Universidade Tecnológica Federal do Paraná
UTFPR – Campus Curitiba

Orientação a Objetos
-
Programação em C++

Grupo de Slides 18 – Parte C: Introdução à *Multithreading.*

Programação OO *multithreading* com *pthreads*.

Exemplos usando a 'biblioteca' *pthread*, que é de acordo com POSIX.

Aluno monitor em 2010: Vagner Vengue

Prof.: Jean Marcelo Simão

Threads Orientadas a Objeto

Thread Orientada a Objeto

- O padrão POSIX de *threads* torna o código fonte mais portável, uma vez que este padrão é aceito pelos principais sistemas operacionais atuais, como Linux, Unix e Windows.

- As pthreads são preparadas para trabalhar com funções, o que é uma vantagem para linguagem estruturadas, como a linguagem C.

- Não obstante, além de ser mais conveniente trabalhar com as *threads* de maneira OO na linguagem C++, a implementação e a compreensão do código se tornam mais simples.

Primeira tentativa

(contra-exemplo)

Primeira tentativa

```
#include <iostream>
using namespace std;

#include <stdlib.h>
#include <pthread.h>                    // biblioteca pthreads;


class Thread
{
private:
    pthread_t _threadID;                // pthread - ponteiro ;
    pthread_attr_t _tAttribute;         // atributo da thread, pode conter
                                        // informações sobre o funcionamento da thread;
private:
    void* runThread(void* pThread);// inicia a thread;
    virtual void run();                 // método que executará;
public:
    Thread();
    virtual ~Thread();
    void start();
    void join();                        // espera a thread acabar;
    void yield();                       // libera o processador;
};
```

Primeira tentativa

```
Thread::Thread()    {   }
Thread::~Thread()   {   }
void Thread::run()  {   }           // deve ser redefinido;

void Thread::start()
{
    // inicia o atributo;
    int status = pthread_attr_init( &_tAttribute );
    status = pthread_attr_setscope( &_tAttribute,
                                    PTHREAD_SCOPE_SYSTEM );
    // cria uma thread
    status = pthread_create( &_threadID,
                             &_tAttribute,
                             Thread::runThread,
                             NULL );
    // destrói o atributo;
    status = pthread_attr_destroy( &_tAttribute );
}

void Thread::yield()
{
    sched_yield();
}
```

Por definição das pthreads, deve ser usado um ponteiro de função e esta deve receber como parâmetro um ponteiro do tipo void.

Primeira tentativa

```
void Thread::join()
{
    pthread_join( _threadID, NULL );
}

void* Thread::runThread(void* pThread)
{
    run();                    // executa a thread;
}

class MinhaT : public Thread        // classe derivada de Thread;
{
public:
    MinhaT()     {    }
    ~MinhaT()    {    }

    void run()                      // define o método virtual para
    {                               // executar o código desejado;
        cout << "Oi" << endl;
    }
};
```

Primeira tentativa

```
int main()
{
    MinhaT *t = new MinhaT();       // cria um objeto da classe derivada;
    t->start();                     // inicia a thread;
    t->join();                      // comando join, para que a thread
                                    // main espere a 't' acabar;
    cout << endl << "Oi! Eu sou a thread principal." << endl;

    cout << endl << "pressione ENTER para continuar ...";
    cin.get();
    delete t;
    return 0;
}
```

Message

```
In member function `void Thread::start()':
no matches converting function `runThread' to type `
candidates are: void* Thread::runThread(void*)
[Build Error] [main.o] Error 1
```

Mensagem de erro do compilador (Dev C++), que indica que o ponteiro de função passado como parâmetro não é compatível. Na verdade, todo o método de um objeto tem um parâmetro adicional implícito (o parâmetro *this*), o que atrapalha o uso da função **pthread_create.** Para entender em detalhes, veja o Exemplo 01.

