UFMG Engenharia de Controle e Automação

Automação em Tempo Real

Módulo 2:
Processos e Threads

Things, which matter most, must never be at the mercy of things, which matter least - Goethe

Prof: Constantino Seixas Filho segunda-feira, 1 de maio de 2006

1

Primeiras Arquiteturas de micros em Automação

8085

TRS-80

Z-80

6502

Apple II

Teletype

Z-80

CP/M

Intel 8088

QNX

DOS

"I think there is a world market for maybe five computers" -- Thomas Watson, Chairman of IBM, 1943

2

Primeiras Arquiteturas de micros em Automação

- So we went to Atari and said, "Hey, we've got this amazing thing, even built with some of your parts, and what do you think about funding us? Or we'll give it to you. We just want to do it. Pay our salary, we'll come work for you." And they said, "No." So then we went to Hewlett-Packard, and they said, "Hey, we don't need you. You haven't got through college yet." -- Apple Computer founder Steve Jobs on attempts to get Atari and HP interested in his and Steve Wozniak's personal computer

3

Primeiras Arquiteturas de micros em Automação

- Primeiros PCs usavam sistema operacional CP/M
- Esta geração foi pouco usada em automação
- O IBM-PC oferecia um HW padrão e barato com sistema operacional MS-DOS desenvolvido pela Microsoft.
- Começa a dobradinha Wintel

Intel

8088

Multitarefas não preemptivo

DOS → Windows 3.1

"There is no reason for any individual to have a computer in their home"
-- Ken Olson, President & Founder, Digital Equipment Corp., 1977

4

Microsoft 1978

Fila de trás: Steve Wood, Bob Wallace and Jim Lane;

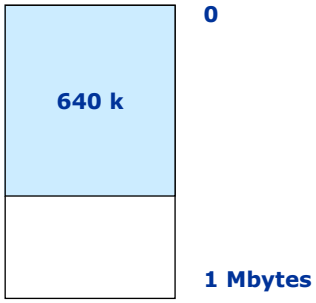
Segunda fila Bob O'Rear, Bob Greenberg, March McDonald and Gordon Letwin;

Fila da frente: Bill Gates, Andrea Lewis, Marla Wood and Paul Allen

5

Primeiras Arquiteturas de micros em Automação

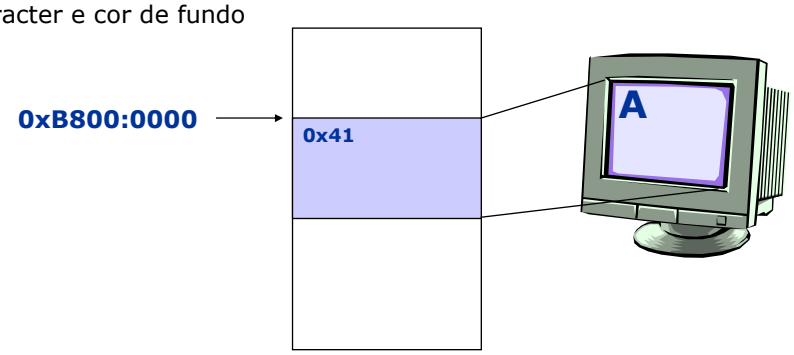
- Espaço de endereçamento inicial de 1M bytes com 640k bytes dedicados para o usuário se mostra rapidamente insuficiente. Esta limitação irá acompanhar o Windows 3.1 mesmo em arquiteturas superiores (286/386).
- DOS era monousuário e monotarefa.
- QNX já utiliza melhor os recursos do PC e das arquiteturas seguintes. Era multiusário/multitarefas



6

Endereçamento em modo real

- Acesso direto à memória através do endereço formado por duas partes: Segmento:Offset
- Por exemplo: O endereço 0xB800:0000 correspondia ao endereço da memória de vídeo alfanumérico (CGA). Existiam 25 * 80 caracteres cada um ocupado um byte para o código ASCII e um byte para cor do caracter e cor de fundo



Qual o tamanho ocupado pela memória de vídeo ? _____

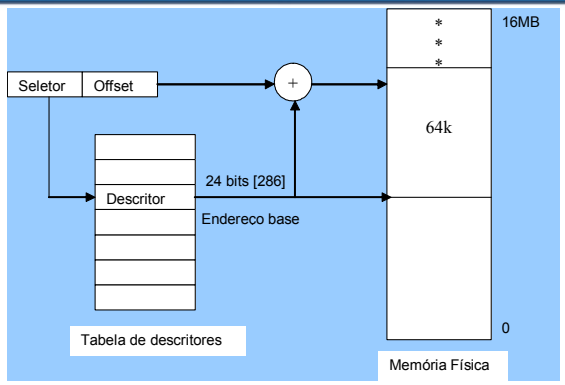
7

Endereçamento em modo real

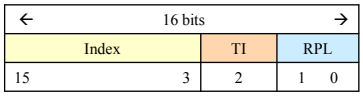
- Problemas do endereçamento em modo real:
  - Uma task/processo de um usuário podia escrever na área de código ou dados de um outro programa, inclusive em áreas reservadas do sistema operacional
  - Um ponteiro desgovernado poderia causar danos muito grandes
  - Estes programas eram difíceis de serem depurados exigindo em geral a participação de mais de um programador em terminais diferentes
  - Bugs não detectados iam despertar em horas inconvenientes Você conhece as leis de Murphy ?

8

Transcrição de endereços em modo protegido na arquitetura 286/386



Seletor na arquitetura 286

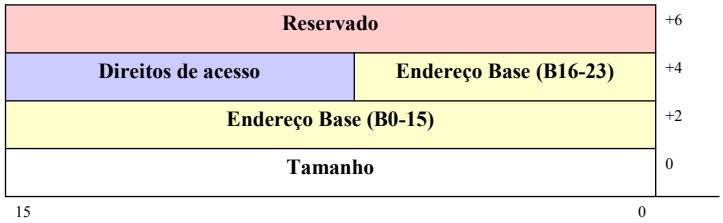


TI: GDT/LDT
RPL: Requestor Privilege Level
Index: Offset para tabela (8k)

