Sistemas Operativos 2

2021/22

Bibliotecas dinâmicas em Windows

1

Tópicos

Bibliotecas dinâmicas

- Conceito e funcionamento geral
- Ligação explícita
- Ligação implícita

Bibliografia específica para este capítulo:

- Windows System Programming; Johnson M. Hart - (4th Edition)
- Advanced Windows (3rd Edition); Jeffrey Richter
- WindowsNT 4 Programming; Herbert Schildt
- MSDNAA Library (online) – PlattformSDK: DLLs, Processes, and Threads

2

Windows NT – DLL : *Dinamic Link Libraries*

DLL – ***D**ynamic **L**ink **L**ibrary*

- Biblioteca (conjunto) de funções ou recursos para um determinado fim
- Podem ser utilizadas por diversos processos
- São mapeadas nos espaço de endereçamento dos processos em ***runtime*** (ligação dinâmica)

Uma DLL é mapeada (ligada) num processo de duas formas

- **Implicitamente** (***load time*** *linking*)
  São dadas indicações ao compilador acerca do uso da biblioteca
- **Explicitamente** (***run-time*** *linking*)
  Gestão explicita feita pelo programador em runtime

Após o mapeamento, as funções exportadas pela DLL podem ser invocadas pelas *threads* do processo em que foi mapeada.

Os recursos (handles, objectos, memória) usados pelas DLL pertencem aos processos que usam/invocam as funções da DLL, e não à DLL propriamente dita

Windows NT – DLL : *Dinamic Link Libraries*

Criação de uma DLL com o Visual Studio

- Projecto Win32, maioritariamente da forma habitual
- File -> New -> Project
  - **Tipo de aplicação: DLL**
- Opcionalmente, configurações adicionais
  - *Empty project*, sem *precompiled headers*, sem SDL

As DLL e os executáveis Win32 partilham o mesmo formato de ficheiro binário ("PE – *Portable Executable*")

Windows NT – DLL : *Dinamic Link Libraries*

Exportação de funções e variáveis

- Uma DLL pode ter funções e variáveis.
- Destas, todas ou apenas algumas podem ser exportadas para processos interessados em as utilizar

O formato do ficheiro "binário" (executável) da DLL é muito semelhante ao de um executável normal. Existe uma estrutura interna que identifica e eventualmente publica variáveis externas, funções, recursos, etc.

- A DLL tem uma tabela de exportação
  - A tabela de exportação contém os nomes das funções a exportar a executáveis que usam a DLL.
  - As funções que não são mencionadas na tabela de exportação tornam-se privadas à DLL

Windows NT – DLL : *Dinamic Link Libraries*

As funções a exportar pela DLL devem ser identificadas (no código fonte da DLL) com **`__declspec(dllexport)`**

`__declspec`

- Palavra chave para as extensões de C/C++ específicas à Microsoft
- Especifica como o armazenamento de determinado objecto é caracterizado através de um atributo/parâmetro
  - Exemplos de atributos: dllexport, dllimport, nothrow, novtable
- Deve ser colocado no início das declarações, caso contrário será ignorado sem aviso

  Exemplo

  ```
  __declspec(dllimport) class ABC { } varXYZ;
  ```

  (neste exemplo, a declaração afecta a variável *varXYZ*)

Windows NT – DLL : *Dinamic Link Libraries*

Exemplo genérico: Exportação de uma função e de uma variável

No código fonte da dll:

Funções

```
__declspec(dllexport) void FuncaoExportada() {
  ...
}
```

Variáveis

```
__declspec(dllexport) int Var = ...;
```

Na prática, o que é exportado são os ponteiros para as funções/variáveis.
Este aspecto pode ser oculto ao programador dependendo da forma como se usa a DLL

Windows NT – DLL : *Dinamic Link Libraries*

Exemplo (continuação)

No código fonte do programa que vai usar a DLL deve existir:

Funções

"**import**" – a função é **importada**

```
__declspec(dllimport) __cdecl void FunçãoExportada();
```

Só o protótipo (a função já se encontra implementada na DLL)

Variáveis

```
extern __declspec(dllimport) int Var;
```

Este "**extern**" significa que o espaço para a variável já existe noutro local (na DLL)

