

# Sistemas Operacionais Embarcados

## Threads e mutexes

# Processos e threads

- Processos são programas em execução
- Cada processo tem seus próprios:
  - Program Counter, memória alocada, registradores em uso, pilha, PID etc. (*vide aula anterior sobre processos*)
  - Threads: unidades concorrentes de execução
- Threads são mecanismos que permitem um programa realizar mais de uma operação "simultaneamente" (**até agora, só vimos processos com uma única thread**)

# Processos e threads

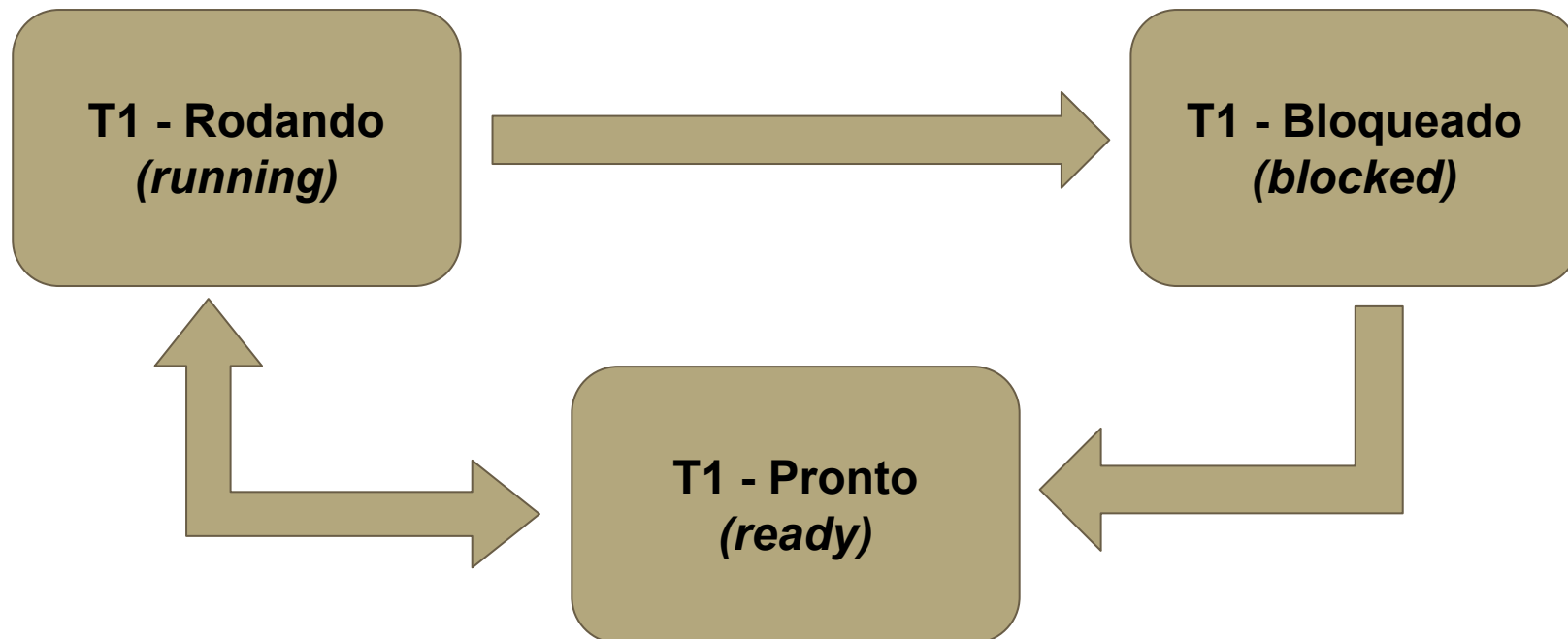
	Processos	Threads
<b>Memória</b>	Cada processo tem seu próprio espaço de memória protegido	As threads de um mesmo processo compartilham o mesmo espaço em memória
<b>Comunicação</b>	Através de mecanismos de comunicação: pipes, sinais etc (IPC - <i>Inter-process communication</i> )	Através da memória compartilhada
<b>Risco</b>	Processos-zumbi	Corrupção de dados
<b>Troca de contexto</b>	Pesada	Leve

# Processos e threads

<b>Ambiente do processo</b>	<ul style="list-style-type: none"><li>- Espaço de endereçamento</li><li>- Processo pai/filho</li><li>- Proprietário</li><li>- Arquivos abertos (descritores de arquivo)</li><li>- Parâmetros de chamadas em andamento</li><li>- Sinais</li></ul>
<b>Execução da thread</b>	<ul style="list-style-type: none"><li>- Identificador do processo (<b>PID</b>)</li><li>- Contador de programa (<b>PC - program counter</b>)</li><li>- Apontador de pilha (<b>SP - stack pointer</b>)</li><li>- Registradores</li><li>- Estado de execução</li><li>- Momento de início do processo</li><li>- Estatísticas de uso (tempo de processador utilizado etc.)</li></ul>

# Estados das threads

- A entidade que realmente se executa é a **thread**. O **processo** se refere ao ambiente.
- As **threads** compartilham as variáveis globais do programa, os descritores abertos, etc.
  - Requer mecanismos de sincronização



# Biblioteca **pthread**

- O GNU/Linux oferece uma biblioteca de threads, seguindo o padrão POSIX. Ela não faz parte das bibliotecas-padrão de C, requerindo:
  - Incluir **pthread.h** no código:  
`#include <pthread.h>`
  - Incluir **-lpthread** na chamada ao GCC:  
`gcc -lpthread [CÓDIGOS-FONTE] -o [NOME_EXEC]`

# Biblioteca **pthread**

- Cada *thread* é identificada por um **thread ID**
- O que cada *thread* criada deve fazer é definido em uma função específica em C
- A *thread* encerra sua execução quando retorna o valor da função

```

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

void* print_1(void* dummy_ptr);
void* print_2(void* dummy_ptr);

int main()
{
    pthread_t thread_id1;
    pthread_t thread_id2;

    printf("Pressione CONTROL+C para"
           "\nsair do programa\n\n");
    pthread_create(&thread_id1, NULL,
                  &print_1, NULL);
    pthread_create(&thread_id2, NULL,
                  &print_2, NULL);
    while(1)
    {
        fputc('-', stderr);
        usleep(50000);
    }
    return 0;
}

```

```

void* print_1(void* dummy_ptr)
{
    while(1)
    {
        fputc('1', stderr);
        usleep(50000);
    }
    return NULL;
}

void* print_2(void* dummy_ptr)
{
    while(1)
    {
        fputc('2', stderr);
        usleep(50000);
    }
    return NULL;
}

```

(Continuação)



```

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

void* print_1(void* dummy_ptr);
void* print_2(void* dummy_ptr);

int main()
{
    pthread_t thread_id1;
    pthread_t thread_id2;

    printf("Pressione CONTROL+C para"
           "sair do programa\n\n");
    pthread_create(&thread_id1, NULL,
                  &print_1, NULL);
    pthread_create(&thread_id2, NULL,
                  &print_2, NULL);
    while(1)
    {
        fputc('-', stderr);
        usleep(50000);
    }
    return 0;
}

```

**Identificadores das threads  
que serão criadas**

```

void* print_1(void* dummy_ptr)
{
    while(1)
    {
        fputc('1', stderr);
        usleep(50000);
    }
    return NULL;
}

```

```

void* print_2(void* dummy_ptr)
{
    while(1)
    {
        fputc('2', stderr);
        usleep(50000);
    }
    return NULL;
}

```

(Continuação)

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

void* print_1(void* dummy_ptr);
void* print_2(void* dummy_ptr);

int main()
{
    pthread_t thread_id1;
    pthread_t thread_id2;

    printf("Pressione CONTROL+C para"
           "\nsair do programa\n\n");
    pthread_create(&thread_id1, NULL,
                  &print_1, NULL);
    pthread_create(&thread_id2, NULL,
                  &print_2, NULL);
    while(1)
    {
        fputc('-', stderr);
        usleep(50000);
    }
    return 0;
}
```

```
void* print_1(void* dummy_ptr)
{
    while(1)
    {
        fputc('1', stderr);
        usleep(50000);
    }
    return NULL;
}

void* print_2(void* dummy_ptr)
{

```

**Criação da primeira thread,  
que executará a função  
print\_1()**

(Continuação)

```

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

void* print_1(void* dummy_ptr);
void* print_2(void* dummy_ptr);

int main()
{
    pthread_t thread_id1;
    pthread_t thread_id2;

    printf("Pressione CONTROL+C para"
           "\nsair do programa\n\n");
    pthread_create(&thread_id1, NULL,
                  &print_1, NULL);
    pthread_create(&thread_id2, NULL,
                  &print_2, NULL);
    while(1)
    {
        fputc('-', stderr);
        usleep(50000);
    }
    return 0;
}

```

```

void* print_1(void* dummy_ptr)
{
    while(1)
    {
        fputc('1', stderr);
        usleep(50000);
    }
    return NULL;
}

void* print_2(void* dummy_ptr)
{

```

**Ou seja, print\_1() será executada em paralelo à função main()**

(Continuação)

```

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

void* print_1(void* dummy_ptr);
void* print_2(void* dummy_ptr);

int main()
{
    pthread_t thread_id1;
    pthread_t thread_id2;

    printf("Pressione CONTROL+C para
           \"sair do programa\n\n");
    pthread_create(&thread_id1, NULL,
                  &print_1, NULL);
    pthread_create(&thread_id2, NULL,
                  &print_2, NULL);
    while(1)
    {
        fputc('-', stderr);
        usleep(50000);
    }
    return 0;
}

```

```

void* print_1(void* dummy_ptr)
{
    while(1)
    {
        fputc('1', stderr);
        usleep(50000);
    }
    return NULL;
}

```

**A variável `thread_id1` receberá o identificador desta thread, que pode ser usada para controlar a thread (bloquear, desativar, ativar etc.)**

(Continuação)

