
Sistemas Operativos 2

2021/22

Threads em Windows

DEIS/ISEC

Sistemas Operativos 2 – 2021/22

João Durães

1

Tópicos

Threads em Win32

Bibliografia específica para este capítulo:

- Windows System Programming; Johnson M. Hart - (4th Edition)
- WindowsNT 4 Programming; Herbert Schildt
- MSDN Library (online) – PlatformSDK: Processes, and Threads

DEIS/ISEC

Sistemas Operativos 2 – 2021/22

João Durães

2

Windows NT – *Threads* em Win32

Threads:

- Correspondem ao conceito de linha de execução.
- Entidade que executa as instruções de um programa no contexto de um processo.
- Intuitivamente pode assumir-se que um processo terá uma *thread* – é a *thread* que executa as instruções do programa no interior desse processo
- No entanto, normalmente (Windows, Linux, etc.), um processo pode ter mais do que uma *thread* em simultâneo
 - Para fazer várias tarefas em simultâneo
 - Cada *thread* pode executar uma zona de código diferente.
 - Muitos dos recursos do processo serão automaticamente partilhados (ex., variáveis globais) – é necessário usar sincronização

DEIS/ISEC

Sistemas Operativos 2 – 2021/22

João Durães

3

Windows NT – *Threads* em Win32

Threads em Windows (família NT):

- São as verdadeiras entidades de execução no sistema NT
- Cada processo tem pelo menos uma *thread* (*thread* inicial ou principal)
- A partir de uma *thread* pode-se lançar outra *thread*
- O facto da *thread* inicial ser a primeira (ou ser chamada de principal) não lhe confere nenhum privilégio em particular sobre outras *threads* que venham a existir no processo
- A criação da *thread* está associada a uma função. Quando essa função terminar, a *thread* também termina.

DEIS/ISEC

Sistemas Operativos 2 – 2021/22

João Durães

4

Windows NT – *Threads* em Win32

Diferenças entre processos e *threads*

Um processo tem:

- Um espaço de endereçamento (zonas de código e de dados)
- Uma identidade única
- Uma prioridade (usada como base de cálculo da prioridade das suas *threads*)
- Uma ou mais *threads*

Uma *thread* tem:

- Pilha própria dentro do espaço do processo a que pertence
- Uma identidade própria
- Recursos atribuídos (pode incluir zonas de dados próprias)
- Capacidade de executar instruções
- Uma prioridade

DEIS/ISEC

Sistemas Operativos 2 – 2021/22

João Durães

5

Windows NT – *Threads* em Win32

API Win32 principal para lidar com *threads*

- | | |
|-----------------------------|---|
| • CreateThread | Cria uma <i>thread</i> nova |
| • ExitThread | Termina a <i>thread</i> |
| • TerminateThread | Termina a uma <i>thread</i> (evitar usar) |
| • GetExitCodeThread | Obtém o código de terminação |
| • SuspendThread | Suspende a execução de uma <i>thread</i> |
| • ResumeThread | Recomeça a execução de uma <i>thread</i> |
| • GetCurrentThread | Obtém um <i>handle</i> local da <i>thread</i> |
| • GetCurrentThreadId | Obtém o identificador da <i>thread</i> |
| • OpenThread | Obtém um <i>handle</i> para uma <i>thread</i> |

DEIS/ISEC

Sistemas Operativos 2 – 2021/22

João Durães

6

Windows NT – *Threads* em Win32

Criação de *threads*

```
HANDLE CreateThread(  
    LPSECURITY_ATTRIBUTES lpThreadAttributes, // descr. segurança  
    DWORD dwStackSize, // tam. inicial pilha  
    LPTHREAD_START_ROUTINE lpStartAddress, // função da thread  
    LPVOID lpParameter, // argumento da func.  
    DWORD dwCreationFlags, // opções de criação  
    LPDWORD lpThreadId // ptr para thread ID  
);
```

Observações

- Tamanho *default* da pilha: 1MB
- Exemplo de opções de criação: **CREATE_SUSPENDED**
- lpStartAddress → Endereço para a **função da thread**
- lpParameter → Parâmetros a passar a função da *thread* quando esta inicia a execução. Pode-se ser qualquer tipo de dados desde que caiba num LPVOID (com *typecast*). Se exceder o tamanho de um LPVOID pode passar um ponteiro para uma estrutura qualquer com vários campos.

