Aula 5 - Pthreads em C

Sistemas Operacionais
Ciência da Computação
IFB - Campus Taguatinga

Professor João Victor de A. Oliveira

Hoje

- Funcionalidades da Biblioteca Pthreads a serem exploradas
  - pthread_t
  - pthread_create
  - pthread_exit
  - pthread_self
  - pthread_detach
  - pthread_join
  - pthread_mutex_init
  - pthread_mutex_lock
  - pthread_mutex_unlock
  - pthread_mutex_destroy

Meu primeiro programa multithread :)

- Disponível em **thread1.c**
- Como compilar:

gcc thread1.c **-pthread** -o thread1

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


void* PrintHello(void* data)
{
    int *my_data = (int *)data;

    pthread_detach(pthread_self());
    printf("Ola, sou a nova thread! Recebi o dado %d\n", *my_data);
    pthread_exit(NULL);
}


int main(int argc, char* argv[])
{
    int         rc;
    pthread_t   thread_id;
    int         t = 10;

    rc = pthread_create(&thread_id, NULL, PrintHello, (void *) &t);
    if(rc)  /* Erro na criação da thread */
    {
        printf("\n Erro! Código: %d \n", rc);
        exit(1);
    }
    printf("\n Foi criada a thread (%lu) ... \n", thread_id);

    pthread_exit(NULL);

    return 0;
}
```

Principais funcionalidades

- Tipo de variável: **pthread_t**
- **Definida em** /usr/include/x86_64-linux-gnu/bits/pthreadtypes.h:

typedef unsigned long int pthread_t;

- usada para identificar uma thread.

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


void* PrintHello(void* data)
{
    int *my_data = (int *)data;

    pthread_detach(pthread_self());
    printf("Ola, sou a nova thread! Recebi o dado %d\n", *my_data);
    pthread_exit(NULL);
}


int main(int argc, char* argv[])
{
    int         rc;
    pthread_t   thread_id;
    int         t = 10;

    rc = pthread_create(&thread_id, NULL, PrintHello, (void *) &t);
    if(rc)  /* Erro na criação da thread */
    {
        printf("\n Erro! Código: %d \n", rc);
        exit(1);
    }
    printf("\n Foi criada a thread (%lu) ... \n", thread_id);

    pthread_exit(NULL);

    return 0;
}
```

Principais funcionalidades

- **pthread_create**
- Possui 4 argumentos
  1. **pthread_t *thread**
     - Identificador da thread criada
  2. **const pthread_attr_t *attr**
     - Estrutura usada para definir alguns atributos da thread (por hora deixemos como NULL)
  3. **void *(*start_routine) (void *)**
     - nome da subrotina a ser executada pela thread
  4. **void *arg**
     - Argumentos da subrotina a ser executada

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


void* PrintHello(void* data)
{
    int *my_data = (int *)data;

    pthread_detach(pthread_self());
    printf("Ola, sou a nova thread! Recebi o dado %d\n", *my_data);
    pthread_exit(NULL);
}


int main(int argc, char* argv[])
{
    int         rc;
    pthread_t   thread_id;
    int         t = 10;

    rc = pthread_create(&thread_id, NULL, PrintHello, (void *) &t);
    if(rc)  /* Erro na criação da thread */
    {
        printf("\n Erro! Código: %d \n", rc);
        exit(1);
    }
    printf("\n Foi criada a thread (%lu) ... \n", thread_id);

    pthread_exit(NULL);

    return 0;
}
```

Principais funcionalidades

- **pthread_self**
  - Retorna o id da thread em execução
- **pthread_detach**
  - Caso em que a thread ao ser encerrada libera seus recursos automaticamente
- **pthread_exit**
  - Finaliza a thread
  - Note que a thread principal precisa ser encerrada!

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


void* PrintHello(void* data)
{
    int *my_data = (int *)data;

    pthread_detach(pthread_self());
    printf("Ola, sou a nova thread! Recebi o dado %d\n", *my_data);
    pthread_exit(NULL);
}


int main(int argc, char* argv[])
{
    int         rc;
    pthread_t   thread_id;
    int         t = 10;

    rc = pthread_create(&thread_id, NULL, PrintHello, (void *) &t);
    if(rc)  /* Erro na criação da thread */
    {
        printf("\n Erro! Código: %d \n", rc);
        exit(1);
    }
    printf("\n Foi criada a thread (%lu) ... \n", thread_id);

    pthread_exit(NULL);

    return 0;
}
```

thread2.c

- Note que podemos usar **pthread_self** para identificar até mesmo a thread principal
- O que acontecerá se comentarmos o **pthread_exit da** função principal?
- Neste programa de duas threads, se quisermos dar uma ordem de execução o que podemos usar?
  - Sleep seria suficiente?