```

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

void* print_1(void* dummy_ptr);
void* print_2(void* dummy_ptr);

int main()
{
    pthread_t thread_id1;
    pthread_t thread_id2;

    printf("Pressione CONTROL+C para"
           "sair do programa\n");
    pthread_create(&thread_id1,
                  &print_1, NULL);
    pthread_create(&thread_id2,
                  &print_2, NULL);
    while(1)
    {
        fputc('-', stderr);
        usleep(50000);
    }
    return 0;
}

```

```

void* print_1(void* dummy_ptr)
{
    while(1)
    {
        fputc('1', stderr);
        usleep(50000);
    }
    return NULL;
}

void* print_2(void* dummy_ptr)
{
    while(1)
    {
        fputc('2', stderr);
        usleep(50000);
    }
    return NULL;
}

```

**Repare que passamos o endereço  
da função `print_1()`**

(Continuação)

```

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

void* print_1(void* dummy_ptr);
void* print_2(void* dummy_ptr);

int main()
{
    pthread_t thread_id1;
    pthread_t thread_id2;

    printf("Pressione CONTROL+C para"
           "sair do programa\n\n");
    pthread_create(&thread_id1, NULL,
                  &print_1, NULL);
    pthread_create(&thread_id2, NULL,
                  &print_2, NULL);
    while(1)
    {
        fputc('-', stderr);
        usleep(50000);
    }
    return 0;
}

```

```

void* print_1(void* dummy_ptr)
{
    while(1)
    {
        fputc('1', stderr);
        usleep(50000);
    }
    return NULL;
}

```

```

void* print_2(void* dummy_ptr)
{
    while(1)
    {
        fputc('2', stderr);
        usleep(50000);
    }
    return NULL;
}

```

Criação da segunda thread, que executará a função `print_2()`

Ou seja, `print_2()` será executada **em paralelo** às funções `main()` e `print_1()`

```

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

void* print_1(void* dummy_ptr);
void* print_2(void* dummy_ptr);

int main()
{
    pthread_t thread_id1;
    pthread_t thread_id2;

    printf("Pressione CONTROL+C para"
           "sair do programa\n\n");
    pthread_create(&thread_id1, NULL,
                  &print_1, NULL);
    pthread_create(&thread_id2, NULL,
                  &print_2, NULL);
    while(1)
    {
        fputc('-', stderr);
        usleep(50000);
    }
    return 0;
}

```

```

void* print_1(void* dummy_ptr)
{
    while(1)
    {
        fputc('1', stderr);
        usleep(50000);
    }
    return NULL;
}

void* print_2(void* dummy_ptr)
{
    while(1)
    {
        fputc('2', stderr);
        usleep(50000);
    }
    return NULL;
}

```

**A função `main()`  
escreverá continuamente  
o caractere ' - ' na tela,  
em intervalos de 50000 us  
(50 ms)**

```

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

void* print_1(void* dummy_ptr);
void* print_2(void* dummy_ptr);

int main()
{
    pthread_t thread_id1;
    pthread_t thread_id2;

    printf("Pressione CONTROL+C p
        \"sair do programa\n\n");
    pthread_create(&thread_id1, N
        &print_1, NULL);
    pthread_create(&thread_id2, N
        &print_2, NULL);
    while(1)
    {
        fputc('-', stderr);
        usleep(50000);
    }
    return 0;
}

```

```

void* print_1(void* dummy_ptr)
{
    while(1)
    {
        fputc('1', stderr);
        usleep(50000);
    }
}

```

**Usamos a escrita no arquivo `stderr` para garantir que o caractere seja apresentado.**

**Um `printf()` escreve no arquivo `stdout`, e o resultado só aparece na tela depois de uma quebra de linha. O `stderr` não tem esse “problema”**



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

```
void* pr
void* pr
```

```
int main
{
```

```
pthread_t thread_id1;
pthread_t thread_id2;
```

```
printf("Pressione CONTROL+C para"
      "sair do programa\n\n");
```

```
pthread_create(&thread_id1, NULL,
              &print_1, NULL);
```

```
pthread_create(&thread_id2, NULL,
              &print_2, NULL);
```

```
while(1)
{
    fputc('-', stderr);
    usleep(50000);
}
```

```
return 0;
```

```
}
```

**A primeira thread criada  
escreverá continuamente  
o caractere '1' na tela,  
em intervalos de 50 ms**

```
void* print_1(void* dummy_ptr)
{
    while(1)
    {
        fputc('1', stderr);
        usleep(50000);
    }
    return NULL;
}
```

```
void* print_2(void* dummy_ptr)
{
    while(1)
    {
        fputc('2', stderr);
        usleep(50000);
    }
    return NULL;
}
```

(Continuação)

```

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

void* print_1(void* dummy_ptr);
void* print_2(void* dummy_ptr);

int main()
{
    pthread_t t1;
    pthread_t t2;

    print_1(&t1);
    print_2(&t2);

    pthread_join(t1, NULL);
    pthread_join(t2, NULL);

    while(1)
    {
        fputc('-', stderr);
        usleep(50000);
    }
    return 0;
}

```

**A segunda thread criada  
escreverá continuamente  
o caractere '2' na tela,  
em intervalos de 50 ms**

```

void* print_1(void* dummy_ptr)
{
    while(1)
    {
        fputc('1', stderr);
        usleep(50000);
    }
    return NULL;
}

void* print_2(void* dummy_ptr)
{
    while(1)
    {
        fputc('2', stderr);
        usleep(50000);
    }
    return NULL;
}

```

(Continuação)

**As *threads* concorrem por uso de CPU e pela escrita no terminal, sendo difícil prever a ordem com que os caracteres serão escritos**

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

```
void* print_1(void* dummy_ptr)
void* print_2(void* dummy_ptr)
```

```
int main()
{
```

```
    pthread_t thread_id1;
    pthread_t thread_id2;
```

```
    printf("Pressione CONTROL+C para"
           "sair do programa\n\n");
```

```
    pthread_create(&thread_id1, NULL,
                  &print_1, NULL);
    pthread_create(&thread_id2, NULL,
                  &print_2, NULL);
```

```
    while(1)
    {
        fputc('-', stderr);
        usleep(50000);
    }
```

```
    return 0;
```

```
}
```

```
void* print_1(void* dummy_ptr)
{
```

```
    fputc('1', stderr);
    usleep(50000);
```

```
void* print_2(void* dummy_ptr)
{
    while(1)
    {
        fputc('2', stderr);
        usleep(50000);
    }
    return NULL;
}
```

(Continuação)

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

```
void* print_1(void*)
void* print_2(void*)
```

```
int main()
{
```

```
pthread_t thread_id1;  
pthread_t thread_id2;
```

```
void* print_1(void* dummy_ptr)
{
```

```
    stderr);  
    00);
```

```
void* print_2(void* dummy_ptr)
{
```

As *threads* concorrem por uso de CPU e pela escrita no terminal, sendo difícil prever a ordem com que os caracteres serão escritos

```
~/Code/08_Threads_Mutexes $ ./Ex0.out
Pressione CONTROL+C para sair do programa

1-212-12-12-12-12-12-12-12-21-2-12-12-12-121-21-2-12-12-12-12
1-21-21-2-12-121-2-12-12-12-12-12-12-12-12-12-12-12-12-12-
12-1-21-21-121-212-12-21-21-21-21-21-2-1-2121-2-1-21-21-12
-12-12-12-21-12-12-12-21-21-21-21-12-121-2-12-21-21-12-12-
121-21-21-21-212-1-21-212-12-2-12-1-21-12-21-12^C
~/Code/08_Threads_Mutexes $
```

}

```
void* print_2(void* dummy_ptr)
{
```

Code/08 Threads Mutexes/Ex0.c

```

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

void* print_xs (void* c);
int main ()
{
    pthread_t thread_id1;
    pthread_t thread_id2;
    char c1 = 'A';
    char c2 = 'B';
    printf("Pressione CONTROL+C para
sair do programa\n\n");
    pthread_create (&thread_id1, NULL,
&print_xs, &c1);
    pthread_create (&thread_id2, NULL,
&print_xs, &c2);
    while(1)
    {
        fputc('-', stderr);
        usleep(50000);
    }
    return 0;
}

```

```

void* print_xs (void* c)
{
    char *character = (char *) c;
    while (1)
    {
        fputc(*character, stderr);
        usleep(50000);
    }

    return NULL;
}

```

(Continuação)

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

```
void* print_1(void* c)
int main ()
{
```

```
    pthread_t t1;
    pthread_t t2;
    char c1 = '1';
    char c2 = '2';
    printf("Caracteres: %c e %c\n", c1, c2);
    // Sair do programa
    pthread_create(&t1, NULL, print_xs, &c1);
    pthread_create(&t2, NULL, print_xs, &c2);
    while(1)
    {
        fputc('-', stderr);
        usleep(50000);
    }
    return 0;
}
```

```
void* print_xs (void* c)
{
    char *caractere = (char *) c;

    while(1)
    {
        fputc(*caractere, stderr);
        usleep(50000);
    }
}
```

No exemplo anterior, precisamos criar duas funções diferentes, `print_1()` e `print_2()`, para escrever caracteres diferentes na tela

O ideal seria ter uma função que escreve um caractere arbitrário

É o que fazemos no segundo exemplo

**A função  
print\_xs()  
recebe uma  
variável de  
entrada do tipo  
void\*, que é  
um tipo genérico**

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

void* print_xs (void* c)
int main ()
{
    pthread_t thread_id1;
    pthread_t thread_id2;
    char c1 = 'A';
    char c2 = 'B';
    printf("Pressione CONTROL+C para
sair do programa\n\n");
    pthread_create (&thread_id1, NULL,
&print_xs, &c1);
    pthread_create (&thread_id2, NULL,
&print_xs, &c2);
    while(1)
    {
        fputc('-', stderr);
        usleep(50000);
    }
    return 0;
}
```


```
void* print_xs (void* c)
{
    char *character = (char *) c;
    while (1)
    {
        fputc(*character, stderr);
        usleep(50000);
    }

    return NULL;
}
```

(Continuação)



