

# OO – Engenharia Eletrônica

---

## Programação em C/C++

---

Slides 18: Introdução à *Multithreading*.

Introdução à *Multithreading*: execução concorrente de tarefas.

Exemplos usando a ‘biblioteca’ [pthread](#) que é de acordo com POSIX.

**Prof. Jean Marcelo SIMÃO**  
**Aluno monitor: Vagner Vengue**

# *POSIX Threads* (Pthreads)

**Obs.: Material inicial elaborado por Murilo S. Holtman no 2o Semestre de 2007. O Holtman era então aluno da disciplina em questão e estagiário em projeto do Professor da disciplina.**

# *POSIX Threads* (Pthreads)

---

- Uma padronização desenvolvida pelo comitê POSIX (*Portable Operating System Interface*) para tornar os programas que utilizam *threads* **portáveis entre várias plataformas**.
- Inclui suporte à criação e controle de *threads* e a alguns objetos de sincronização básicos.

# Elementos Básico de *threading* na ‘biblioteca’ Pthreads

---

## *pthread\_create* e *pthread\_exit*

---

- `pthread_create()` cria uma nova *thread*, que pode (ou não) começar a executar imediatamente após a chamada ter sido concluída.

```
int pthread_create (
                                pthread_t *thread,
                                const pthread_attr_t *attr,
                                void    *(*start_routine)(void*),
                                void    *arg
                                );
```

- `pthread_exit()` é chamada no contexto de uma *thread* criada por `pthread_create()` e faz com que essa *thread* seja finalizada.

```
void pthread_exit(
                                void *value_ptr
                                );
```

# Exemplos de implementação de *threads* (utilizando Pthreads)

# Exemplo 01

---

```
#include <iostream>
#include <pthread.h>
using namespace std;

bool thread_ligada;
pthread_t minha_thread;

// função passada como parâmetro para nova
// thread;
void* escreveAlgo(void *p)
{
    while ( thread_ligada )
    {
        cout << "Executando thread..."
              << endl;
    }

    // termina a thread;
    pthread_exit(&minha_thread);
}
```

```
int main()
{
    cout << "Digite ENTER para iniciar e parar
a thread..." << endl;

    cin.get(); // aguarda um ENTER do usuário;
    thread_ligada = true;

    // inicia uma nova thread, passando como
    // parâmetro a thread e a função que se deseja
    // executar;
    pthread_create(&minha_thread, NULL,
                  escreveAlgo, NULL);

    cin.get(); // aguarda um ENTER do usuário;

    // informa a thread que ela deve parar;
    thread_ligada = false;

    return 0;
}
```

# Exemplo 01

[illegible]

# Elementos Básico de *threading* na ‘biblioteca’ Pthreads

---

## *join e detach*

---

- Uma thread pode aguardar pela finalização de outra *thread* através de `pthread_join()`.

```
int pthread_join(  
                                pthread_t thread,  
                                void **value_ptr  
                                );
```

- `pthread_detach()` avisa à implementação subjacente que a *thread* especificada deve ser liberada assim que terminar.

```
int pthread_detach (  
                                pthread_t *thread  
                                );
```



# Exemplo 02 (Região Crítica)

```
#include <iostream>
#include <pthread.h>
using namespace std;

pthread_t thread_1;
pthread_t thread_2;
// função passada para a thread_1;
void* tarefa_1(void *p) {

    //----- REGIÃO CRÍTICA
    for (char* s = "123456"; *s != '\0'; s++){
        cout << *s;
    }
    //-----
}
// função passada para a thread_2;
void* tarefa_2(void* p) {

    //----- REGIÃO CRÍTICA
    for (char* s = "ABCDEF"; *s != '\0'; s++){
        cout << *s;
    }
    //-----
}
```

