

Introdução POSIX Threads

Sistemas Operacionais

João Vicente Ferreira Lima

Departamento de Linguagens e Sistemas de Computação
Universidade Federal de Santa Maria
`jvlima@inf.ufsm.br`

2020/2

- 1 POSIX Threads
- 2 Mutexes
- 3 Variáveis de condição
- 4 Sincronização
- 5 Thread Local Storage

- 1 POSIX Threads
- 2 Mutexes
- 3 Variáveis de condição
- 4 Sincronização
- 5 Thread Local Storage

- Padrão POSIX para threads definido em 1995 em C.
- Implementado por diversos sistemas UNIX-Like:
 - FreeBSD, NetBSD, OpenBSD, Linux, MacOS, Android, Solaris, etc.
- Categorizado em 4 grupos:
 - Gerenciamento de threads
 - Mutexes
 - Variáveis de condição
 - Sincronização

- Principais funções

- `pthread_create`
- `pthread_exit`
- `pthread_join`
- `pthread_yield`
- `pthread_cancel`

Criando threads

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#define NUM_THREADS      5

int main(int argc, char *argv[])
{
    pthread_t threads[NUM_THREADS];
    int rc;
    long t;
    for( t = 0; t < NUM_THREADS; t++ ) {
        rc = pthread_create( &threads[t], NULL, PrintHello ,
                             (void *)t );
        // testa rc
    }

    pthread_exit(NULL); // return 0
}
```

Criando threads

```
void *PrintHello(void *threadid)
{
    long tid;
    tid = (long)threadid;
    printf("Hello World! It's me, thread #%ld!\n", tid);
    pthread_exit(NULL);
}
```

Criar e esperar

```
int main (int argc, char *argv[])
{
    pthread_t thread[NUM_THREADS];
    long t;
    void *status;

    for(t=0; t<NUM_THREADS; t++) {
        pthread_create(&thread[t], NULL, PrintHello,
            (void *)t);
    }

    for(t=0; t<NUM_THREADS; t++)
        pthread_join(thread[t], &status);

    pthread_exit(NULL);
}
```


- 1 POSIX Threads
- 2 Mutexes**
- 3 Variáveis de condição
- 4 Sincronização
- 5 Thread Local Storage

- São variáveis de trava do tipo dormir/acordar
- Variações são de testar, ou travas leituras/escrita
- Funções:
 - `pthread_mutex_init`
 - `pthread_mutex_destroy`
 - `pthread_mutex_lock`
 - `pthread_mutex_unlock`
 - `pthread_mutex_trylock`

Exemplo mutex

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

pthread_mutex_t m; // lock
int main(void) {
    pthread_t tid1, tid2;

    pthread_mutex_init(&m, NULL);

    pthread_create(&tid1, NULL, countgold, NULL);
    pthread_create(&tid2, NULL, countgold, NULL);
    // espera as duas threads
    pthread_join(tid1, NULL);
    pthread_join(tid2, NULL);

    pthread_mutex_destroy(&m);
    printf("The sum is %d\n", sum);

    return 0;
}
```

Exemplo mutex

```
int sum = 0; // global

void *countgold(void *param) {
    int i;

    // regioao critica
    pthread_mutex_lock(&m);

    for (i = 0; i < 100000000; i++) {
        sum += 1;
    }

    pthread_mutex_unlock(&m);
    return NULL;
}
```

- 1 POSIX Threads
- 2 Mutexes
- 3 Variáveis de condição**
- 4 Sincronização
- 5 Thread Local Storage

- Variáveis que permitem bloquear/esperar um evento
- Sinalizar uma thread ou várias
- Funções importantes:
 - `pthread_cond_wait`
 - `pthread_cond_signal`
 - `pthread_cond_broadcast`

Variáveis de condição

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

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

```
#include <stdlib.h>
```

```
#define NUM_THREADS 3
```

```
#define TCOUNT 10
```

```
#define COUNT_LIMIT 12
```

```
int      count = 0;
```

```
pthread_mutex_t count_mutex;
```

```
pthread_cond_t count_threshold_cv;
```

Variáveis de condição

```
int main(int argc, char *argv[])
{
    int i, rc;
    long t1=1, t2=2, t3=3;
    pthread_t threads[3];
    pthread_mutex_init(&count_mutex, NULL);
    pthread_cond_init (&count_threshold_cv, NULL);

    pthread_create(&threads[0], NULL, watch_count, (void *)t1);
    pthread_create(&threads[1], NULL, inc_count, (void *)t2);
    pthread_create(&threads[2], NULL, inc_count, (void *)t3);
    for (i = 0; i < NUM_THREADS; i++)
        pthread_join(threads[i], NULL);

    printf ("Joined with %d threads. Final value = %d. Done.\n",
        NUM_THREADS, count);

    pthread_mutex_destroy(&count_mutex);
    pthread_cond_destroy(&count_threshold_cv);
    pthread_exit (NULL);
}
```