**Essa variável é  
então convertida  
em um char\*...**



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

```
void* print_xs
int main ()
{
    pthread_t thread_id1;
    pthread_t thread_id2;
    char c1 = 'A';
    char c2 = 'B';
    printf("Pressione CONTROL+C para
sair do programa\n\n");
    pthread_create (&thread_id1, NULL,
&print_xs, &c1);
    pthread_create (&thread_id2, NULL,
&print_xs, &c2);
    while(1)
    {
        fputc('-', stderr);
        usleep(50000);
    }
    return 0;
}
```

```
void* print_xs (void* c)
{
    char *character = (char *) c;
    while (1)
    {
        fputc(*character, stderr);
        usleep(50000);
    }

    return NULL;
}
```

(Continuação)

**Essa variável é  
então convertida  
em um char\*...**

**e é usada para  
escrever o  
caractere na tela**

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

```
void* print_xs
int main ()
{
    pthread_t thread_id1;
    pthread_t thread_id2;
    char c1 = 'A';
    char c2 = 'B';
    printf("Pressione CONTROL+C para
sair do programa\n\n");
    pthread_create (&thread_id1, NULL,
&print_xs, &c1);
    pthread_create (&thread_id2, NULL,
&print_xs, &c2);
    while(1)
    {
        fputc('-', stderr);
        usleep(50000);
    }
    return 0;
}
```

```
void* print_xs (void* c)
{
    char *character = (char *) c;
    while (1)
    {
        fputc(*character, stderr);
        usleep(50000);
    }

    return NULL;
}
```

(Continuação)

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

```
void* print_xs (void* c);
int main ()
{
    pthread_t thread_id1;
    pthread_t thread_id2;
    char c1 = 'A';
    char c2 = 'B';
    printf("Pressione CONTROL+C p
sair do programa\n\n");
    pthread_create (&thread_id1,
&print_xs, &c1);
    pthread_create (&thread_id2,
&print_xs, &c2);
    while(1)
    {
        fputc('-', stderr);
        usleep(50000);
    }
    return 0;
}
```

```
void* print_xs (void* c)
{
    char *character = (char *) c;
    while (1)
    {
        fputc(*character, stderr);
        usleep(50000);
    }
}
```

Na criação da *thread*, passamos o endereço de um caractere (&c1) para escreve-lo na tela

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

```
void* print_xs (void* c);
int main ()
{
```

```
    pthread_t thread_id1;
    pthread_t thread_id2;
    char c1 = 'A';
    char c2 = 'B';
    printf("Pressione CONTROL+C p
sair do programa\n\n");
    pthread_create (&thread_id1,
&print_xs, &c1);
    pthread_create (&thread_id2,
&print_xs, &c2);
    while(1)
    {
        fputc('-', stderr);
        usleep(50000);
    }
    return 0;
}
```

```
void* print_xs (void* c)
{
    char *character = (char *) c;
    while (1)
    {
        fputc(*character, stderr);
        usleep(50000);
    }

    return NULL;
}
```

(Continuação)

**Idem para a  
segunda *thread***

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

```
void* print_xs (void* c)
{
    pthread_t thre
    pthread_t thre
    char c1 = 'A';
    char c2 = 'B';
```

```
pr
sair
pt
&prin
pt
&prin
wh
{
}
re
}
```

```
void* print_xs (void* c)
{
    char *character = (char *) c;
    printf("character, stderr);
    00);
```

Novamente, as *threads* concorrem por uso de CPU e pela escrita no terminal, sendo difícil prever a ordem com que os caracteres serão escritos

(Continuação)

```
~/Code/08_Threads_Mutexes $ ./Ex1.out
Pressione CONTROL+C para sair do programa
```

```
A-BA-B-AB-AB-AB-AB-AB-AB-AB-AB-AB-AB-AB-ABA-B-AB-AB-AB-
AB-AB-AB-BA-BA-BA-BA-BA-BAB-AB-AB-AB-AB-AB-AB-AB-AB-AB-
A-BA-BA-BA-BA-BA-BA-BA-BA-BA-BA-BA-BA-BA-BA-BA-BA-BAB-AB-A
B-AB-AB-AB-AB-AB-AB-AB-AB-AB-AB-AB-AB-AB-AB-AB-AB-AB-AB-
-AB-AB-AB-AB-AB-AB-AB-AB-AB-AB-AB-AB-AB-AB-AB-AB-AB-AB-
~/Code/08_Threads_Mutexes $
```

```

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

struct char_print_parms
{
    char character;
    int count;
};

void* char_print (void* parameters)
{
    struct char_print_parms* p =
        (struct char_print_parms*)
        parameters;
    int i;
    for (i = 0; i < p->count; ++i)
        fputc(p->character, stderr);
    return NULL;
}

```

Code/08\_Threads\_Mutexes/Ex2.c

```

int main ()
{
    pthread_t thread1_id;
    struct char_print_parms
        thread1_args;
    thread1_args.character = 'x';
    thread1_args.count = 100;
    pthread_create(&thread1_id, NULL,
        &char_print, &thread1_args);
    pthread_t thread2_id;
    struct char_print_parms
        thread2_args;
    thread2_args.character = 'o';
    thread2_args.count = 80;
    pthread_create(&thread2_id, NULL,
        &char_print, &thread2_args);
    fputs(" Terminando a execucao da
thread principal.\n", stderr);
    return 0;
}

```

(Continuação)

```

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

struct char_print_parms
{
    char character;
    int count;
};

void* char_print (
{
    struct char_pr
        (struct cha
        parameters;
    int i;
    for (i = 0; i <
        fputc(p->char
    return NULL;
}

```

Code/08\_Threads\_Mutexes/Ex2.c

```

int main ()
{
    pthread_t thread1_id;
    struct char_print_parms
        thread1_args;
    thread1_args.character = 'x';
    int = 100;
    pthread1_id, NULL,
    &thread1_args);
    pthread2_id;
    struct char_print_parms
        thread2_args;
    thread2_args.character = 'o';
    int = 80;
    pthread2_id, NULL,
    &thread2_args);
    printf("Fim da execucao da\n", stderr);
    return 0;
}

```

(Continuação)

**No exemplo anterior,  
conseguimos passar um caractere  
da função `main()` para a função  
das *threads***

**E se precisarmos passar mais de  
uma variável?**

```

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

struct char_print_parms
{
    char character;
    int count;
};

void* char_print(void* parameters)
{
    struct char_print_parms* p =
        (struct char_print_parms*)
        parameters;
    int i;
    for (i = 0; i < p->count; ++i)
        fputc(p->character, stderr);
    return NULL;
}

```

**Basta passar uma estrutura**

```

int main ()
{
    pthread_t thread1_id;
    struct char_print_parms
        thread1_args;
    thread1_args.character = 'x';
    thread1_args.count = 100;
    pthread_create(&thread1_id, NULL,
        char_print, &thread1_args);

    pthread_t thread2_id;
    struct char_print_parms
        thread2_args;
    thread2_args.character = 'o';
    thread2_args.count = 80;
    pthread_create(&thread2_id, NULL,
        &char_print, &thread2_args);
    fputs(" Terminando a execucao da
thread principal.\n", stderr);
    return 0;
}

```

Code/08\_Threads\_Mutexes/Ex2.c

(Continuação)



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

struct char_print_parms
{
    char character;
    int count;
};

void* char_print (void* parameters)
{
    struct char_print_parms* p =
        (struct char_print_parms*)
        parameters;
    int i;
    for (i = 0; i < p->count; ++i)
        fputc(p->character, stderr);
    return NULL;
}
```

Code/08\_Threads\_Mutexes/Ex2.c

```
int main ()
{
    pthread_t thread1_id;
    struct char_print_parms
        thread1_args;
    thread1_args.character = 'x';
    thread1_args.count = 100;
    pthread_create(&thread1_id, NULL,
        &char_print, &thread1_args);
    pthread_t thread2_id;
    struct char_print_parms
        thread2_args;
    thread2_args.character = 'o';
    thread2_args.count = 80;
```

**É por isso que o parâmetro de entrada é do tipo `void*`: ele permite passar qualquer tipo de variável, incluindo estruturas**

```

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

struct char_print_parms
{
    char character;
    int count;
};

void* char_print (void* parameters)
{
    struct char_print_parms* p =
        (struct char_print_parms*)
        parameters;
    int i;
    for (i = 0; i < p->count; ++i)
        fputc(p->character, stderr);
    return NULL;
}

```

```

int main ()
{
    pthread_t thread1_id;
    struct char_print_parms
        thread1_args;
    thread1_args.character = 'x';
    thread1_args.count = 100;
    pthread_create(&thread1_id, NULL,
        &char_print, &thread1_args);
    pthread_t thread2_id;
    struct char_print_parms
        thread2_args;
    thread2_args.character = 'o';
    thread2_args.count = 80;
    pthread_create(&thread2_id, NULL,
        &char_print, &thread2_args);
    fputs(" Terminando a execucao da
thread principal.\n", stderr);
    return 0;
}

```

No exemplo anterior, as *threads* entravam em *loop* infinito. Neste exemplo, as *threads* escrevem um caractere uma quantidade finita de vezes

```

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

struct char_print_parms
{
    char character;
    int count;
};

void* char_print (void* parameters)
{
    struct char_print_parms* p =
        (struct char_print_parms*)
        parameters;
    int i;
    for (i = 0; i < p->count; ++i)
        fputc(p->character, stderr)
    return NULL;
}

```

Code/08\_Threads\_Mutexes/Ex2.c

```

int main ()
{
    pthread_t thread1_id;
    struct char_print_parms
        thread1_args;
    thread1_args.character = 'x';
    thread1_args.count = 100;
    pthread_create(&thread1_id, NULL,
        &char_print, &thread1_args);
    pthread_t thread2_id;
    struct char_print_parms
        thread2_args;
    thread2_args.character = 'o';
    thread2_args.count = 80;
    pthread_create(&thread2_id, NULL,
        &char_print, &thread2_args);
}

```

(Continuação)

**Por isso passamos um caractere  
e um valor inteiro**

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

```
struct
{
```

```
};
```

```
void
{
```

- No código principal:**
- criamos uma estrutura,
  - a preenchemos com o caractere 'x' e a contagem 100,
  - e a passamos como entrada para a função de *threads*