Normalmente usam-se macros para simplificar as declarações. Por exemplo:

```
# define DLL_EXPORT __declspec(dllexport)
# define DLL_IMPORT __declspec(dllimport)
```

Windows NT – DLL : *Dinamic Link Libraries*

No programa que vai usar a DLL

- São necessárias as declarações referentes a ***import*** e ***extern*** (exemplos no slide anterior) – normalmente num ***.h***
- Esse .h deverá, em princípio, ter sido construído pelo programador da DLL, cabendo a quem a usa incluir esse .h

Os ficheiros envolvidos numa DLL dependem muito de:

- A forma como os autores da DLL escolheram tornar visíveis as funções
- A forma como os autores do programa que vai usar a DLL carrega a DLL, e identifica e invoca as funções

O próximo slide apresenta um resumo acerca dos ficheiros envolvidos que depois será melhor detalhado ao longo dos slides desta matéria

Windows NT – DLL : *Dinamic Link Libraries*

Normalmente não se fornece o código fonte da DLL (a não ser que se trate de um projecto *open-source*).

O que é fornecido pelos autores da DLL (aos clientes que a usam):

- **O ficheiro .DLL**. Este ficheiro é o "binário" / "executável". Contém as funções, variáveis e recursos. Constitui o mínimo essencial e é sempre necessário.
- **O ficheiro .h**. Apresenta a interface visível (usável) da DLL. Normalmente é fornecida, sendo pouco usual que se obrigue o utilizador da DLL a reconstrui-lo pois faltar-lhe-iam dados para tal. Note-se que **este .h não é o mesmo que é usado na construção da DLL** (por exemplo, num caso usa-se dllexport, noutro dllimport)

Windows NT – DLL : *Dinamic Link Libraries*

Normalmente não se fornece o código fonte da DLL (a não ser que se trate de um projecto *open-source*).

O que é fornecido pelos autores da DLL:

- **O ficheiro .lib** (opcional). Usado para os casos de ***ligação implícita*** (a ver mais adiante)
- **O ficheiro .DEF** (opcional). Método alternativo para identificar as funções em ***ligação explícita*** (identificação por *número de ordem* em vez de *pelo nome*)

Windows NT – DLL : *Dinamic Link Libraries*

Identificação e utilização das funções existentes na DLL
→ Método habitual de identificação: **por nome**

O método mais habitual é o de identificar as funções pelo nome. A identificação posterior por parte dos programas que usam a DLL é feita então de duas formas: **ligação implícita** e **ligação explícita**

Em ambos os casos a DLL tem que **estar sempre presente** durante a execução

Ligação implícita

- É usado um ficheiro auxiliar **.lib** = biblioteca de **ligação estática** para estabelecer a **ponte** entre o **programa** e as funções na **DLL**
- O programa que usa as funções apenas precisa do .h e do .lib durante a compilação e usa as funções quase como se elas fizessem parte do seu código.
- É a forma mais confortável, mas menos flexível.

Windows NT – DLL : *Dinamic Link Libraries*

Identificação e utilização das funções existentes na DLL

Ligação explícita

- Não é necessário nenhum .lib, e, eventualmente, nem sequer um .h durante a compilação
- O programa que usa a DLL tem que ***carregar explicitamente*** a DLL e depois ***procurar*** as funções pelo nome, invocando-as por ponteiro.
- É menos confortável para o programador, mas é mais flexível.

Nota:
As ligações **implícita** e **explícita** são detalhadas mais adiante nestes slides

Windows NT – DLL : *Dinamic Link Libraries*

Identificação e utilização das funções existentes na DLL

→ Método alternativo: **por número ordinal**

Também se podem exportar funções por um *ordinal* em vez de pelo nome

- Atribui-se um número de ordem a cada função.
- As funções serão mais tarde identificadas por esse número e não pelo seu nome original
- Usa-se o parâmetro NONAME na declaração __declspec

Este método faz sentido quando

- Há um grande número de funções a exportar e pretende-se poupar alguma memória (não se guardam os nomes das funções)
- A DLL está em desenvolvimento e prevê-se acrescentar ou modificar funções

Este método exige a criação de um ficheiro .DEF com a identificação das funções e dos ordinais (exemplo no próximo slide)