Exemplo 01

Exemplo 01

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

class Thread{
private:
    pthread_t _threadID;              // pthread;
    pthread_attr_t _tAttribute;       // atributo da thread, pode conter
                                      // informações sobre o funcionamento da thread;
private:
    // método que inicia as thread;
    static void* runThread(void* pThread);
    virtual void run();               // código para execução;
    void printError(const string msg) const;
public:
    Thread();
    virtual ~Thread();

    void start();                     // cria a thread;
    void join();                      // espera a thread acabar;
    void yield();                     // libera o processador;
};
```

A mensagem de erro da primeira tentativa pode ser solucionada com um método estático.

Exemplo 01

```
Thread::Thread()    {    }
Thread::~Thread()   {    }
void Thread::run()  {    }

void Thread::yield(){
    sched_yield();
}

void Thread::start(){
    // inicia o atributo;
    int status = pthread_attr_init( &_tAttribute );
    status = pthread_attr_setscope( &_tAttribute, PTHREAD_SCOPE_SYSTEM );
    if ( status != 0)
        printError("falha ao iniciar atributo da thread.");

    // cria uma thread;
    status = pthread_create( &_threadID, &_tAttribute, Thread::runThread, (void*)this);
    if ( status != 0)
        printError("falha ao iniciar a thread.");

    // destrói o atributo;
    status = pthread_attr_destroy( &_tAttribute );
    if ( status != 0)
        printError("falha ao destruir atributo da thread.");
}
```

Exemplo 01

```
void Thread::join() {
    int status = pthread_join( _threadID, NULL );
    if ( status != 0)
        printError("comando join falhou.");
}

void* Thread::runThread(void* pThread) {
    Thread* sThread = static_cast<Thread*>( pThread );

    if ( NULL == sThread )  { cout << "thread falhou." <<endl;      }
    else {     sThread->run();  /* executa a thread; */             }
}

void Thread::printError(const string msg) const {
    cout << "Erro: " << msg << endl;
}
```

- Quando um método é declarado, implicitamente também é declarado um ponteiro da classe como parâmetro, o ponteiro '*this*'. Desta forma, um método que supostamente tem um parâmetro na definição de uma classe, possui na verdade **dois** ponteiros, como é o caso do método feito no contra-exemplo na primeira tentativa.

- Entretanto, a função *pthread_create* de *pthread* devem receber como parâmetro apenas **um** ponteiro como parâmetro. Isso gerou o erro do contra-exemplo. Para contornar esse problema, deve-se utilizar um método estático, que não possui o ponteiro 'this'.

- Na verdade, tanto métodos estáticos quanto funções simples de programação estruturada (modularizada) não recebem o ponteiro *this*...

Exemplo 01

```
// qualquer classe derivada de Thread é uma classe de thread;
class MinhaT : public Thread
{
public:
    MinhaT()    {   }
    ~MinhaT()   {   }

    void run() {
        for (int i = 0; i < 10; i++)
            cout << i+1 << " Oi! " << endl;
    }
};

int main() {
    MinhaT *t;
    t = new MinhaT();

    t->start();
    t->join();

    cout << endl << "Oi! Eu sou a thread principal." << endl;

    cout << endl << "pressione ENTER para continuar ...";
    cin.get();
    delete t;
    return 0;
}
```

Após definida a classe Thread, todas as *threads* do programa podem ser derivadas dela, como pode ser visto ao lado.

Exemplo 01

Resultado do programa Exemplo 01.

Exemplo 02

Thread Orientada a Objeto

- No mundo de competições da Formula 1, os dois irmãos Michael e Ralf, ambos pilotos, conseguem ter uma plena relação de harmonia. Umas das razões é o fato de não conversarem sobre as corridas quando saem juntos. Vamos supor que em uma corrida de 100 voltas, cada um deles pare de acelerar a cada 10 voltas para que o outro irmão não fique para trás.

- Isto considerado, construa um programa que simule esta situação da seguinte forma:

  - Cada um dos dois pilotos deve ser uma *thread*.
  - A cada dez voltas eles param de acelerar (processar).
  - A *thread* principal deve esperar os dois pilotos acabarem a corrida antes de terminar o programa.
  - Cada piloto deve informar no Console quando acabar as 100 voltas.
  - Desenvolva o programa usando programação orientada a objeto.

Exemplo 02

```
#ifndef _CORREDOR_H_
#define _CORREDOR_H_

#include "Thread.h"