```
parameters;
```

```
int i;
```

```
for (i = 0; i < p->count; ++i)
```

```
    fputc(p->character, stderr);
```

```
return NULL;
```

```
}
```

```
int main ()
```

```
{
```

```
pthread_t thread1_id;
```

```
struct char_print_parms  
    thread1_args;
```

```
thread1_args.character = 'x';
```

```
thread1_args.count = 100;
```

```
pthread_create(&thread1_id, NULL,  
    &char_print, &thread1_args);
```

```
pthread_t thread2_id;
```

```
struct char_print_parms  
    thread2_args;
```

```
thread2_args.character = 'o';
```

```
thread2_args.count = 80;
```

```
pthread_create(&thread2_id, NULL,  
    &char_print, &thread2_args);
```

```
fputs(" Terminando a execucao da  
thread principal.\n", stderr);
```

```
return 0;
```

```
}
```

Code/08\_Threads\_Mutexes/Ex2.c

(Continuação)

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

struct char_print_parms
{
    char character;
    int count;
};

void char_print(struct char_print_parms *p)
{
    int i;
    for (i = 0; i < p->count; i++)
        putchar(p->character);
    putchar('\n');
}
```

- No código principal:**
- criamos uma estrutura,
  - a preenchemos com o caractere 'o' e a contagem 100,
  - e a passamos como entrada para a função de *threads*

```
int main ()
{
    pthread_t thread1_id;
    struct char_print_parms
        thread1_args;
    thread1_args.character = 'x';
    thread1_args.count = 100;
    pthread_create(&thread1_id, NULL,
        &char_print, &thread1_args);

    pthread_t thread2_id;
    struct char_print_parms
        thread2_args;
    thread2_args.character = 'o';
    thread2_args.count = 80;
    pthread_create(&thread2_id, NULL,
        &char_print, &thread2_args);

    fputs(" Terminando a execucao da
thread principal.\n", stderr);
    return 0;
}
```

Code/08\_Threads\_Mutexes/Ex2.c

(Continuação)

```

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

struct char_print_parms
{
    char character;
    int count;
};

void* char_print (void* parameters)
{
    struct char_print_parms* p =
        (struct char_print_parms*)
        parameters;

    while (p->count > 0)
    {
        putchar(p->character);
        p->count--;
    }

    return NULL;
}

```

**Em seguida,  
terminamos a  
execução da *thread*  
principal**

Code/08\_Threads\_Mutexes/Ex2.c

```

int main ()
{
    pthread_t thread1_id;
    struct char_print_parms
        thread1_args;
    thread1_args.character = 'x';
    thread1_args.count = 100;
    pthread_create(&thread1_id, NULL,
        &char_print, &thread1_args);
    pthread_t thread2_id;
    struct char_print_parms
        thread2_args;
    thread2_args.character = 'o';
    thread2_args.count = 80;
    pthread_create(&thread2_id, NULL,
        &char_print, &thread2_args);
    fputs(" Terminando a execucao da
thread principal.\n", stderr);
    return 0;
}

```

(Continuação)

```

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

struct char_print_params {
    char character;
    int count;
};

void* char_print(void* p)
{
    struct char_print_params *p_params = (struct char_print_params *) p;
    parameters;
    int i;
    for (i = 0; i < p->count; ++i)
        fputc(p->character, stderr);
    return NULL;
}

```

Code/08\_Threads\_Mutexes/Ex2.c

```

int main ()
{
    pthread_t thread2_id;
    struct char_print_params thread2_args;
    thread2_args.count = 80,
    pthread_create(&thread2_id, NULL,
        &char_print, &thread2_args);
    fputs(" Terminando a execucao da
thread principal.\n", stderr);
    return 0;
}

```

(Continuação)

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

```
struct
{
```

```
    char
    int
};
```

```
void*
{
```

```
    struct char_print_params *
        parameters;
    int i;
    for (i = 0; i < p->count; ++i)
        fputc(p->character, stderr);
    return NULL;
}
```

Code/08\_Threads\_Mutexes/Ex2.c

```
int main ()
{
```

```
~/Code/08_Threads_Mutexes $ ./Ex2.out
xxxxxx Terminando a execucao da thread principal.
~/Code/08_Threads_Mutexes $
```

```
    pthread_create(&thread2_id, NULL,
        &char_print, &thread2_args);
    fputs(" Terminando a execucao da
thread principal.\n", stderr);
    return 0;
}
```

(Continuação)



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

```
struct
{
    ch
    in
};
```

```
void*
{
    st
    (struct char_print_params *)
    parameters;
    int i;
    for (i = 0; i < p->count; ++i)
        fputc(p->character, stderr);
    return NULL;
}
```

Code/08\_Threads\_Mutexes/Ex2.c

```
int main ()
{
```

```
~/Code/08_Threads_Mutexes $ ./Ex2.out
xxxxxx Terminando a execucao da thread principal.
o~/Code/08_Threads_Mutexes $ ./Ex2.out
xxxxxxxxxxx Terminando a execucao da thread principal.
oo~/Code/08_Threads_Mutexes $
```

```
NULL,
s);

thread2_args.count = 80,
pthread_create(&thread2_id, NULL,
    &char_print, &thread2_args);
fputs(" Terminando a execucao da
thread principal.\n", stderr);
return 0;
}
```

(Continuação)

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

```
struct
{
    ch
    in
};
```

```
void*
{
    st
    (struct char_print_params *)
    parameters;
    int i;
    for (i = 0; i < p->count; ++i)
        fputc(p->character, stderr);
    return NULL;
}
```

Code/08\_Threads\_Mutexes/Ex2.c

```
int main ()
{
```

```
NULL,
s);
```

```
thread2_args.count = 80,
pthread_create(&thread2_id, NULL,
    &char_print, &thread2_args);
fputs(" Terminando a execucao da
thread principal.\n", stderr);
return 0;
}
```

(Continuação)



```

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

struct char_print_parms
{
    char character;
    int count;
};

void* char_print (void* parameters)
{
    struct char_print_parms* p =
        (struct char_print_parms*)
        parameters;
    int i;
    for (i = 0; i < p->count; ++i)
        fputc(p->character, stderr);
    return NULL;
}

```

Code/08\_Threads\_Mutexes/Ex3.c

```

int main ()
{
    pthread_t thread1_id;
    struct char_print_parms
        thread1_args;
    thread1_args.character = 'X';
    thread1_args.count = 10;
    pthread_create(&thread1_id, NULL,
        &char_print, &thread1_args);
    pthread_t thread2_id;
    struct char_print_parms
        thread2_args;
    thread2_args.character = 'Y';
    thread2_args.count = 8;
    pthread_create (&thread2_id, NULL,
        &char_print, &thread2_args);
    pthread_join (thread1_id, NULL);
    pthread_join (thread2_id, NULL);
    fputs(" Terminando a execucao da
thread principal.\n", stderr);
    return 0;
}

```

(Continuação)

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

struct char_print_parms
{
    char character;
    int count;
};

void* char_print (void* parameters)
{

```

Para garantir que a *thread* principal não termine sua execução antes das *threads* criadas, usamos a função `pthread_join()`

Ela trava a *thread* principal até que a *thread* indicada termine sua execução

```
int main ()
{
    pthread_t thread1_id;
    struct char_print_parms
        thread1_args;
    thread1_args.character = 'X';
    thread1_args.count = 10;
    pthread_create(&thread1_id, NULL,
        &char_print, &thread1_args);
    pthread_t thread2_id;
    struct char_print_parms
        thread2_args;
    thread2_args.character = 'Y';
    thread2_args.count = 8;
    pthread_create (&thread2_id, NULL,
        &char_print, &thread2_args);
    pthread_join (thread1_id, NULL);
    pthread_join (thread2_id, NULL);
    fputs(" Terminando a execucao da
thread principal.\n", stderr);
    return 0;
}
```

(Continuação)

```

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

struct char_print_parms
{
    char character;
    int count;
};

void* print_char(void* parameters)
{
    struct char_print_parms* p = (struct char_print_parms*)
        parameters;
    int i;
    for (i = 0; i < p->count; ++i)
        fputc(p->character, stderr);
    return NULL;
}

```

**Neste exemplo, pedimos para as *threads* escreverem os caracteres 'X' e 'Y' 10 e 8 vezes, respectivamente**

Code/08\_Threads\_Mutexes/Ex3.c

```

int main ()
{
    pthread_t thread1_id;
    struct char_print_parms
        thread1_args;
    thread1_args.character = 'X';
    thread1_args.count = 10;
    pthread_create(&thread1_id, NULL,
        &print_char, &thread1_args);
    pthread_t thread2_id;
    struct char_print_parms
        thread2_args;
    thread2_args.character = 'Y';
    thread2_args.count = 8;
    pthread_create (&thread2_id, NULL,
        &print_char, &thread2_args);
    pthread_join (thread1_id, NULL);
    pthread_join (thread2_id, NULL);
    fputs(" Terminando a execucao da
thread principal.\n", stderr);
    return 0;
}

```

(Continuação)

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

```
struct char_print_parms
{
```

```
};
```

```
void
{
```

```
    parameters;
    int i;
    for (i = 0; i < p->count; ++i)
        fputc(p->character, stderr);
    return NULL;
}
```

Code/08\_Threads\_Mutexes/Ex3.c

```
int main ()
{
```

```
    pthread_t thread1_id;
    struct char_print_parms
        thread1_args;
    thread1_args.character = 'X';
```

```
~/Code/08_Threads_Mutexes $ ./Ex3.out
XXXXXXXXXXYYYYYYY Terminando a execucao da thread principal.
~/Code/08_Threads_Mutexes $
```

```
    pthread_create (&thread2_id, NULL,
        &char_print, &thread2_args);
    pthread_join (thread1_id, NULL);
    pthread_join (thread2_id, NULL);
    fputs(" Terminando a execucao da
thread principal.\n", stderr);
    return 0;
}
```

(Continuação)

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

struct char_print_parms
{
```

```
};

void
{
    ~/Code/08_Threads_Mutexes $ ./Ex3.out
XXXXXXXXXXYYYYYYY Terminando a execucao da thread principal.
~/Code/08_Threads_Mutexes $ ./Ex3.out
XXXXXXXXXXYYYYYYY Terminando a execucao da thread principal.
~/Code/08_Threads_Mutexes $
```

```
    parameters;
    int i;
    for (i = 0; i < p->count; ++i)
        fputc(p->character, stderr);
    return NULL;
}
```