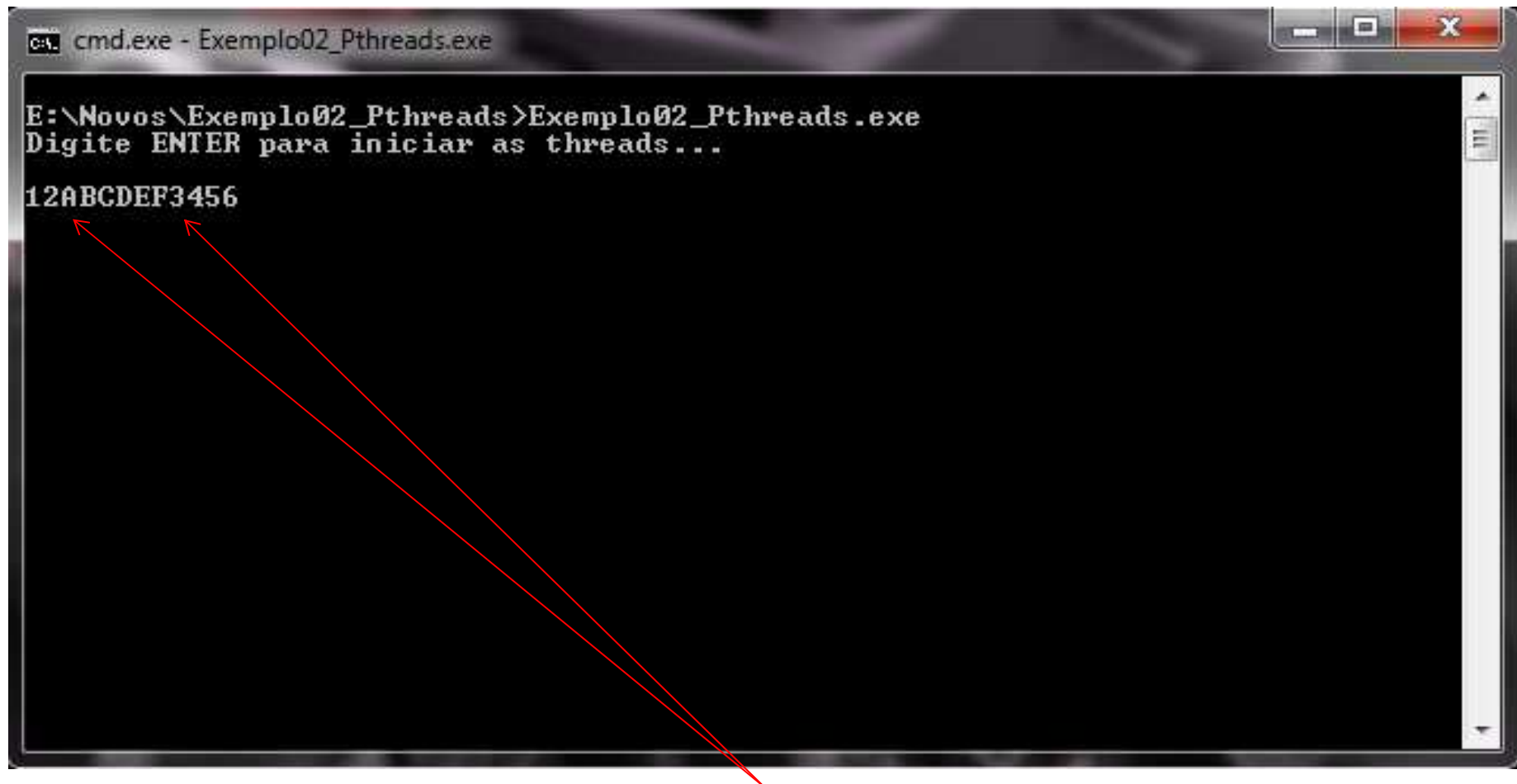
```
int main()
{
    cout << "Digite ENTER para iniciar as
    threads..." << endl;
    cin.get(); // aguarda um ENTER do usuário;

    // inicia as duas threads, passando como
    parâmetro a thread e a função que cada uma
    deve executar;
    pthread_create(&thread_1, NULL,
        tarefa_1, NULL);
    pthread_create(&thread_2, NULL,
        tarefa_2, NULL);

    // faz com que a thread principal espere a
    'thread_1' e a 'thread_2' acabarem;
    pthread_join(thread_1, NULL);
    pthread_join(thread_2, NULL);

    cin.get(); // aguarda um ENTER do usuário;
    return 0;
}
```

## Exemplo 02 (Região Crítica)



```
cmd.exe - Exemplo02_Pthreads.exe
E:\Novos\Exemplo02_Pthreads>Exemplo02_Pthreads.exe
Digite ENTER para iniciar as threads...
12ABCDEF3456
```

- Como o Console (recurso) é compartilhado pelas duas *threads*, os textos se misturam, alterando o resultado correto.