// classe Corredor derivada de Thread;
class Corredor : public Thread
{
private:
    string _nome;
    int _nVoltas;
    void run();
public:
    Corredor(const int voltas = 0);
    ~Corredor();

    void setNome(const string nome);
    string getNome() const;
    void setVoltas(const int voltas);
    int getVoltas() const;
};
#endif
```

- Neste exemplo foi utilizada a classe Thread do Exemplo 1, que serve de base para a classe Corredor (*thread*), vista ao lado.

Exemplo 02

```
#include "Corredor.h"
#include "Principal.h"

Corredor::Corredor(const int voltas) {
    setVoltas( voltas );
}

Corredor::~Corredor() {   }

// redefinição do método virtual de Thread;
void Corredor::run() {
    for(int i = 0; i < _nVoltas; i++) {
        if ( i % 10 == 0 )
            yield(); // irmãos sendo generosos;
    }

    while( true ) {
        if ( ! Principal::consoleOcupado )
        {
            Principal::consoleOcupado = true;
            cout << getNome()
                << " chegou!" << endl;
            Principal::consoleOcupado = false;
            break;
        }
    }
}
```

- Na classe Corredor é preciso apenas redefinir o método virtual 'run' com o código que se deseja executar.

- A cada 10 voltas o "corredor" chama o método yield, que informa ao programa que naquele momento ele está "oferecendo" os recursos de processamento para outra *thread*.

- Para evitar problemas de concorrência das *threads* pelo Console, foi criada uma variável estática que informa se o Console está ocupado.

- **Está solução ameniza o problema de concorrência, mas não o resolve**, uma vez que a *thread* pode ser interrompida exatamente entre o comando 'if' e a linha seguinte, onde a variável deveria mudar para o estado ocupado (região crítica). Além de desperdiçar processamento.

Thread Orientada a Objeto com mutex

- Duas ou mais *threads* podem existir no mesmo processo sem interferir uma no trabalho da outra, porém, na maioria das vezes elas precisam compartilhar recursos, uma vez que esse compartilhamento pode ser uma vantagem no uso de múltiplas *threads* em um programa.

- No exemplo anterior foi utilizada uma técnica chamada *Busy Waiting* para gerenciar o uso do Console pelas duas *threads*. Essa técnica além de não eliminar o problema de concorrência, ainda consome processamento.

- O *mutex* é uma forma desenvolvida pelos fabricantes de sistemas operacionais, que utiliza uma técnica parecida com o *Busy Waiting*, porém as chances de apresentar problemas são praticamente zero. Essa técnica também tem como principal vantagem o melhor aproveitamento do processador. No momento em que a *thread* não consegue acessar a região crítica, ela chama o comando "*yield*" automaticamente, liberando o processador.

Exemplo 03

(Exemplo de uso de mutex)

Thread Orientada a Objeto com mutex

- No exemplo anterior foi desenvolvido um programa que simulava uma situação de corrida entre os irmãos Michael e Ralf, adapte-o para funcionar com mais corredores, utilizando um mutex e com as seguintes condições:
  - Cada um dos pilotos deve ser uma *thread*.
  - Todos devem correr até que seja pressionado ENTER.
  - Ao final deve ser informado o número de voltas que cada um fez e o vencedor.
  - Desenvolva o programa usando programação orientada a objeto.

Exemplo 03

```
// includes ...

class Thread
{
private:
    pthread_t _threadID;            // pthread;
    static pthread_mutex_t _mutex;  // mutex estático;
    // atributo da thread, pode conter
    // informações sobre o funcionamento da thread;
    pthread_attr_t _tAttribute;
    // método que inicia as thread;
    static void* runThread(void* pThread);
    virtual void run();             // código para execução;
    void printError(const string msg);
public:
    Thread();
    virtual ~Thread();

    void start();                   // cria a thread;
    void join();                    // espera a thread acabar;
    void yield();                   // libera o processador;
    void lock();
    void unlock();
};
#endif
```

- Objeto estático de mutex, ou seja, comum a todas as *threads*.

Exemplo 03