Code/08\_Threads\_Mutexes/Ex3.c

```
int main ()
{
    pthread_t thread1_id;
    struct char_print_parms
        thread1_args;
    thread1_args.character = 'X';

    pthread_create (&thread2_id, NULL,
        &char_print, &thread2_args);
    pthread_join (thread1_id, NULL);
    pthread_join (thread2_id, NULL);
    fputs(" Terminando a execucao da
thread principal.\n", stderr);
    return 0;
}
```

(Continuação)



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

struct char_print_parms
{
```

```
int main ()
{
    pthread_t thread1_id;
    struct char_print_parms
        thread1_args;
    thread1_args.character = 'X';
```

```
~/Code/08_Threads_Mutexes $ ./Ex3.out
XXXXXXXXXXYYYYYYY Terminando a execucao da thread principal.
~/Code/08_Threads_Mutexes $ ./Ex3.out
XXXXXXXXXXYYYYYYY Terminando a execucao da thread principal.
~/Code/08_Threads_Mutexes $ ./Ex3.out
XXYYYYXX Terminando a execucao da thread principal.
~/Code/08_Threads_Mutexes $
```

```
        parameters;
    int i;
    for (i = 0; i < parameters->n; i++)
        fputc(p->character, &thread2_args);
    pthread_create (&thread2_id, NULL,
                    char_print, &thread2_args);
    pthread_join (thread1_id, NULL);
    pthread_join (thread2_id, NULL);
    printf ("Terminando a execucao da thread principal.\n", stderr);
    return NULL;
}
```

Code/08\_Threads

Novamente, as *threads* concorrem por uso de CPU e pela escrita no terminal, sendo difícil prever a ordem com que os caracteres serão escritos

(continuação)

# Concorrência em *threads*

- Vimos que processos-filho não compartilham memória com o processo-pai, necessitando de mecanismos de comunicação (pipes, sinais etc.) para que eles troquem informações
- Também vimos que *threads* compartilham memória
- Se duas ou mais *threads* tentarem alterar a mesma posição na memória, podem acontecer erros

```

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

volatile int varCompartilhada=0;

void* incrementa_contador(void *arg)
{
    for(unsigned int i=0; i<10000; i++)
        varCompartilhada++;
    return NULL;
}

void* decrementa_contador (void *arg)
{
    for(unsigned int i=0; i<10000; i++)
        varCompartilhada--;
    return NULL;
}

```

Code/08\_Threads\_Mutexes/Ex4.c

```

int main (int argc, char** argv) {

    pthread_t t0;
    pthread_t t1;

    int res0, res1;
    printf("Valor inicial: %d\n",
        varCompartilhada);
    res0 = pthread_create(&t0,
        NULL,
        &incrementa_contador,
        NULL);
    res1 = pthread_create(&t1,
        NULL,
        &decrementa_contador,
        NULL);
    res0 = pthread_join(t0, NULL);
    res1 = pthread_join(t1, NULL);
    printf("Valor final: %d\n",
        varCompartilhada);
    return 0;
}

```

(Continuação)

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

```
volatile int varCompartilhada=0;
```

```
void* incrementa_contador(void *arg)
{
    for(unsigned int i=0; i<10000; i++)
        varCompartilhada++;
    return NULL;
}
```

```
void* decrementa_contador (void *arg)
{
    for(unsigned int i=0; i<10000; i++)
        varCompartilhada--;
    return NULL;
}
```

Code/08\_Threads\_Mutexes/Ex4.c

**Temos uma variável global, que é acessível para todas as *threads***

```
int main (int argc, char** argv) {
    pthread_t t0;
```

```
    printf("Valor inicial: %d\n", varCompartilhada);
    pthread_create(&t0, NULL, &incrementa_contador,
```

```
        &incrementa_contador,
        NULL);
    res1 = pthread_create(&t1,
        NULL,
        &decrementa_contador,
        NULL);
    res0 = pthread_join(t0, NULL);
    res1 = pthread_join(t1, NULL);
    printf("Valor final: %d\n", varCompartilhada);
    return 0;
}
```

(Continuação)

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

```
volatile int varCompartilhada=0;
```

```
void* incrementa_contador(void *arg)
{
    for(unsigned int i=0; i<10000; i++)
        varCompartilhada++;
    return NULL;
}
```

```
void* decrementa_contador (void *arg)
{
    for(unsigned int i=0; i<10000; i++)
        varCompartilhada--;
    return NULL;
}
```

Code/08\_Threads\_Mutexes/Ex4.c

```
int main (int argc, char** argv) {


    pthread_t t0;
    pthread_t t1;

    int res0, res1;
    printf("Valor inicial: %d\n",
        varCompartilhada);

    pthread_create(&t0,
        NULL, incrementa_contador,
        NULL);
    pthread_create(&t1,
        NULL, decrementa_contador,
        NULL);

    res0 = pthread_join(t0, NULL);
    res1 = pthread_join(t1, NULL);
    printf("Valor final: %d\n",
        varCompartilhada);
    return 0;
}
```

Uma das *threads* irá  
incrementar essa  
variável global  
10000 vezes



(Continuação)

```

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

volatile int varCompartilhada=0;

void* incrementa_contador(void *arg)
{
    for(unsigned int i=0; i<10000; i++)
        varCompartilhada++;
    return NULL;
}

void* decrementa_contador (void *arg)
{
    for(unsigned int i=0; i<10000; i++)
        varCompartilhada--;
    return NULL;
}

```

Code/08\_Threads\_Mutexes/Ex4.c

```

int main (int argc, char** argv) {


    pthread_t t0;
    pthread_t t1;

    int res0, res1;
    printf("Valor inicial: %d\n",
        varCompartilhada);
    res0 = pthread_create(&t0,
        NULL,
        &incrementa_contador,
        NULL);
    res1 = pthread_create(&t1,
        NULL,
        &decrementa_contador,
        NULL);

    pthread_join(t0, NULL);
    pthread_join(t1, NULL);
    printf("Valor final: %d\n",
        varCompartilhada);
}

```

**Outra *thread* irá  
decrementar essa  
variável global  
10000 vezes**



(Continuação)

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

```
volatile int varCompartilhada=0;
```

```
void*
{
    for
    re
}

void* decrementa_contador (void *arg)
{
    for(unsigned int i=0; i<10000; i++)
        varCompartilhada--;
    return NULL;
}
```

**Criamos as duas *threads*...**

```
int main (int argc, char** argv) {

    pthread_t t0;
    pthread_t t1;

    int res0, res1;
    printf("Valor inicial: %d\n",
        varCompartilhada);
    res0 = pthread_create(&t0,
        NULL,
        &incrementa_contador,
        NULL);
    res1 = pthread_create(&t1,
        NULL,
        &decrementa_contador,
        NULL);
    res0 = pthread_join(t0, NULL);
    res1 = pthread_join(t1, NULL);
    printf("Valor final: %d\n",
        varCompartilhada);
    return 0;
}
```

Code/08\_Threads\_Mutexes/Ex4.c

(Continuação)

```

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

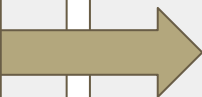
volatile int varCompartilhada=0;

void* incrementa_contador(void *arg)
{
    for(unsigned int i=0; i<10000; i++)
        varCompartilhada++;
    return NULL;
}

void*
{
    fo
    re
}

```

**... e aguardamos a  
execução de ambas.**



```

int main (int argc, char** argv) {

    pthread_t t0;
    pthread_t t1;

    int res0, res1;
    printf("Valor inicial: %d\n",
        varCompartilhada);
    res0 = pthread_create(&t0,
        NULL,
        &incrementa_contador,
        NULL);
    res1 = pthread_create(&t1,
        NULL,
        &decrementa_contador,
        NULL);
    res0 = pthread_join(t0, NULL);
    res1 = pthread_join(t1, NULL);
    printf("Valor final: %d\n",
        varCompartilhada);
    return 0;
}

```

Code/08\_Threads\_Mutexes/Ex4.c

(Continuação)



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

```
volatile int varCompartilhada=0;
```

```
void* incrementa_contador(void *arg)
{
    for(unsigned int i=0; i<10000; i++)
        varCompartilhada++;
    return NULL;
}
```

```
void*
{
    fo
    re
}
```

**A *thread* principal mostra o valor inicial e final da variável global**

```
int main (int argc, char** argv) {

    pthread_t t0;
    pthread_t t1;

    int res0, res1;
    printf("Valor inicial: %d\n",
        varCompartilhada);
    res0 = pthread_create(&t0,
        NULL,
        &incrementa_contador,
        NULL);
    res1 = pthread_create(&t1,
        NULL,
        &decrementa_contador,
        NULL);
    res0 = pthread_join(t0, NULL);
    res1 = pthread_join(t1, NULL);
    printf("Valor final: %d\n",
        varCompartilhada);
    return 0;
}
```

Code/08\_Threads\_Mutexes/Ex4.c

(Continuação)

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

```
volatile
```

```
void
```

```
{
    ~/Code/08_Threads_Mutexes $ ./Ex4.out
    Valor inicial: 0
    Valor final: 86
    ~/Code/08_Threads_Mutexes $
```

```
}
```

```
void
```

```
{
```

```
    varCompartilhada--;
    return NULL;
}
```

```
}
```

```
int main (int argc, char** argv) {
```

```
    pthread_t t0;
```

```
    pthread_t t1;
```

```
    int res0, res1;
```

```
    "
```

```
    printf("Valor final: %d\n",
           varCompartilhada);
    return 0;
```

```
}
```

```
    L);
```

```
    L);
```

Code/08\_Threads\_Mutexes/Ex4.c

(Continuação)

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

```
void
```

```
void
```

```
{
    ~/Code/08_Threads_Mutexes $ ./Ex4.out
    Valor inicial: 0
    Valor final: 86
    ~/Code/08_Threads_Mutexes $ ./Ex4.out
    Valor inicial: 0
    Valor final: 13
    ~/Code/08_Threads_Mutexes $
```