# Mutexes (Pthreads)

---

## Revisando:

- São utilizados para **exclusão mutua**, ou seja, permitem que apenas uma *thread* por vez acesse um recurso.
- Se uma *thread* tenta acessar um recurso que outra *thread* está bloqueando, é impedida e libera o processador para que outras *threads* executem. Isso garante que uma *thread* não desperdice processamento porque está aguardando por um recurso bloqueado por outra *thread*.

# Mutexes (Pthreads)

---

- Um mutex permite acesso de uma *thread* por vez a um recurso através de uma trava mutualmente exclusiva (*mutually exclusive*).
- A função `pthread_mutex_init()` e a função `pthread_mutex_destroy()` respectivamente criam e liberam um objeto mutex.

```
int pthread_mutex_init (
                        pthread_mutex_t *mutex,
                        const pthread_mutexattr_t *attr
                        );

int pthread_mutex_destroy ( pthread_mutex_t *mutex );
```

- Mutexes devem ter sido inicializados antes de serem utilizados (no contexto das *threads*) pelas seguintes funções:

```
int pthread_mutex_lock      ( pthread_mutex_t *mutex );

int pthread_mutex_trylock   ( pthread_mutex_t *mutex );

int pthread_mutex_unlock    ( pthread_mutex_t *mutex );
```

# Exemplo 03 (Mutex)

```
#include <iostream>
#include <pthread.h>
using namespace std;

pthread_t thread_1;
pthread_t thread_2;
pthread_mutex_t meu_mutex;

// função passada para a thread 1;
void* tarefa_1(void *p)
{
    //----- REGIÃO CRÍTICA
    pthread_mutex_lock(&meu_mutex);
    for (char* s = "123456"; *s != '\0'; s++)
    {
        cout << *s;
    }
    pthread_mutex_unlock(&meu_mutex);
    //-----
}
```

```
// função passada para a thread 2;
void* tarefa_2(void* p)
{
    //----- REGIÃO CRÍTICA
    pthread_mutex_lock(&meu_mutex);
    for (char* s = "ABCDEF"; *s != '\0'; s++)
    {
        cout << *s;
    }
    pthread_mutex_unlock(&meu_mutex);
    //-----
}
```

- Com um objeto de Mutex, pode-se sincronizar as *threads*, pois ele só permite que uma *thread* por vez acesse o recurso que esta sendo compartilhado.

# Exemplo 03 (Mutex)

---

```
int main()
{
    cout << "Digite ENTER para iniciar as threads..." << endl;
    cin.get(); // aguarda um ENTER do usuário;

    pthread_mutex_init(&meu_mutex, NULL);

    // inicia as duas threads, passando como parâmetro
    // a thread e a função que cada uma deve executar;
    pthread_create(&thread_1, NULL, tarefa_1, NULL);
    pthread_create(&thread_2, NULL, tarefa_2, NULL);

    // faz com que a thread principal espere a 'thread_1' e a 'thread_2'
    // acabarem;
    pthread_join(thread_1, NULL);
    pthread_join(thread_2, NULL);

    pthread_mutex_destroy(&meu_mutex);

    cin.get(); // aguarda um ENTER do usuário;
    return 0;
}
```

# Exemplo 03 (Mutex)



```
cmd.exe - Exemplo03_Pthreads.exe
E:\Novos\Exemplo03_Pthreads>Exemplo03_Pthreads.exe
Digite ENTER para iniciar as threads...
123456ABCDEF
```

