

```
int pthread_mutex_unlock(pthread_mutex_t *mp);
```

POSIX Threads: Soma de Vetor Multithread

- Gerar uma soma total de todos os elementos do vetor



Exemplo: Soma elementos vetor com MUTEX (1/3)

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#define TAM_VETOR 1000000
#define NUM_THREADS 4
int elem[TAM_VETOR];
int somaTotal;

pthread_mutex_t mut; //declaração do MUTEX
```

Declarado como global

Exemplo: Soma elementos vetor com MUTEX (2/3)

```
void *SomaElementos(void *id) //funcão executada pelos threads
 {
    int i            = 0;
    int somaParcial = 0;

    for (i = id; i < TAM_VETOR; i = i + NUM_THREADS)
        somaParcial = somaParcial + elem[i];

    pthread_mutex_lock (&mut);
      somaTotal = somaTotal + somaParcial;
    pthread_mutex_unlock (&mut);

    pthread_exit(NULL);

 }
```

Exemplo: Soma elementos vetor com MUTEX (3/3)

```
int main (int argc, char *argv[]) {
     int t, rc;
    /* Inicializa vetor com números randômicos */
    for (t = 0; t < TAM_VETOR; t++)
        elem[t] = rand() % 100;

    //inicialização do mutex com atributos default
    pthread_mutex_init (&mut, NULL);

    /* Cria threads para somar elementos do vetor em paralelo /*
    for(t = 0; t < NUM_THREADS; t++) {
       rc = pthread_create(&thread[t], NULL, SomaElementos, (void *)t);


    }

    /* Aguarda todos os threads terminarem a sua parte do cálculo */
    for(t = 0; t < NUM_THREADS; t++) {
       rc = pthread_join(thread[t], NULL);


    }

    //destrói o mutex
    pthread_mutex_destroy (&mut);

     printf (“Resultado Final %d”, somaTotal); /* Imprime resultado*/
}
```

Sincronização através de SEMÁFOROS (1/3)

- SEMÁFOROS são usados para resolver problemas do tipo Produtor/Consumidor
- Ou seja, quando 2 threads precisam cooperar para resolver um problema e um é mais lento que o outro
  - Ex: Gravação de CD/DVD
- 2 Primitivas
  - `sem_wait` (sleep) e `sem_post` (wakeup)
- É uma variável do tipo `sem_t` que deve ser inicializada antes do uso (e destruída após isto):
  - `sem_init` e
  - `sem_destroy`

Exemplo: Produtor/Consumidor com Semáforos (1/4)

```
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>

#define TAMANHOBUFFER 5
#define TAMANHOMIDIA 20

// inicializacao dos semaforos
pthread_mutex_t  mutexBuffer; // proteger buffer
sem_t cheio, livre;

// inicializacao do buffer
int buffer[TAMANHOBUFFER];
int quantBytesEscritos = 0;
int quantBytesLidos = 0;
```

Exemplo: Produtor/Consumidor com Semáforos (2/4)

```
void *produtor( void *id ) {

    while( quantBytesEscritos < TAMANHOMIDIA ) {
        // se o buffer estiver cheio
        // aguarda um sinal do consumidor
        sem_wait(&livre);

        // Pegar a posicao do buffer que sera modificada
        int posicao = quantBytesEscritos % TAMANHOBUFFER;

        // Modificar o buffer
        pthread_mutex_lock(&mutexBuffer);
        buffer[posicao] = (int) rand(324);
        printf("Info colocada no buffer na posicao %d: %d\n", posicao, buffer[posicao]);
        pthread_mutex_unlock(&mutexBuffer);

        quantBytesEscritos++;

        sem_post(&cheio);
    }
    pthread_exit(NULL);
}
```

Exemplo: Produtor/Consumidor com Semáforos (3/4)

```
void *consumidor(void *id) {

    while (quantBytesLidos < TAMANHOMIDIA) {
        sem_wait(&cheio);

        // Pegar a posicao do buffer que sera lida
        int posicao = quantBytesLidos % TAMANHOBUFFER;

        // Retirar dados do buffer
        pthread_mutex_lock(&mutexBuffer);
        printf("Info retirada do buffer na posicao %d: %d\n", posicao,
   buffer[posicao]);
        pthread_mutex_unlock(&mutexBuffer);

        sleep(1000);     // va dormir por 2 segundos

        quantBytesLidos++;
        sem_post(&livre);
    }
    pthread_exit(NULL);
}
```

Exemplo: Produtor/Consumidor com Semáforos (4/4)

```
int main( int argc, char *argv[] ) {
    pthread_t tConsumidor, tProdutor;

    printf( "Inicializando Semaforos e mutex\n");
    sem_init(&cheio, 0, 0);
    sem_init(&livre, 0, TAMANHOBUFFER);
    pthread_mutex_init( &mutexBuffer, NULL );

    printf( "Criando thread produtor\n");
    int rc = pthread_create(&tProdutor, NULL, produtor, NULL);
    printf( "Criando thread consumidor\n");
    rc = pthread_create(&tConsumidor, NULL, consumidor, NULL);