Windows NT – DLL : *Dinamic Link Libraries*

Identificação por ordinal

Exemplo de ficheiro .DEF
(funções exportadas *insereValor*, *mudaValor* e *comparaCom*)

```
LIBRARY Exemplo
EXPORTS
    insereValor  @1
    mudaValor    @2
    comparaCom   @3
```

Notas:
- A identificação **por nome** é a mais usada
- A identificação ordinal não é usada nos exemplos em SO2

Windows NT – DLL : *Dinamic Link Libraries*

Código da DLL

- Uma DLL é uma variante de um ficheiro executável.
- Não se destina a executar por si só.
- Age como um repositório de funções e recursos para ser usado por outros programas

→ Assim, não existe propriamente uma lógica típica associada aos programas (função *main* etc.). No entanto, existe uma função principal cuja existência pode ser útil

Os próximos slides apresentam exemplos de código de uma DLL numa aplicação win32

Windows NT – DLL : *Dinamic Link Libraries*

Função "principal" da DLL

- Não é obrigatória a sua existência (o compilador fornece uma por omissão)
- Invocada automaticamente em ocasiões específicas

```
BOOL APIENTRY DllMain( HANDLE hModule, // proc. em questão
                       DWORD  ul_reason_for_call,
                       LPVOID lpReserved) {

  switch (ul_reason_for_call) {

    case DLL_PROCESS_ATTACH: // o proc. mapeou a DLL
    case DLL_THREAD_ATTACH:  // proc. criou uma nova thread
    case DLL_THREAD_DETACH:  // uma thread do proc terminou
    case DLL_PROCESS_DETACH: // o proc. des-mapeou
        break;
  }
  return TRUE;
}
```


Windows NT – DLL : *Dinamic Link Libraries*

Nesta DLL, para este exemplo:

- **Mostra uma mensagem quando é mapeada**
- **Exporta uma função e uma variável**
- **Tem uma função que não exporta (destina-se a uso interno à DLL)**

```
#include <windows.h>

// Função "local“ -> não vai ser exportada

int factorial (int n) {  // calcula factorial de n
   int res = n;
   n--;
   while (n>1) {
        res = res * n;
        n--;
   }
   return res;
}
```

Windows NT – DLL : *Dinamic Link Libraries*

Exemplo (continuação) – função DllMain

```
BOOL APIENTRY DllMain( HANDLE hModule,
                       DWORD  ul_reason_for_call,
                       LPVOID lpReserved ) {

    switch (ul_reason_for_call)      {
       case DLL_PROCESS_ATTACH:
              _tprintf(_T("DLL attached\n"));
              break;
       case DLL_THREAD_ATTACH:
       case DLL_THREAD_DETACH:
       case DLL_PROCESS_DETACH:
              break;
    }
    return TRUE;
}
```


Windows NT – DLL : *Dinamic Link Libraries*

Exemplo (continuação) – função e variável exportadas

```
// Variável exportada

__declspec(dllexport) int nExemplo = 0;


// Função exportada

__declspec(dllexport) int fnExemplo(int n) {
   if (n<0)
        n = -1;
   return factorial(n);
}
```

Resultado da compilação

- **exemplo.dll** → Código da DLL
- **exemplo.lib** → Ficheiro auxiliar para ligação implícita

Windows NT – DLL : *Dinamic Link Libraries*

Utilização da DLL - Ligação do código da DLL ao código do programa

Ligação implícita *versus* **ligação explícita**

Ligação implícita

Forma de uso da DLL que oculta o facto de se estarem a usar recursos externos ao programa. As funções aparentam estar no próprio programa e o seu uso é muito simplificado.

> A carga da DLL, procura de funções e libertação da DLL são feitas automaticamente por código disponibilizado numa pequena biblioteca de **ligação estática** (o .lib) que faz a ponte entre o programa e a DLL.
>
> O .lib contém funções com o mesmo nome que as existem no .dll, reencaminhando as chamadas para lá

O código adaptador presente no .lib:

- Oculta e simplifica a gestão da DLL: efectua internamente os passos descritos na ligação explícita
- Remove flexibilidade, ocupa recursos (memória) durante mais tempo

21

Windows NT – DLL : *Dinamic Link Libraries*

Utilização da DLL - Ligação do código da DLL ao código do programa

Ligação implícita *versus* **ligação explícita**

Ligação implícita → Acções do programador

- A DLL é carregada automaticamente no início da execução e libertada no fim. Se não estiver presente, o programa não corre de todo.
- As funções são invocadas de forma natural pelo nome, sendo necessário apenas o .h e o .lib durante a compilação (e a DLL durante a execução).