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

void* PrintHello(void* data)
{
    int *my_data = (int *)data;
    pthread_t tid;
    tid = pthread_self();
    sleep(2)
    pthread_detach(pthread_self());
    printf("Sou a thread (%lu) e recebi o dado %d\n",tid, *my_data);
    pthread_exit(NULL);
}

int main(int argc, char* argv[])
{
    int         rc;
    pthread_t   thread_id;
    int         t = 10;
    pthread_t tid;

    tid = pthread_self();
    rc = pthread_create(&thread_id, NULL, PrintHello, (void *) &t);
    printf("\nSou a thread (%lu) e criei a thread (%lu)\n",tid, thread_id);

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

thread3.c

- **Pthread_join**
  - Aguarda uma thread ser encerrada
  - Similar à função wait nos processos
  - Dois parâmetros
    1. pthread_t thread
    2. void **retval

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

void* PrintHello(void* data)
{
    pthread_t *tpai = (pthread_t *)data;
    pthread_t tid;

    tid = pthread_self();
    pthread_join(*tpai, NULL);

    printf("Sou a thread (%lu) e fui criado por (%lu)\n",tid,*tpai);
    pthread_exit(NULL);
}


int main(int argc, char* argv[])
{
    pthread_t  thread_id;
    pthread_t tid;

    tid = pthread_self();
    pthread_create(&thread_id, NULL, PrintHello, (void *) &tid);
    //pthread_join(thread_id, NULL);
    printf("\nSou a thread (%lu) e criei a thread (%lu) ... \n",tid, thread_id);

    pthread_exit(NULL);

    return 0;
}
```

thread4.c

- Podemos criar várias threads a partir da thread principal
- Neste exemplo podemos usar **thread_join** na thread principal para esperar que todas threads tenham finalizado.

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

#define N_THREADS 10
int count = 0;

void* contador(void *id)
{
    int * i = (int *) id ;

    pthread_detach(pthread_self());
    printf("Thread %d: count = %d\n",*i, ++count);
    pthread_exit(NULL);
}


int main(int argc, char* argv[])
{
    pthread_t  thread_ids[N_THREADS];
    int i;
    for (i=0;i<N_THREADS; i++){
        pthread_create(&thread_ids[i], NULL, contador, &i);
    }
    for (int j=0;j<N_THREADS; i++){
        pthread_join(thread_ids[j], NULL);
    }
    printf("Fim!\n");
    pthread_exit(NULL);

    return 0;
}
```

Exclusão mútua em threads

- Note que no código anterior, estávamos acessando uma variável global em diferentes threads
  - Pode gerar condição de corrida!
    - Precisamos garantir a exclusão mútua... usando semáforos (ou melhor... Mutexes)
      - pthread_mutex_init
      - pthread_mutex_lock
      - pthread_mutex_unlock
      - pthread_mutex_destroy

thread5.c

- Note que a saída deste programa pode ser bastante inesperada para quem costuma programar com uma única thread....

```
Thread 2: count = 1
Thread 5: count = 3
Thread 5: count = 2
Thread 5: count = 4
Thread 5: count = 5
Thread 5: count = 6
Thread 5: count = 7
Thread 5: count = 8
Thread 5: count = 9
Thread 5: count = 10
Fim!
```

```c
#define N_THREADS 5
int count = 0;

void* contador(void *id)
{
    int * i = (int *) id;
    unsigned long delay = 0xCAFEBABE;

    count = count + 1;
    printf("Thread %d: count = %d\n",*i, count);
    for (int j=0; j<delay; j++);
    count = count + 1;
    printf("Thread %d: count = %d\n",*i, count);

    pthread_exit(NULL);
}


int main(int argc, char* argv[])
{
    pthread_t  thread_ids[N_THREADS];
    int i;
    for (i=0;i<N_THREADS; i++){
        pthread_create(&thread_ids[i], NULL, contador, &i);
    }
    for (int j=0;j<N_THREADS; j++){
        pthread_join(thread_ids[j], NULL);
    }
    printf("Fim!\n");

    pthread_exit(NULL);
}
```

thread6.c

- Neste código garantimos a exclusão mútua para a **variável global count**.
- Usamos as seguintes funcionalidades
  - **pthread_mutex_t**
  - pthread_mutex_init
  - pthread_mutex_destroy
  - pthread_mutex_lock
  - pthread_mutex_unlock

```c
#define N_THREADS 5
int count = 0;
pthread_mutex_t lock;

void* contador(void *id)
{
    int * i = (int *) id;
    unsigned long delay = 0xCAFEBABE;

    pthread_mutex_lock(&lock);

    count = count + 1;
    printf("Thread %d: count = %d\n",*i, count);
    for (int j=0; j<delay; j++);
    count = count + 1;
    printf("Thread %d: count = %d\n",*i, count);

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

```c
int main(int argc, char* argv[])
{
    pthread_t  thread_ids[N_THREADS];
    int i;

    if (pthread_mutex_init(&lock, NULL) != 0)
    {
        printf("\n Erro na inicialização do mutex\n");
        return 1;
    }

    for (i=0;i<N_THREADS; i++){
        pthread_create(&thread_ids[i], NULL, contador, &i);
    }
    for (int j=0;j<N_THREADS; j++){
        pthread_join(thread_ids[j], NULL);
    }
    printf("Fim!\n");
    pthread_mutex_destroy(&lock);
    pthread_exit(NULL);
}
```

thread6.c

- Note que ainda nosso resultado é um pouco estranho....
- **Exercício:** Garanta a exclusão mútua para a variável i.

