

# Ferramentas para Programação em Processadores Multi-Core

Prof. Dr. Gerson Geraldo H. Cavalheiro

Departamento de Informática  
Universidade Federal de Pelotas



Programa de Verão 2008

Petrópolis, 14 a 17 de janeiro de 2008.

1

Ferramentas de programação

## Sumário

- ✓ Introdução
- ✓ Arquiteturas multi-core
- ✓ Programação multithread
- **Ferramentas de programação**
- Prática de programação
- Considerações finais

Gerson Geraldo H. Cavalheiro

Ferramentas para Programação em Processadores Multi-Core  
LNCC – Programa de Verão 2008  
14 e 17 de janeiro de 2008

2

Ferramentas de programação

## Ferramentas de programação

- **POSIX threads – pthreads**
  - Biblioteca de funções
  - Paralelismo não estruturado
- **OpenMP**
  - Diretivas de compilação / Biblioteca de serviços
  - Paralelismo aninhado
- **.NET Framework**
  - Integrado com a linguagem
  - Threads definidas no contexto de objetos

Gerson Geraldo H. Cavalheiro

Ferramentas para Programação em Processadores Multi-Core  
LNCC – Programa de Verão 2008  
14 e 17 de janeiro de 2008

3

Ferramentas de programação

## Pthreads

- Padrão IEEE POSIX 1003.1c 1995
  - Definido para permitir compatibilidade de programas multithread entre diferentes plataformas
  - Define a interface de serviço (API)
  - É de uso geral
  - Linux
    - LinuxThreads
    - NPTL – Native POSIX Threads Library
  - Windows
    - pthreads\_win32

Gerson Geraldo H. Cavalheiro

Ferramentas para Programação em Processadores Multi-Core  
LNCC – Programa de Verão 2008  
14 e 17 de janeiro de 2008

4

## Pthread

## ■ Corpo de um thread

- Função C/C++ convencional
- Recebe e retorna endereços de memória
- Variáveis locais visíveis apenas no escopo da função

```
void *foo(void *args)
{
    ...
    // Código C/C++
    ...
}
```

**Dois threads podem ser criados a partir da mesma função, no entanto, são instâncias diferentes!!!**

## Pthreads

## ■ Interface básica

- Serviços
  - pthread\_XXXX
- Tipos de dados
  - pthread\_t\_XXXX
- Macros
  - PTHREAD\_XXXXXX
- Retorno dos serviços
  - Código de erro. Execução sem erro retorna 0 (zero).
  - Manipula a variável errno
- Conceito
  - Estrutura de dados opaca

## Pthreads

## ■ Manipulação de Threads

## □ Criação:

```
int pthread_create(..., void *(*foo)(void *),
                  void *arg );
```

## □ Término:

```
void pthread_exit(void *retval );
return (void *)retval;
```

## □ Sincronização:

```
int pthread_join(pthread_t tid, void **ret);
```

## □ Identificação:

```
pthread_t* pthread_self(void);
```

## Pthreads

## ■ Manipulação de Threads

## □ Criação:


```
int pthread_create( pthread_t *tid,
                  pthread_attr_t *atrib,
                  void *(*func)(void *),
                  void *args );
```

Cria um novo fluxo de execução (um novo thread). O novo fluxo executa de forma concorrente com o thread original

## ■ Manipulação de Threads

### □ Criação:

```
int pthread_create( pthread_t *tid,  
                  pthread_attr_t *atrib,  
                  void *(*func)(void *),  
                  void *args );
```




**func:** nome da função que contém o código a ser executado pelo thread

## ■ Manipulação de Threads

### □ Criação:

```
int pthread_create( pthread_t *tid,  
                  pthread_attr_t *atrib,  
                  void *(*func)(void *),  
                  void *args );
```

