MONITORIA DE INFRAESTRUTURA DE SOFTWARE

POSIX Threads (PThreads)

LISTA DE E-MAIL

if677cc+subscribe@googlegroups.com

REPOSITÓRIO

github.com/pvsaragao/if677cc

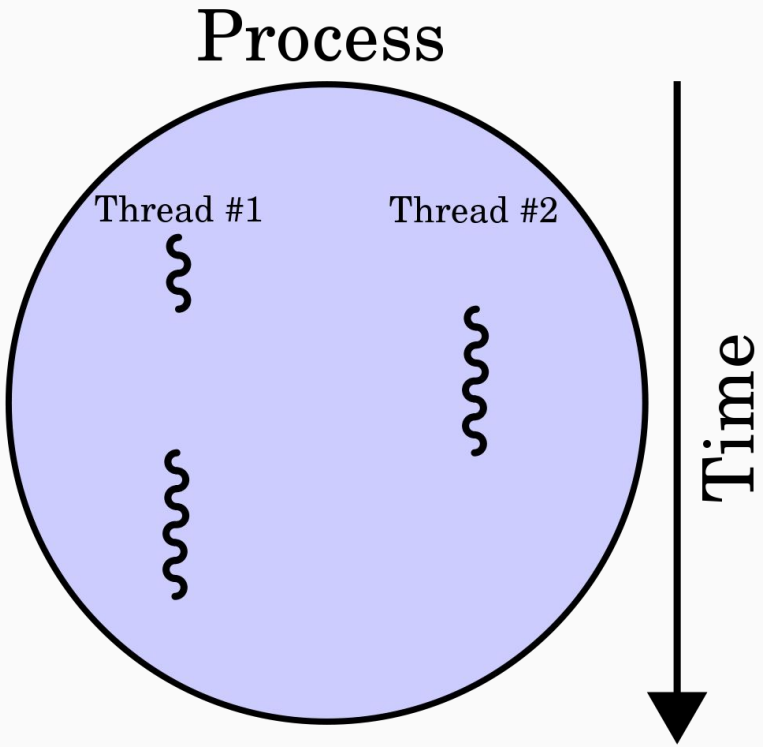
INTRODUÇÃO

O que é uma thread?

Também conhecida como linha de execução, é uma forma de um processo dividir a si mesmo em duas ou mais tarefas que podem ser executadas concorrencialmente.

Várias linguagens têm suporte; em C temos a biblioteca <pthread.h>.

ILUSTRAÇÃO



pthread.h

- Definido como um conjunto de tipos de programação da linguagem C e chamadas de procedimentos
- Implementa funções úteis, como gerenciamento de threads, mutexes, variáveis condicionais e barreiras

VISÃO GERAL

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

DECLARAÇÃO, CRIAÇÃO E TÉRMINO DE THREADS

pthread_t thread

Declaração de uma thread. Também podemos criar arrays de threads.

pthread_create(*thread, attr, start_routine, void *arg)

- **thread**: An opaque, unique identifier for the new thread returned by the subroutine.
- **attr**: An opaque attribute object that may be used to set thread attributes. You can specify a thread attributes object, or NULL for the default values.
- **start_routine**: the C routine that the thread will execute once it is created.
- **arg**: A single argument that may be passed to start_routine. It must be passed by reference as a pointer cast of type void. NULL may be used if no argument is to be passed.

pthread_exit(void *value_ptr)

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

/* Variaveis globais podem ser declaradas fora das funcoes */
int count = 0;

/* Quando utilizamos pthreads, tanto o retorno quanto os argumentos das funcoes
 * devem ser um ponteiro VOID */
void *hello(void *arg)
{
    count = count + 1;
    printf("Hello world! (Thread %d)\n", count);
    pthread_exit(NULL);
}

int main()
{
    /* Declarando um array de threads */
    pthread_t threads[10];