```
Thread 2: count = 1
Thread 5: count = 2
Thread 5: count = 3
Thread 5: count = 4
Thread 5: count = 5
Thread 5: count = 6
Thread 5: count = 7
Thread 5: count = 8
Thread 5: count = 9
Thread 5: count = 10
Fim!
```

```c
#define N_THREADS 5
int count = 0;
pthread_mutex_t lock;

void* contador(void *id)
{
    int * i = (int *) id;
    unsigned long delay = 0xCAFEBABE;

    pthread_mutex_lock(&lock);

    count = count + 1;
    printf("Thread %d: count = %d\n",*i, count);
    for (int j=0; j<delay; j++);
    count = count + 1;
    printf("Thread %d: count = %d\n",*i, count);

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

```c
int main(int argc, char* argv[])
{
    pthread_t  thread_ids[N_THREADS];
    int i;

    if (pthread_mutex_init(&lock, NULL) != 0)
    {
        printf("\n Erro na inicialização do mutex\n");
        return 1;
    }

    for (i=0;i<N_THREADS; i++){
        pthread_create(&thread_ids[i], NULL, contador, &i);
    }
    for (int j=0;j<N_THREADS; j++){
        pthread_join(thread_ids[j], NULL);
    }
    printf("Fim!\n");
    pthread_mutex_destroy(&lock);
    pthread_exit(NULL);
}
```

Variáveis de Condição

- São primitivas que permitem a sincronização entre threads
  - Duas operações essenciais: **wait** e **signal**
  - **wait:** libera o mutex previamente usado e bloqueia a thread até que receba um determinado sinal.

```
int pthread_cond_wait(pthread_cond_t *restrict cond,pthread_mutex_t *restrict mutex);
```

  - **signal:** envia um sinal a uma thread bloqueada, de forma a despertá-la

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

- **Criando e destruindo de variáveis de condição:**
  - **Criar**: `pthread_cond_t cond;`

    `int pthread_cond_init (pthread_cond_t *cond, pthread_condattr_t *attr)`
  - **Destruir**: `int pthread_cond_destroy (pthread_cond_t *cond)`

Produtor X Consumidor (1/)

- thread7.c

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

#define N 10

pthread_mutex_t lock;
pthread_cond_t empty, full;
int buffer[N];
int qtd_buffer = 0;


void* produtor(void *id)
{
   int item = 1;

   while (1){
    sleep(rand() % 5); //Produzindo o item;
    pthread_mutex_lock(&lock);
    if (qtd_buffer ==  N) pthread_cond_wait(&full,&lock);
    buffer[qtd_buffer] = item;
    qtd_buffer++;
    printf("Produtor: adicionado %d item(s)\n",qtd_buffer);
    if (qtd_buffer == 1) pthread_cond_signal(&empty);
    pthread_mutex_unlock(&lock);
   }

   pthread_exit(NULL);
}
```

Produtor X Consumidor (2/3)

```
void* consumidor(void *id)
{
    int item;

    while(1){
        sleep(rand() % 5);

        pthread_mutex_lock(&lock);
        if (qtd_buffer ==  0) pthread_cond_wait(&empty,&lock);
        item = buffer[qtd_buffer-1];
        qtd_buffer--;
        printf("Consumidor: consumido um item. %d item(s) restantes\n",qtd_buffer);
        if (qtd_buffer == N-1) pthread_cond_signal(&full);

        pthread_mutex_unlock(&lock);

    }


    pthread_exit(NULL);
}
```

Produtor X Consumidor (3/3)

```c
int main(int argc, char* argv[])
{
    pthread_t t_produtor,t_consumidor;
    srand(time(NULL));

    pthread_mutex_init(&lock, NULL);
    pthread_cond_init(&empty,NULL);
    pthread_cond_init(&full,NULL);

    pthread_create(&t_produtor, NULL,produtor ,NULL);
    pthread_create(&t_consumidor, NULL,consumidor ,NULL);

    pthread_join(t_produtor, NULL);
    pthread_join(t_consumidor, NULL);


    pthread_mutex_destroy(&lock);
    pthread_cond_destroy(&empty);
    pthread_cond_destroy(&full);

    pthread_exit(NULL);

}
```

Links úteis (Referências)

- https://www.geeksforgeeks.org/thread-functions-in-c-c/?ref=lbp
- https://www.thegeekstuff.com/2012/05/c-mutex-examples/
- http://www.csc.villanova.edu/~mdamian/threads/posixthreads.html
- https://www.ibm.com/docs/en/aix/7.2?topic=programming-using-condition-variables