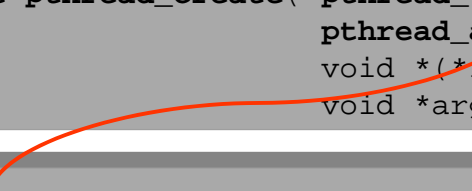


**args:** ponteiro para uma região de memória com dados de entrada para a função

## ■ Manipulação de Threads

### □ Criação:

```
int pthread_create( pthread_t *tid,  
                  pthread_attr_t *atrib,  
                  void *(*func)(void *),  
                  void *args );
```

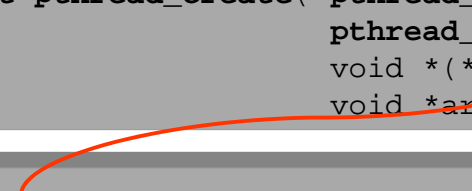


**tid:** identificador (único) do novo thread

## ■ Manipulação de Threads

### □ Criação:

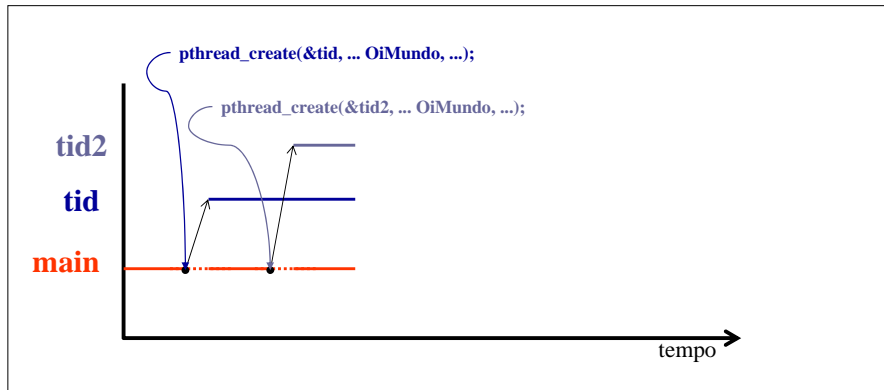
```
int pthread_create( pthread_t *tid,  
                  pthread_attr_t *atrib,  
                  void *(*func)(void *),  
                  void *args );
```