```
void Thread::lock()
{
    if ( NULL == Thread::_mutex )
        pthread_mutex_init( &Thread::_mutex, NULL );

    pthread_mutex_lock( &Thread::_mutex );
}

void Thread::unlock()
{
    if ( NULL != Thread::_mutex )
        pthread_mutex_unlock( &Thread::_mutex );
}

void Thread::printError(const string msg)
{
    lock();     // pára para a mensagem;
    cout << "Erro: " << msg << endl;
    unlock();
}
```

Exemplo 03

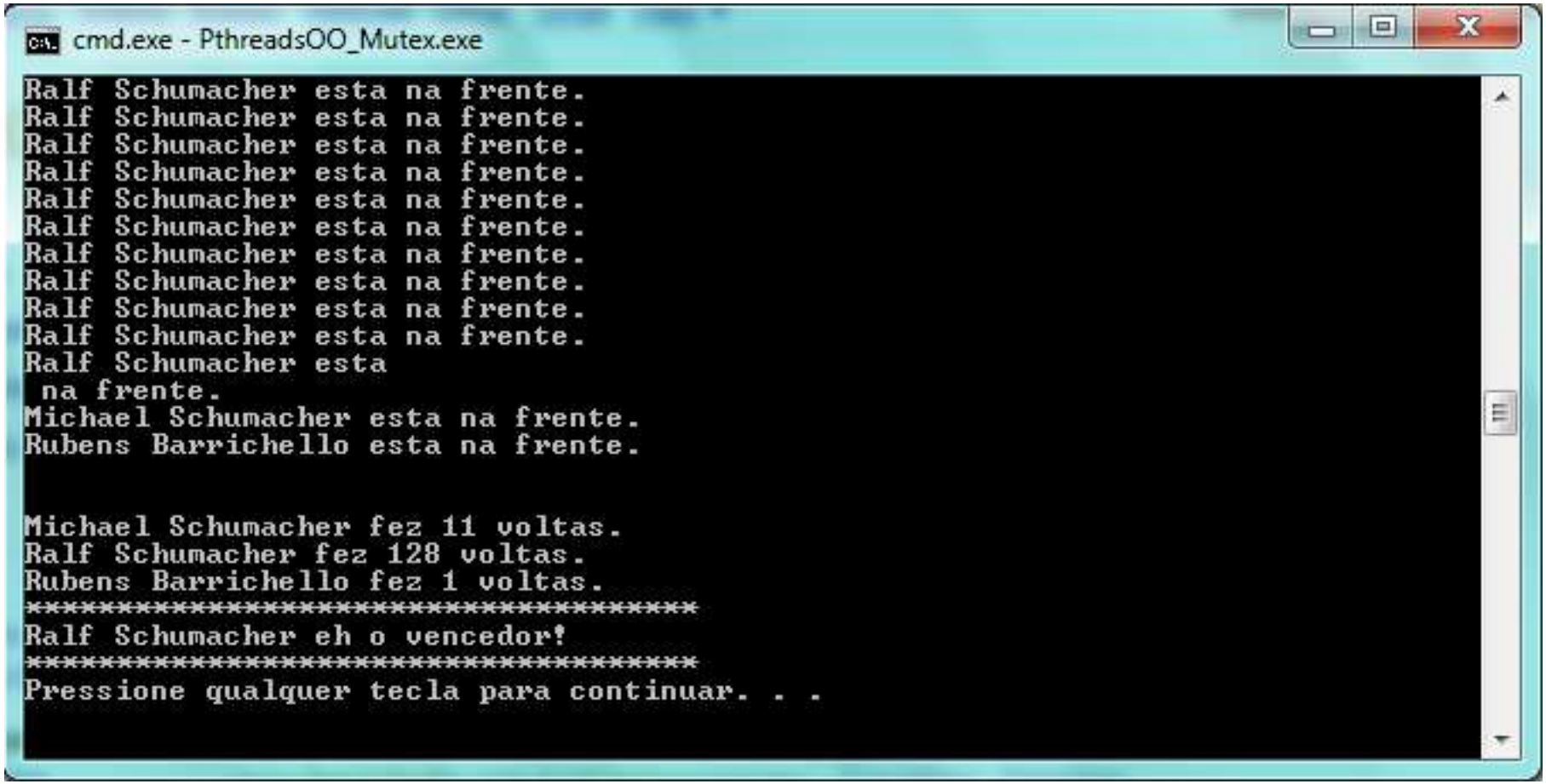
```
Corredor::Corredor(const string nome) {
    _nome = nome;   _voltas = 0;
}

