

Programação Concorrente

Odorico Machado Mendizabal

Universidade Federal de Santa Catarina – UFSC
Departamento de Informática e Estatística – INE

Threads

Objetivo da aula

- Introduzir a biblioteca *pthread* para programação com *threads* na linguagem C

POSIX (*Portable Operating System Interface*)

- Um padrão aberto de interface de operação aceito mundialmente
- É mantido pela IEEE e tem reconhecimento ISO e ANSI
- Diferentes sistemas oferecem implementações deste padrão, por exemplo, a biblioteca [pthread.h](#) para a linguagem C

POSIX – *Threads*

POSIX *Threads*

- Interface padrão para uso de threads
- Suportada nativamente pelo Linux e outros SOs
 - Esta padronização proporciona portabilidade de código entre plataformas

Threads – Biblioteca pthread.h

- Para criar uma *Thread*:

```
pthread_create( <thread>, <atrib>, <rotina>, <args>);
```

- Obter Identificação da *Thread*:

```
pthread_self();
```

- Suspende Execução:

```
pthread_delay_np(<tempo>);
```

- Definir um ponto de junção (aguardar o termino de um *Thread*):

```
pthread_join (<thread>, <retorno_thread>);
```

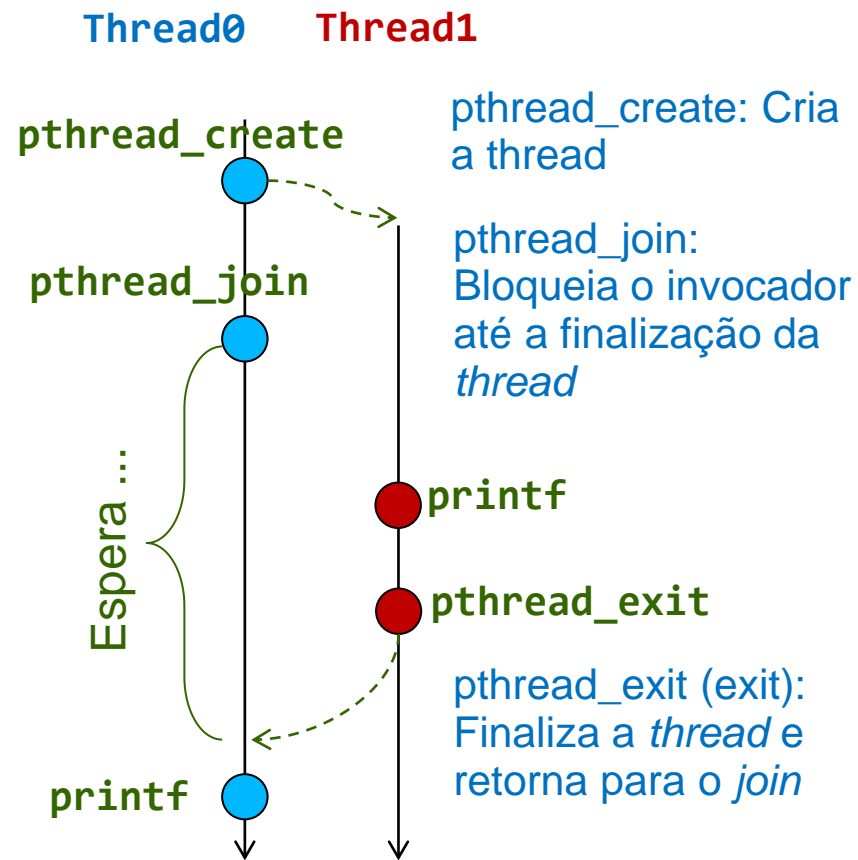
- Finalizar a *Thread*:

```
pthread_exit (<retorno>);
```

Threads – sincronização entre *threads*

- Primitivas `pthread_exit` e `pthread_join` são sincronizantes (análogo ao `exit` e `wait` para processos)

```
void *thread_function(void *arg) {  
    printf("thread_function estah executando.\n");  
    pthread_exit("Obrigado pelo tempo de CPU!\n");  
}  
  
int main() {  
    int res;  
    pthread_t a_thread;  
    void *thread_result;  
  
    res = pthread_create(&a_thread, NULL,  
        thread_function, NULL);  
    res = pthread_join(a_thread, &thread_result);  
    printf("Thread joined, retornou %s\n", (char  
*)thread_result);  
    printf("Mensagem agora eh: %s\n", message);  
    exit(EXIT_SUCCESS);  
}
```



Exemplo com *threads* – POSIX

```
// compilar com gcc -pthread
#include <pthread.h>
#include <stdio.h>
#define NUM_THREADS 5

void *print_alo(void *threadid);

int main (int argc, char *argv[]) {
    pthread_t threads[NUM_THREADS];
    int rc;
    long t;
    for(t=0; t<NUM_THREADS; t++){
        printf("In main: creating thread %ld\n", t);
        rc = pthread_create(&threads[t], NULL, print_alo, (void *)t);
        if (rc){
            printf("ERROR; return code from pthread_create() is %d\n", rc);
            exit(-1);
        }
    }
    for(t=0; t<NUM_THREADS; t++){
        pthread_join(threads[t], NULL);
    }
    pthread_exit(NULL);
}
```

// continua no proximo slide ...

Exemplo com *threads* – POSIX

```
// .. continuacao

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

Exemplo com *threads* – POSIX (saída do programa)

```
In main: creating thread 0
In main: creating thread 1
Hello World! It's me, thread #0!
In main: creating thread 2
Hello World! It's me, thread #1!
Hello World! It's me, thread #2!
In main: creating thread 3
In main: creating thread 4
Hello World! It's me, thread #3!
Hello World! It's me, thread #4!
```

POSIX – *Threads* escopo de variáveis

- Respeitam o escopo de variáveis da linguagem
 - Variáveis visíveis no escopo da chamada são visíveis por todas as *threads*, as demais são locais

```
#include <stdio.h>

int x;           // escopo - variável global

void *func_thread1 (void *arg) {
    long t1;           // não é visível pela thread2
    pthread_exit(NULL);
}

void *func_thread2 (void *arg) {
    long t2;           // não é visível pela thread1
    pthread_exit(NULL);
}
```

Threads no Windows

- Para criar uma *Thread*:

`CreateThread (<atribos>, <tam_stack>, <rotina>,<params>,
<flags>,<thread_id>)`

- Obter Identificação da *Thread*:

`GetCurrentThreadId()`

- Suspende Execução:

`SuspendThread (<thread>)`

- Finalizar a *Thread*:

`ExitThread(<retorno>)`