
THREADS

Monitoria de Infraestrutura de Software

Introdução

- Mini processos, compartilhando recursos e endereços de memória
- Têm um mesmo objetivo, trabalham juntos (processos competem)
- Um exemplo?
- Várias linguagens têm suporte; em C temos a biblioteca **pthread.h**
- Pthreads é um padrão para UNIX
- Implementa funções úteis, como gerenciamento de threads, mutexes, variáveis de condição e barreiras
- MAIS: https://computing.llnl.gov/tutorials/pthreads/

PTHREAD.H

Chamada de thread	Descrição
pthread_create	Cria um novo thread
pthread_exit	Conclui a chamada de thread
pthread_join	Espera que um thread específico seja abandonado
pthread_yield	Libera a CPU para que outro thread seja executado
pthread_attr_init	Cria e inicializa uma estrutura de atributos do thread
pthread_attr_destroy	Remove uma estrutura de atributos do thread

HELLO WORLD

```c
#include <pthread.h>
#include <stdio.h>
void *PrintHello(void *threadid){
  printf("Olá, mundo!\n");
  pthread_exit(NULL);
}
int main (int argc, char *argv[]){
  pthread_t thread;
  int rc;
  rc = pthread_create(&thread, NULL, PrintHello, NULL);
  if (rc){
    printf("ERRO; código de retorno é %d\n", rc);
    exit(-1);
  }
  pthread_exit(NULL);
}
```

PTHREAD.H

- Compilando:
  - **$ gcc -lpthread prog.c -o prog**
  - **$ gcc -pthread prog.c -o prog**
  - **$ gcc -pthread -o prog prog.c**
  - **$ gcc prog.c -o prog -pthread**
- Executando:
  - **$ ./prog**

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

//#define NUM_THREADS 5

void *funcao(void *threadId){ //funcao executada pelas threads. o parametro é um ponteiro do tipo void
    long tid;
    tid = (long)threadId;
    printf ("Hello #%ld!\n", tid);
    pthread_exit(NULL); //encerra a thread
}

int main(){
    int NUM_THREADS;
    printf("Defina o número de threads: \n");
    scanf("%d", &NUM_THREADS);

    pthread_t thread[NUM_THREADS]; // pthread_t identifica a thread; criamos um array de tamanho 5
    int contador;
    long i;

    for (i=0; i<NUM_THREADS; i++){
        printf ("Estou na main, criando a thread de número %ld\n", i);
        contador = pthread_create(&thread[i], NULL, *funcao, (void *)i); //funcao que cria a thread.
        //os parametros sao: (espaço que guarda infos da thread, propriedades (null é default), funcao

        //esse if vai ajudar se der algum erro na execução do pthread_create
        if(contador!=0){
            printf ("Deu ruim na thread %d\n", contador);
            exit(-1);
        }

    }
}
```

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

void *print_message_function( void *ptr )
{
     char *message;
     message = (char *) ptr;
     printf("%s \n", message);
}

int main()
{
     pthread_t thread1, thread2;
     char message1 = "Thread 1";
     char message2 = "Thread 2";
     int  iret1, iret2;

     /* Create independent threads each of which will execute function */

     iret1 = pthread_create(&thread1, NULL, *print_message_function, (void*)message1);
     iret2 = pthread_create(&thread2, NULL, *print_message_function, (void*)message2);

     /* Wait till threads are complete before main continues. Unless we  */
     /* wait we run the risk of executing an exit which will terminate   */
     /* the process and all threads before the threads have completed.   */

     pthread_join(thread1, NULL);
     pthread_join(thread2, NULL);

     printf("Thread 1 returns: %d\n",iret1);
     printf("Thread 2 returns: %d\n",iret2);
     exit(-1);
}
```

PTHREAD.H

Routine Prefix	Functional Group
pthread_	Threads themselves and miscellaneous subroutines
pthread_attr_	Thread attributes objects
pthread_mutex_	Mutexes
pthread_mutexattr_	Mutex attributes objects.
pthread_cond_	Condition variables
pthread_condattr_	Condition attributes objects
pthread_key_	Thread-specific data keys
pthread_rwlock_	Read/write locks
pthread_barrier_	Synchronization barriers

Concorrência

-Mutexes

-Variáveis condicionais (cond)

-Barrier

Mutex

- Garantem acesso exclusivo à uma região crítica
- Criação e inicialização:

  Estática:

  - `pthread_mutex_t nome_mutex = PTHREAD_MUTEX_INITIALIZER;`

  Dinâmica:

  - `pthread_mutex_t mymutex;`
  - `…`
  - `pthread_mutex_init(&mymutex, const pthread_mutexattr_t *attr);`

Mutex

Gerenciamento:

- `int pthread_mutex_lock( pthread_mutex_t *mutex);`
- `int pthread_mutex_trylock( pthread_mutex_t *mutex);`
- `int pthread_mutex_unlock( pthread_mutex_t *mutex);`
- `int pthread_mutex_destroy( pthread_mutex_t *mutex);`

Documents

mutex.c

```c
#include<pthread.h>
#include<stdio.h>
#include<stdlib.h>
#define Num 5
int posicao = 0;

void *showThread(void *id){
    //As linhas abaixo são uma região crítica sem controle de acesso

    posicao ++;
    printf("olá da thread: %d\n",posicao);

    pthread_exit(NULL);
}
int main(){
    long i = 0;
    pthread_t threads[Num];

    for(i=0;i<Num;i++){
        pthread_create(&threads[i], NULL, showThread, NULL);
    }
    for(i=0;i<Num;i++){
        pthread_join(threads[i],NULL);
    }
    return 0;
}