Corredor::~Corredor()   {   }

void Corredor::run()
{   // enquanto o semáforo estiver aberto;
    // este semáforo é o de "trânsito" da corrida...
    // não confundir com semáforos de threads...
    while ( Principal::_semaforoAberto )
    {
        lock();
        cout << _nome << " esta na frente." << endl;
        unlock();
        _voltas ++;     // incrementas as voltas

        // a cada 10 voltas, avisa o SO que
        // está oferecendo o processador para
        // outra thread;
        if ( _voltas % 10 == 0)
            yield();
    }
}
```

- Cada vez que é preciso imprimir uma mensagem, a *thread* chama a função lock.

- É, até certo ponto, o desenvolvedor gerenciar um tanto o período em que as *threads* devem executar, com os comando "yield", "sleep" ou, os objetos de sincronização.

Exemplo 03

Resultado do programa Exemplo 03.



Exemplo 04

(Exemplo de uso de mutex)

Exemplo 04

```
#ifndef _PRINCIPAL_H_
#define _PRINCIPAL_H_

#include "Escritora.h"

class Principal
{
private:
    Escritora escriNum;
    Escritora escriLetras;
public:
    Principal();
    ~Principal();

    void executar();
};
#endif
```

```
#ifndef _ESCRITORA_H_
#define _ESCRITORA_H_

#include "Thread.h"

class Escritora : public Thread
{
private:
    string _frase;

    void run();
public:
    Escritora(const string f = "");
    ~Escritora();

};
#endif
```

Exemplo 04

```
#include "Escritora.h"

Escritora::Escritora(const string f)
{
    _frase = f;
}

Escritora::~Escritora()
{   }

void Escritora::run()
{
    //---------------------------------------- REGIÃO CRÍTICA
    lock();
    for (string::iterator it = _frase.begin(); it != _frase.end(); it++)
    {
        cout << *it;
    }
    unlock();
    //----------------------------------------
}
```

Exemplo 04

```
#include "Principal.h"

Principal::Principal() :
escriNum("123456"),
escriLetras("ABCDEF")
{   }

Principal::~Principal()
{   }

void Principal::executar()
{
    escriNum.start();
    escriLetras.start();

    escriNum.join();
    escriLetras.join();

    cin.get();
}
```

```
#include "Principal.h"

int main()
{
    Principal p;
    p.executar();

    return 0;
}
```

Exemplo 04

Resultado do programa Exemplo 04.

Exemplo 05

(Exemplo de uso de semáforo)

Exemplo 05

```
#ifndef _PRINCIPAL_H_
#define _PRINCIPAL_H_

#include "Escritora.h"

class Principal
{
private:
    Escritora escriNum;
    Escritora escriLetras;
    sem_t meu_semaforo;
public:
    Principal();
    ~Principal();

    void executar();
};
#endif
```

```
#ifndef _ESCRITORA_H_
#define _ESCRITORA_H_

#include "Thread.h"
#include <semaphore.h>

class Escritora : public Thread
{
private:
    string _frase;
    sem_t* _semaforo;

    void run();
public:
    Escritora(const string f = "",
              sem_t* const sem = NULL);
    ~Escritora();

};
#endif
```

Exemplo 05

```
#include "Escritora.h"

Escritora::Escritora(const string f, sem_t* const sem)
{
    _frase = f;
    _semaforo = sem;
}

Escritora::~Escritora()
{   }

void Escritora::run()
{
    //---------------------------------------- REGIÃO CRÍTICA
    sem_wait( _semaforo );
    for (string::iterator it = _frase.begin(); it != _frase.end(); it++)
    {
        cout << *it;
    }
    sem_post( _semaforo );
    //----------------------------------------
}
```

Exemplo 05