**atrib:** atributos de execução para o novo thread

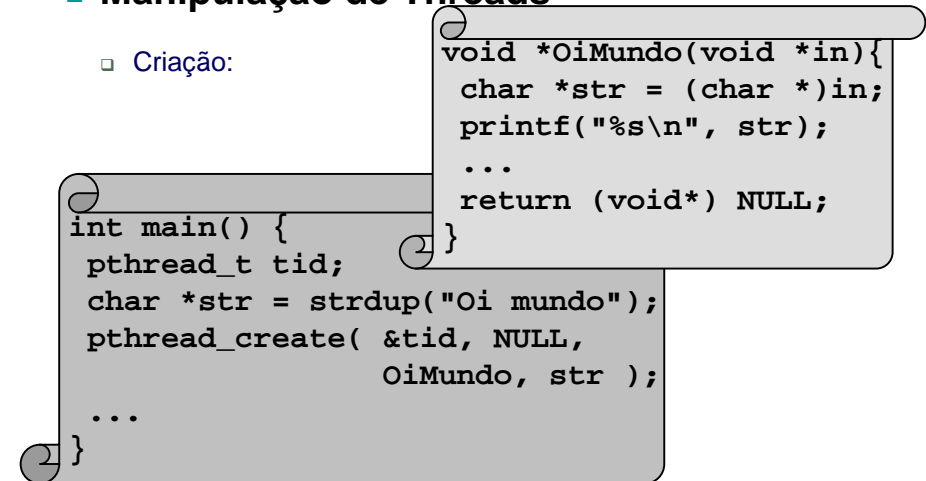
## ■ Manipulação de Threads

### □ Criação:



## ■ Manipulação de Threads

### □ Criação:



## ■ Manipulação de Threads

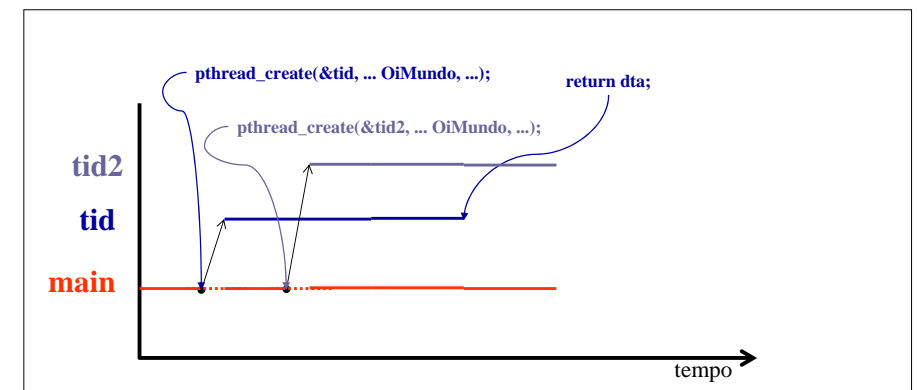
### □ Término:

```
void pthread_exit(void *retval );
OU
return (void *)retval;
```

Termina a execução do thread que executou a chamada.  
Joinable ou Detached  
**retval**: endereço de uma posição de memória contendo o dado a ser retornado

## ■ Manipulação de Threads

### □ Término:



## ■ Manipulação de Threads

### □ Término:

#### ■ pthread\_exit

- A execução do thread é interrompida e o processo de computação abandonado
- No caso de utilização com programas C++, escopos de funções/métodos não são terminados e há risco de objetos não serem destruídos com invocação ao destrutor.

#### ■ return

- Abandona a execução do thread corrente fechando o contexto de memória utilizado.

## ■ Manipulação de Threads

### □ Sincronização:

```
int pthread_join( pthread_t tid, void **ret );
```

Aguarda o término de um thread (se ele ainda não terminou) e recupera o resultado produzido.

## ■ Manipulação de Threads

### □ Sincronização:

```
int pthread_join( pthread_t tid, void **ret );
```

tid: identificador do thread a ter seu término sincronizado  
 ret: endereço de um ponteiro que será atualizado com a posição de memória que contém os dados retornados

## ■ Manipulação de Threads

### □ Sincronização:

```
int pthread_join( pthread_t tid, void **ret );
```

Cada thread suporta, no máximo, uma operação de *join*

## ■ Manipulação de Threads

### □ Sincronização:

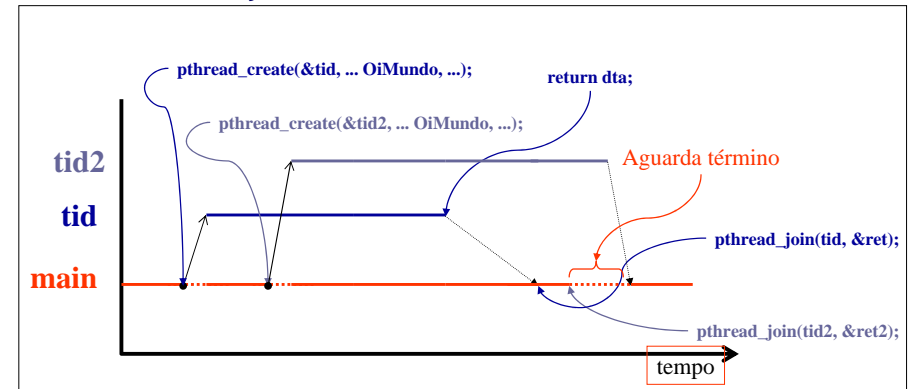
```
int pthread_detach( pthread_t tid );
```

Operação inversa ao *join*:

