OO – Engenharia Eletrônica

-

Programação em C/C++

Slides 19 B: Introdução à *Multithreading*.

Introdução à *Multithreading*: execução concorrente de tarefas.

Exemplos usando a ‘biblioteca’ C Run-time library.

Prof. Jean Marcelo SIMÃO, DAELN/UTFPR
Aluno monitor: Vagner Vengue.

C Run-time library

- C Run-time libraries faz parte da Microsoft Run-time Library.

- "O Microsoft Run-time Library provê rotinas para programar o sistema operacional Microsoft Windows. Estas rotinas automatizam muitas tarefas comuns de programação que não são providas pelas linguagens C e C++" [Microsoft Visual C++ Help].

C Run-time library

- Esta biblioteca provê funções para a criação de *threads* e processos no Windows, ou seja, internamente ela utiliza as funções da Win32 API.

- A sua vantagem está na forma simples como as funções são apresentadas, fornecendo um certo nível de abstração da API do Windows.

- **Para operações mais avançadas** sobre *threads*, tais como criação de objetos de sincronização ou controle de prioridade, deve ser utilizado a **Win32 API**.

Funções básicas para *threads*

Threads – C Run-time library

_beginthread

```
uintptr_t  _beginthread(
                void( *start_address )( void * ),
                unsigned stack_size,
                void *arglist
);
```

- Cria e executa uma *thread*.
- Parâmetros:
  - *start_address*: função que a *thread* deve executar. (ponteiro de função)
  - *stack_size*: quantidade inicial de memória que a *thread* precisará. Pode ser passado 0(zero) para usar o valor padrão.
  - *arglist* : parâmetro passado para a função que executa a *thread*.

Threads – C Run-time library

_beginthreadex

```
uintptr_t  _beginthreadex(
                    void *security,
                    unsigned stack_size,
                    unsigned ( *start_address )( void * ),
                    void *arglist,
                    unsigned initflag,
                    unsigned *thrdaddr
);
```

- Cria e executa uma *thread*.
- Parâmetros:
  - *security*: atributos de segurança. Pode ser passado NULL para usar os valores padrões. (mesmos atributos da Win32 API).
  - *stack_size*: quantidade inicial de memória que a *thread* precisará. Pode ser passado 0(zero) para usar o valor padrão.
  - *start_address* : função que a *thread* deve executar (ponteiro de função).
  - *arglist* : parâmetro passado para a função que executa a *thread*.
  - *initflag*: flag que indicam o estado inicial da *thread*. Deve ser passado 0(zero) para indicar que a *thread* deve ser iniciada automaticamente.
  - *thrdaddr* : ponteiro para um inteiro identificador da *thread*. Pode ser NULL.

Threads – C Run-time library

_endthread e _endthreadex

- As *threads* são encerradas ao completar a "tarefa" ou com o término do processo, no entanto, também podem terminar a si mesmas, utilizando funções próprias para isto.

- As funções a seguir encerram as *threads* criadas por _beginthread e _beginthreadex respectivamente.

Threads – C Run-time library

_endthread e _endthreadex

```
void  _endthread( void );
```

- Encerra uma *thread* criada por _beginthread, liberando os recursos alocados para a *thread*.

```
void  _endthreadex(
            unsigned retval
);
```

- Encerra uma *thread* criada por _beginthreadex, porém, não libera os recursos alocados para a *thread*, exigindo a utilização da função CloseHandle da Win32 API.
- Parâmetros:
  - *retval* : código de saída da *thread*. Zero indica o término normal.

Exemplos

Exemplo 01

```
#include <iostream>
// biblioteca para CRuntime Library;
#include <process.h>
using namespace std;

bool thread_ligada;

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

```
int main()
{
    cout << "Digite ENTER para iniciar e "
    << "parar a thread..." << endl;
    // aguarda um ENTER do usuário;
    cin.get();
    thread_ligada = true;

    // inicia uma thread, passando como
    parâmetro a função que deve ser executada;
    _beginthread(escreveAlgo, 0, NULL);

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

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

    return 0;
}
```

Exemplo 01

Resultado do programa Exemplo 01.

Exemplo 02

```
#include <iostream>
// biblioteca para CRuntime Library;
#include <process.h>
using namespace std;

bool thread_ligada;