Variáveis de condição

```
void *watch_count(void *t)
{
    long my_id = (long)t;

    pthread_mutex_lock(&count_mutex);
    while (count < COUNT_LIMIT) {
        // esperando sinal
        pthread_cond_wait(&count_threshold_cv , &count_mutex);
    }

    count += 125;

    pthread_mutex_unlock(&count_mutex);
    pthread_exit(NULL);
}
```

Variáveis de condição

```
void *inc_count(void *t)
{
    int i;
    long my_id = (long)t;

    for (i=0; i < TCOUNT; i++) {
        pthread_mutex_lock(&count_mutex);
        count++;

        if (count == COUNT_LIMIT)
            // limite: avisa thread que monitora
            pthread_cond_signal(&count_threshold_cv);

        pthread_mutex_unlock(&count_mutex);
        sleep(1);
    }
    pthread_exit(NULL);
}
```

- 1 POSIX Threads
- 2 Mutexes
- 3 Variáveis de condição
- 4 Sincronização**
- 5 Thread Local Storage

- Pthreads suportam barreiras de Sincronização
- As thread esperam todas chegarem até certo ponto para continuar
- Funções:
 - `pthread_barrier_init`
 - `pthread_barrier_destroy`
 - `pthread_barrier_wait`

```
pthread_barrier_t barreira;  
  
int main(int argc, char *argv[])  
{  
    pthread_t threads[NUM_THREADS];  
    long t;  
  
    pthread_barrier_init(&barreira, NULL, NUM_THREADS);  
  
    for( t= 0; t < NUM_THREADS; t++ ) {  
        rc = pthread_create( &threads[t], NULL, PrintHello ,  
                             (void *)t );  
  
        // testa rc  
    }  
  
    pthread_barrier_destroy(&barreira);  
    pthread_exit(NULL); // return 0  
}
```

```
void *PrintHello(void *threadid)
{
    long tid;
    tid = (long)threadid;
    printf("Thread_#%ld_antes_da_barreira!\n", tid);

    pthread_barrier_wait(&barreira);

    printf("Thread_#%ld_passou!\n", tid);
    pthread_exit(NULL);
}
```

- 1 POSIX Threads
- 2 Mutexes
- 3 Variáveis de condição
- 4 Sincronização
- 5 Thread Local Storage**

- Pthreads permite ter dados globais mas específicos para cada thread.
- As funções são:
 - `pthread_key_create`
 - `pthread_getspecific`
 - `pthread_setspecific`
 - `pthread_key_delete`

Exemplo TLS

```
static pthread_key_t  chave;
static pthread_once_t inicia_once = PTHREAD_ONCE_INIT;
// esta e chamada apenas quando destroi o dado
void libera_no_fim (void *buffer){
    free(buffer);
}
// esta e chamada por apenas uma thread
void inicializa (void){
    pthread_key_create(&chave, libera_no_fim);
}
// varias threads chamam esta funcao
void *aloca(void *param) {
    char* texto;
    pthread_once(&inicia_once, inicializa); // apenas uma thread
    texto = pthread_getspecific(chave);
    if(texto == NULL){
        texto = (char*)malloc( sizeof(char)*100 );
        pthread_setspecific(chave, texto);
    }
    pthread_exit(NULL);
}
```

Outra alternativa é o `__thread` do C11.

```
__thread char *texto = NULL;
```

```
// varias threads chamam esta funcao
```

```
void *aloca(void *param) {
```

```
    char* texto;
```

```
    if(texto == NULL){
```

```
        texto = (char*)malloc( sizeof(char)*100 );
```

```
        // cada thread tem sua copia
```

```
    }
```

```
    pthread_exit(NULL);
```

```
}
```

GCC possui funções built-in do compilador como abaixo:

```
type __sync_fetch_and_add (type *ptr, type value, ...)  
type __sync_fetch_and_sub (type *ptr, type value, ...)  
type __sync_fetch_and_or (type *ptr, type value, ...)  
type __sync_fetch_and_and (type *ptr, type value, ...)  
type __sync_fetch_and_xor (type *ptr, type value, ...)  
type __sync_fetch_and_nand (type *ptr, type value, ...)
```

```
type __sync_add_and_fetch (type *ptr, type value, ...)  
type __sync_sub_and_fetch (type *ptr, type value, ...)  
type __sync_or_and_fetch (type *ptr, type value, ...)  
type __sync_and_and_fetch (type *ptr, type value, ...)  
type __sync_xor_and_fetch (type *ptr, type value, ...)  
type __sync_nand_and_fetch (type *ptr, type value, ...)
```

GCC possui funções built-in do compilador como abaixo:

```
bool __sync_bool_compare_and_swap (type *ptr ,  
    type oldval type newval, ...)
```

```
type __sync_val_compare_and_swap (type *ptr ,  
    type oldval type newval, ...)
```

```
__sync_synchronize (...)
```

```
void __sync_lock_release (type *ptr , ...)
```

Exemplo de soma com GCC atomics

```
int sum = 0; // global

void *countgold(void *param) {
    int i;

    // regioao critica
    for (i = 0; i < 100000000; i++)
        __sync_fetch_and_add( &sum, 1 ) ;

    return NULL;
}
```