Endereço:
Seletor:Deslocamento

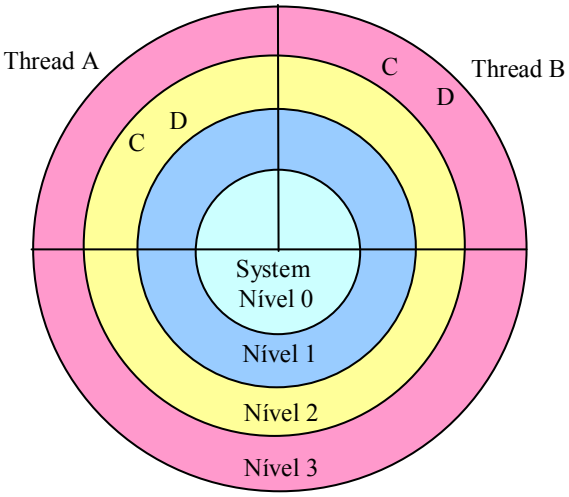
9

Descritor de segmento da arquitetura 286



10

Níveis de privilégios na arquitetura Intel 80x86

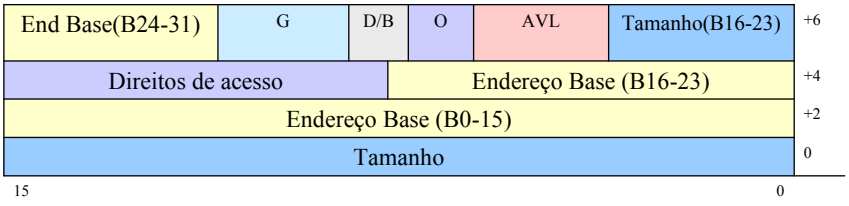


C = Código

D = Dado

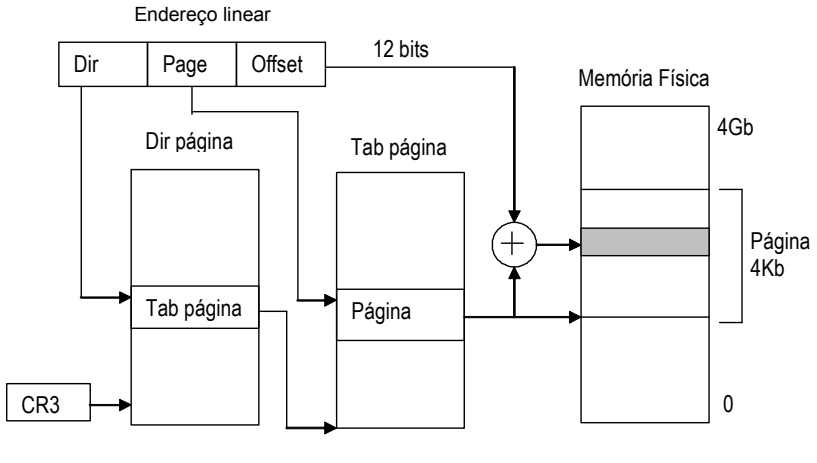
11

Descritor de segmento na arquitetura 386



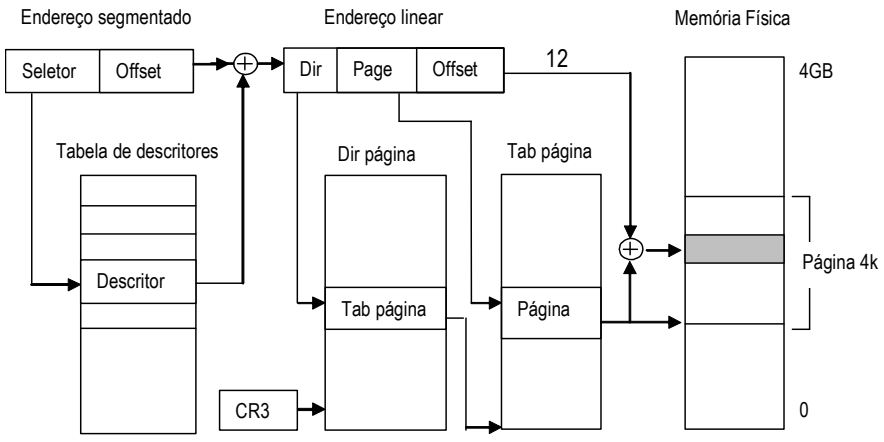
12

Tradução de endereço com paginação de memória no 80386



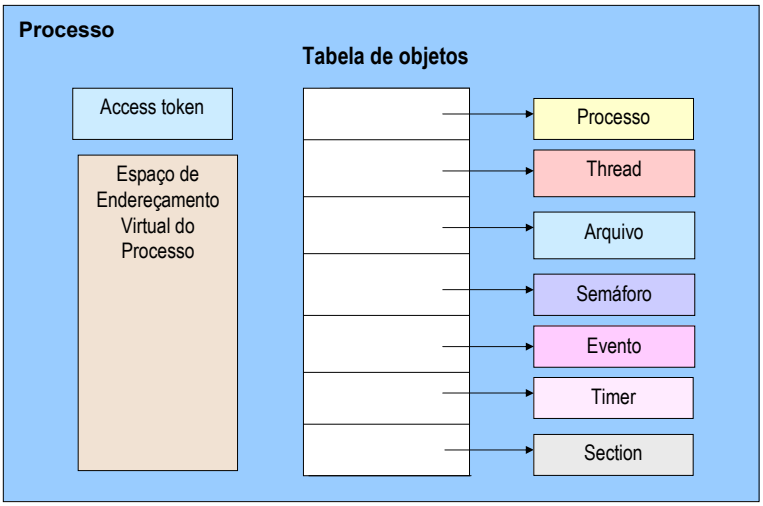
13

Tradução dupla de endereço no 80386



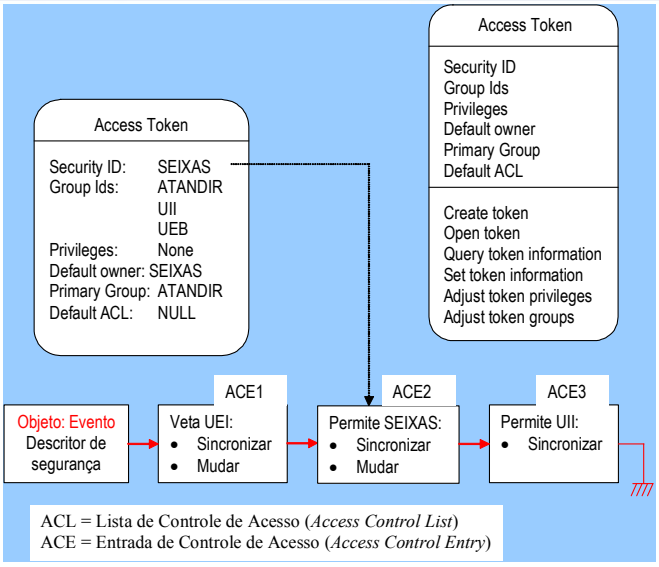
14

Recursos de um Processo



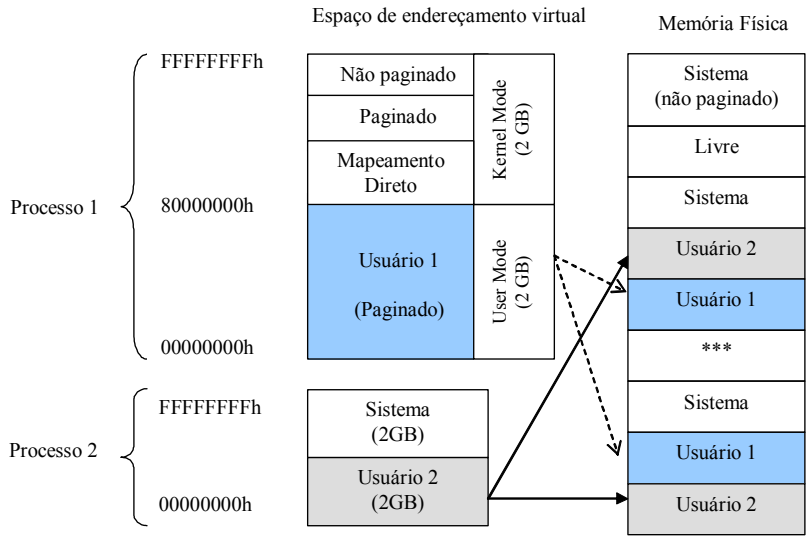
15

Proteção de objetos no WNT



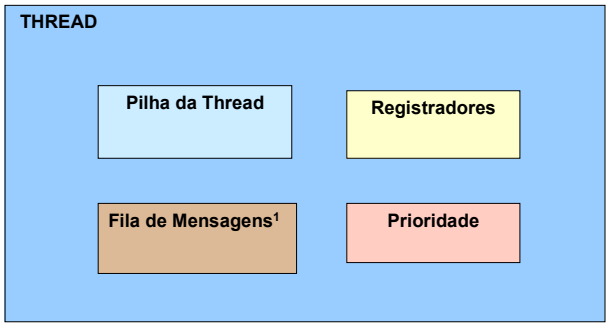
16

Memória virtual no WNT



17

Threads



[1] GUI thread: thread responsável pela interface com o usuário
Working Threads não têm fila de mensagens

18

Programas x Processos x Threads

Preparo de um Jantar	Programação Concorrente
Receita	Programa
Cozinha	Processo
Preparos: ▪ Arroz ▪ Steak Diane ▪ Salada Niçoise ▪ Crème Brulée	Threads

19

Estratégias de Escalonamento

Round Robin	Time slicing apenas
FIFO	Tarefas escalonadas pela ordem de chegada. Sem preeempção
Preemptivo baseado em prioridades estabelecidas pelo programador	Thread de maior prioridade preempta as demais
Menor tempo para completar	Escalona tarefa que necessita de menos tempo para completar sua execução
Menor data limite	Escalona tarefa que apresenta menor deadline
Menor Job primeiro	Escalonador escolhe a thread de menor duração estimada