```
void
```

```
{
```

```
    varCompartilhada--;
    return NULL;
}
```

Code/08\_Threads\_Mutexes/Ex4.c

```
int main (int argc, char** argv) {
```

```
    pthread_t t0;
```

```
    pthread_t t1;
```

```
    int res0, res1;
```

```
    "
```

```
    L);
```

```
    L);
```

```
    printf("Valor final: %d\n",
           varCompartilhada);
    return 0;
```

```
}
```

(Continuação)

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

```
void
```

```
void
```

```
{
```

```
}
```

```
void
```

```
{
```

```
}
```

```
~/Code/08_Threads_Mutexes $ ./Ex4.out
Valor inicial: 0
Valor final: 86
~/Code/08_Threads_Mutexes $ ./Ex4.out
Valor inicial: 0
Valor final: 13
~/Code/08_Threads_Mutexes $ ./Ex4.out
Valor inicial: 0
Valor final: -4581
~/Code/08_Threads_Mutexes $
```

```
varCompartilhada--;
return NULL;
```

```
int main (int argc, char** argv) {

    pthread_t t0;
    pthread_t t1;

    int res0, res1;
```

```
    printf("Valor inicial: %d\n",
```

```
    pthread_create(&t0, NULL, funcao, (void*)varCompartilhada);
    pthread_create(&t1, NULL, funcao, (void*)varCompartilhada);
```

```
    printf("Valor final: %d\n",
           varCompartilhada);
    return 0;
```

```
}
```

Code/08\_Threads\_Mutexes/Ex4.c

(Continuação)

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

```
int main (int argc, char** argv) {

    pthread_t t0;
    pthread_t t1;

    int res0, res1;
```

```
void* funcao0(void* arg) {
    ~/Code/08_Threads_Mutexes $ ./Ex4.out
    Valor inicial: 0
    Valor final: 86
    ~/Code/08_Threads_Mutexes $ ./Ex4.out
    Valor inicial: 0
    Valor final: 13
    ~/Code/08_Threads_Mutexes $ ./Ex4.out
    Valor inicial: 0
    Valor final: -4581
    ~/Code/08_Threads_Mutexes $
```

```
    varCompartilhada--;
    return NULL;
}
```

```
    printf("Valor final: %d\n",
           *varCompartilhada);
    return NULL;
}
```

Code/08\_Threads\_Mu

(Continuação)

Esperava-se que a  
variável global  
terminasse valendo 0  
em todos os casos

## Intel Disassembly

### **varCompartilhada++**

```
movl _varCompartilhada(%rip), %eax
```

```
addl $1, %eax
```

```
movl %eax, _varCompartilhada(%rip)
```

### **varCompartilhada--**

```
movl _varCompartilhada(%rip), %eax
```

```
subl $1, %eax
```

```
movl %eax, _varCompartilhada(%rip)
```

Estes são os códigos Assembly correspondentes ao incremento e ao decremento da variável global

```
int main (int argc, char** argv) {
```

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

```
volatile
```

```
void* inc
```

```
{
```

```
for(u
```

```
va
```

```
return
```

```
}
```

```
void* dec
```

```
{
```

```
for(unsigned int i=0; i<10000; i++)
```

```
varCompartilhada--;
```

```
return NULL;
```

```
}
```

```
%d\n",
```

```
&t0,
```

```
or,
```

```
&t1,
```

```
or,
```

```
pthread_join(t0, NULL);
```

```
res1 = pthread_join(t1, NULL);
```

```
printf("Valor final: %d\n",
```

```
varCompartilhada);
```

Code/08\_Threa

inuação)

## Intel Disassembly

### **varCompartilhada++**

```
movl _varCompartilhada(%rip), %eax
```

```
addl $1, %eax
```

```
movl %eax, _varCompartilhada(%rip)
```

### **varCompartilhada--**

```
movl _varCompartilhada(%rip), %eax
```

```
subl $1, %eax
```

```
movl %eax, _varCompartilhada(%rip)
```

### Repare que é necessário

1. copiar o conteúdo da variável global para um registrador,
2. incrementar ou decrementar o registrador e
3. copiar o registrador atualizado para a variável global

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

```
volatile int
```

```
void* incre
```

```
{
    for(unsigned int i=0; i<10000; i++)
        varCompartilhada++;
    return 1;
}
```

```
void* decre
```

```
{
    for(unsigned int i=0; i<10000; i++)
        varCompartilhada--;
    return NULL;
}
```

Thread 0: varCompartilhada++

```
reg1 = varCompartilhada
```

```
reg1 = reg1 + 1
```

```
varCompartilhada = reg1
```

Bloqueado

Thread 1: varCompartilhada--

```
reg1 = varCompartilhada
```

```
reg1 = reg1 - 1
```

```
varCompartilhada = reg1
```

Bloqueado

```
int main (int argc, char** argv) {

    pthread_t t0;
    pthread_t t1;

    int res0, res1;

    printf("Valor inicial: %d\n",
           varCompartilhada);
    pthread_create(&t0,
                  NULL,
                  incre,
                  NULL);
    pthread_create(&t1,
                  NULL,
                  decre,
                  NULL);
    pthread_join(t0, NULL);
    res1 = pthread_join(t1, NULL);
    printf("Valor final: %d\n",
           varCompartilhada);
    return 0;
}
```

Code/08\_Threads\_Mutexes/Ex4.c

(Continuação)



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

```
volatile int
```

```
void* incre
```

```
{
    for(unsigned int i=0; i<100000; i++)
        varCompartilhada++;
    return 1;
}
```

```
void* decre
```

```
{
    for(unsigned int i=0; i<100000; i++)
        varCompartilhada--;
    return NULL;
}
```

Thread 0: varCompartilhada++

```
reg1 = varCompartilhada
```

```
reg1 = reg1 + 1
```

```
varCompartilhada = reg1
```

Bloqueado

Thread 1: varCompartilhada--

```
reg1 = varCompartilhada
```

```
reg1 = reg1 - 1
```

```
varCompartilhada = reg1
```

Bloqueado

```
int main (int argc, char** argv) {

    pthread_t t0;
    pthread_t t1;

    int res0, res1;
```

```
    printf("Initial: %d\n",
```

```
    );
```

```
    pthread_create(&t0,
```

```
    &incre,
```

```
    &t1,
```

```
    &decre,
```

```
    &t0, NULL);
```

```
    res1 = pthread_join(t1, NULL);
```

```
    printf("Final: %d\n",
```

```
    varCompartilhada);
```

**Neste caso, a variável global é incrementada e decrementada corretamente**

**Se ela valia 0 antes desses comandos, ela termina valendo 0 depois deles**

Code/08\_Thread

(continuação)

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

```
volatile int
```

```
void* incre
```

```
{
    for(unsigned int i=0; i<10000; i++)
        varCompartilhada++;
    return NULL;
}
```

```
void* decre
```

```
{
    for(unsigned int i=0; i<10000; i++)
        varCompartilhada--;
    return NULL;
}
```

Thread 0: varCompartilhada++

reg1 = varCompartilhada

reg1 = reg1 + 1

Bloqueado

varCompartilhada = reg1

Thread 1: varCompartilhada--

Bloqueado

reg2 = varCompartilhada

reg2 = reg2 - 1

varCompartilhada = reg2

Bloqueado

```
int main (int argc, char** argv) {

    pthread_t t0;
    pthread_t t1;

    int res0, res1;

    pthread_create(&t0, NULL, incre, NULL);
    pthread_create(&t1, NULL, decre, NULL);

    pthread_join(t0, NULL);
    pthread_join(t1, NULL);

    printf("Valor final: %d\n",
           varCompartilhada);

    return 0;
}
```

Code/08\_Threads\_Mutexes/Ex4.c

(Continuação)

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

```
volatile int
```

```
void* incre
```

```
{
```

```
for(unsig
```

```
varCom
```

```
return 1
```

```
}
```

```
void* decre
```

```
{
```

```
for(unsigned int i=0; i<100000; i++)
```

```
varCompartilhada
```

```
return NULL;
```

```
}
```

Thread 0: varCompartilhada++

```
reg1 = varCompartilhada
```

```
reg1 = reg1 + 1
```

Bloqueado

```
varCompartilhada = reg1
```

Thread 1: varCompartilhada--

Bloqueado

```
reg2 = varCompartilhada
```

```
reg2 = reg2 - 1
```

```
varCompartilhada = reg2
```

Bloqueado

```
al: %d\n",
```

```
);
```

```
ce(&t0,
```

```
ador,
```

```
ce(&t1,
```

```
ador,
```

```
(t0, NULL);
```

```
res1 = pthread_join(t1, NULL);
```

```
for final: %d\n",
```

```
tilhada);
```

**Neste caso, a variável global é incrementada e decrementada incorretamente**

**Se ela valia 0 antes desses comandos, ela termina valendo 1 depois deles**

Code/08\_Threa

inuação)

# Concorrência em *threads*

- Para evitar esta concorrência no uso da variável global, pode-se utilizar um mutex (*mutual exclusive*)
- O mutex indica que uma *thread* tem acesso exclusivo a uma parte crítica do código, como o acesso à variável global no exemplo anterior
- Considere a seguinte analogia\* para entender o mutex:
  - Um grupo de pessoas discute demais, com várias interrupções
  - O chefe decide então que somente a pessoa segurando sua galinha de borracha pode falar
  - As outras pessoas podem somente pedir a sua vez de segurar a galinha de borracha
  - O mutex é a galinha de borracha



\*<https://stackoverflow.com/a/34558>

```

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include <semaphore.h>

volatile int varCompartilhada=0;
static pthread_mutex_t mutexLock;

void* incrementa_contador (void *arg)
{
    for(unsigned int i=0; i<10000; i++)
    {
        pthread_mutex_lock(&mutexLock);
        varCompartilhada++;
        pthread_mutex_unlock(&mutexLock);
    }
    return NULL;
}

void* decrementa_contador (void *arg)
{
    for(unsigned int i=0; i<10000; i++)
    {
        pthread_mutex_lock(&mutexLock);
        varCompartilhada--;
        pthread_mutex_unlock(&mutexLock);
    }
    return NULL;
}

```