    /* Neste exemplo, nao eh garantido que as threads serao executadas em ordem (1 ao 10) */
    for (int i = 0; i < 10; i++)
    {
        /* Passamos a referencia para a thread no array de threads e tambem o nome da funcao
         * que sera executada.  Os argumentos NULL sao padrao, e nao sera necessario muda-los */
        pthread_create(&threads[i], NULL, hello, NULL);
    }

    return 0;
}
```

HELLO WORLD

SUSPENSÃO DA EXECUÇÃO DE UMA THREAD

pthread_join(thread, void **value_ptr)

Suspende a execução da thread que chamou o pthread_join() até que a execução da **thread** seja finalizada. Por padrão, utiliza-se **NULL** como o valor do ponteiro.

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

int count = 0;

void *hello(void *arg)
{
    count = count + 1;
    printf("Hello world! (Thread %d)\n", count);
    pthread_exit(NULL);
}

int main()
{
    int size = 20;
    pthread_t threads[size];

    /* Com o uso do join, garantimos que as threads serao executadas em ordem */
    for (int i = 0; i < size; i++)
    {
        pthread_create(&threads[i], NULL, &hello, NULL);
        pthread_join(threads[i], NULL);
    }

    return 0;
}
```

EXEMPLO DE USO DO pthread_join()

REGIÃO CRÍTICA

- Área da memória que pode ser acessada por múltiplas threads, gerando a condição de corrida
- Para garantirmos que apenas uma thread por vez acesse, devemos garantir exclusão mútua. Para isso, utilizamos os mutexes

Thread 1	Thread 2	Balance
Read balance: $1000		$1000
	Read balance: $1000	$1000
	Deposit $200	$1000
Deposit $200		$1000
Update balance $1000+$200		$1200
	Update balance $1000+$200	$1200

EXEMPLO DE CONDIÇÃO DE CORRIDA

MUTEX

Podem ser inicializados de duas formas:

ESTÁTICA

```
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
```

DINÂMICA

```
pthread_mutex_t mutex;
pthread_mutex_init(&mutex, *attr);
```

SEM MUTEX

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

void *regiao(void *arg)
{
    /* Podemos fazer cast para qualquer tipo */
    int *ptr = (int *)arg;
    int id = *ptr;
    printf("Thread %d entrando na região crítica...\n", id);
    sleep(2);
    printf("Thread %d finalizada!\n", id);
    pthread_exit(NULL);
}
/* Se executarmos varias vezes esse codigo, perceberemos que o resultado sempre
 * sera diferente. O que ocorre eh que uma thread pode ser encerrada enquanto outra
 * nem sequer comecou */
int main()
{
    int size = 3;
    pthread_t threads[size];

    int i = 0;
    while (i < size)
    {
        /* Quando quisermos passar argumentos para uma funcao, devemos sempre
         * passar a referencia para aquela variavel */
        pthread_create(&threads[i], NULL, regiao, &i);
        i = i + 1;
    }

    for (int j = 0; j < size; j++)
    {
        pthread_join(threads[j], NULL);
    }

    return 0;
}
```

COM MUTEX

```c
/* Inicializacao estatica de um mutex */
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

void *regiao(void *arg)
{
    int *ptr = (int *)arg;
    int id = *ptr;
    /* Usando o LOCK, conseguimos garantir que a thread em execucao acessara a regiao critica
     * e excluira outras de acessarem ate que ela seja finalizada. Ao final, sempre devemos
     * liberar a regiao com o UNLOCK */
    pthread_mutex_lock(&mutex);
    printf("Thread %d entrando na região crítica...\n", id);
    sleep(2);
    printf("Thread %d finalizada!\n", id);
    pthread_mutex_unlock(&mutex);
    pthread_exit(NULL);
}

int main()
{
    int size = 3;
    pthread_t threads[size];

    int i = 0;
    while (i < size)
    {
        pthread_create(&threads[i], NULL, regiao, &i);
        i = i + 1;
    }

    for (int j = 0; j < size; j++)
    {
        pthread_join(threads[j], NULL);
    }

    return 0;
}
```

VARIÁVEIS CONDICIONAIS

- São variáveis que irão sinalizar para alguma thread quando alguma condição for atendida
- Junto com mutexes, podem ser de grande utilidade para “manter a coerência” durante a execução
- Também podem ser inicializadas estática ou dinamicamente

Um exemplo de código usando variáveis condicionais está disponível no repositório do Github.

BARREIRAS

- Servem para sincronizar múltiplas threads
- Quando uma thread chegar à barreira, ela esperará pelas outras threads chegarem até o mesmo ponto
- Quando todas estiverem paralelas, poderão prosseguir com sua execução

Neste link é possível conferir um exemplo de uso das barreiras.

LISTA DE E-MAIL

if677cc+subscribe@googlegroups.com

REPOSITÓRIO

github.com/pvsaragao/if677cc