20

Prioridades no WNT

Prioridade base = Classe de prioridade + Nível de prioridade

Classe	Prioridade
REALTIME_PRIORITY_CLASS	24
HIGH_PRIORITY_CLASS	13
NORMAL_PRIORITY_CLASS	7 OU 9
IDLE_PRIORITY_CLASS	4

Classes de prioridade de um processo

Nível de Prioridade	Ajuste em relação à prioridade do processo
THREAD_PRIORITY_HIGHEST	+2
THREAD_PRIORITY_ABOVE_NORMAL	+1
THREAD_PRIORITY_NORMAL	0
THREAD_PRIORITY_BELOW_NORMAL	-1
THREAD_PRIORITY_LOWEST	-2
THREAD_PRIORITY_IDLE	Força para 1
THREAD_PRIORITY_TIME_CRITICAL	Força para 15

Níveis de prioridade de uma thread

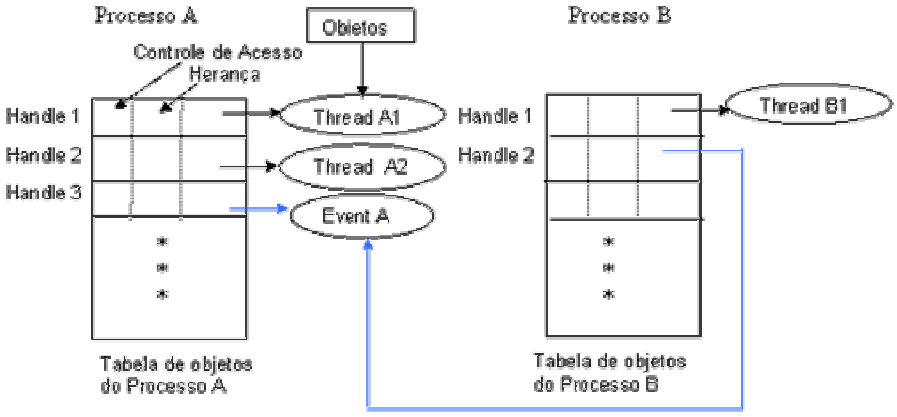
21

Prioridades no WNT

PRI	CLASSES DE PRIORIDADES				
	REAL TIME	HIGH	NORMAL FOREGROUND	NORMAL BACKGROUND	IDLE
31	TIME_CRITICAL				
30					
29					
28					
27					
26	HIGHEST				
25	ABOVE_NORMAL				
24	**NORMAL**				
23	BELOW_NORMAL	*Cursor*			
22	LOWEST	*Sistema de arquivos, Drivers*			
21					
20					
19					
18					
17					
16	IDLE				
15		TIME_CRITICAL	TIME_CRITICAL	TIME_CRITICAL	TIME_CRITICAL
14		ABOVE_NORMAL			
13		**NORMAL**			
12		BELOW_NORMAL			
11		LOWEST	HIGHEST		
10			ABOVE_NORMAL		
09			**NORMAL**	HIGHEST	
08			BELOW_NORMAL	ABOVE_NORMAL	
07			LOWEST	**NORMAL**	
06				BELOW_NORMAL	HIGHEST
05				LOWEST	ABOVE_NORMAL
04					**NORMAL**
03					BELOW_NORMAL
02					LOWEST
01		IDLE	IDLE	IDLE	IDLE
00	RESERVADO				

22

Handles e compartilhamento de objetos entre processos



23

Create Thread

```
HANDLE CreateThread (
LPSECURITY_ATTRIBUTES lpThread Attributes,   // Apontador para os atributos de segurança
DWORD dwStackSize                            // Tamanho inicial da pilha em bytes
LPTHREAD_START_ROUTINE lpStartAddress,       // Apontador para o ponto de entrada da thread
LPVOID lpParameter,                          // Argumento para a nova thread
DWORD dwCreationFlags,                       // Flags de criação
LPDWORD lpThreadId                           // Retorna o ID da nova thread criada
);
```

Retorno da função:

Status	Interpretação
Handle para a thread criada	Sucesso
NULL	Falha

24

Create Thread

Comentários sobre os parâmetros:

lpThread Attributes	Apontador para atributos de segurança (estrutura SECURITY_ATTRIBUTES) a serem aplicados à nova thread. NULL utiliza descritor default, mas handle não será herdável. Win95/98 ignora este parâmetro.
dwStackSize	Especifica o tamanho da pilha da nova thread em bytes. Default: 0 – especifica tamanho dado pela opção /STACK do linker (1M bytes).
lpStartAddress	Nome da função que representa a thread Deve ser declarada como: DWORD ThreadFunc(LPVOID lpParameter)
lpParameter	Parâmetro de 32 bits passado à thread. Pode ser um ponteiro ou um valor qualquer.
dwCreationFlags	0: thread começa imediatamente CREATE_SUSPENDED: thread fica bloqueada até que *ResumeThread()* seja chamada. *SuspendThread()* suspende uma thread a qualquer momento.
lpThreadId	Apontador para variável que receberá o Id da thread.

25

Outras Funções

- DWORD GetCurrentThreadId(VOID);

- HANDLE GetCurrentThread(VOID);

- BOOL CloseHandle (
  HANDLE hObject);

```
// Handle para objeto a ser desconectado
```

- VOID ExitThread(
  DWORD dwExitCode);

```
// Código de saída retornado pela thread
```

- BOOL TerminateThread(
  HANDLE hThread
  DWORD dwExitCode);

```
// Handle para a thread a ser terminada
// Código de saída retornado pela thread
```

26

Outras Funções

```
BOOL GetExitCodeThread (
        HANDLE hThread                  // Handle para a thread da qual se deseja saber o
                                           código de saída
        LPDWORD lpExitCode);            // Apontador para o código a ser retornado
                                           STILL_ACTIVE = thread não terminada
```

Retorno da função:

Status	Interpretação
TRUE	Função retornou com SUCESSO, mas pode estar em execução ainda.
FALSE	houve algum ERRO

27

Terminação de Threads

Uma thread termina quando um dos seguintes eventos ocorre:

- **A função da thread retorna**
- **A função *ExitThread()* é chamada**
- **A função *TerminateThread()* é chamada**
- **Alguma thread do processo chama a função *ExitProcess()***
- **Alguma thread do processo chama a função *TerminateProcess()***

- ***Qual a diferença entre ExitThread e TerminateThread ?***

28

Criando Threads

```
#define WIN32_LEAN_AND_MEAN
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>                      // _getch

DWORD WINAPI TestFunc(LPVOID);          // declaração da função

int main()
{
    HANDLE hThreads[3];
    DWORD dwThreadId;
    DWORD dwExitCode = 0;
    BOOL bFlag[3] = {TRUE, TRUE, TRUE};
    int i;

    // Define o título da janela
    SetConsoleTitle("Programa 2.1 - Criando Threads");
    for (i=0; i<3; ++i) {               // cria 3 threads
           hThreads[i] = CreateThread(NULL, 0, TestFunc, (LPVOID)i, 0, &dwThreadId);
           if (hThreads[i]) printf("Thread %d criada com Id= %0x \n", i, dwThreadId);
    } // for
```

29

Criando Threads

```
   i = 0; // espera término das threads
   while (bFlag[0] | bFlag[1] | bFlag[2]) {
       if (bFlag[i]){
          GetExitCodeThread(hThreads[i], &dwExitCode);
          if (dwExitCode != STILL_ACTIVE) {
          printf("thread %d terminou com codigo de saida %d\n", i, dwExitCode);
          bFlag[i] = FALSE;
          CloseHandle(hThreads[i]); // apaga ref. ao objeto
          };
       };
       i = (i + 1) % 3;
   }  // while
   printf("\nAcione uma tecla para terminar");
   _getch(); // Espere aqui se não estiver no ambiente MDS
   return EXIT_SUCCESS;
}   // main
```

30

Criando Threads

```
DWORD WINAPI TestFunc(LPVOID index)
{  int i;

   for(i=0; i<50; ++i) {
          printf("%d ", index);
          // Sleep(10);       // delay de 10 ms
          }
   printf("\n");
   ExitThread((DWORD) index);
   return(0);
} // TestFunc
```

31

Bibliotecas runtime do WNT

LIBC.LIB	Biblioteca default do sistema. Versão de ligação estática para aplicações de thread única.
LIBCD.LIB	Versão de depuração da biblioteca anterior.
LIBCMT.LIB	Versão de ligação estática para uso em aplicações multithreaded
LIBCMTD.LIB	Versão de ligação estática de depuração para aplicações multithreaded.
MSVCRT.LIB	Versão de ligação dinâmica da versão definitiva da biblioteca MSVCRT.DLL. Suporta aplicações de thread única e multithreaded.
MSVCRTD.LIB	Versão de ligação dinâmica da versão de depuração da biblioteca MSVCRT.DLL. Suporta aplicações de thread única e multithreaded.