    // aguarda todos os threads terminarem
    pthread_join(tProdutor, NULL);
    pthread_join(tConsumidor, NULL);

    printf("Processamento terminado: %d - %d\n", quantBytesEscritos, quantBytesLidos);

    pthread_mutex_destroy( &mutexBuffer );
    sem_destroy( &cheio );
    sem_destroy( &livre );
    getchar();
    pthread_exit( NULL );
}
```

Sincronização através de SEMÁFOROS (2/3)

- SEMÁFOROS também podem ser implementados com VARIÁVEIS DE CONDIÇÃO
- Bloquear threads até que uma condição seja satisfeita.
- 2 Primitivas
  - wait (sleep) e
  - signal (wakeup)
- Em Pthreads: É uma variável do tipo `pthread_cond_t` que deve ser inicializada antes do uso (e destruída após isto)

Sincronização com SEMÁFOROS (2/2)

Observação

O acesso à uma variável de condição deve ser controlado por um mutex.

```
int pthread_cond_init (pthread_cond_t *cv,
                       const pthread_condattr_t
```

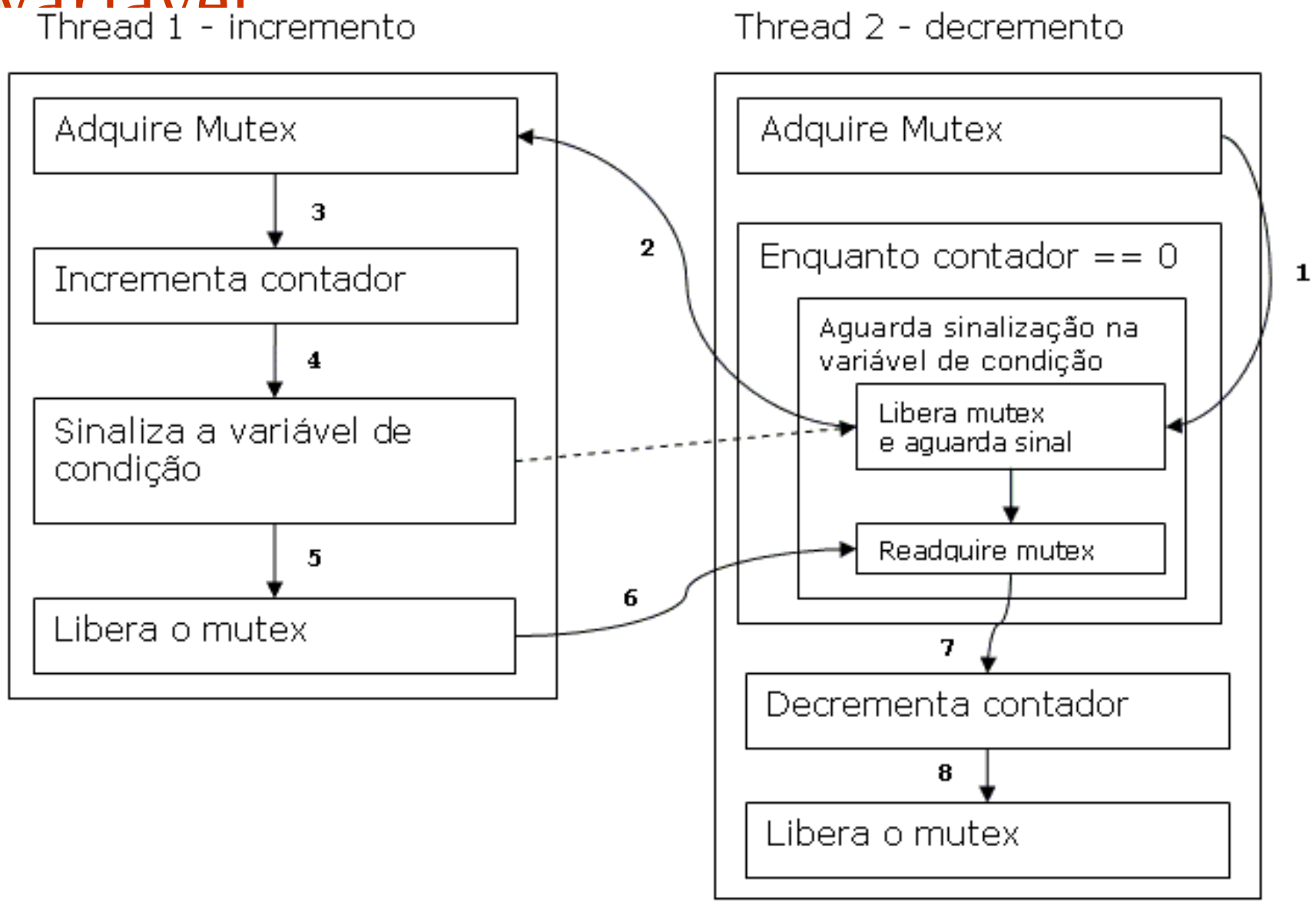
```
int pthread_cond_destroy(pthread_cond_t *cv);
```

É necessário passar o mutex pois ele será liberado no início da função e readquirido antes da função terminar