# Exemplo 04 (Mutex)

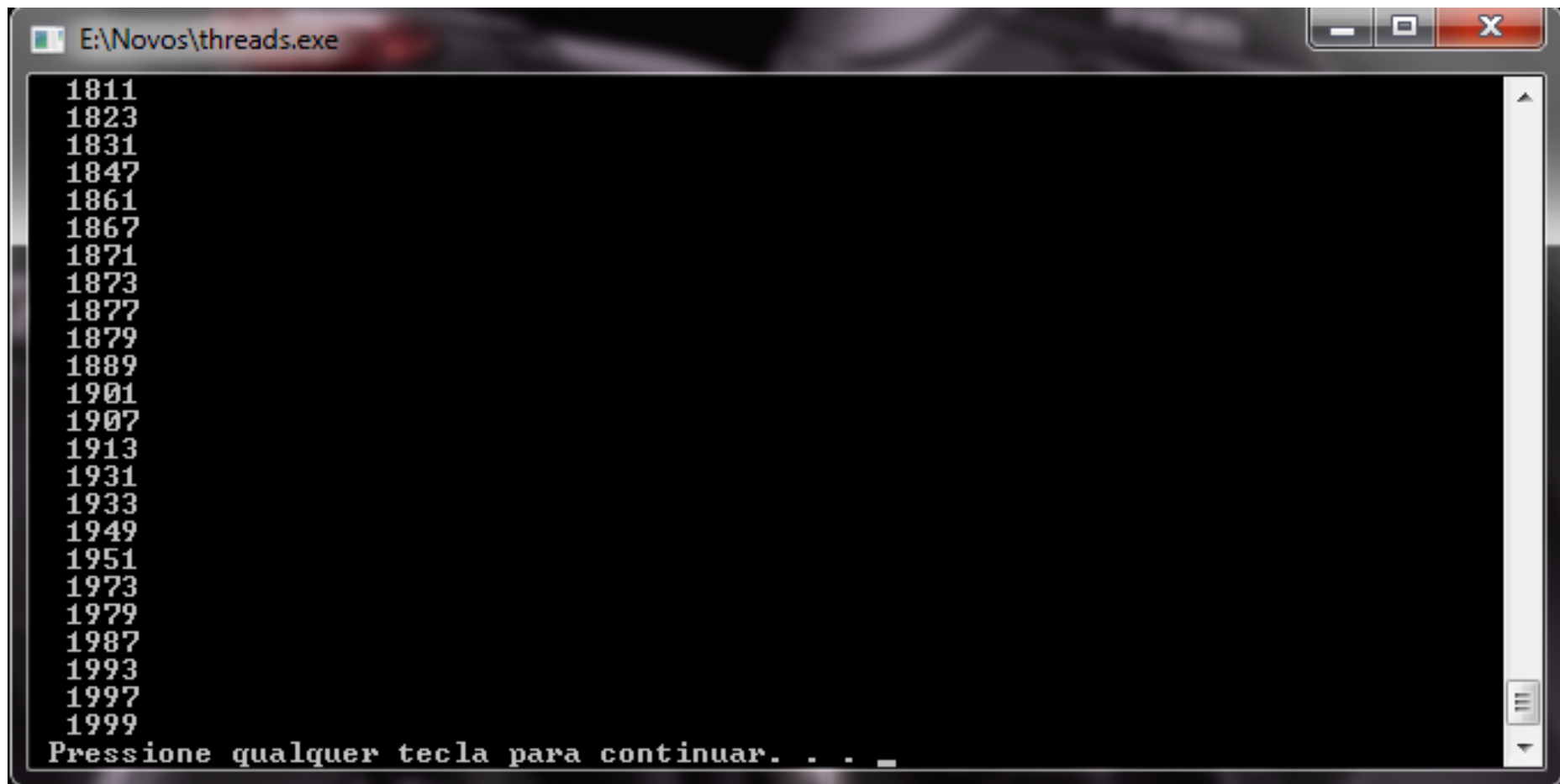
```
#include <stdio.h>
#include <pthread.h>
pthread_mutex_t meu_mutex;
struct limite {
    int baixo;
    int alto;
};
struct limite limite_1, limite_2;
void* encontra_primos( void *param );
int main()
{
    pthread_t thd1, thd2;
    pthread_mutex_init( &meu_mutex, NULL );
    limite_1.baixo = 0;
    limite_1.alto = 1000;
    pthread_create(&thd1,NULL,encontra_primos,&limite_1;
    limite_2.baixo = 1000;
    limite_2.alto = 2000;
    pthread_create(&thd2,NULL,encontra_primos,&limite_2;
    pthread_join( thd1, NULL );
    pthread_join( thd2, NULL );
    pthread_mutex_destroy( &meu_mutex );