Configuração do Microsoft Visual C++:

1. Selecione o tab C/C++ e escolha a categoria ***Code Generation***
2. Abra as opções sob a legenda ***Use run-time library***
3. Selecione a biblioteca a ser utilizada: ***Multithreaded*** ou ***Debug Multithreaded***
4. Valide com a tecla OK e está pronto para compilar.

32

_beginthreadex

```
unsigned long _beginthreadex (
void *security,                                  // Apontador para os atributos de segurança
unsigned stack_size,                             // Tamanho inicial da pilha em bytes
unsigned (stdall *start_address)(void *),        // Apontador para o ponto de entrada da thread
void *arglist,                                   // Argumento para a nova thread
unsigned initflag,                               // Flags de criação
unsigned *threaddr                               // Retorna o ID da nova thread criada
);
```

```
void _endthreadex(
unsigned retval);                                // Código de saída retornado pela thread
```

33

Uso de _beginthreadex

```
typedef unsigned (WINAPI *CAST_FUNCTION)(LPVOID);
typedef unsigned *CAST_LPDWORD;


hThread = (HANDLE) _beginthreadex(
                  NULL,
                  0,
                  (CAST_FUNCTION)TestFunc,
                  (LPVOID)i,
                  0,
                  (CAST_LPDWORD)&dwThreadId);
```

34

Thread como função membro de um objeto

- **Threads podem ser utilizadas como funções membro de um objeto**
- **Este objetos são úteis para modelar os *sprites* de um game, os personagens/equipamentos de uma simulação, etc**
- **Entretanto o uso de uma thread em C++ requer certos cuidados e não é tão simples como parece**

35

Primeira tentativa

```
class ThreadObject
{
public:
    void StartThread();
    virtual DWORD WINAPI ThreadFunc(LPVOID param);
private:
    HANDLE      m_hThread;
    DWORD       m_ThreadId;
};
```

A função ThreadFunc é virtual e assim um objeto real pode ser obtido derivando-se uma classe de ThreadObject e definindo-se a função ThreadFunc como desejado.

36

Primeira tentativa

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

typedef unsigned (WINAPI *PBEGINTHREADEX_THREADFUNC)(
   LPVOID lpThreadParameter
 );
typedef unsigned *PBEGINTHREADEX_THREADID;

class ThreadObject
{
public:
   ThreadObject();                  // Construtor
   void StartThread();
   virtual DWORD WINAPI ThreadFunc(LPVOID param);
   void WaitForExit();
private:
   HANDLE  m_hThread;               // Handle para thread criada
   DWORD m_ThreadId;                // Identificador da thread
};

ThreadObject::ThreadObject()       // Inicializa membros privados da classe
{  m_hThread = NULL;
   m_ThreadId = 0;
}
```


Exemplo - Sistema supervisório

```
void ThreadObject::StartThread()  // Cria Thread
{
   m_hThread = (HANDLE)_beginthreadex(NULL,
                    0,  (PBEGINTHREADEX_THREADFUNC)ThreadFunc,  0,   0,
                    (PBEGINTHREADEX_THREADID)&m_ThreadId );
   if (m_hThread) printf("Thread launched\n");
}

void ThreadObject::WaitForExit()  // Espera fim da thread
{
   WaitForSingleObject(m_hThread, INFINITE);
   CloseHandle(m_hThread);
}

DWORD WINAPI ThreadObject::ThreadFunc(LPVOID param)
{  // Faça algo útil
   return 0;
}

void main()
{
   ThreadObject obj;
   obj.StartThread();   // Cria thread
   obj.WaitForExit();   // Espera fim da thread
}
```

Ao compilar ....

ERRO:

```
:\Ufmg2\BadClass\BadClass.cpp(51) : error C2440: 'type cast' : cannot convert
    from 'overloaded function type' to 'unsigned int (__stdcall *)(void *)'
None of the functions with this name in scope match the target type
```

Onde está o erro ?

- Toda função membro não estática tem um parâmetro oculto.
- Este parâmetro é usado toda vez que a função acessa um membro de dado ou quando o programador utiliza o ponteiro *this* dentro da função membro.
- A função ThreadObject::ThreadFunc(LPVOID param) tem efetivamente dois parâmetros: o ponteiro *this* e *param*.
- Quando o WNT cria uma nova thread ele também cria um novo stack para a nova thread. e depois recria a chamada da função no novo stack, isto é, ele deve transferir os parâmetros da nova thread para o seu stack particular. *Param* é transferido corretamente, mas *this* não é copiado. O WNT não sabe que desta vez você está criando uma função thread, para uso num ambiente orientado a objeto, e que *this* deve ser empilhado como o primeiro parâmetro

39

Funções membro não estáticas

Observações:

- Em uma função membro não estática, a palavra chave *this* é um ponteiro para o objeto através do qual a função é chamada
- Dada uma classe A, uma chamada à função membro A::f, por exemplo,

```
A *pa;
pa-> f(2);

poderia ser transformada em
f(pa, 2);                  // código gerado
```

- Assim, dentro de uma função membro, a palavra this aponta para o objeto sobre o qual a função membro foi invocada.

Bjarne Stroustrup, C++ Manual de referência comentado

40

Convenção de chamada

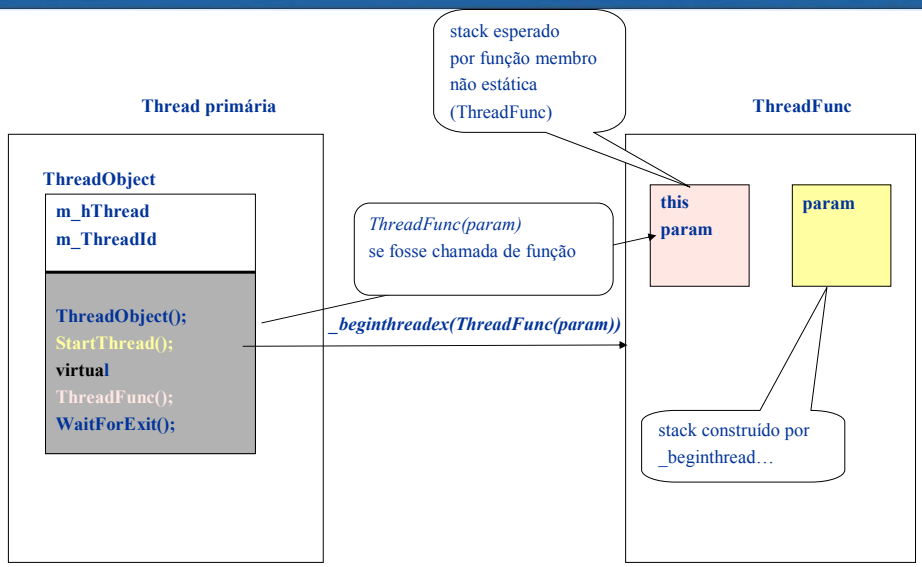
A convenção de chamada WINAPI (#define WINAPI __stdcall) implica:

- Os argumentos são passados da direita para a esquerda (convenção PASCAL).
- A função chamada retira os seus próprios argumentos do stack.
- O nome decorado é formado por nome da função + @ + número de bytes dos parâmetros.
  Exemplo:
  int func(int a, double b) é decorado como: _func@12
- Na convenção de chamada _cdecl, o stack deve ser limpo por quem chama a função

Qual convenção de chamada de função resulta em maior código para o programa principal ?

41

Visão do Stack Frame



42

Solução para o problema:

Para iniciar um objeto thread de uma função membro:

- Use uma função membro estática, que chamará a função membro desejada.
- Use uma função no estilo C, que chamará a função membro desejada

Objetivo:

- Uma função auxiliar irá construir um *stack frame* correto para ser usado pela thread.

43

Função membro estática

- Uma função membro estática não possui um ponteiro ***this***, assim ela só pode acessar membros não estáticos de sua classe pelo uso de . ou ->
- A finalidade do uso de membros *static* é reduzir a necessidade de variáveis globais, proporcionando alternativas que são locais a uma classe
- Uma função membro ou variável *static* atua como global para membros de sua classe, sem afetar o restante do programa. Seu nome não entra em choque com os nomes de variáveis ou funções globais nem com os nomes de outras classes
- Bjarne Stroustrup, C++ Manual de referência comentado

44

Estratégia 1: Função membro estática