```

int main ()
{
    pthread_t t0;
    pthread_t t1;

    printf("Valor inicial: %d\n",
        varCompartilhada);
    pthread_mutex_init(&mutexLock,
        NULL);
    pthread_create(&t0, NULL,
        incrementa_contador,
        NULL);
    pthread_create(&t1, NULL,
        decrementa_contador,
        NULL);
    pthread_join(t0, NULL);
    pthread_join(t1, NULL);
    pthread_mutex_destroy(
        &mutexLock);
    printf("Valor final: %d\n",
        varCompartilhada);
    return 0;
}

```

(Continuação)

```

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include <semaphore.h>

volatile int varCompartilhada=0;
static pthread_mutex_t mutexLock;

void* incrementa_contador(void *arg)
{
    for(unsigned int i=0; i<10000; i++)
    {
        pthread_mutex_lock(&mutexLock);
        varCompartilhada++;
        pthread_mutex_unlock(&mutexLock);
    }
    return NULL;
}

void* decrementa_contador(void *arg)
{
    for(unsigned int i=0; i<10000; i++)
    {
        pthread_mutex_lock(&mutexLock);
        varCompartilhada--;
        pthread_mutex_unlock(&mutexLock);
    }
    return NULL;
}

```

O mutex tem escopo global, para que todas as *threads* o enxerguem

```

int main ()
{
    pthread_t t0;
    pthread_t t1;

    printf("Valor inicial: %d\n",
        varCompartilhada);
    pthread_mutex_init(&mutexLock,
        NULL);
    pthread_create(&t0, NULL,
        incrementa_contador,
        NULL);
    pthread_create(&t1, NULL,
        decrementa_contador,
        NULL);
    pthread_join(t0, NULL);
    pthread_join(t1, NULL);
    pthread_mutex_destroy(
        &mutexLock);
    printf("Valor final: %d\n",
        varCompartilhada);
    return 0;
}

```

(Continuação)

```

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include <semaphore.h>

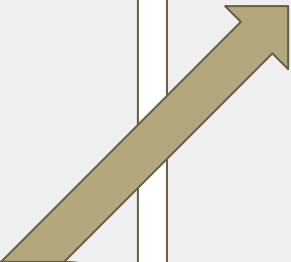
volatile int varCompartilhada=0;
static pthread_mutex_t mutexLock;

void* incrementa_contador (void *arg)
{
    for(unsigned int i=0; i<10000; i++)
    {
        pthread_mutex_lock(&mutexLock);
        varCompartilhada++;
        pthread_mutex_unlock(&mutexLock);
    }
    return NULL;
}

void* decrementa_contador (void *arg)
{
    for(unsigned int i=0; i<10000; i++)
    {
        pthread_mutex_lock(&mutexLock);
        varCompartilhada--;
        pthread_mutex_unlock(&mutexLock);
    }
    return NULL;
}

```

**Ele é inicializado na  
*thread* principal...**



```

int main ()
{
    pthread_t t0;
    pthread_t t1;

    printf("Valor inicial: %d\n",
        varCompartilhada);
    pthread_mutex_init(&mutexLock,
        NULL);
    pthread_create(&t0, NULL,
        incrementa_contador,
        NULL);
    pthread_create(&t1, NULL,
        decrementa_contador,
        NULL);
    pthread_join(t0, NULL);
    pthread_join(t1, NULL);
    pthread_mutex_destroy(
        &mutexLock);
    printf("Valor final: %d\n",
        varCompartilhada);
    return 0;
}

```

(Continuação)

```

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include <semaphore.h>

volatile int varCompartilhada=0;
static pthread_mutex_t mutexLock;

void* incrementa_contador (void *arg)
{
    for(unsigned int i=0; i<10000; i++)
    {
        pthread_mutex_lock(&mutexLock);

        ret

    }
}

void* d
{
    for(unsigned int i=0; i<10000; i++)
    {
        pthread_mutex_lock(&mutexLock);
        varCompartilhada--;
        pthread_mutex_unlock(&mutexLock);
    }
    return NULL;
}

```

**... e é finalizado  
depois que as *threads*  
criadas terminam suas  
execuções**

```

int main ()
{
    pthread_t t0;
    pthread_t t1;

    printf("Valor inicial: %d\n",
        varCompartilhada);
    pthread_mutex_init(&mutexLock,
        NULL);
    pthread_create(&t0, NULL,
        incrementa_contador,
        NULL);
    pthread_create(&t1, NULL,
        decrementa_contador,
        NULL);
    pthread_join(t0, NULL);
    pthread_join(t1, NULL);
    pthread_mutex_destroy(
        &mutexLock);
    printf("Valor final: %d\n",
        varCompartilhada);
    return 0;
}

```

(Continuação)



```

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include <semaphore.h>

volatile int varCompartilhada=0;
static pthread_mutex_t mutexLock;

void* incrementa_contador (void *arg)
{
    for(unsigned int i=0; i<10000; i++)
    {
        pthread_mutex_lock(&mutexLock);
        varCompartilhada++;
        pthread_mutex_unlock(&mutexLock);
    }
    return NULL;
}

void* decrementa_contador (void *arg)
{
    for(unsigned int i=0; i<10000; i++)
    {
        pthread_mutex_lock(&mutexLock);
        varCompartilhada--;
        pthread_mutex_unlock(&mutexLock);
    }
    return NULL;
}

```

```

int main ()
{
    pthread_t t0;
    pthread_t t1;

    printf("Valor inicial: %d\n",
        varCompartilhada);
    pthread_mutex_init(&mutexLock,
        NULL);
    pthread_create(&t0, NULL,
        incrementa_contador,
        varCompartilhada);
    pthread_create(&t1, NULL,
        decrementa_contador,
        varCompartilhada);
    return 0;
}

```

**A função  
pthread\_mutex\_lock()  
“segura a galinha de  
borracha”**

(Continuação)

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

volatile int varCompartilhada=0;
static pthread_mutex_t mutexLock;

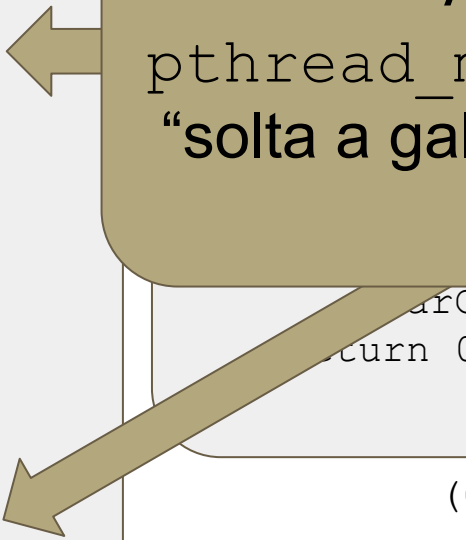
void* incrementa_contador (void *arg)
{
    for(unsigned int i=0; i<10000; i++)
    {
        pthread_mutex_lock(&mutexLock);
        varCompartilhada++;
        pthread_mutex_unlock(&mutexLock);
    }
    return NULL;
}

void* decrementa_contador (void *arg)
{
    for(unsigned int i=0; i<10000; i++)
    {
        pthread_mutex_lock(&mutexLock);
        varCompartilhada--;
        pthread_mutex_unlock(&mutexLock);
    }
    return NULL;
}
```

```
int main ()
{
    pthread_t t0;
    pthread_t t1;

    printf("Valor inicial: %d\n",
        varCompartilhada);
    pthread_mutex_init(&mutexLock,
        NULL);
    pthread_create(&t0, NULL,
        incrementa_contador,
```

**A função  
pthread\_mutex\_unlock()  
“solta a galinha de borracha”**



(Continuação)

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

```
volatile int varCompartilhada=0;
```

```
static
```

```
void
```

```
{
```

```
}
```

```
void
```

```
{
```

```
    for(unsigned int i=0; i<10000; i++)
    {
        pthread_mutex_lock(&mutexLock);
        varCompartilhada--;
        pthread_mutex_unlock(&mutexLock);
    }
    return NULL;
```

```
}
```

```
int main ()
```

```
{
```

```
    pthread_t t0;
```

```
    pthread_t t1;
```

```
    printf("Valor inicial: %d\n",
           varCompartilhada);
```

```
    pthread_t t2;
```

```
~/Code/08_Threads_Mutexes $ ./Ex5.out
```

```
Valor inicial: 0
```

```
Valor final: 0
```

```
~/Code/08_Threads_Mutexes $
```

```
    return 0;
```

```
}
```

(Continuação)

Code/08\_Threads\_Mutexes/Ex5.c

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

```
volatile int varCompartilhada=0;
```

```
static
```

```
void
```

```
{
```

```
}
```

```
void
```

```
{
```

```
for(unsigned int i=0; i<10000; i++)
{
    pthread_mutex_lock(&mutexLock);
    varCompartilhada--;
    pthread_mutex_unlock(&mutexLock);
}
return NULL;
```

```
}
```

```
int main ()
```

```
{
```

```
pthread_t t0;
```

```
pthread_t t1;
```

```
printf("Valor inicial: %d\n",
       varCompartilhada);
```

```
lock,
```

```
~/Code/08_Threads_Mutexes $ ./Ex5.out
```

```
Valor inicial: 0
```

```
Valor final: 0
```

```
~/Code/08_Threads_Mutexes $ ./Ex5.out
```

```
Valor inicial: 0
```

```
Valor final: 0
```

```
~/Code/08_Threads_Mutexes $
```

```
return 0;
```

```
}
```

(Continuação)

Code/08\_Threads\_Mutexes/Ex5.c

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

```
volatile int varCompartilhada=0;
```

```
static
```

```
void
```

```
{
```

```
}
```

```
void
```

```
{
```

```
    for(unsigned int i=0; i<10000; i++)
    {
        pthread_mutex_lock(&mutexLock);
        varCompartilhada--;
        pthread_mutex_unlock(&mutexLock);
    }
    return NULL;
}
```

```
int main ()
```

```
{
```

```
    pthread_t t0;
```

```
    pthread_t t1;
```

```
    printf("Valor inicial: %d\n",
           varCompartilhada);
```

```
    return 0;
```

```
}
```

(Continuação)

