Sistemas Operativos 2

2023/24

Aplicações nativas Win32/WinNT

Criação de processos

1

Tópicos

API para criação de processos

Bibliografia específica para este capítulo:

- WindowsNT 4 Programming; Herbert Schildt
- Windows System Programming
- Microsoft Docs – PlattformSDK: DLLs, Processes, and Threads (disponível online)

2

Windows NT – Modelo(s) de programação

Aplicações Win32

- Constituem o modelo principal de aplicações nativas para windows

- Modelo computacional: "Win API" (Win32 ou Win64)
- Win32 e Win64: Essencialmente o mesmo conjunto de funções
  - Apenas tamanho de dados e de ponteiros varia
  - Código de 32 bits é compatível com aplicações 64 bits
  - "Win32 API" → "Win API"

Alternativas ao Win API:

- Frameworks tais como MFC ou .NET
- Oferecem um conjunto de funcionalidade já implementada
- Não cobrem toda a funcionalidade do windows, pelo que é imprescindível entender a lógica Win API

Windows NT – Modelo(s) de programação

O API Win32 é a base para diversos modelos de programação

- Um ou mais por cada um dos sub-sistemas

Exemplos

- *Win32 console application – Aplicações CUI* (*Console User Interface*)
- *Win32 application – Aplicações GUI* (*Graphical User Interface*)
- **Ambos directamente suportados pelo subsistema win32**

A diferença entre aplicações Win32 CUI e GUI é mínima

- *Uma aplicação consola pode lançar janelas*
- *Uma aplicação gráfica pode lançar consolas*
- **Uma consola é uma janela que simplesmente já tem um comportamento predefinido**

Windows NT – Modelo(s) de programação

Win32 Console Application **(começa-se por este tipo de aplicação)**
(mantém o paradigma de programação habitual)

- Programação sequencial seguindo os mesmos paradigmas interacção definidos na lógica de consola
- Interacção com o utilizador via consola em modo texto

Win32 Application **(muda-se para este tipo mais adiante)**
(novo paradigma de programação orientada a eventos)

- Programação orientada por eventos, em que a sequência de acontecimentos no programa deixa de ser directamente controlada pelo programador
- Interacção como utilizador via interface gráfica

Os nomes ***Win32 Console Application*** e ***Win32 Application*** podem aparecer sob outra forma dependendo do IDE

Windows NT – *Win32 application*

Criação de processos

```
int APIENTRY WinMain (
     HINSTANCE hInstance,     // Instância actual da aplicação
     HINSTANCE hPrevInstance, // Instância anterior. É sempre NULL
     LPSTR lpCmdLine,         // Linha de commandos excp. Nome do prog
     int nCmdShow             // Forma de apresentar a janela (sugest)
)
```

Reparar que continua a existir o conceito de linha de comandos e parâmetros de linha de commandos

GetCommandLine -> Permite obter a linha de commandos completa

Para detectar outra instância já a correr -> Criar um recurso único, por exemplo, um Mutex, e ver se o código de erro retornado é ERROR_ALREADY_EXISTS

Windows NT – *Win32 application*

Criação de processos

```
BOOL CreateProcess(
  LPCTSTR lpApplicationName,    // nome do executavel
  LPTSTR lpCommandLine,         // command line string
  LPSECURITY_ATTRIBUTES lpProcessAttributes, // Sec. Descript.
  LPSECURITY_ATTRIBUTES lpThreadAttributes,  // Sec. Descript.
  BOOL bInheritHandles,         // opções de herança de handles
  DWORD dwCreationFlags,        // flags de creation
  LPVOID lpEnvironment,         // variáveis de ambiente
  LPCTSTR lpCurrentDirectory,   // directoria
  LPSTARTUPINFO lpStartupInfo,  // startup info
  LPPROCESS_INFORMATION lpProcessInformation // process info
);
```

Reparar que continua a existir o conceito de linha de comandos e parâmetros de linha de comandos

Windows NT – *Win32 application*

Acerca dos nomes dos tipos de dados :

- O nome da variável deve reflectir os seu tipo de dados.
  - Isto ajuda o programador a lembrar-se do tipo e do uso pretendido para essa variável em zonas de código longe do local onde a variável foi definida
  - Diminui o esforço do programador e possibilidade de bugs
  - É considerado má prática de programação não seguir esta prática
- Exemplo
  - LPCTSTR → Ponteiro para string (constante) de TCHAR (*)
    - **L**ong
    - **P**ointer (to)
    - **C**onstant
    - **T**char
    - **STR**ing

(*) Todos os ponteiros são Long. Os ponteiros “Near” já não têm expressão na arquitectura actual do Windows)

Windows NT – *Win32 application*

Acerca dos nomes das variáveis:

- Para os tipos de dados devem-se usar macros e typedefs já existentes (no windows.h e outros)
- Traduzem para os tipos de dados habituais de C.
- Os nomes (dessas macros) reflectem com grande detalhe o tipo de dados de uma forma bastante compacta e que ajuda o esforço do programador
- Exemplo
  - **lp**ApplicationName
    - A variável é um ponteiro (**l**ong **p**ointer)
  - **b**InheritHandles
    - A variável é booleana (**B**OOL)
  - **dw**CreationFlags
    - A variável é uma **d**ouble **w**orld (inteiro sem sinal de 32 bits)

→ Deve seguir-se esta lógica na programação para Windows

Windows NT – *Win32 application*

Parâmetros da função **`CreateProcess`**

`lpApplicationName` `// nome do executavel`

- Nome do programa a executar
- Pode incluir o caminho completo
- Pode ser NULL – nesse caso o nome do programa é a primeira string do parâmetro **`lpCommandLine`**

`lpCommandLine` `// command line string`

- Equivalente ao conceito de linha de comandos
- Pode ser NULL – nesse caso o parâmetro **`lpApplicationName`** é considerado a linha de comandos

Windows NT – *Win32 application*

Parâmetros da função **CreateProcess**

lpProcessAttributes // Sec. Descript.