```
// Só os .h mais básicos serão incluídos.
#define WIN32_LEAN_AND_MEAN
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
#include <process.h>

typedef unsigned (WINAPI *PBEGINTHREADEX_THREADFUNC)(
   LPVOID lpThreadParameter
   );

typedef unsigned *PBEGINTHREADEX_THREADID;
//
// This ThreadObject is created by a thread that wants to start another  thread. All public member functions
// except ThreadFunc() are called by that original thread. The virtual function ThreadMemberFunc() is
// the start of the new thread.
//
class ThreadObject
{
public:
   ThreadObject();
   void StartThread();
   void WaitForExit();
   static DWORD WINAPI ThreadFunc(LPVOID param);
protected:
   virtual DWORD ThreadMemberFunc();
   HANDLE  m_hThread;
   DWORD   m_ThreadId;
};
```

45

Estratégia 1: Função membro estática

```
ThreadObject::ThreadObject()       // Aqui nada muda
{
   m_hThread = NULL;
   m_ThreadId = 0;
}

void ThreadObject::StartThread()
{
   m_hThread = (HANDLE)_beginthreadex(NULL,
      0,
      (PBEGINTHREADEX_THREADFUNC) ThreadObject::ThreadFunc,
      (LPVOID)this, // passa pointer para objeto como parâmetro
      0,
      (PBEGINTHREADEX_THREADID) &m_ThreadId );

    if (m_hThread) {
      printf("Thread launched\n");
   }
}

void ThreadObject::WaitForExit()                  // Aqui nada muda
{
   WaitForSingleObject(m_hThread, INFINITE);
   CloseHandle(m_hThread);
}
```

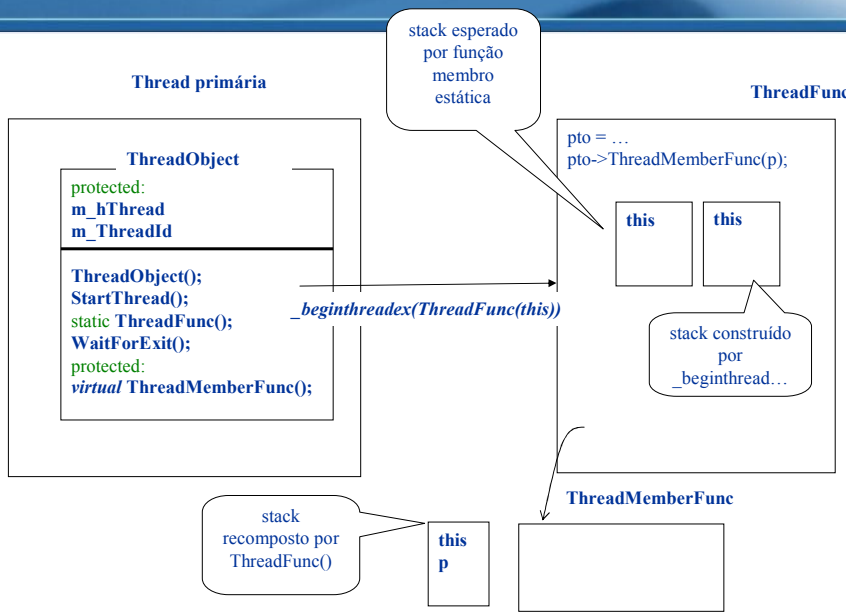
46

Estratégia 1: Função membro estática

```
//
// This is a static member function.  Unlike C static functions, you only
// place the static declaration on the function declaration in the class, not on // its implementation.
// Static member functions have no "this" pointer, but do have access rights.
//
DWORD WINAPI ThreadObject::ThreadFunc(LPVOID param)
{
   // Use the param as the address of the object
   ThreadObject* pto = (ThreadObject*)param;
   // Call the member function. Since we have a proper object pointer,
   // even virtual functions will be called properly.
   return pto->ThreadMemberFunc();
}

// This above function ThreadObject::ThreadFunc()  calls this function after the thread starts up.
DWORD ThreadObject::ThreadMemberFunc()
// Função que desempenhará as funções da thread
{
   // Do something useful ...
   return 0;
}

void main()
{
   ThreadObject obj;
   obj.StartThread();
   obj.WaitForExit();
}
```

47

Stack Frame



48

Estratégia 2: Função membro no estilo C

```
// Só os .h mais básicos serão incluídos.
#define WIN32_LEAN_AND_MEAN
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
#include <process.h>

typedef unsigned (WINAPI *PBEGINTHREADEX_THREADFUNC)(
   LPVOID lpThreadParameter
   );

typedef unsigned *PBEGINTHREADEX_THREADID;

// Define the prototype for the function used to start the thread.
// Função no estilo C que chamrá a função membro

DWORD WINAPI ThreadFunc(LPVOID param);
class ThreadObject
{
public:
   ThreadObject();
   void StartThread();
   void WaitForExit();
// A função membro deve ser declarada como pública ou a função no estilo C não terá direito de acessá-la.
virtual DWORD ThreadMemberFunc();
protected:
   HANDLE  m_hThread;
   DWORD   m_ThreadId;
};
```

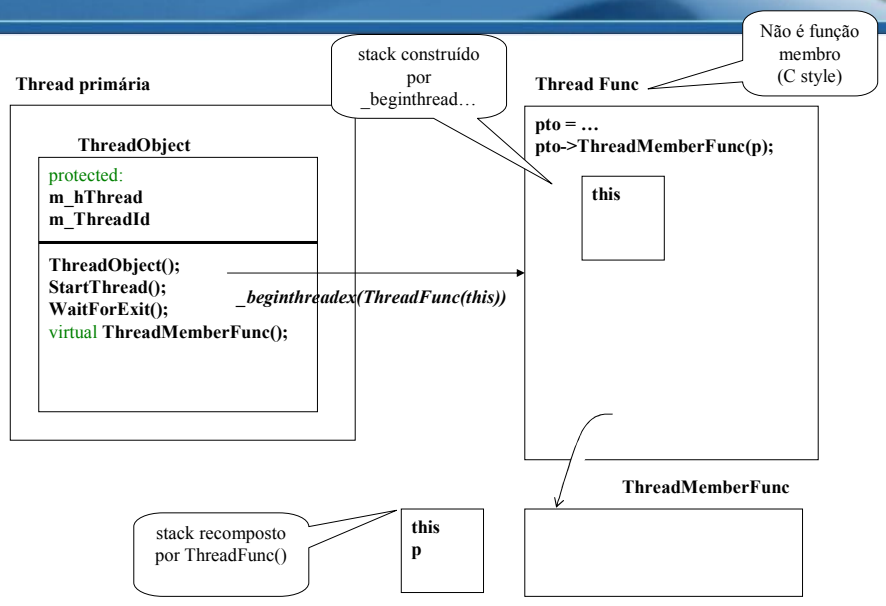

Estratégia 2: Função membro no estilo C

