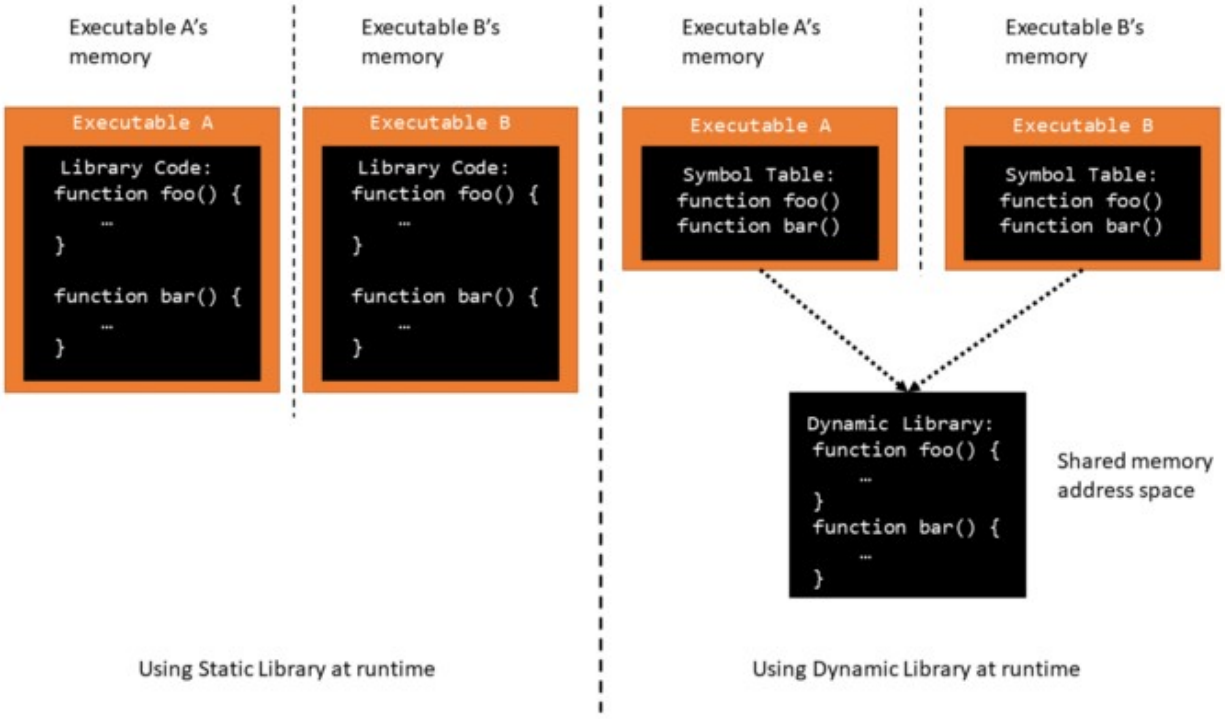
LIBRERIE

STATICHE E DINAMICHE IN WINDOWS E LINUX

Prof. Fischetti Pietro.



Nella parte sinistra della figura abbiamo l'utilizzo di librerie statiche mentre a destra una libreria dinamica (o condivisa).

Si osservi che ogni eseguibile avra' all'interno il codice delle funzioni che andra' ad utilizzare, mentre nel caso di una libreria dinamica l'eseguibile chiamera' durante l'esecuzione le funzioni presenti esternamente in un file separato. Cio' comporta che le librerie statiche produrranno codice piu' veloce, ma programmi piu' grossi e piu' sicuri per le modifiche, mentre una libreria dinamica causera' programmi leggermente piu' lenti ma eseguibili meno corposi comunque insidiosi per le modifiche (basti pensare alla distribuzione di librerie modificate). Per modifiche si intende cambiamenti nel codice delle librerie, i programmi che utilizzano librerie statiche dovranno essere ricompilati e quindi redistribuiti ai vari clienti, mentre i programmi che utilizzano quelle dinamiche non dovranno essere ricompilati se viene mantenuta la compatibilita', ad esempio se una funzione xyz(int a) si vuole modificare per accettare due parametri cioe' xyzw(int a, char b) la nuova libreria dinamica dovra' contenere sia la versione xyz che la xyzw per garantire che i vecchi programmi che utilizzavano la xyz continuino a funzionare,. Infine la lbreria dinamica deve fisicamente esistere per permettere ad un programma che la usa di funzionare, mentre quella statica si puo' eliminare dopo aver costruito i programmi che la usano. Ovviamente un programma puo' contenere sia librerie statiche che dinamiche.