DEIS/ISEC

Sistemas Operativos 2 – 2021/22

João Durães

7

Windows NT – *Threads* em Win32

Função da *thread*

- É a função por onde começa a execução da nova *thread*
- Tem o seguinte aspecto

```
DWORD WINAPI ThreadFunc( LPVOID lpParam ) {  
    ... // variáveis locais;  
    ... // código da função  
    return valor; // código de terminação da thread  
}
```

Observações

- O parâmetro recebido é o especificado na altura da criação da *thread*
- Pode-se utilizar a mesma função para *threads* diferentes. Cada *thread* terá a sua cópia das variáveis locais (armazenadas na pilha dessa *thread*). *Threads* diferentes poderão também ter parâmetros diferentes na sua função
- Cada *thread* é dona dos recursos criados durante a sua execução. Mesmo que no caso das *threads* usarem (partilharem) a mesma função

DEIS/ISEC

Sistemas Operativos 2 – 2021/22

João Durães

8

Windows NT – *Threads* em Win32

Terminação da *thread* → ocorre quando:

- A sua função termina
- O processo que a contém é terminado
- É invocada uma das seguintes funções:

```
BOOL TerminateThread(  
    HANDLE hThread,    // handle da thread a terminar  
    DWORD dwExitCode   // código de terminação  
);
```

Permite terminar uma *thread* (handle tem que ter `THREAD_TERMINATE`)

```
VOID ExitThread(  
    DWORD dwExitCode   // código de terminação da thread  
);
```

Termina a *thread* que invoca esta função

DEIS/ISEC

Sistemas Operativos 2 – 2021/22

João Durães

9

Windows NT – *Threads* em Win32

Terminação de *threads* através de `TerminateThread` :

→ É desaconselhada

- A *thread* não tem hipótese de executar código *clean-up*
- As DLL anexadas não são notificadas que a *thread* terminou
- A pilha da *thread* pode não ser libertada
- Se a *thread* possuir alguma secção crítica, essa secção permanecerá ocupada (mas as outras *threads* que estivessem à espera que a *thread* terminasse continuam a ser notificadas)
- Algumas funções da DLL `Kernel32` podem ficar num estado incoerente se estivessem a ser executadas no contexto da *thread* que foi terminada

DEIS/ISEC

Sistemas Operativos 2 – 2021/22

João Durães

10

Windows NT – *Threads* em Win32

Terminação de um *thread* (a partir de outra)

→ Método aconselhado (válido em qualquer sistema/linguagem)

- Usa-se uma *variável de condição* acessível às *threads* envolvidas
 - Pelo menos duas: a *thread* cuja execução se deseja controlar e a *thread* que vai controlar a primeira
 - Para simplicidade vai-se assumir que se trata de uma variável global (mas não precisa de ser assim → ponteiro passado por parâmetro)
- A *thread* cuja execução é controlada vai periodicamente averiguar o valor dessa variável. Se tiver um determinado valor termina
 - Uma forma de ver este mecanismo é a de imaginar a existência de um ciclo cuja condição de paragem é baseada nessa variável. Tal ciclo existirá na função da *thread*
- A *thread* que deseja controlar a execução da primeira apenas precisará de colocar na variável o valor que indica que a primeira *thread* deve parar.

DEIS/ISEC

Sistemas Operativos 2 – 2021/22

João Durães

11

Windows NT – *Threads* em Win32

Exemplo do método aconselhado para terminar *threads*

Contexto global

```
int DeveContinuar = 1; // variável global
```

Thread A – Thread cuja execução é controlada – função da *thread*

```
while (DeveContinuar == 1) {  
    ... // etc  
}  
... // código clean-up - a thread termina normalmente
```

Thread B – Thread que vai controlar a execução da primeira *thread*

```
... // etc  
DeveContinuar = 0; // na próxima iteração a thread A  
... // etc // termina
```

DEIS/ISEC

Sistemas Operativos 2 – 2021/22

João Durães

12

Windows NT – *Threads* em Win32

Obtenção do código de terminação de uma *thread*:

```
BOOL GetExitCodeThread(  
    HANDLE hThread,      // handle da thread terminada  
    LPDWORD lpExitCode   // ptr para receber o código de terminação  
);
```

Suspensão de uma *thread*:

```
DWORD SuspendThread(  
    HANDLE hThread       // handle da thread a suspender  
);
```

Recomeço de uma *thread* suspensa

```
DWORD ResumeThread(  
    HANDLE hThread       // handle da thread a recomeçar  
);
```