```
ThreadObject::ThreadObject()
{
   m_hThread = NULL;
   m_ThreadId = 0;
}

void ThreadObject::StartThread()
{
   m_hThread = (HANDLE)_beginthreadex(NULL,
      0,
      (PBEGINTHREADEX_THREADFUNC) ThreadFunc,
      (LPVOID)this,
      0,
      (PBEGINTHREADEX_THREADID) &m_ThreadId );
   if (m_hThread) {
      printf("Thread launched\n");
   }
}

void ThreadObject::WaitForExit()
{
   WaitForSingleObject(m_hThread, INFINITE);
   CloseHandle(m_hThread);
}
```

Estratégia 2:
Função membro no estilo C

```
// Esta função é chamada quando a thread inicia
DWORD WINAPI ThreadFunc(LPVOID param)
{
     // Use the param as the address of the object
     ThreadObject* pto = (ThreadObject*)param;
     // Call the member function. Since we have a proper object pointer, even
     // virtual functions will be called properly.
     return pto->ThreadMemberFunc();
}

// A função ThreadFunc() chama esta função assim que a thread inicia
DWORD ThreadObject::ThreadMemberFunc()
{
   // Do something useful ...
   return 0;
}

void main()
{
   ThreadObject obj;
   obj.StartThread();
   obj.WaitForExit();
}
```


Estratégia 2:
Função membro no estilo C



Criação de Processos

```
BOOL CreateProcess (
LPCTSTR lpApplicationName,
LPTSTR lpCommandLine,
LPSECURITY_ATTRIBUTES lpProcess,
LPSECURITY_ATTRIBUTES lpThread,
BOOL bInheritHandles,
DWORD dwCreationFlags,
LPVOID lpEnvironment,
LPTSTR lpCurrentDirectory,
LPSTARUPINFO lpStartupInfo,
LPPROCESS_INFORMATION lpProcessInformation
);
```

Retorno da função:

Status	Interpretação
TRUE	Sucesso
FALSE	Falha

53

Criação de Processos

lpApplicationName	Nome do programa executável. NULL se aplicação for 16 bits
lpCommandLine	Se lpApplicationName for NULL, especifica linha de comando da chamada da aplicação. Caso contrário, possui lista de argumentos para novo processo no formato argv.
lpProcess	Apontador para descritor de segurança para o handle do novo processo (contém a ACL do novo handle). Se for NULL, a estrutura default será utilizada e o handle para o processo será herdável
lpThread,	O mesmo do parâmetro anterior aplicado à thread.
bInheritHandles	Define se os handles de objetos criados por este processo, com atributo herdável, serão herdados pelo novo processo. Valores: TRUE/FALSE.
dwCreationFlags	Flags definindo status e classe de prioridade do processo criado. É uma combinação de: CREATE_SUSPENDED: thread primária ficará bloqueada até que função *ResumeThread* seja chamada. CREATE_NEW_CONSOLE: novo processo criará uma nova console ou CREATE_DETACHED_PROCESS: o novo processo não terá console (válido para processos compilados para rodar em consoles). CREATE_SEPARATE_WOW_VDM: processos de 16 bis serão criados em máquinas virtuais distintas.
lpEnvironment	Default: NULL. O processo filho herda exatamente o mesmo ambiente do processo pai (recebe uma cópia). Use *GetEnvironmentStrings()* para obter um apontador para o bloco de parâmetros de ambiente e passe este apontador como parâmetro. O efeito é o mesmo de se passar NULL.
lpCurrentDirectory	NULL: diretório corrente do filho é o mesmo diretório do pai. Exemplo: "c:\\Ufmg\\Automacao"
lpStartupInfo	Apontador para a estrutura STARTUPINFO.

lpProcessInformation	Apontador para estrutura _PROCESS_INFORMATION contendo info sobre novo processo e sua thread primária.
	typedef struct _PROCESS_INFORMATION {
	HANDLE hProcess // handle para o novo processo
	HANDLE hThread // handle para thread primária
	DWORD dwProcessId // Identificação do processo
	DWORD dwThreadId} // Identificação da thread
	Os handles para o processo e para a thread deverão ser fechados utilizando *CloseHandle()*.

54

Estrutura STARTUPINFO

```
typedef struct _STTARTUPINFO {
DWORD     cb;               // Tamanho da estrutura em bytes
LPTSTR    lpReserved;       // Sempre igual a NULL
LPTSTR    lpDesktop;        // NULL para implementar herança de desktop
LPTSTR    lpTitle;          // Título do novo console
DWORD     dwX;              // Posição X da nova janela (console)
DWORD     dwY;              // Posição Y da nova janela (console)
DWORD     dwXSize;          // Largura do novo console
DWORD     dwYSize;          // Altura do novo console
DWORD     dwXCountChars;    // Tamanho do buffer em colunas
DWORD     dwYCountChars;    // Tamanho do buffer em linhas
DWORD     dwFillAttribute;  // Cor de background
DWORD     dwFlags;          // Especifica membros utilizados desta estrutura
WORD      wShowWindow;      // Utilizado em aplicações GUI
WORD      cbReserved2;      // Sempre igual a 0
LPBYTE    lpReserved2;      // Sempre igual a NULL
HANDLE    hStdInput;        // Handle para standard input
HANDLE    hStdOutput;       // Handle para standard output
HANDLE    hStdError         // Handle para standard error
} STARTUPINFO, *LPSTARTUPINFO;
```

55

Terminação de Processos

Um processo termina quando um dos seguintes eventos ocorre:

- **A thread primária do processo retorna naturalmente (sem evocar *ExitThread()*)**
- **A função *ExitProcess()* é chamada por qualquer thread do processo**
- **A última thread do processo termina**
- **A função *TerminateProcess()* é chamada**

```
VOID ExitProcess (
UINT uExitCode);                    // Código de saída retornado pelo processo

BOOL TerminateProcess (
HANDLE hProcess                     // Handle para o processo a ser terminado
UINT uExitCode);                    // Código de saída retornado pelo processo e todas suas threads
```

56

Terminação de Processos

```
BOOL GetExitCodeProcess
HANDLE hProcess                 // Handle para o processo do qual se deseja saber o código de saída
LPDWORD lpExitCode              // Apontador para o código a ser retornado
                                   STILL_ACTIVE = processo não terminado
);
```

- DWORD GetCurrentProcessId(VOID);
- HANDLE GetCurrentProcess(VOID);

57

OpenProcess

```
HANDLE OpenProcess
DWORD dwDesiredAccess           // Tipo de acesso
BOOL bInhritHandle,             // TRUE: o handle será herdável
DWORD dwProcessId,              // Id do processo alvo
);
```

Tipo de Acesso	Significado
PROCESS_ALL_ACCESS	especifica todos flags descritos a seguir
PROCESS_CREATE_THREAD	permite utilizar o handle para criar uma thread no processo representado por ele, através da função *CreateRemoteThread()*
PROCESS_DUP_HANDLE	o handle pode ser utilizado na função *DuplicateHandle()*
PROCESS_QUERY_INFORMATION	o handle poderá ser utilizado para obter a prioridade e código de saída do processo
PROCESS_SET_INFORMATION	o handle poderá ser utilizado para ajustar a classe de prioridade do processo
PROCESS_TERMINATE	o handle poderá ser utilizado na função *TerminateProcess()*
SYNCHRONIZE	permite a utilização do handle em qualquer função Wait

58

Alterando a prioridade

```
DWORD GetPriorityClass (                    // Handle para o processo alvo
        HANDLE hProcess);

BOOL SetPriority Class (                    // Handle para o processo alvo
        HANDLE hProcess,
        DWORD dwPriorityClass);             // Nova classe desejada

int GetThreadPriority
        HANDLE hThread);                    // Handle para a thread alvo

BOOL SetThreadPriority
        HANDLE hThread,                     // Handle para a thread alvo
        int nPriority);                     // Nível de prioridade desejado
```

59

He never sleeps, he never slumbers ...