```
#include "Principal.h"

Principal::Principal() :
escriNum("123456", &meu_semaforo),
escriLetras("ABCDEF", &meu_semaforo)
{
    sem_init(&meu_semaforo, 0, 1);
}

Principal::~Principal()
{
    sem_destroy( &meu_semaforo );
}

void Principal::executar()
{
    escriNum.start();
    escriLetras.start();

    escriNum.join();
    escriLetras.join();

    cin.get();
}
```

```
#include "Principal.h"

int main()
{
    Principal p;
    p.executar();

    return 0;
}
```

Exemplo 05

Resultado do programa Exemplo 05.

Exemplo 06

(Exemplo de uso de semáforo)

Exemplo 06

```
#ifndef _PRINCIPAL_H_
#define _PRINCIPAL_H_

#include "MinhaThread.h"
#include "Thread.h"
#include <windows.h>

class Principal : public Thread
{
private:
    int num_threads;
    int max_semaforo;

    MinhaThread* vt_thread[5];
    sem_t meu_semaforo;

    void run();
public:
    Principal();
    ~Principal();

    void executar();
};
#endif
```

```
#ifndef _MINHATHREAD_H_
#define _MINHATHREAD_H_

#include "Thread.h"
#include <windows.h>
#include <semaphore.h>

class MinhaThread : public Thread
{
private:
    int _id;
    int _ativa;
    sem_t* _semaforo;

    void run();
public:
    MinhaThread(const int id = -1,
                sem_t* const sem = NULL);
    ~MinhaThread();

    void setEstado(const int e);
    int getEstado() const;
};
#endif
```

Exemplo 06

```
#include "MinhaThread.h"

MinhaThread::MinhaThread(const int id, sem_t* const sem) {
    _id = id;
    _semaforo = sem;
    _ativa = 0;
}