```
int pthread_cond_wait(pthread_cond_t *cv, pthread_mutex_t *m);
```

```
int pthread_cond_signal(pthread_cond_t *cv);
```

Exemplo: Incremento e Decremento de Variável



Exemplo: Variáveis de Condição 1/3

```
#include <stdio.h>
#include <pthread.h>
#define NUM_THREADSI 10  // número de threads de incremento
#define NUM_THREADSD 15  // número de threads de decremento

int count = 0;
pthread_mutex_t mut;
pthread_cond_t cond;

void *decrement( void *id ) {
    int p = (int *)id;
    // adquire o mutex
    pthread_mutex_lock( &mut );
        // enquanto o contador for zero, espera
        // (caso outro thread seja mais rapido...)
        while( count == 0 )
            pthread_cond_wait( &cond, &mut );
    count--;
    printf( "DEC #%d - %d\n", p, count );
    pthread_mutex_unlock( &mut );  // libera o mutex
    pthread_exit(NULL);
}
```

Exemplo: Variáveis de Condição 2/3

```
 void *increment( void *id ) {
    int p = (int *)id;
    pthread_mutex_lock( &mut );
        count++;
        printf( "INC #%d - %d\n", p, count );
        pthread_cond_signal( &cond );
    pthread_mutex_unlock( &mut );
    pthread_exit(NULL);
}

int main( int argc, char *argv[] ) {
    pthread_t threadsi[NUM_THREADSI];
    pthread_t threadsd[NUM_THREADSD];
    int rc, t;

    pthread_mutex_init( &mut,NULL );
    pthread_cond_init( &cond, NULL );

    for( t = 0; t < NUM_THREADSI; t++ ) {
        printf( "Criando thread incremento %d\n", t );
        rc = pthread_create(&threadsi[t],NULL,increment,(void *)t );
    }
```

Exemplo: Variáveis de Condição 3/3

```
    for( t = 0; t < NUM_THREADSD; t++ ) {
        printf( "Criando thread decremento %d\n", t );
        rc = pthread_create(&threadsd[t],NULL,decrement,(void *)t );
    }
    sleep(2);      // va dormir por 2 segundos
    for( t = 0; t < NUM_THREADSD - NUM_THREADSI; t++ ) {
        printf( "Criando mais threads de incremento %d\n", t );
        rc = pthread_create(&threadsi[t],NULL,increment,(void *)t );
    }
    // aguarda todos os threads terminarem
    for( t = 0; t<NUM_THREADSD;t++) pthread_join(threadsd[t], NULL);
    for( t = 0; t<NUM_THREADSI;t++) pthread_join(threadsi[t],NULL);

    printf( "Valor final: %d\n", count );

    pthread_cond_destroy( &cond );
    pthread_mutex_destroy( &mut );
    pthread_exit( NULL );
}
```

Exemplo: Saída

Exemplo com 2 threads de incremento e 4 de decremento

```
Criando thread incremento 0
INC #0 - 1
Criando thread incremento 1
INC #1 - 2
Criando thread decremento 0
DEC #0 - 1
Criando thread decremento 1
DEC #1 - 0
Criando thread decremento 2
Criando thread decremento 3
Criando mais threads de incremento 0
INC #0 - 1
Criando mais threads de incremento 1
INC #1 - 2
DEC #2 - 1
DEC #3 - 0
Valor final: 0
```

Referências bibliográficas

GUBITOSO, M. D. *Introdução ao processamento paralelo e distribuído*. Disponível em <http://www.ime.usp.br/~gubi>. Acesso em: 13 setembro 2003.

IEEE Computer Society. *A Backgrounder on IEEE Std 1003.1, 2003 Edition*. Copyright 2003 por IEEE e Open Group. Disponível em: <http://www.opengroup.org/austin/papers/backgrounder.html>. Acessado em: 01 out 2003.

LLNL - LAURENCE LIVERMORE NATIONAL LABORATORY. *POSIX Threads programming*. Copyright 2003. Disponível em: <http://www.llnl.gov/computing/tutorials/workshops/workshop/pthreads/MAIN.html>. Acessado em: 28 set 2003.

MOREIRA, D. A. *Operating System*. Material da disciplina sistemas operacionais ICMC-USP. Copyright 2003. Disponível em <http://java.icmc.usp.br/~os_course>. Acesso em: 01 outubro 2003.

MSDN. *Process and Threads.* Copyright 2003. Disponível em <http://msdn.microsoft.com/library>. Acesso em: 05 outubro 2003.

SUN Microsystems. *Multithreaded Programming Guide.* Califórnia:SUN Microsystems, 1998. 361p..

TANENBAUM, A. S. *Modern Operating Systems*. 2ed. Upper Saddle River, N. J.: Prentice Hall, c2001. 951p.