```
void WINAPI Sleep(
DWORD dwMilliseconds);      // Atraso em milisegundos
                            0: thread irá abandonar o seu time slice em favor de outra thread
```

60

Desabilitando a promoção de prioridades

```
BOOL SetProcessPriorityBoost(
        HANDLE hProcess,        // Handle para o processo alvo
        BOOL fDisableBoost);    // TRUE: desabilita promoção de prioridades
```

```
BOOL SetThreadPriorityBoost(
        HANDLE hThread,         // Handle para a thread alvo
        BOOL fDisableBoost);    // TRUE: desabilita promoção de prioridades
```

61

Suspendendo e Acordando uma Thread

```
BOOL SuspendThread(
        HANDLE hThread);        // Handle para a thread alvo
```

```
BOOL ResumeThread(
        HANDLE hThread);        // Handle para a thread alvo
```

62

Programando com Segurança

```
DWORD WINAPI GetLastError (VOID);
```

```
#define MY_ERROR                0x20000000
DWORD WINAPI SetLastError (
        DWORD dwErrorCode);
```
// Código de Erro a ser retornado para a aplicação

Conferindo o retorno das funções

```
#define WIN32_LEAN_AND_MEAN
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <process.h>          // _beginthreadex() e _endthreadex()
#include <conio.h>            // _getch

#define _CHECKERROR 1         // Ativa função CheckForError
#include "CheckForError.h"
// Casting para terceiro e sexto parâmetros da função _beginthreadex
typedef unsigned (WINAPI *CAST_FUNCTION)(LPVOID);
typedef unsigned *CAST_LPDWORD;

DWORD WINAPI TestFunc(LPVOID);          // declaração da função
int main()
{
    HANDLE hThreads[3];
    DWORD dwThreadId;
    DWORD dwExitCode = 0;
    BOOL bFlag[3] = {TRUE, TRUE, TRUE};
    int i;

    // Define o título da janela
    SetConsoleTitle("Programa 2.3 - GetLastError...");
```

Conferindo o retorno das funções

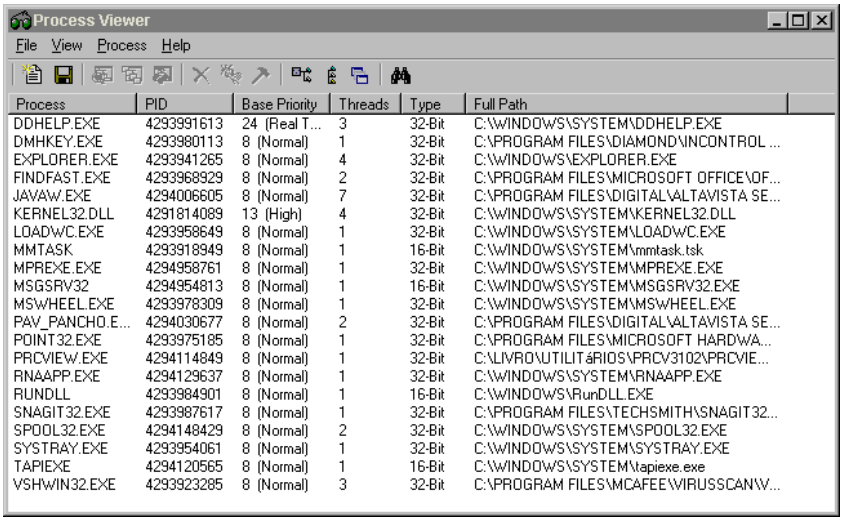
```
for (i=0; i<3; ++i) {      // cria 3 threads
      hThreads[i] = (HANDLE) _beginthreadex(
                NULL, 0,
                (CAST_FUNCTION)TestFunc,
                (LPVOID)i, 0, (CAST_LPDWORD)&dwThreadId
      );
      CheckForError(hThreads[i]);
      printf("\nThread %d criada com Id= %0x\n",i,dwThreadId);
}     // for
// Apaga Referência ao objeto (Handles) para forçar ERRO da função GetExitCode
for (i=0; i<3; ++i) CloseHandle(hThreads[i]); // Apaga referência ao objeto
i = 0;          // espera término das threads
while (bFlag[0] + bFlag[1] + bFlag[2] >0) {
  BOOL bRet;
  if (bFlag[i]) {
      bRet = GetExitCodeThread(hThreads[i], &dwExitCode);
      CheckForError(bRet);
      if (bRet == FALSE) bFlag[i]= FALSE;
      if (bRet == TRUE && dwExitCode != STILL_ACTIVE){
                printf("thread %d terminou com codigo de saida %d\n", i, dwExitCode);
                bFlag[i] = FALSE;
      }; // if
  }; // if
  i = (i + 1) % 3;
} // while
```

Conferindo o retorno das funções

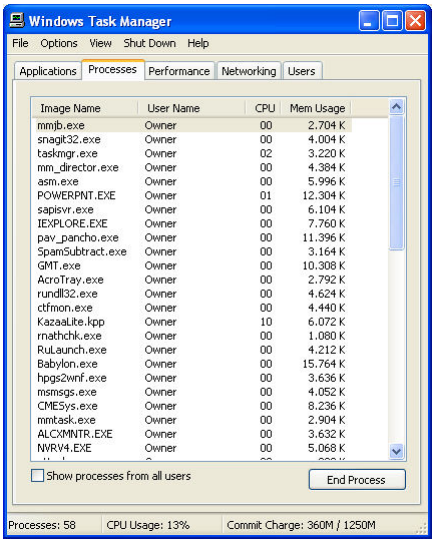
```
    printf("\nAcione uma tecla para terminar\n");
    _getch(); // Espere aqui, caso não esteja no ambiente MDS
    return EXIT_SUCCESS;
}   // main


DWORD WINAPI TestFunc(LPVOID index)
{
    // Sleep(50);    // Da um tempo ...
    printf("Thread %d em operacao\n", index);

    _endthreadex((DWORD) index);

    return(0);
} // TestFunc
```

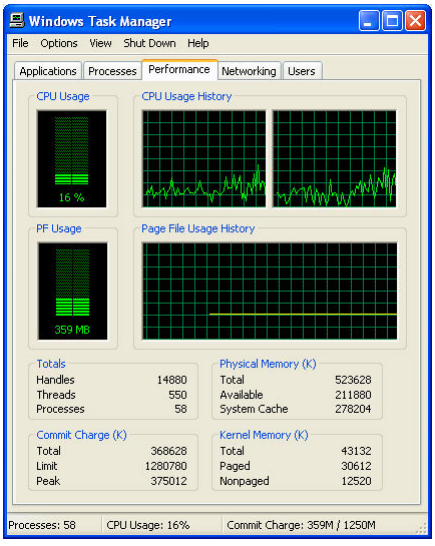
66

Visualizando Processos e Threads (95/98)



Visualizando Processos e Threads (XP)





Exemplo 1: CriaProcesso.exe: Criando processos

69

CriaProcesso.exe: Criando processos

```
void CCriaProcessoDlg::OnCriaProcesso()
{
   CEdit * edNomeProcesso;
   char szProcesso[200], szMsgErro[100];
   STARTUPINFO siStartinfo;

   // obtém o nome do processo a ser criado e o coloca na variável szProcesso
   edNomeProcesso = (CEdit *) GetDlgItem (IDC_NOME_PROCESSO);
   edNomeProcesso->GetWindowText (szProcesso, 200);
   // preencher estrutura de incialização do processo filho
   (STARTUPINFO)siStartinfo.cb = sizeof (STARTUPINFO);
   siStartinfo.lpReserved = NULL;
   siStartinfo.lpDesktop = NULL;
   siStartinfo.dwFlags = 0;
   siStartinfo.cbReserved2 = 0;
   siStartinfo.lpReserved2 = NULL;
   if (!CreateProcess (NULL, szProcesso, NULL, NULL, FALSE, 0, NULL, NULL,
   &siStartinfo, &piFilho))
        MensagemErro ("Erro ao criar processo");
}
```

70

CriaProcesso.exe: Criando processos