Suspensão/recomeço: O *handle* deverá ter o privilégio **THREAD_SUSPEND_RESUME**

DEIS/ISEC

Sistemas Operativos 2 – 2021/22

João Durães

13

Windows NT – *Threads* em Win32

Obtenção de um *handle* local (*pseudo-handle*) para a *thread*:

```
HANDLE GetCurrentThread(VOID);
```

O *handle* apenas serve no contexto da própria *thread*. Pode-se obter, através de **DuplicateHandle**, um verdadeiro *handle* utilizável noutras *threads*

Obtenção do ID da *thread* (único a nível de sistema)

```
DWORD GetCurrentThreadId(VOID);
```

Obtenção de um *handle* para uma *thread* dado o identificador

```
HANDLE OpenThread(  
    DWORD dwDesiredAccess, // priv. de acesso  
    BOOL bInheritHandle,   // herança de handle  
    DWORD dwThreadId       // identificador da thread  
);
```

O *handle* obtido é um verdadeiro *handle* e não um *pseudo-handle*

DEIS/ISEC

Sistemas Operativos 2 – 2021/22

João Durães

14

Windows NT – *Threads* em Win32

Privilégios de acesso – alguns exemplos

- **SYNCHRONIZE**
Permite usar o *handle* em funções de sincronização
- **THREAD_ALL_ACCESS**
Todos os acessos suportados pelo sistema
- **THREAD_QUERY_INFORMATION**
Permite usar o *handle* para obter o código de terminação e outro tipos de informação acerca da *thread*
- **THREAD_SUSPEND_RESUME**
Permite suspender/des-suspender a *thread* através do *handle* obtido
- **THREAD_TERMINATE**
Permite terminar a *thread* através do *handle* obtido

DEIS/ISEC

Sistemas Operativos 2 – 2021/22

João Durães

15

Windows NT – *Threads* em Win32

Acerca das funções associadas às *threads*:

- Cada *thread* está associada a uma determinada função
- Diferentes *threads* podem estar associadas a diferentes *threads*
 - Este é o cenário normal quando as várias *threads* desempenham tarefas diferentes entre si (código e dados diferentes)
- Diferentes *threads* podem estar associadas à mesma função
 - Cenário normal quando várias *threads* desempenham a mesma tarefa, variando apenas nos dados que manipulam (mesmo código, dados diferentes)
 - Cada *thread* tem uma “execução” da função
 - Não há qualquer atropelo ou partilha de variáveis locais. Cada execução da função ocorre no espaço local da *thread* e as variáveis locais são duplicadas para cada execução (este é o comportamento habitual)

DEIS/ISEC

Sistemas Operativos 2 – 2021/22

João Durães

16

Windows NT – *Threads* em Win32

Questões de compatibilidade:

A utilização de funções da biblioteca C standard dentro de *threads* pode levar a que haja pequenas perdas de memória (*memory leaks*) se as *threads* tiverem sido baseadas nas funções **CreateThread** e **ExitThread**

Soluções:

- Usar **_beginthreadex** em vez de **CreateThread**
- Usar **_endthreadex** em vez de **ExitThread**

Em muitos casos, estes problemas vão sendo resolvidos em novas edições do Visual Studio e do Windows. No entanto alguns casos podem perdurar

Consultar sempre a documentação pormenorizada acerca do API quanto ao seu uso no contexto *multi-thread*

DEIS/ISEC

Sistemas Operativos 2 – 2021/22

João Durães

17

Windows NT – *Threads* em Win32

Criação de *threads* com **_beginthread** / **_beginthreadex**

```
uintptr_t _beginthread(  
    void(__cdecl *start_address)(void *),    // função da thread  
    unsigned stack_size,                     // tam. inicial pilha  
    void *arglist,                           // argumento da func.  
);
```

```
uintptr_t _beginthreadex(  
    void *security,                          // descr. segurança  
    unsigned stack_size,                     // tam. inicial pilha  
    void (__stdcall *start_address)(void *), // função da thread  
    void *arglist,                           // argumento da func.  
    unsigned initflag,                       // opções de criação  
    unsigned *thrdaddr                       // ptr para thread ID  
);
```

DEIS/ISEC

Sistemas Operativos 2 – 2021/22

João Durães

18

Windows NT – *Threads* em Win32

Função da thread com `_endthread` / `_endthreadex`

Se for usada `_beginthread`

```
void __cdecl funcao (void *) { ... }
```

Se for usada `_beginthreadex`

```
void __stdcall funcao(void *) { ... }
```

Se se terminar explicitamente a thread (não recomendado), deve-se usar

`_endthread` - se tiver sido criada com `_beginthread`

`_endthreadex` - se tiver sido criada com `_beginthreadex`