- Ponteiro para uma estrutura SECURITY_ATTRIBUTES que define o descritor de segurança do processo novo
- NULL → descritor de segurança *default*

lpThreadAttributes, // Sec. Descript.

- Equivalente ao anterior mas relativo à *thread* principal do novo processo

bInheritHandles, // opções de herança de handles

- Indica se o novo processo herda os handles do processo que invoca a função

Windows NT – *Win32 application*

Parâmetros da função **CreateProcess**

dwCreationFlags // flags de criação do proc.

- Definição de opções do novo processo
- Exemplos de flags neste parâmetro
  - **CREATE_DEFAULT_ERROR_MODE**
    O novo processo não herda o *error mode* do processo pai (um processo pode mudar o seu *error mode* com a função **SetErrorMode**)
  - **CREATE_NEW_CONSOLE**
    Nas aplicações de consola faz com que o novo processo tenha a sua própria consola em vez de herdar a do processo pai
  - **CREATE_NO_WINDOW**
    Nas aplicações consola faz com que o novo processo não tenha consola inicial
  - **CREATE_SUSPENDED**
    A *thread* principal do novo processo fica no estado inicial suspensa (A função *ResumeThread* permite inicia-la).

Windows NT – *Win32 application*

Parâmetros da função **CreateProcess**

dwCreationFlags // (continuação)

- Permite também definir a prioridade do novo processo
- Prioridades possíveis:
  - REALTIME_PRIORITY_CLASS **(*)**
  - HIGH_PRIORITY_CLASS
  - ABOVE_NORMAL_PRIORITY_CLASS
  - NORMAL_PRIORITY_CLASS
  - BELOW_NORMAL_PRIORITY_CLASS
  - IDLE_PRIORITY_CLASS

(*) Nesta prioridade as *threads* do processo ficam com mais prioridade que o próprio sistema operativo e podem impedir que tarefas importantes (gestão de *cache*, rato, etc.) deixem de ser efectuadas

Windows NT – *Win32 application*

Parâmetros da função **CreateProcess**

lpEnvironment // variáveis de ambiente

- Variáveis de ambiente do novo processo
- Conjunto de *strings* terminadas por \0. O conjunto é terminado por \0
  Exemplo

```
Variavel1=valor\0
Variavel2=valor\0
varEtc=valoretc\0\0
```

Neste caso as variáveis do processo pai não são propagadas

NULL → o processo novo herda as variáveis de ambiente do processo pai

Windows NT – *Win32 application*

Parâmetros da função **`CreateProcess`**

`lpCurrentDirectory` `// directoria de trabalho`

- Indica qual a directoria de trabalho do novo processo
- NULL → o novo processo fica com a mesma directoria que o processo pai

`lpStartupInfo,` `// startup info`

- Ponteiro para uma estrutura STARTUPINFO que indica como é que a janela principal do novo processo deve aparecer

  Exemplos:
  - título
  - coordenadas
  - dimensão

Windows NT – *Win32 application*

Parâmetros da função **`CreateProcess`**

`lpProcessInformation` `// process info`

- Ponteiro para uma estrutura PROCESS_INFORMATION que será preenchida com dados acerca do novo processo

  Conteúdo:
  - `HANDLE hProcess;`
  - `HANDLE hThread;`
  - `DWORD dwProcessId;`
  - `DWORD dwThreadId`

Handle leaking → Fechar explicitamente os handles obtidos do processo/thread quando já não forem precisos

Windows NT – *Win32 application*

Outras funções relacionadas com a gestão de processos

```
UINT WinExec(
  LPCSTR lpCmdLine,  // linha de comandos
  UINT uCmdShow      // estilo da janela
);
```

```
DWORD LoadModule(
  LPCSTR lpModuleName,      // file name do programa
  LPVOID lpParameterBlock   // parametros
);
```

► Implementadas à custa da função **CreateProcess**

Load module também é usada para trabalhar com DLL

Windows NT – *Win32 application*

Outras funções relacionadas com a gestão de processos

```
HANDLE OpenProcess(
  DWORD dwDesiredAccess,  // flags de acesso
  BOOL bInheritHandle,    // herança de handles
  DWORD dwProcessId       // identificador do processo
);
```

► Obtém um handle para um processo já em execução dado o seu ID. Preenche uma estrutura **PROCESS_INFORMATION**

```
DWORD WaitForInputIdle(
  HANDLE hProcess,       // handle do processo
  DWORD dwMilliseconds   // time-out
);
```

► Aguarda que o novo processo complete a sua inicialização e esteja à espera de input

Windows NT – *Win32 application*

Outras funções relacionadas com a gestão de processos

```
VOID ExitProcess(
  UINT uExitCode   // exit code para todas as threads
);
```

- Termina o processo (termina todas as *threads* do processo) com indicação do código de terminação (exit code)

```
BOOL GetExitCodeProcess(
  HANDLE hProcess,     // handle do processo
  LPDWORD lpExitCode   // pont. p/ código de terminação
);
```

- Obtém o código de terminação do processo (dado o handle dele)

Windows NT – *Win32 application*

Relacionadas com processos mas não só

```
DWORD WaitForSingleObject(
        HANDLE hHandle,        // neste caso, do processo
        DWORD  dwMilliseconds  // timeout (INFINITE=sempre)
);
```

- Função para aguardar em objectos de sincronização (mutexes, semáforos, etc.)

  Quando usada num processo aguarda que este termine