MinhaThread::~MinhaThread(){   }

void MinhaThread::run() {
    //---------------------------------------- REGIÃO CRÍTICA
    sem_wait( _semaforo );
    for (int i = 0; i < 5; i++)
    {
        _ativa = 1;
        // marca a thread como ativa e aguarda até ser desmarcada pela thread principal;
        while ( 1 == _ativa )
        {
            Sleep(100);
        }
    }
    _ativa = -1;// informa a thread principal que já acabou o seu trabalho;
    sem_post( _semaforo );
    //----------------------------------------
    pthread_exit(0);
}
```

Exemplo 06

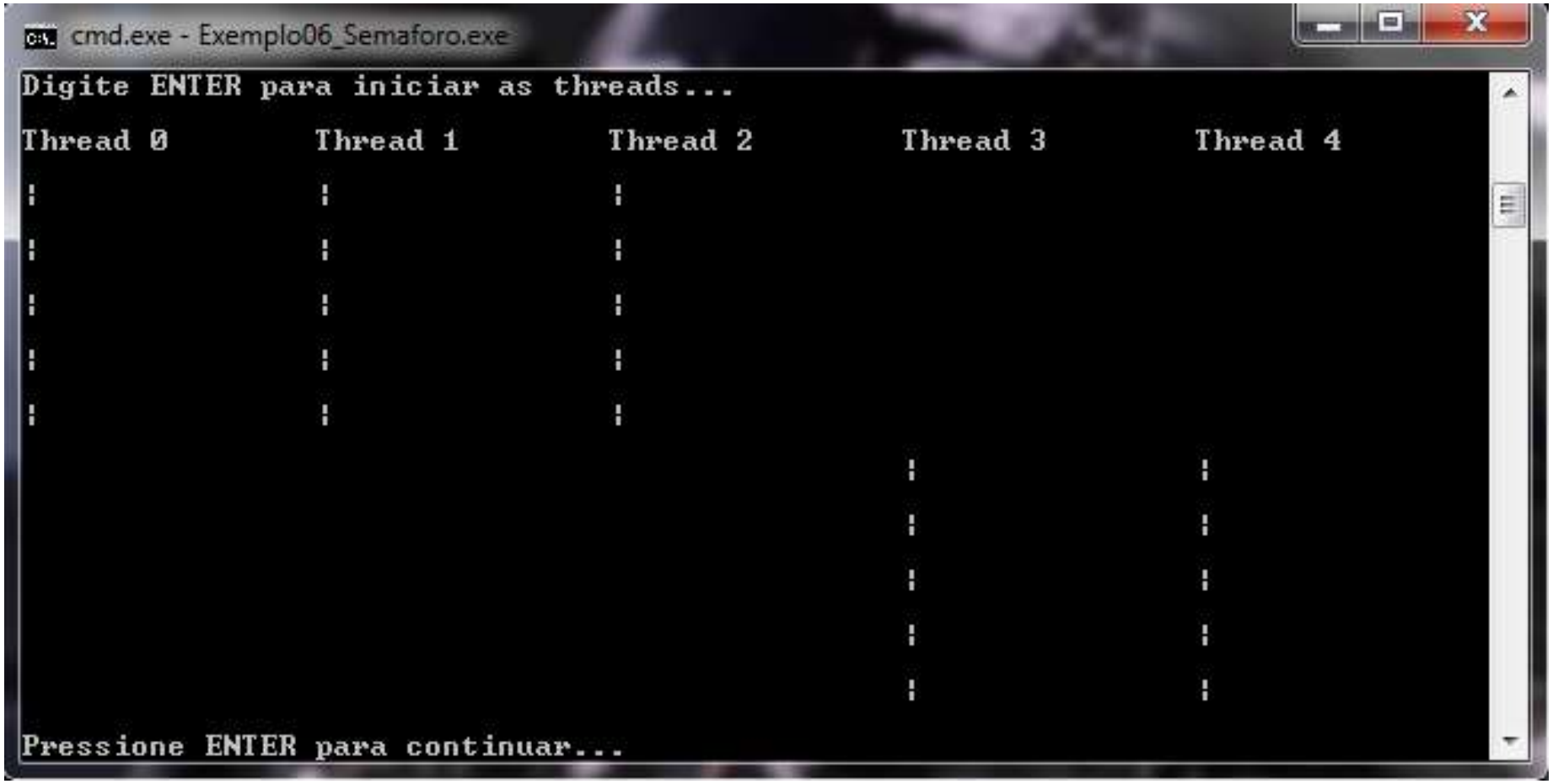
```
void Principal::run()
{
    for (int i = 0; i < num_threads; i++)
    {
        cout << "Thread " << i << "\t";
    }
    cout << endl;

    bool threads_executando;
    do{
        // verifica se há pelo menos 1 thread
        // executando;
        threads_executando = false;
        for (int i = 0; i < num_threads; i++)
        {
            if(vt_thread[i]->getEstado() != -1)
            {
                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(vt_thread[i]->getEstado() == 1)
              {
                  cout << "|";
                  vt_thread[i]->setEstado( 0 );
              }
              cout << "\t\t";
            }
            cout << endl;
            Sleep(500);
        }
    }while ( threads_executando );
}
```

Exemplo 06

```
void Principal::executar()
{
    cout << "Digite ENTER para iniciar as threads..." << endl;
    cin.get();

    // inicia as threads, passando como parâmetro
    // a thread e a função que cada uma deve executar;
    for (int i = 0; i < num_threads; i++)
    {
        vt_thread[ i ]->start();
    }

    start();    // inicia esta thread para que ela monitore as demais;
    join();

    cout << "Pressione ENTER para continuar..." << endl;
    cin.get();
}
```

Exemplo 06

Resultado do programa Exemplo 06.



Atividades – Exercícios

- Estudar os programas dos exemplos procurando entender as diferenças entre cada um.

- Desenvolva outros programas para testar a classe Thread criada.

Bibliografias relativas a *Threads*.

- TANENBAUM, Andrew Stuart: **Sistemas Operacionais Modernos.** Prentice Hall. 3ª Edição 2010.

- HUGHES C; HUGHES T.: **Professional Multicore Programming: Design and Implementation for C++ Developers**. Wrox (Paperback) 2008. ISBN-10: 0470289627