Per i sistemi Windows i file che contengono le librerie statiche hanno estensione .lib, mentre quelle dinamiche estensione .dll, mentre in Linux le statiche estensione .a mentre quelle dinamiche .so

ESEMPIO

Realizziamo un semplice programma che utilizza delle funzioni aritmetiche (somma, differenza, moltiplicazioni e divisione)inserite in una libreria statica e dinamica in entrambi gli ambienti Windows e Linux. Ie funzioni aritmetiche le scriviamo in 4 files separati sum.c , diff.c, mult.c e div.c (non e' obbligatorio potrebbero essere messi in un unico file o in due file, come si preferisce), il programma principale lo chiamo use.c d e funzionera da linea di comando soecificado l'operazioe, cioe': use 3+4 scrivera' 7.

Comandi di compilazione

Questo e' il contenuto della directory:

aritme.h
diff.c
div.c
mult.c
sum.c
use.c

<table>
<tr><th colspan="2">WINDOWS</th></tr>
<tr><td>LIBRERIA STATICA (.lib)</td><td>LIBRERIA DINAMICA (.dll)</td></tr>
<tr><td></td><td>useDLL.c</td></tr>
<tr><td>cl -c sum.c
cl -c diff.c
cl -c mult.c
cl -c div.c

lib sum.obj diff.obj mult.obj div.obj /OUT:aritm.lib
cl use.c aritm.lib</td><td>cl -c dllmain.c
cl -c sum.c
cl -c diff.c
cl -c div.c
cl -c mult.c

link -nologo -dll /OUT:aritm.dll dllmain.obj sum.obj diff.obj div.obj mult.obj /def:source.def

cl /DUNICODE /D_UNICODE use.c</td></tr>
<tr><td>Per visualizzare le funzioni presenti in una libreria statica windows:
C:\>dumpbin /symbols /exports aritm.lib</td><td>Viene creata una libreria:aritm.dll
Per controllare le funzioni contenute o meglio esportate dare dal prompt dei comandi di Visual Studio:
C:>dumpbin /EXPORTS aritm.dll
Microsoft (R) COFF/PE Dumper Version 14.24.28315.0
Copyright (C) Microsoft Corporation. All rights reserved.

Dump of file aritm.dll

File Type: DLL

Section contains the following exports for aritm.dll

00000000 characteristics
FFFFFFFF time date stamp
0.00 version
1 ordinal base
4 number of functions
4 number of names

ordinal hint RVA name

1 0 00005D10 diffe
2 1 00005D40 divid
3 2 00005D70 prodo
4 3 00001140 somma</td></tr>
</table>

	Se cancello la libreria dinamica aritm.dll il programma va ovviamente in errore

LINUX	
LIBRERIA STATICA (.a)	LIBRERIA DINAMICA (.so)
gcc -c sum.c gcc -c diff.c gcc -c mult.c gcc -c div.c ar rcs libaritm.a sum.o diff.o mult.o div.o gcc -c use.c gcc use.o -L. -laritm	gcc -c sum.c gcc -c diff.c gcc -c mult.c gcc -c div.c gcc -shared -o libaritm.so sum.o diff.o mult.o div.o gcc -o useSO use.c -L. -laritm -Wl,-rpath=.

Per stampare le funzioni contenute nella libreria statica:
$ nm libaritm.a
	Viene creata una libreria libaritm.so e un programma cxon nome useSO
	Per controllare le funzioni esportate:
	$ nm libaritm.so
	Oppure:
	$ objdump –x libaritm.so
	Per vedere le librerie dinamiche utilizzate da un programma ad es useSO:
	$ ldd useSO
	
	Ovviamente se non e' presente la libreria libaritm.so il programma va in errore

Contenuti dei files:

sum.c

```
float somma(float a, float b){
        float c;
        c=a+b;
        return c;
}
```

diff.c

```
float diffe(float a, float b){
        float c;
        c=a-b;
        return c;
}
```

mult.c

```
float prodo(float a, float b){
        float c;
        c=a*b;
```

```
	return c;
}
```

div.c

```
float divid(float a, float b){
	float c;
	c=a/b;
	return c;
}
```

aritme.h

```
#ifndef ARITME_H
#define ARITME_H
float somma(float a, float b);
float diffe(float a, float b);
float prodo(float a, float b);
float divid(float a, float b);
#endif
```

aritme.h

```
#ifndef ARITME_H
#define ARITME_H
typedef float(*somma)(float, float);
typedef float(*diffe)(float, float);
typedef float(*prodo)(float, float);
typedef float(*divid)(float, float);
#endif
```