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

        if ( !thread_ligada ){
        // termina a thread
explicitamente;
            _endthread();
        }
    }
}
```

```
int main()
{
    cout << "Digite ENTER para iniciar e "
    << "parar a thread..." << endl;
    // aguarda um ENTER do usuário;
    cin.get();
    thread_ligada = true;

    // inicia uma thread, passando como
    parâmetro a função que deve ser executada;
    _beginthread(escreveAlgo, 0, NULL);

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

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

    return 0;
}
```

Exemplo 02

Resultado do programa Exemplo 02.

Exemplo 03

```
#include <iostream>
#include <process.h>        // biblioteca para usar funções de CRuntime Library;
#include <windows.h>
using namespace std;

bool thread_ligada;
HANDLE hnd_thread;          // identificador da thread;

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

Exemplo 03

```
int main()
{
    cout << "Digite ENTER para iniciar e parar a thread..." << endl;
    cin.get();              // aguarda um ENTER do usuário;
    thread_ligada = true;
    // inicia uma thread, passando como parâmetro
    // a função que deve ser executada;
    hnd_thread = (HANDLE) _beginthreadex(
                    NULL,       // atributo de segurança;
                    0,          // quantidade inicial de memória para a thread;
                    &escreveAlgo,// função a ser executada;
                    NULL,       // parâmetro da thread;
                    0,          // estado inicial da thread;
                    NULL        // identificador da thread;
                    );

    cin.get();              // aguarda um ENTER do usuário;
    thread_ligada = false;  // informa a thread que ela deve parar;

    if ( NULL != hnd_thread )
        WaitForSingleObject(&hnd_thread, INFINITE);
    CloseHandle( hnd_thread );
    return 0;
}
```

Exemplo 03

Resultado do programa Exemplo 03.

Exemplo 04

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

HANDLE meu_mutex;

// função passada para a thread 1;
unsigned WINAPI tarefa_1( LPVOID lpParam )
{
    //---------------------- REGIÃO CRÍTICA
    WaitForSingleObject(meu_mutex,
INFINITE);
    for (char* s = "123456"; *s != '\0';
s++)
    {
        cout << *s;
    }
    ReleaseMutex( meu_mutex );
    //-------------------------------------
}
```

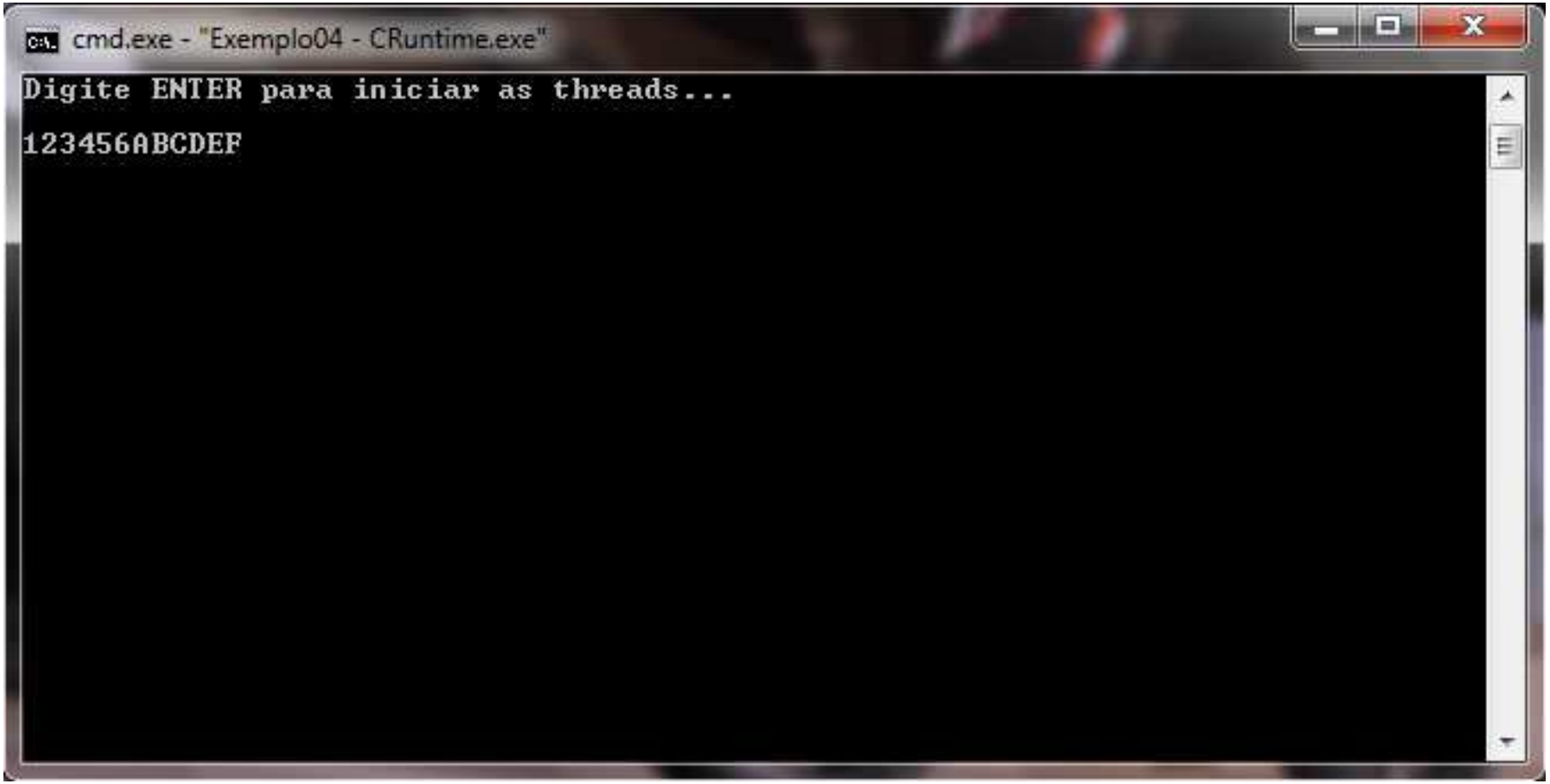
```
// função passada para a thread 2;
unsigned WINAPI tarefa_2( LPVOID lpParam )
{
 //---------------------- REGIÃO CRÍTICA
 WaitForSingleObject(meu_mutex, INFINITE);
 for (char* s = "ABCDEF"; *s != '\0'; s++)
 {
     cout << *s;
 }
 ReleaseMutex( meu_mutex );
 //-------------------------------------
}
```

Exemplo 04