    system ("pause");
    return 0;
}
```

```
void* encontra_primos(void *param )
{
    struct limite *aux;
    aux = (struct limite *) param;
    int n, k;
    int primo;

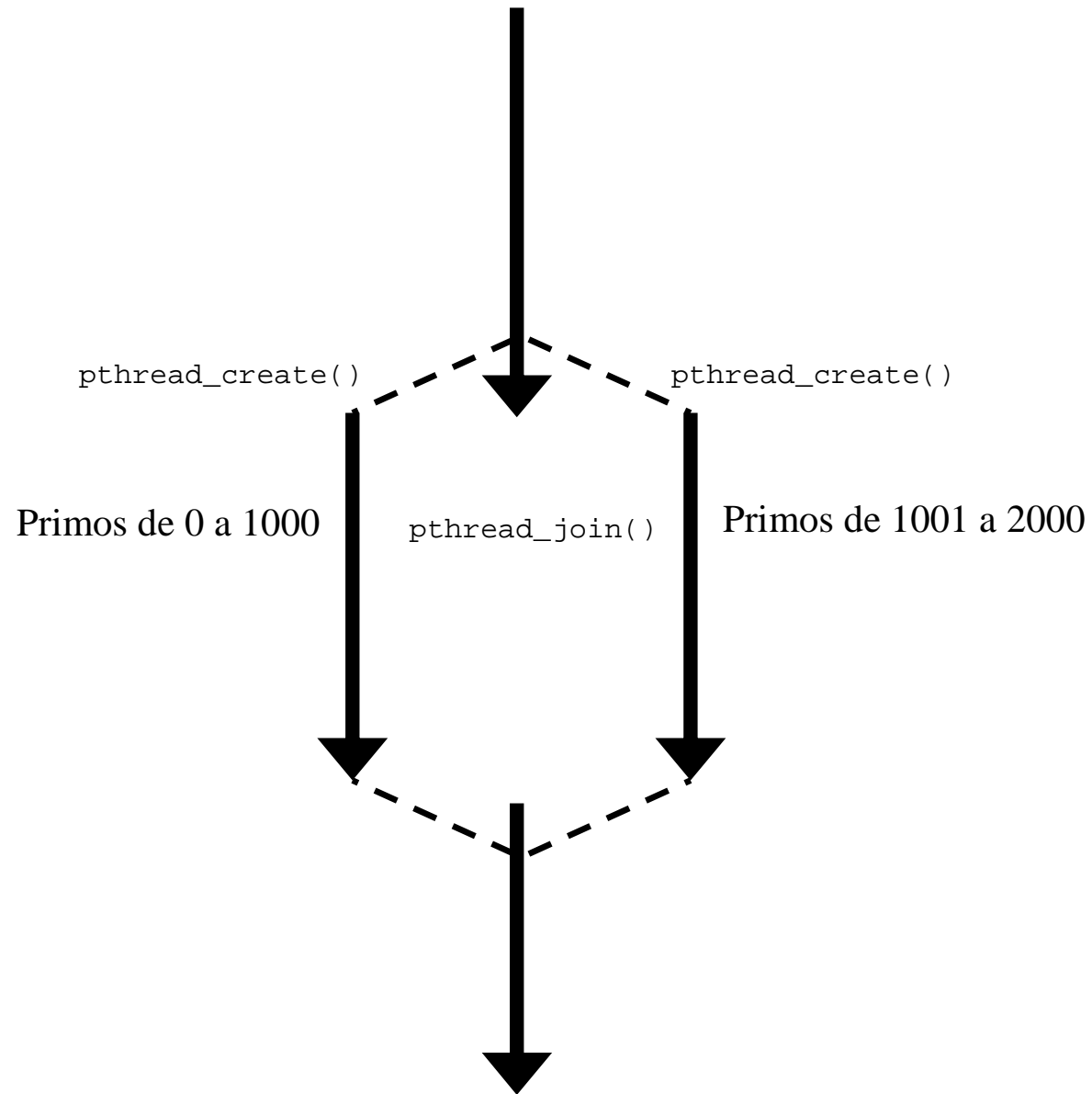
    for (n = aux->baixo; n <= aux->alto; n++)
    {
        primo = 1;
        for (k = 2; k < n; k++) {
            if ( (n % k) == 0 ){
                primo = 0;
                break;
            }
        }
        if ( primo ){
            pthread_mutex_lock(&meu_mutex);
            printf(" %u ", n );
            printf("\n");
            pthread_mutex_unlock(&meu_mutex);
        }
    }
    pthread_exit( NULL );
}
```