```
void CCriaProcessoDlg::OnEsperaProcessoTerminar()
{
   DWORD dwExitCode;
   char szMsgErro[200];
   /* Realizar uma espera ocupada até que o processo termine.
   Quando isto ocorrer, exibir uma mensagem */
   while (TRUE)
   {
      if (!GetExitCodeProcess (piFilho.hProcess, &dwExitCode)) {
        MensagemErro ("Erro ao tentar esperar pelo término do processo");
        break;
      }
      else
      {   if (dwExitCode != STILL_ACTIVE)
            {    AfxMessageBox ("Processo terminado!");
                 break;
            }
        }
   }  // while
}
```

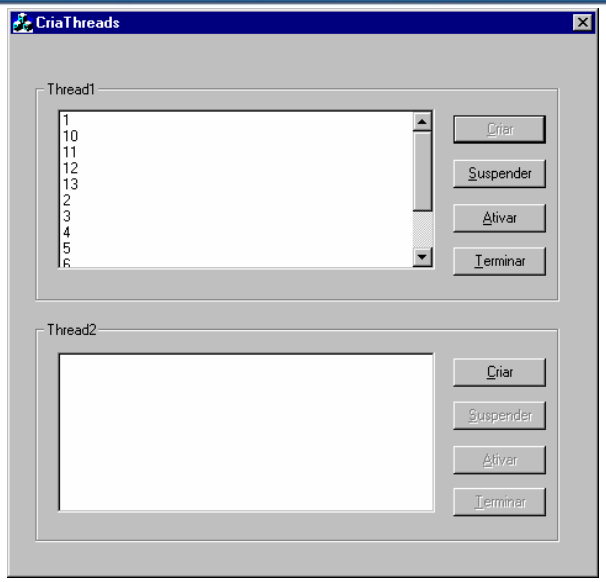
71

CriaProcesso.exe: Criando processos

```
void CCriaProcessoDlg::OnTerminaProcesso()
{
   /* terminar o processo */
   if (!TerminateProcess (piFilho.hProcess, 0))
        MensagemErro ("Erro ao tentar terminar processo");
}
```

72

Exemplo 2:
CriaThreads.exe: Criando threads



73

CriaThreads.exe: Criando threads

```
void CCriaThreadsDlg::OnCriarThread1()
{
    CListBox * Lista;
    Lista = (CListBox *) GetDlgItem (IDC_LIST1);
    Lista->ResetContent();
/*
Criar a thread1, passando como parâmetro para ela o list box no qual deverá escrever
(variável Lista). Se não conseguir criar, exibir uma mensagem de erro. Guardar o apontador
para o objeto na variável global pThread. A thread deverá ser criada suspensa.
*/
    pThreads[0]= AfxBeginThread(
          Thread,
          (LPVOID) Lista,
          THREAD_PRIORITY_NORMAL,
          0,
          CREATE_SUSPENDED);
    if (pThreads[0] == NULL)
          MensagemErro ("Não foi possível criar a thread1");
    else{
          pThreads[0]->m_bAutoDelete = FALSE;
          hThreads[0] = pThreads[0]->m_hThread;
          pThreads[0]->ResumeThread();
          /* habilita os botões */
          btCriarThread1->EnableWindow (FALSE);
          btSuspenderThread1->EnableWindow (TRUE);
          btAtivarThread1->EnableWindow (TRUE);
          btTerminarThread1->EnableWindow (TRUE);
          }
}
```

74

CriaThreads.exe: Criando threads

```
/* Thread que será executada. Recebe como parâmetro apontador para list box no
   qual deverá escrever.
*/
UINT Thread (LPVOID Param)
{
   CListBox * Lista;
   int i;
   char szAux[5];
   Lista = (CListBox *) Param;
   i = 0;
   while (TRUE)
   {
        Sleep (1000);
        i ++;
        Lista->AddString (itoa (i, szAux, 10));
   }
   return 0;
}
```

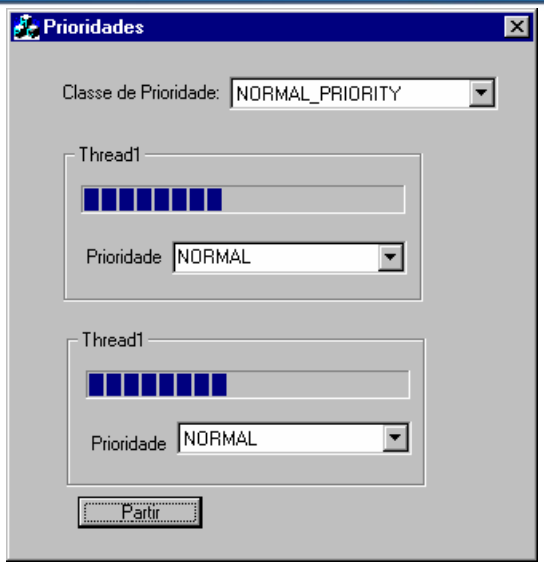
75

CriaThreads.exe: Criando threads

```
void CCriaThreadsDlg::OnSuspenderThread1() {
  if (pThreads[0]->SuspendThread() == 0xFFFFFFFF)
        MensagemErro ("Erro ao tentar suspender thread1");
}

void CCriaThreadsDlg::OnAtivarThread1() {
  if (pThreads[0]->ResumeThread() == 0xFFFFFFFF)
        MensagemErro ("Erro ao tentar ativar thread1");
}

void CCriaThreadsDlg::OnTerminarThread1() {
   if (!TerminateThread(hThreads[0], 0))
        MensagemErro ("Erro ao tentar terminar thread1");
   else
   {    delete pThreads[0];
        btCriarThread1->EnableWindow (TRUE);       /* Habilita os botões */
        btSuspenderThread1->EnableWindow (FALSE);
        btAtivarThread1->EnableWindow (FALSE);
        btTerminarThread1->EnableWindow (FALSE);
   }
}
```

76

Exemplo 3: Prioridades.exe
Alterando as prioridades de threads



77

Alterando as prioridades de threads

Teste:

		Processo 1	Processo 2
Classe de prioridades		NORMAL_PRIORITY	NORMAL_PRIORITY
Nível de prioridades	Thead1	ABOVE_NORMAL	HIGHEST
	Thread2	BELOW_NORMAL	IDLE

78

Alterando as prioridades de threads

```
void CPrioridadesDlg::OnSelchangeClassePrioridade()
{
   DWORD dwPrioridade;
   switch ( cbClassePrioridade->GetCurSel() )
   { case 0:     dwPrioridade = REALTIME_PRIORITY_CLASS;
                 break;
   case 1:       dwPrioridade = HIGH_PRIORITY_CLASS;
                 break;
   case 2:       dwPrioridade = NORMAL_PRIORITY_CLASS;
                 break;
   case 3:       dwPrioridade = IDLE_PRIORITY_CLASS;
                 break;
   }
   // modificar a classe de prioridade
   // (a prioridade está na variável dwPrioridade)
   SetPriorityClass (GetCurrentProcess(), dwPrioridade);}
```

79

Alterando as prioridades de threads

```
void CPrioridadesDlg::OnSelchangePrioridade1()
{
   DWORD dwPrioridade;
   switch (cbPrioridade1->GetCurSel())
   {     case 0: dwPrioridade = THREAD_PRIORITY_TIME_CRITICAL;
                 reak;
         case 1:dwPrioridade = THREAD_PRIORITY_HIGHEST;
                 break;
         case 2:dwPrioridade = THREAD_PRIORITY_ABOVE_NORMAL;
                 break;
         case 3: dwPrioridade = THREAD_PRIORITY_NORMAL;
                 break;
         case 4: dwPrioridade = THREAD_PRIORITY_BELOW_NORMAL;
                 break;
         case 5: dwPrioridade = THREAD_PRIORITY_LOWEST;
                 break;
         case 6: dwPrioridade = THREAD_PRIORITY_IDLE;
                 break;
   }
   /* mudar a prioridade da thread (está em dwPrioridade) */
   SetThreadPriority (hThread1, dwPrioridade);
}
```

80

Muito Obrigado

Perguntas?

Constantino Seixas Filho

constantino.seixas@task.com.br