  > **A existência do .h permite ao compilador validar a sintaxe/parâmetros das funções, ajudando o programador**

- A DLL mantem-se sempre em memória e utilizando assim mais memória

Nesta forma, os recursos da DLL aparentam estar no programa tal como se de uma biblioteca de ligação estática se tratasse.

22

Windows NT – DLL : *Dinamic Link Libraries*

Ligação implícita

Tarefas do programador:

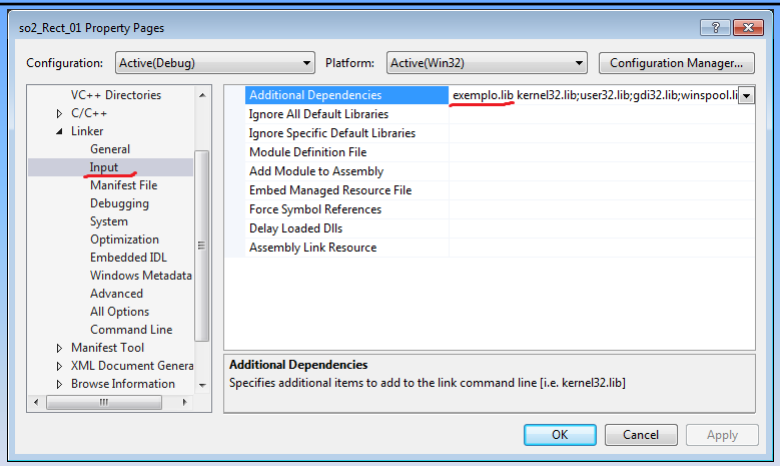
- Incluir as declarações das funções e variáveis que quer usar usando **dllimport**
  Geralmente corresponde à tarefa de incluir o .h fornecido juntamente com a DLL
- Indicar ao compilador/linker que deve incluir o ficheiro .lib correspondente à pequena biblioteca de ligação estática que tem o código que faz a gestão da DLL
  (figura no slide seguinte)
  Sem esta indicação ocorreria um erro de *linker* tal como:

```
error LNK2001: unresolved external symbol _fnExemplo
```

DEIS/ISEC Sistemas Operativos 2 – 2021/22 João Durães

23

Windows NT – DLL : *Dinamic Link Libraries*



A ligação à DLL mantém-se **dinâmica**: se a DLL não estiver presente quando o programa é lançado é assinalado um erro e o programa não prossegue

Apenas o ficheiro .lib é que é ligado de forma estática ao programa

DEIS/ISEC Sistemas Operativos 2 – 2021/22 João Durães

24

Windows NT – DLL : *Dinamic Link Libraries*

Utilização da DLL - Ligação do código da DLL ao código do programa

Ligação implícita *versus* **ligação explícita**

Ligação explícita

Forma de uso dos recursos da DLL que está mais próxima daquilo que a DLL é na realidade.

> A DLL é tratada como um conjunto de recursos que é trazida para memória e usada quando é necessário de uma forma explícita e totalmente controlada pelo programador do programa "cliente" da DLL.

Ou seja, a DLL é tratada como aquilo que realmente é.