## Exemplo 04 (Mutex)



```
E:\Novos\threads.exe
1811
1823
1831
1847
1861
1867
1871
1873
1877
1879
1889
1901
1907
1913
1931
1933
1949
1951
1973
1979
1987
1993
1997
1999
Pressione qualquer tecla para continuar. . . _
```



# Semáforos (Pthreads)

---

## Revisando:

- São mecanismos que **permitem que um determinado número de *threads* tenham acesso a um recurso**. Agindo como um contador que não deixa ultrapassar um limite.
- No momento em que um objeto de semáforo é criado, é especificada a quantidade máxima de *threads* que ele deve permitir. Então cada *thread* que queira acessar o recurso, deve chamar uma função que decrementa em 1 o semáforo (*down*) e, após utilizar o recurso, chamar uma função que incrementa em 1 o semáforo (*up*). Quando o contador do semáforo chega a zero, significa que o número de *threads* chegou ao limite e o recurso ficará bloqueado para as *threads* que chegarem depois, até que pelo menos uma das *threads* que estão utilizando o recurso o libere, incrementado o contador do semáforo.

# Semáforos (Pthreads)

---

- Inicializa um objeto de semáforo.

```
int sem_init(      sem_t *semaforo,  
                  int pshared,  
                  int value );
```

- Parâmetros:
  - *semaforo*: deve ser um ponteiro para um objeto de semáforo.
  - *pshared*: define o escopo do semáforo, que também pode ser utilizado para sincronizar *threads* em processos diferentes. Se todas as *threads* são do mesmo processo, então o valor deve ser 0 (zero).
  - *value*: valor do semáforo, ou seja, a quantidade de *threads* que ele deve suportar.

- Destrói um objeto de semáforo, liberando a memória utilizada.

```
int sem_destroy( sem_t *semaforo );
```

- Parâmetro:  
*semaforo*: ponteiro para o objeto de semáforo.

# Semáforos (Pthreads)

---

- A função `sem_wait()` faz uma requisição de acesso ao semáforo, se o número de *threads* ainda não chegou ao limite, então a thread obtém o acesso, senão aguarda até receber acesso.

```
int sem_wait(      sem_t *semaforo );
```

- Parâmetro:
  - *semaforo*: ponteiro para o objeto de semáforo.

- A função `sem_post()` deve ser usada após a thread ter chamado a função `sem_wait()`, indicando que não precisa mais de acesso ao recurso compartilhado e permitindo que outras *threads* que estejam esperando pelo recurso, obtenham acesso.

```
int sem_post(      sem_t *semaforo );
```

- Parâmetro:
  - semaforo*: ponteiro para o objeto de semáforo.

# Exemplo 05 (Semáforo)

```
#include <iostream>
#include <pthread.h>
#include <semaphore.h>
using namespace std;

pthread_t thread_1;
pthread_t thread_2;
sem_t meu_semaforo;

// função passada como parâmetro para a thread
// 1;
void* tarefa_1(void *p)
{
    //-----
    REGIÃO CRÍTICA
    sem_wait(&meu_semaforo);
    for (char* s = "123456"; *s != '\0'; s++)
    {
        cout << *s;
    }
    sem_post(&meu_semaforo);
    //-----
}
```

```
// função passada como parâmetro para a thread
// 2;
void* tarefa_2(void* p)
{
    //-----
    REGIÃO CRÍTICA
    sem_wait(&meu_semaforo);
    for (char* s = "ABCDEF"; *s != '\0'; s++)
    {
        cout << *s;
    }
    sem_post(&meu_semaforo);
    //-----
}
```

- Um objeto de semáforo também pode ser utilizado para sincronizar *threads* através de exclusão mutua, definindo-se o seu valor como 1.
- Este tipo de semáforo é conhecido como **semáforo binário**, pois ele só tem dois estados: ocupado e desocupado.

# Exemplo 05 (Semáforo)

---

```
int main()
{
    cout << "Digite ENTER para iniciar as threads..." << endl;
    cin.get();          // aguarda um ENTER do usuário;

    sem_init(&meu_semaforo, 0, 1);

    // inicia as duas threads, passando como parâmetro
    // a thread e a função que cada uma deve executar;
    pthread_create(&thread_1, NULL, tarefa_1, NULL);
    pthread_create(&thread_2, NULL, tarefa_2, NULL);

    // faz com que a thread principal espere a 'thread_1' e a 'thread_2'
    // acabarem;
    pthread_join(thread_1, NULL);
    pthread_join(thread_2, NULL);

    sem_destroy(&meu_semaforo);

    cin.get();          // aguarda um ENTER do usuário;
    return 0;
}
```