```

Linha 5, Coluna 17 INSERIR Tabulações emuladas: 4 (8) UTF-8

Search and Replace

Documents

mutex.c

```c
#include<pthread.h>
#include<stdio.h>
#include<stdlib.h>
#define Num 5
int posicao = 0;
//criação do Mutex
pthread_mutex_t myMutex = PTHREAD_MUTEX_INITIALIZER;

void *showThread(void *id){
    //As linhas abaixo são uma região crítica com controle de acesso
    pthread_mutex_trylock(&myMutex);
        posicao ++;
        printf("olá da thread: %d\n",posicao);
    pthread_mutex_unlock(&myMutex);
    pthread_exit(NULL);
}
int main(){
    long i = 0;
    pthread_t threads[Num];

    for(i=0;i<Num;i++){
        pthread_create(&threads[i], NULL, showThread, NULL);
    }
    for(i=0;i<Num;i++){
        pthread_join(threads[i],NULL);
    }
    return 0;
}
```

Linha 29, Coluna 1 INSERIR Tabulações emuladas: 4 (8) UTF-8 C

Search and Replace

Variáveis de condição

- Permite que uma thread continue após determinada condição
- Evita checagem contínua de dados
- Variáveis condicionais trabalham junto com mutexes

Criação e destruição

- Estática:
  - `pthread_cond_t myCv = PTHREAD_COND_INITIALIZER;`
- Dinâmica:
  - `pthread_cond_t myCv;`
  - `pthread_cond_init(&myCv,NULL);`
- Destruição:
  - `pthread_cond_destroy(&myCv);`

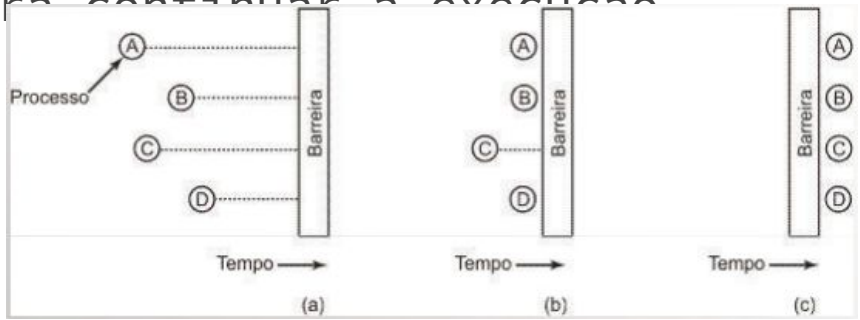
Gerenciamento de variáveis condicionais

- - Esperar que certa condição torne-se verdadeira
  - `pthread_cond_wait(cond_var, mutex);`
- sinalizar que uma condição tornou-se verdadeira
  - `pthread_cond_signal(cond_var) ;`
  - `pthread_cond_broadcast(cond_var);`

```c
#include <pthread.h>
#define TCOUNT 10
#define WATCH_COUNT 12
int count = 0;
pthread_mutex_t count_mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t count_threshold_cv = PTHREAD_COND_INITIALIZER;
int  thread_ids[3] = {0,1,2};
extern int
main(void)
{
     int      i;
     pthread_t threads[3];
     pthread_create(&threads[0],NULL,inc_count, &thread_ids[0]);
     pthread_create(&threads[1],NULL,inc_count, &thread_ids[1]);
     pthread_create(&threads[2],NULL,watch_count, &thread_ids[2]);
     for (i = 0; i < 3; i++) {
               pthread_join(threads[i], NULL);
     }
     return 0;
}
void watch_count(int *idp)
{
     pthread_mutex_lock(&count_mutex)
     while (count <= WATCH_COUNT) {
               pthread_cond_wait(&count_threshold_cv,
                                 &count_mutex);
               printf("watch_count(): Thread %d,Count is %d\n",
                    *idp, count);
     }
     pthread_mutex_unlock(&count_mutex);
}
void inc_count(int *idp)
{
     for (i =0; i < TCOUNT; i++) {
               pthread_mutex_lock(&count_mutex);
               count++;
               printf("inc_count(): Thread %d, old count %d,\
                    new count %d\n", *idp, count - 1, count );
               if (count == WATCH_COUNT)
                    pthread_cond_signal(&count_threshold_cv);
               pthread_mutex_unlock(&count_mutex);
     }
```

Barreiras

- Mecanismo de sincronização entre threads
- threads executam até chegarem à barreira, quando isso ocorre, elas “adormecem”
- Quando todas as threads “alcançam” a barreira, elas são acordadas para continuar a execução



Gerenciamento de Barriers

- Na inicialização deve ser especificado o número de threads
- Inicialização:
  - `int pthread_barrier_init(pthread_barrier_t *barrier, const pthread_barrierattr_t *restrict attr, unsigned count);`
- Destruição:
  - `int pthread_barrier_destroy(pthread_barrier_t *barrier);`
- Aguardar outras threads:
  - `int pthread_barrier_wait(pthread_barrier_t *barrier);`