Windows NT – DLL : *Dinamic Link Libraries*

Utilização da DLL - Ligação do código da DLL ao código do programa

Ligação implícita *versus* **ligação explícita**

Ligação explícita → Acções do programador

- O programador decide quando é que precisa de carregar a DLL e carrega-a explicitamente e apenas nessa altura
- O programador obtém acesso aos recursos da DLL procurando explicitamente as funções que precisa e invoca-as através de um ponteiro

  > **O compilador não oferece apoio na validação de parâmetros da função.**

- O programador liberta a DLL quando já não precisa mais dela, libertando recursos ao processo e ao sistema

Nesta forma torna-se óbvio ao programador que está a usar recursos externos ao seu programa

Windows NT – DLL : *Dinamic Link Libraries*

Ligação explícita

- O programador não precisa de indicar nada ao compilador/linker
- É preciso carregar "manualmente" a biblioteca (dll) para o seu processo e depois procurar as funções pretendidas

API Win32:

- **`LoadLibrary / LoadLibraryEx`**
  Mapeia um módulo (DLL) no espaço do processo
- **`GetProcAddress`**
  Obtém o endereço de uma função/variável exportada
- **`FindResource`**
  Obtém um recurso existente no módulo
- **`FreeLibrary`**
  Liberta ("des-mapeia") o módulo do espaço do processo

Windows NT – DLL : *Dinamic Link Libraries*

```
HMODULE LoadLibraryEx(
  LPCTSTR lpFileName,     // filename do módulo
  HANDLE hFile,           // reservado, sempre NULL
  DWORD dwFlags           // flags e opções
);  // Exemplo de flag: LOAD_LIBRARY_AS_DATAFILE
```

Em caso de erro a função retorna NULL

```
BOOL FreeLibrary(
  HMODULE hModule         // handle do módulo DLL
);
```

Depois de invocada a função **`FreeLibrary`** já não é possível usar as funções e variáveis da DLL no processo em questão

Windows NT – DLL : *Dinamic Link Libraries*

```
FARPROC GetProcAddress(
  HMODULE hModule,    // handle do módulo DLL
  LPCSTR lpProcName   // nome EXACTO da função/var
);
```

Para invocar a função é necessário efectuar o **typecast** do ponteiro obtido com **GetProcAddress** (compilador não valida)

```
HRSRC FindResource(
  HMODULE hModule,    // handle do módulo
  LPCTSTR lpName,     // nome do recurso
  LPCTSTR lpType      // tipo do recurso
);  // exemplo de tipo de recurso: RT_DIALOG (dialog box)
```


Windows NT – DLL : *Dinamic Link Libraries*

Reparar que

- O *typecast* do ponteiro retornado por GetProcAdress é necessário para converter um ponteiro genérico num ponteiro para a função com os parâmetros e tipo de retorno adequados (para o compilador gerar o código correcto na invocação dessa função).
  O compilador não consegue validar e exactidão do typecast. Uma discrepância neste ponto terá consequências apenas em *runtime*
- Se a DLL não estiver presente, ou se houver algum erro que impeça a sua utilização, o programa não tem que terminar. Apenas a funcionalidade associada à DLL fica inviabilizada. No caso da ligação implícita, se a DLL não estiver presente, todo o programa é impedido de correr.

Windows NT – DLL : *Dinamic Link Libraries*

Ligação implícita versus ligação explícita

A ligação explícita é mais trabalhosa de usar pelo programador da aplicação que usa a DLL. Mas tem algumas vantagens.

- Usando ligação explícita, o nome da DLL pode ser determinado apenas em *runtime* (por exemplo, um ficheiro de configuração). Esta característica pode ser importante.
- Se a DLL não estiver presente, o programa decide o que acontece. Na ligação implícita o programa não correria de todo.
- Se a função *DLLMain* falhar, na ligação implícita o processo termina. Na ligação explícita isso não acontece.
- Na ligação implícita, o compilador consegue efectuar algumas validações no uso das funções (pela existência do ficheiro .h).

31

Windows NT – DLL : *Dinamic Link Libraries*

Ligação implícita versus ligação explícita (continuação)

- Se existirem muitas DLL com ligação implícita, o processo tem um início demorado pois todas essas DLL são carregadas no início. Na ligação explícita as DLL são carregadas em qualquer altura.
- A ligação implícita requer a biblioteca estática para fazer a ponte com a DLL. Esta biblioteca pode tornar-se desajustada se a DLL for modificada (por exemplo, se os ordinais das funções exportadas mudarem). A ligação explícita não usa essa biblioteca.
- A ligação explícita permite carregar as DLL apenas quando necessário e libertá-las assim que não são necessárias. Isto permite poupar recursos ao processo e sistema. Na ligação implícita as DLL mantêm-se mapeadas durante a execução do processo.

32