# Exemplo 05 (Semáforo)



```
cmd.exe - Exemplo04_Pthreads.exe
E:\Novos\Exemplo04_Pthreads>Exemplo04_Pthreads.exe
Digite ENTER para iniciar as threads...
123456ABCDEF
```



# Exemplo 06 (Semáforo)

```
#include <iostream>
#include <windows.h> // para de Sleep();
#include <pthread.h> // funções de pthreads;
#include <semaphore.h> // funções de semáforos;
using namespace std;

// número de threads;
#define NUM_THREADS      5
// número de threads permitidas pelo semáforo;
#define NUM_SEMAFORO     3

pthread_t vt_thread[NUM_THREADS];
int thread_ativa[NUM_THREADS];

sem_t meu_semaforo; // objeto de semáforo;
```

```
// função passada para as threads;
void* tarefa(void *p)
{
    // recupera o id da thread;
    int* valor = static_cast<int*>( p );
    int id = *valor;
    //----- REGIÃO CRÍTICA
    sem_wait(&meu_semaforo);
    for (int i = 0; i < 5; i++)
    {
        thread_ativa[ id ] = 1;
        // marca a thread como ativa e aguarda
        até ser desmarcada pela thread principal;
        while ( 1 == thread_ativa[ id ] )
        {
            Sleep(100);
        }
        thread_ativa[ id ] = -1; // informa a thread
        principal que já acabou o seu trabalho;
        sem_post(&meu_semaforo);
    }
    //-----
    pthread_exit(0);
}
```

# Exemplo 06 (Semáforo)

```
void monitorar()
{
    for (int i = 0; i < NUM_THREADS; i++)
    {
        cout << "Thread " << i << "\t";
    }
    cout << endl;

    bool threads_executando;
    do{
        // verifica se há pelo menos 1 thread
        executando;
        threads_executando = false;
        for (int i = 0; i < NUM_THREADS; i++)
        {
            if ( -1 != thread_ativa[ i ] )
            {
                threads_executando = true;
            }
        }
    }
```

```
        if ( threads_executando )
        {
            // imprime no Console as threads que
            estão executando e que estão ativas, ou
            seja, estão com acesso ao semáforo;
            for (int i=0; i < NUM_THREADS; i++)
            {
                if ( 1 == thread_ativa[i] )
                {
                    cout << "|";
                    thread_ativa[ i ] = 0;
                }
                cout << "\t\t";
            }
            cout << endl;
            Sleep(500);
        }
    }while ( threads_executando );
}
```

# Exemplo 06 (Semáforo)

---

```
int main()
{
    cout << "Digite ENTER para iniciar as threads..." << endl;
    cin.get();

    for (int i = 0; i < NUM_THREADS; i++) {
        thread_ativa[ i ] = 0;
    }
    // inicializa o semáforo;
    sem_init(&meu_semaforo, 0, NUM_SEMAFORO);
    // inicia as threads, passando como parâmetro
    // a thread e a função que cada uma deve executar;
    for (int i = 0; i < NUM_THREADS; i++) {
        pthread_create(&vt_thread[ i ], NULL, tarefa, (void*) &i );
        Sleep(100);
    }

    monitorar();    // função que mostra as threads que estão ativas;

    sem_destroy(&meu_semaforo);
    cout << "Pressione ENTER para continuar..." << endl;
    cin.get();
    return 0;
}
```

## Exemplo 06 (Semáforo)



```
cmd.exe - Exemplo05_Pthreads.exe

Thread 0      Thread 1      Thread 2      Thread 3      Thread 4
!             !             !             !             !
!             !             !             !             !
!             !             !             !             !
!             !             !             !             !
!             !             !             !             !
!             !             !             !             !

!             !
!             !
!             !
!             !
!             !

Pressione ENTER para continuar...
```

- Esta é uma possível saída do programa, mostrando a sequência em que 5 *threads* obtiveram acesso ao semáforo (e a um possível recurso associado a ele, como o próprio Console).
- Como o objeto de semáforo criado só permitiu um máximo de 3 *threads*, as *threads* 3 e 4 tiveram que aguardar as três primeiras acabarem os seus trabalhos e liberarem o semáforo.