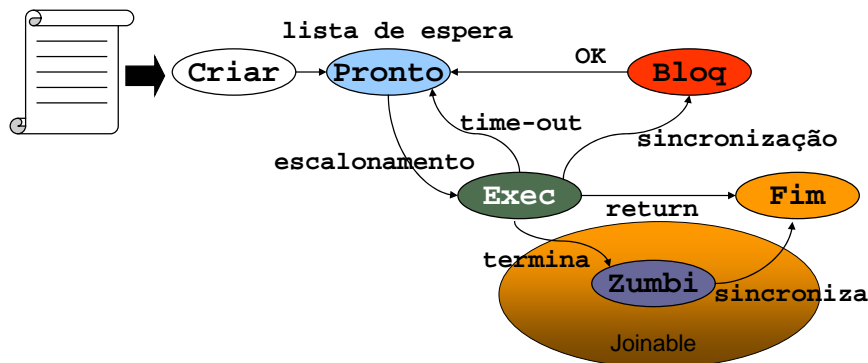
- O thread *tid* não sofrerá nenhuma sincronização por *join*

## ■ Manipulação de Threads

### □ Sincronização:



## ■ Ciclo de vida



## ■ Exemplo: produtor/consumidor

```
void *prod(void *args) {
    Buf *buf = malloc;

    produz item

    return buf;
}
```

```
void *cons(void *args) {
    Buf *buf;
    pthread_t p;

    p = (pthread_t)* args;
    pthread_join( p, &buf );

    consome item

    return NULL;
}
```

```
main(){
    pthread_t p[5], c[5];
    for(i = 0 ; i < 5 ; i++ ) {
        pthread_create(&p[i]), NULL, prod, NULL );
        pthread_create(&c[i]), NULL, cons, &p[i]);
    }

    for(i = 0 ; i < 5 ; i++ )
        pthread_join(c[i], NULL);
}
```

## ■ Exemplo: produtor/consumidor

```
void *prod(void *args) {
    Buf *buf = malloc;

    produz item

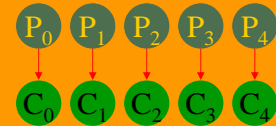
    return buf;
}
```

```
void *cons(void *args) {
    Buf *buf;
    pthread_t p;

    p = (pthread_t)* args;
    pthread_join( p, &buf );

    consome item
}
```

A sincronização por **join** garante a correta comunicação entre as tarefas executadas (controle da comunicação).



```
pthread_create(&(c[i]), NULL, cons, &(p[i]));
}

for(i = 0 ; i < 5 ; i++)
    pthread_join(c[i], NULL);
}
```

## ■ Descritor de threads

- Identificador do thread (Thread ID)
- Registradores
  - Contador de Programa (PC)
  - Ponteiro de Pilha (SP)
  - Registradores Gerais
- Pilha de execução local
  - Dados locais aos escopos
  - Endereços de retorno para chamadas de subrotinas
- Endereço de retorno após completar a chamada
- Endereço dos outros threads
- Ponteiro do PCB do processo
- Informações de escalonamento
  - Prioridade
  - Estado
  - Tipo de escalonamento
  - ....

## ■ Atributos de threads

- Alguns atributos podem ser definidos para execução
    - Tamanho da pilha
    - Política de escalonamento
    - Prioridade
    - Escopo de execução
- } Pode não ser garantido
- Depende da implementação

pthread\_attr\_t

```
int pthread_attr_init( pthread_attr_t* atrib );
int pthread_attr_setXXXX( pthread_attr_t* atrib,
                          int valor );
```

## ■ Atributos de threads

- Alguns atributos podem ser definidos para execução
  - Escopo de execução
    - Local ao processo: **PTHREAD\_SCOPE\_PROCESS**
      - O thread deverá ser escalonado no escopo do processo
    - Sistema: **PTHREAD\_SCOPE\_SYSTEM**
      - É definida uma unidade de escalonamento próprio ao thread

```
int pthread_attr_setscope( &atrib, XXXX );
```