```
int main()
{
    HANDLE hndThread_1, hndThread_2;

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

    meu_mutex = CreateMutex(NULL, FALSE, NULL);
    if ( NULL == meu_mutex ) {
        cout << "Erro ao criar o mutex." << endl;
        return 0;
    }
    // inicia as threads, passando como parâmetro
    // o "id" da thread e a função que cada uma deve executar;
    hndThread_1 = (HANDLE) _beginthreadex(NULL, 0, tarefa_1, NULL, 0, NULL);
    hndThread_2 = (HANDLE) _beginthreadex(NULL, 0, tarefa_2, NULL, 0, NULL);

    // faz com que a thread principal espere a 'thread_1' e a 'thread_2' acabarem;
    if ( NULL != hndThread_1 ){    WaitForSingleObject(hndThread_1, INFINITE);        }
    if ( NULL != hndThread_2 ){    WaitForSingleObject(hndThread_2, INFINITE);        }

    CloseHandle( hndThread_1 );
    CloseHandle( hndThread_2 );
    CloseHandle( meu_mutex );
    cin.get();              // aguarda um ENTER do usuário;
    return 0;
}
```

Exemplo 04

Resultado do programa Exemplo 04.



Bibliografias relativas a *Threads*.

- Microsoft Developer Network – C Run-time Functions for Thread Control. Disponível em: <http://msdn.microsoft.com/en-us/library/7t9ha0zh(v=VS.80).aspx>. Acesso em: 19 nov 2010.

- Microsoft Developer Network – Multithreading with C and Win32. Disponível em: <http://msdn.microsoft.com/en-us/library/y6h8hye8(v=VS.80).aspx>. Acesso em: 19 nov 2010.

- Microsoft Developer Network – C Run-time Reference. Disponível em: <http://msdn.microsoft.com/en-us/library/59ey50w6(v=VS.80).aspx>. Acesso em: 19 nov 2010.