# Mais informações (técnicas)

---

- Para instalar em Dev C++
  - Ir no menu Ferramentas e depois no sub-menu Atualizações (++). No campo *'Select devpack server'*, selecionar *'devpaks.org Community Devpacks'* e então clicar no botão *'Check for updates'*.
  - Após as atualizações, selecionar item relativo a *Pthreads* e clicar no botão *'Download selected'*. Uma vez realizado o *download*, ativa-se um instalador...
  - Ao compilar um projeto com pthread, pode acontecer um erro causado pela falta dos arquivos libpthreadGC2.a e pthreadGC2.dll. O primeiro se encontra no diretório (default): *'...\DevC++\Dev-Cpp\lib'* e o segundo pode ser encontrado no link: *'ftp://sourceware.org/pub/pthreads-win32/dll-latest/lib/'*. (último acesso em 20/05/2010).
  - O arquivo libpthreadGC2.a deve ser adicionado ao projeto indo no menu Projeto, depois no sub-menu Opções de Projeto e, na guia superior Parâmetros, clicando-se em Adicionar Biblioteca.
  - O arquivo pthreadGC2.dll deve ser adicionado aos arquivos do Dev C++ (*\DevC++\Dev-Cpp\bin*) ou as pastas do sistema operacional, como *'C:\Windows\System32'*, que além de não ser recomendado, ainda pode ser mais trabalhoso, devido aos sistemas de segurança dos sistemas operacionais modernos.

# Mais informações (técnicas)

---

- Para instalar *Pthread* em outros...
  - Pesquisar na internet...
  - Talvez em: [http://koti.mbnet.fi/outgun/documentation/compiler\\_install.html#pthreads](http://koti.mbnet.fi/outgun/documentation/compiler_install.html#pthreads)  
Obs. : Último acesso foi em 19/05/2008, às 21:05.
- Infos
  - <http://sources.redhat.com/pthreads-win32/>  
Obs. : Último acesso foi em 19/05/2008, às 21:05.
- Infos
  - <http://www.opengroup.org/onlinepubs/007908799/xsh/pthread.h.html>
  - Todas as *man pages* relativas a Pthreads.
  - Obs. : Último acesso foi em 19/05/2008, às 20:07.

# Apenas Lembrando: Número primo

---

**Número primo** é um [número inteiro](#) com apenas quatro divisores inteiros: 1, -1, seu oposto e ele mesmo. Por exemplo, o número 3 é um número primo pois seus dois únicos divisores inteiros são 1 e 3, -1 e -3. Se um número inteiro tem módulo maior que 1 e não é primo, diz-se que é **composto**. Os números [0](#) e [1](#) não são considerados primos nem compostos.

O conceito de número primo é muito importante na [teoria dos números](#). Um dos resultados da teoria dos números é o [Teorema Fundamental da Aritmética](#), que afirma que qualquer número natural pode ser escrito de forma única (desconsiderando a ordem) como um produto de números primos (chamados **fatores primos**): este processo se chama decomposição em fatores primos ([fatoração](#)).

Os 25 primeiros números primos positivos são:

[2](#), [3](#), [5](#), [7](#), [11](#), [13](#), [17](#), [19](#), [23](#), [29](#), [31](#), [37](#), [41](#), [43](#), [47](#), [53](#), [59](#), [61](#), [67](#), [71](#), [73](#), [79](#), [83](#), [89](#), [97](#)...

# Bibliografias relativas a *Threads*.

- SCHILDT, H.: **The Art of C++**. McGraw-Hill Osborne Media. 1ª Edição (Paperback) 2004. ISBN-10: 0072255129
- RICHARD, H. C.; KUO-CHUNG, T.: **Modern Multithreading: Implementing, Testing, and Debugging Multithreaded Java and C++/Pthreads/Win32 Programs**. Wiley-Interscience (Paperback) 2005. ISBN-10: 0471725048
- HUGHES C; HUGHES T.: **Object-Oriented Multithreading Using C++**. Wiley; (Paperback) 1997. ISBN-10: 0471180122.
- HUGHES C; HUGHES T.: **Professional Multicore Programming: Design and Implementation for C++ Developers**. Wrox (Paperback) 2008. ISBN-10: 0470289627



# Atividades – Exercícios

- Estudar o exemplo 4, testando o mesmo programa sem o uso de mutex.
- Estudar o exemplo 6, testando o programa com diferentes valores para o semáforo.
- Pesquisar/Estudar como programar com Pthread de maneira orientada a objetos.
- Vide
  - <http://www.linuxselfhelp.com/HOWTO/C++Programming-HOWTO-18.html>
  - <http://threads.sourceforge.net/>