## ■ Atributos de threads

- Alguns atributos podem ser definidos para execução
  - Estratégia de escalonamento
    - `SCHED_FF`
    - `SCHED_RR`
    - `SCHED_OTHER` (default)
  - No GNU-Linux: `SCHED_FF` e `SCHED_RR` apenas como super-usuário.

```
int pthread_attr_setschedpolicy( &atrib, XXXX );
```

## ■ Atributos de threads

- Alguns atributos podem ser definidos para execução
  - Modo de execução
    - Autônoma: `PTHREAD_CREATE_DETACHED`
      - O thread não sofrerá operação de sincronização por *join*
    - Sincronizável: `PTHREAD_CREATE_JOINABLE` (default)
      - Algum thread deverá efetuar *join* sobre o thread

```
int pthread_attr_setdetachstate( &atrib, XXXX );
```

## ■ Atributos de threads

- Manipulação da pilha do thread
  - Identificação de um tamanho alternativo para a pilha

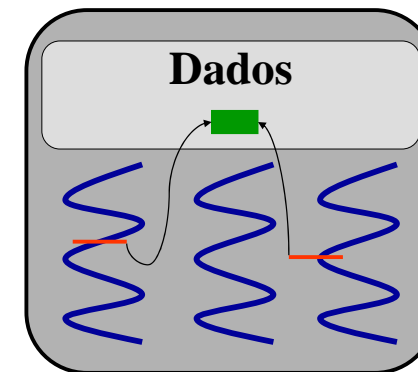
```
int pthread_attr_setstacksize(
    pthread_attr_t* atrib, size_t tam );
```

- Identificação de uma área de memória alternativa (com tamanho mínimo de `PTHREAD_STACK_MIN`)

```
int pthread_attr_setstackaddr(
    pthread_attr_t* atrib, void* end );
```

- Atenção: o padrão não define se `end` corresponde ao endereço baixo ou alto da memória

## ■ Concorrência no acesso a memória



**O controle do acesso aos dados é de responsabilidade do programador.**



## Pthreads

- Concorrência no acesso a memória
  - Os acessos à memória se dão através de operações de leitura e escrita convencionais:
    - Escrita: `DadoCompartilhado = valor;`
    - Leitura: `variável = DadoCompartilhado;`

As instruções que acessam os dados compartilhados são considerados  
**Seções Críticas**

## Pthreads

- Concorrência no acesso a memória
- Seção crítica

```
int x = 313;
```

```
// thread A
```

```
a = x;  
a = a + 1;  
x = a;
```

```
// thread B
```

```
b = x;  
b = b - 1;  
x = b;
```

A	a = x	313
A	a = a + 1	313
B	b = x	313
A	x = a	314
B	b = b - 1	314
B	x = b	312

## Pthreads

- Concorrência no acesso a memória
- Seção crítica

## MUTEX

- Exclusão mútua
  - Garantia de que apenas um thread poderá executar instruções que manipulam um dado na memória compartilhada em um determinado instante de tempo
  - Mecanismo oferecido por POSIX, a utilização, no entanto, é de responsabilidade do programador

## Pthreads

- Concorrência no acesso a memória
- Seção crítica

## MUTEX

- Tipo de dado: `pthread_mutex_t`
- Primitivas:
  - `int pthread_mutex_lock(pthread_mutex_t *m);`
  - `int pthread_mutex_unlock(pthread_mutex_t *m);`
  - `int pthread_mutex_init(pthread_mutex_t *m, pthread_mutexattr_t *atrib);`