dllmain.c

```
// dllmain.cpp : Defines the entry point for the DLL application.
#include <Windows.h>
#include <tchar.h>
#include <stdio.h>

// entry point of every dll function
// because there can't be more than one main functions
BOOL APIENTRY DllMain( HMODULE hModule,
                       DWORD  ul_reason_for_call,
                       LPVOID lpReserved
                     )
{
   switch (ul_reason_for_call)
   {
   case DLL_PROCESS_ATTACH:
      // called when libary is loaded with LoadLibrary
      printf("Library loaded\n");
      break;
   case DLL_THREAD_ATTACH:
      printf("Thread attached\n");
```

```
    break;
  case DLL_THREAD_DETACH:
    printf("Thread detached\n");
    break;
  case DLL_PROCESS_DETACH:
    // called when libary is freed with FreeLibrary
    printf("Process detached\n");
    break;
  }
  return TRUE;
}
```

useDLL.c

```
#include <stdio.h>
#include <Windows.h>
#include <tchar.h>
#include <libloaderapi.h>
#include <wchar.h>
#include "aritme.h"

int main(int argc, char *argv[]) {
float a,b;
char c;

        if (argc < 2) {
                printf("usage: %s <x><+|-|/|*><y>\n", argv[0]);
                return 1;
        }

        HMODULE hDll = LoadLibrary(_T("aritm"));
        if (!hDll || hDll == INVALID_HANDLE_VALUE) {
                printf("unable to load libraray");
                return 1;
        }

        // address of loaded dll
        printf("library loaded at 0x%p\n", hDll);

        if (3==sscanf(argv[1],"%f%c%f",&a,&c,&b)){
        switch(c){
        case '+':{
                somma Add = (somma)GetProcAddress(hDll, "somma");
                if (!Add) printf("unable to load somma function\n");
                else printf("%f + %f = %f\n", a, b, Add(a, b)); // calling function
        break;
        }
        case '-':{
                diffe Sub = (diffe)GetProcAddress(hDll, "diffe");
                if (!Sub) printf("unable to load DivFunc\n");
                else printf("%f - %f = %f\n", a, b, Sub(a, b));
        break;
        }
```

```
        case '*':{
                prodo Pro = (prodo)GetProcAddress(hDll, "prodo");
                if (!Pro) printf("unable to load DivFunc\n");
                else printf("%f * %f = %f\n", a, b, Pro(a, b));
        break;
        }
        case '/':{
                divid Div = (divid)GetProcAddress(hDll, "divid");
                if (!Div) printf("unable to load DivFunc\n");
                else printf("%f / %f = %f\n", a, b, Div(a, b));
        break;
        }
        default:{
                printf("invalid operation\n");
        }
        }
    }

        FreeLibrary(hDll);
}
```

MAKEFILE

```
use.exe: use.obj diff.obj div.obj mult.obj sum.obj
        LINK /OUT:USE.EXE use.obj diff.obj div.obj mult.obj sum.obj

diff.obj : diff.c aritme.h
 CL -c diff.c

div.obj : div.c aritme.h
 CL -c div.c

mult.obj : mult.c aritme.h
 CL -c mult.c

sum.obj : sum.c aritme.h
 CL -c sum.c

clean:
        del use.obj diff.obj div.obj mult.obj sum.obj use.exe
```

esercizio

Vediamo in Windows come scrivere un semplice programma che utiizzi le librerie statiche di Windows (Win32 API)ad esempio per produrre una finestra di dialogo modale xon un messaggio. Cercando su internet l'help della funzione all'url ufficiale: https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-messagebox si nota che bisogna includere windows.h e utilizzare la libreria statica: User32.lib oppure la libreria dinamica User32.dll.

```
#include <windows.h>
int main(){
        MessageBox(NULL,"ciao come va?","messaggio",MB_OK|MB_ICONASTERISK);
        return 0;
}
```

Il comando di compilazione sara'quindi:?

UTILIZZO DELLA LIBRERIA DINAMICA IN PYTHON

Utilizziamo ad esempio la libreria dll appena creata in Windows, ma per Linux e' molto simile:

Innanzitutto controllare che l'architettura (32 o 64) di Python sia la stessa della dll altrimenti Python stamperebbe il seguente messaggio di errore:

WindowsError: [Error 193] %1 non è un'applicazione di Win32 valida

mydll.py

```
import ctypes
mydll = ctypes.cdll.LoadLibrary("aritm.dll")

print mydll.somma
mydll.somma.argtypes = [ctypes.c_float, ctypes.c_float]
mydll.somma.restype = ctypes.c_float

print(mydll.somma(3.2,5.1))
```

Output

```
Library loaded
<_FuncPtr object at 0x0000000003838528>
8.30000019073
Process detached
```