Code/08\_Threads\_Mutexes/Ex5.c

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

```
volatile int varCompartilhada=0;
```

```
static
```

```
void
```

```
{
```

```
}
```

```
void
```

```
{
```

```
~/Code/08_Threads_Mutexes $ ./Ex5.out
```

```
Valor inicial: 0
```

```
Valor final: 0
```

```
~/Code/08_Threads_Mutexes $ ./Ex5.out
```

```
Valor inicial: 0
```

```
Valor final: 0
```

```
~/Code/08_Threads_Mutexes $ ./Ex5.out
```

```
Valor inicial: 0
```

```
Valor final: 0
```

```
~/Code/08_Threads_Mutexes $
```

```
for(unsigned int i=0; i<10000; i++)
```

```
{
```

```
pthread_mutex_t
```

```
varCompartilhada
```

```
pthread_mutex_t
```

```
}
```

```
return NULL;
```

```
}
```

```
int main ()
```

```
{
```

```
pthread_t t0;
```

```
pthread_t t1;
```

```
printf("Valor inicial: %d\n",
      varCompartilhada);
```

```
back,
```

A mutex impediu que os valores da variável compartilhada fossem corrompidos, como aconteceu no exemplo anterior

(continuação)

```

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

void *thread_function(void *arg);

int main()
{
    pthread_t a_thread;
    void *thread_result;

    printf("Criando a thread\n");
    pthread_create(&a_thread, NULL,
        thread_function, NULL);
    sleep(5);
    printf("Cancelando a thread\n");
    pthread_cancel(a_thread);
    printf("Thread cancelada!\n");
    return 0;
}

```

Code/08\_Threads\_Mutexes/Ex6.c

```

void *thread_function(void *arg)
{
    int i;
    printf("Nova thread\n");
    for (i = 10; i > 0; i--)
    {
        printf("Faltam %2d
segundos para acabar a thread ...
\n", i);
        sleep(1);
    }
    pthread_exit(0);
}

```

(Continuação)

```

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

void *thread_function(void *arg);

int main()
{
    pthread_t a_thread;
    void *thread_result;

    printf("Criando a thread\n");
    pthread_create(&a_thread,
                  NULL,
                  thread_function,
                  NULL);
    sleep(5);
    printf("Cancelando a thread\n");
    pthread_cancel(a_thread);
    printf("Thread cancelada\n");
    return 0;
}

```

```

void *thread_function(void *arg)
{
    int i;
    printf("Nova thread\n");
    for (i = 10; i > 0; i--)
    {
        printf("Faltam %2d
segundos para acabar a thread ...
\n", i);
        sleep(1);
    }
    printf("Thread acabou\n");
    exit(0);
}

```

**Vejamos agora como uma  
*thread* pode controlar a  
execução de outra *thread***

(continuação)



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

void *thread_function(void *arg);

int main()
{
    pthread_t a_thread;
    void *thread_result;

    printf("Criando a thread\n");
    pthread_create(&a_thread, NULL,
        thread_function, NULL);
    sleep(5);
    printf("Cancelando a thread\n");
    pthread_cancel(a_thread);
    printf("Thread cancelada!\n");
    return 0;
}
```

```
void *thread_function(void *arg)
{
    int i;
    printf("Nova thread\n");
    for (i = 10; i > 0; i--)
    {
        printf("Faltam %2d
segundos para acabar a thread ...
\n", i);
        sleep(1);
    }
    pthread_exit(0);
}
```

**Vamos executar a função  
thread\_function() em  
uma *thread* separada**

Code/08\_Threads\_Mutexes/Ex6.c

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

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

```
#include <unistd.h>
```

```
#include <time.h>
```

```
void main(void)
```

```
{
```

```
    pthread_t a_thread;
```

```
    void *thread_result;
```

```
    printf("Criando a thread\n");
```

```
    pthread_create(&a_thread, NULL,  
                  thread_function, NULL);
```

```
    sleep(5);
```

```
    printf("Cancelando a thread\n");
```

```
    pthread_cancel(a_thread);
```

```
    printf("Thread cancelada!\n");
```

```
    return 0;
```

```
}
```

**Essa função conta de 10 a 0  
em intervalos de 1 segundo...**

```
void *thread_function(void *arg)
```

```
{
```

```
    int i;
```

```
    printf("Nova thread\n");
```

```
    for (i = 10; i > 0; i--)
```

```
    {
```

```
        printf("Faltam %2d
```

```
segundos para acabar a thread ...  
\n", i);
```

```
        sleep(1);
```

```
    }
```

```
    pthread_exit(0);
```

```
}
```

(Continuação)

Code/08\_Threads\_Mutexes/Ex6.c

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

void *thread_function(void *arg);
```

```
int main()
{
```

**... e depois conclui sua  
execução.**

```
    pthread_t a_thread;
    pthread_create(&a_thread, NULL, thread_function, NULL);
    sleep(5);
    printf("Cancelando a thread\n");
    pthread_cancel(a_thread);
    printf("Thread cancelada!\n");
    return 0;
}
```

```
void *thread_function(void *arg)
{
    int i;
    printf("Nova thread\n");
    for (i = 10; i > 0; i--)
    {
        printf("Faltam %2d  
segundos para acabar a thread ...  
\n", i);
        sleep(1);
    }
    pthread_exit(0);
}
```

(Continuação)

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

void *thread_function(void *arg);
```

```
int main()
{
```

**(A explicação da função  
pthread\_exit() será  
oferecida mais à frente.)**

```
    pthread_t a_thread;
    pthread_create(&a_thread, NULL, thread_function, NULL);
    sleep(5);
    printf("Cancelando a thread\n");
    pthread_cancel(a_thread);
    printf("Thread cancelada!\n");
    return 0;
}
```

```
void *thread_function(void *arg)
{
    int i;
    printf("Nova thread\n");
    for (i = 10; i > 0; i--)
    {
        printf("Faltam %2d\n", i);
        sleep(1);
    }
    pthread_exit(0);
}
```

(Continuação)

```

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

void *thread_function(void *arg);

int main()
{
    pthread_t a_thread;
    void *thread_result;

    printf("Criando a thread\n");
    pthread_create(&a_thread,
                  NULL, thread_function, NULL);
    sleep(5);
    printf("Cancelando a thread\n");
    pthread_cancel(a_thread);
    printf("Thread cancelada\n");
    return 0;
}

```

```

void *thread_function(void *arg)
{
    int i;
    printf("Nova thread\n");
    for (i = 10; i > 0; i--)
    {
        printf("Faltam %2d
segundos para acabar a thread ...
\n", i);
        sleep(1);
    }
    pthread_exit(0);
}

```

Neste exemplo, a *thread* principal espera 5 segundos...

(continuação)

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

void *thread_function(void *arg);

int main()
{
    pthread_t a_thread;
    void *thread_result;

    printf("Criando a thread\n");
    pthread_create(&a_thread, NULL,
        thread_function, NULL);
    sleep(5);
    printf("Cancelando a thread\n");
    pthread_cancel(a_thread);
    printf("Thread cancelada!\n");
    return 0;
}
```

```
void *thread_function(void *arg)
{
    int i;
    printf("Nova thread\n");
    for (i = 10; i > 0; i--)
    {
        printf("Faltam %2d
segundos para acabar a thread ...
\n", i);
        sleep(1);
    }
    pthread_exit(0);
}
```

**... e termina a execução da  
*thread* secundária  
prematuramente.**

Code/08\_Threads\_Mutexes/Ex6.c

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

```
#include <unistd.h>
int main(void) {
    pthread_t thread;
```

```
    printf("Criando a thread\n");
```

```
    pthread_create(&thread, NULL, thread_function, NULL);
```

```
    printf("Nova thread\n");
```

```
    printf("Faltam 10 segundos para acabar a thread ...");
```

```
    printf("Faltam 9 segundos para acabar a thread ...");
```

```
    printf("Faltam 8 segundos para acabar a thread ...");
```

```
    printf("Faltam 7 segundos para acabar a thread ...");
```

```
    printf("Faltam 6 segundos para acabar a thread ...");
```

```
    printf("Cancelando a thread\n");
```

```
    pthread_cancel(thread);
```

```
    printf("Thread cancelada!\n");
```

```
    return 0;
```

```
}
```

```
void *thread_function(void *arg)
{
    int i;
```

```
    while(i < 10) {
        printf("Faltam %d segundos para acabar a thread ...", i);
        i++;
    }
```

```
    return NULL;
```

Code/08\_Threads\_Mutexes/Ex6.c

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

```
void *thread_function(void *arg)
{
    int i;
```

```
#include <unistd.h>
~/Code/08_Threads_Mutexes $ ./Ex6.out
```

Criando a thread

```
void *thread_function(void *arg)
{
    Nova thread
```

```
int main(void)
{
    Faltam 10 segundos para acabar a thread ...
```

```
    Faltam 9 segundos para acabar a thread ...
```

```
    Faltam 8 segundos para acabar a thread ...
```

```
    Faltam 7 segundos para acabar a thread ...
```

```
    Faltam 6 segundos para acabar a thread ...
```

```
    Cancelando a thread
```

```
    Thread cancelada!
```

```
~/Code/08_Threads_Mutexes $
```

```
printf("Cancelando a thread\n");
```

```
pthread_cancel(a_thread);
```

```
printf("Thread cancelada\n");
```

```
return 0;
```

```
}
```

Code/08\_Threads\_Mutexes

Ou seja, é possível controlar a execução de uma *thread* secundária.



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

```
void *
```

```
int m
{
```

```
p
v
```

```
p
p
```

```
s
```

```
p
```

```
p
```

```
p
```

```
r
```

```
}
```

**PS: A função `pthread_exit()` permite à *thread* secundária retornar um valor à *thread* que a criou através do segundo parâmetro de entrada da função `pthread_join()`.**

**Com este parâmetro, a *thread* secundária pode, por exemplo, indicar à *thread* primária se houve erro na sua execução**

```
void *thread_function(void *arg)
{
    int i;
    printf("Nova thread\n");
    for (i = 10; i > 0; i--)
    {
        printf("Faltam %2d
segundos para acabar a thread ...
\n", i);
        sleep(1);
    }
    pthread_exit(0);
}
```

(Continuação)