**Uso: `init`, para inicializar o mutex (NULL == aberto), `lock` para adquirir o passe e `unlock` para liberar.**

## Pthreads

- Concorrência no acesso a memória
- Seção crítica

<pre>int x = 313; pthread_mutex_t m; pthread_mutex_init(&amp;m, NULL); // executado no main</pre>	
<pre>// thread A pthread_mutex_lock(&amp;m); a = x; a = a + 1; x = a; pthread_mutex_unlock(&amp;m);</pre>	<pre>// thread B pthread_mutex_lock(&amp;m); b = x; b = b - 1; x = b; pthread_mutex_unlock(&amp;m);</pre>

## Pthreads

- Concorrência no acesso a memória
- Seção crítica

## Variável de Condição

- Permite o acesso a uma seção crítica quando uma determinada condição for satisfeita
- Leva em conta o estado da memória no momento da sincronização

**Uso típico no compartilhamento de um *buffer* por produtores e consumidores.**

## Pthreads

- Concorrência no acesso a memória
- Seção crítica

## Variável de Condição

- **Tipo de dado:** `pthread_cond_t`
- **Primitivas:**
  - `int pthread_cond_wait(pthread_cond_t *c, pthread_mutex_t *m);`
  - `int pthread_cond_signal(pthread_cond_t *c);`
  - `int pthread_cond_broadcast(pthread_cond_t *c);`
  - `int pthread_cond_init(pthread_cond_t *c, pthread_condattr_t *atrib);`

## Pthreads

- Concorrência no acesso a memória
- Seção crítica

## Variável de Condição

- **Uso**
  - `init` : Inicializa a variável de condição (NULL == satisfeita)
  - `wait` : Bloqueia o thread, aguardando a condição
  - `signal` : Sinaliza um dos threads que estão aguardando que a condição seja satisfeita (acorda um *thread*)
  - `broadcast` : Sinaliza todas as *threads* que estão aguardando que a condição foi satisfeita (acorda todos threads)

**Uma variável de condição não garante acesso em exclusão mútua, apenas informa se uma condição foi ou não satisfeita. Portanto, variáveis de condição devem ser utilizadas em conjunto a um mutex.**

## Pthreads

- Concorrência no acesso a memória
- Seção crítica

## Variável de Condição

## □ Uso

```
pthread_mutex_t m;
pthread_cond_t c;

pthread_mutex_lock( &m );
while( teste )
    pthread_cond_wait( &c, &m );
seção crítica
pthread_mutex_unlock( &m );
```

## Pthreads

- Concorrência no acesso a memória
- Seção crítica

## Variável de Condição

## □ Uso

<pre>for(;;) { // Produtor     it = ...;     pthread_mutex_lock(&amp;m);     WrBuffer(it);     nb_itens++;     pthread_cond_signal(&amp;c);     pthread_mutex_unlock(&amp;m); }</pre>	<pre>for(;;) { // Consumidor     pthread_mutex_lock(&amp;m);     while( nb_itens &lt;= 0 )         pthread_cond_wait(&amp;m,&amp;c);     it = InBuffer();     nb_itens--;     pthread_mutex_unlock(&amp;m); }</pre>
---	---

## Pthreads

- Inicialização dinâmica

```
int pthread_once(pthread_once_t *once_control,
                void (*init_routine)(void));
```

Garante uma única execução de uma função em um processo

```
#include <pthread.h>

static pthread_once_t fooInicial = PTHREAD_ONCE_INIT;
extern int fooFuncao();

int random_function() {
    pthread_once( &fooInicial, fooFuncao );
}
```

## Sumário

- ✓ Introdução
- ✓ Arquiteturas multi-core
- ✓ Programação multithread
- **Ferramentas de programação**
  - ✓ **POSIX Threads – pthreads**
    - OpenMP
    - .NET Framework
- Prática de programação
- Considerações finais

# Ferramentas para Programação em Processadores Multi-Core

Prof. Dr. Gerson Geraldo H. Cavaleiro

Departamento de Informática  
Universidade Federal de Pelotas



<http://gersonc.anahy.org>



Programa de Verão 2008

Petrópolis, 14 a 17 de janeiro de 2008.