6. Laborator

DLL (Dynamic Link Library)

6.1. Scopul laboratorului

Acest laborator are un singur scop, acela de a dobândi deprinderile practice în crearea, implementarea, dezvoltarea și utilizarea de DLL-uri.

6.2. Cerințe

1. Dezvoltați un program care să aibă o interfață grafică de tip *dialog*.
2. Pe interfața grafică veți avea 3 elemente de tip *Edit Control* și 2 butoane.
3. În două dintre elementele de tip *Edit Control* se pot introduce numere în virgulă mobilă, în cel de al treilea element veți afișa rezultatul obținut în directă dependență de butonul care a fost apăsat de utilizator;
4. În cadrul funcției asociate primului buton veți realiza următorul calcul numeric $10 * a + b$ iar în cadrul funcției asociate cu cel de al doilea buton veți calcula $a + 10 * b$ (a și b sunt valorile preluate de la cele 2 elementele de tip *Edit Control* menționate anterior). Relațiile prezentate mai sus vor fi implementate în două DLL-uri distincte.
5. Realizați exportarea acestor funcții prin 2 metode diferite.
6. Utilizați placa care este instalată pe sistemul dumneavoastră în testarea proiectelor (EVM3530, BeagleBoard sau eBox).

6.3. Noțiuni introductive DLL-uri

Un DLL (*Dynamic Link Library*) este un fișier binar, care este o colecție de: funcții, date și/sau alte tipuri de resurse (de exemplu o imagine). Un alt program, proces sau fir de execuție (creat în C++, Visual Basic, Delphi sau în orice alt limbaj), poate apela funcții care sunt interne unui DLL. Nu toate funcțiile interne unui DLL pot fi apelate din exterior. Funcțiile interne care pot fi lansate în execuție din exteriorul DLL-ului sunt doar cele exportate. De asemenea mai multe procese sau fire de execuție, ce rulează simultan, pot utiliza aceste funcții interne unui singur DLL.

Formatul intern al unui fișier DLL este similar cu al unui fișier de tip EXE. Prin această modalitate de utilizare a funcționalităților interne existente într-un DLL către exterior, se oferă posibilitatea ca un program sau sistem de operare să-i fie corectate greșelile sau să aibă funcționalități noi prin înlocuirea DLL-urilor existente cu noi versiuni. Astfel, programul respectiv nu mai trebuie să fie recompilat și *link-*

editat și apoi reinstalat. De altfel, întreg SO Windows (indiferent de versiunea lui) este o sumă de diferite fișiere de tip DLL.

6.4. Pași necesari în crearea unui DLL

În continuare vom arăta etapele necesare creării unui DLL în mediul de dezvoltare Visual Studio utilizând limbajul C++. Etapele necesare dezvoltării unui DLL sunt următoarele:

I. Urmați pașii cunoscuți deja: **File** → **New** → **Project …**;
II. Din noua fereastra deschisa executați pașii:
    1. *Name*: **Test** (sau orice alt nume doriți dvs.);
    2. *Location*: calea pe discul **D:** unde lucrați de obicei;
    3. Avem in continuare posibilitatea alegerii a doua tipuri distincte de DLL-uri:
        a. **Win32** (prin opțiunea *Smart Device* → *Win32 Smart Device Project* – veţi avea posibilitatea de a selecta tipul fişierului, fişier DLL-în cazul nostru, dar această selecţie se va realiza puţin mai târziu în unul din meniurile ce vor urma) este o abordare care dă posibilitatea ca funcţiile interne DLL-ul să nu utilizeze sau să utilizeze biblioteca MFC. În această situaţie funcţiile exportate (cele care pot fi apelate din exteriorul fişierului DLL) pot fi apelate de executabile ce sunt sau nu de tipul MFC.
        b. **MFC** (prin opțiunea MFC Smart Device DLL).
    4. Vom alege opțiunea **Win32 Smart Device Project**;
    5. Apăsați butonul **OK**;
III. Din noua fereastra deschisa executați următoarea secvență de comenzi:
    1. Apăsați butonul **OK** urmat de **Next**;
    2. În cadrul opțiunii **Selected SDKs** alegeti SDK-ul specific plăcii dvs.
    3. Apăsați butonul **Next**;
    4. Selectați opțiunea **DLL** (din cadrul secţiunii **Application type**), tot aici prin selectarea sau nu a opțiunii MFC (din cadrul secţiunii **Add Suport For:**) se poate alege includerea sau nu a suportului pentru MFC. Noi nu vom alege suport MFC;
    5. In final apăsați butonul **Finish**;

Orice funcţie internă unui fişier DLL este similară cu orice altă funcţie care este utilizată în cadrul unui program scris în C respectând aceleaşi reguli de declarare a variabilelor, a tipurilor de date ce pot fi utilizate, a modalităţii de întoarcere a diferitelor valori, etc.

După cum am menționat anterior, nu toate funcțiile existente în interiorul DLL-ului pot fi utilizate de alte fire de execuție – altele decât cele lansate din interiorul DLL-ului. Pentru a fi utilizate aceste funcții trebuie ca mai întâi să fie exportate. Există două modalități de a exporta una sau mai multe funcții din interiorul unui DLL. Astfel (aceste acțiuni vor fi realizate în proiectul în care dezvoltăm DLL-ul):

1. Prin utilizarea declaratorului __declspec(dllexport) în fața funcției/funcțiilor ce vor fi exportate.
2. Prin crearea unui fișier de tipul **Module-Definition File** (.DEF).

Pentru a crea un fișier de tipul ".DEF" se urmăresc etapele:

a. Se dă click dreapta pe numele proiectului, în fereastra Solution Explorer şi se selectează **Add | New Item;**
b. Se creează un fişier de tipul Code | **Module Definitions File** cu numele exports.def sau cu orice alt nume se dorește;
c. Se include în cadrul fişierul ".def" o linie de tipul:

```
EXPORTS fctMat
```

Unde fctMat este numele funcţiei care implementează una din relaţiile matematice anterioare. În cazul în care se exportă mai multe funcţii din interiorul DLL-ului fiecare din acestea va avea o linie corespondentă de tipul de mai sus în fişierul ".def";
d. Se verifică corectitudinea includerii acestui fişier prin: se dă *click* dreapta pe numele proiectului, în fereastra Solution Explorer şi se selectează Properties;
e. În pagina Property pe calea **Linker | Input** verificaţi că în câmpul M**odule Definition File** este un fişier cu extensia .def ce are un nume similar cu cel creat anterior.

De asemenea, fiecare DLL mai are o funcţie predefinită: DllMain. Această funcţie, internă fiecărui DDL, este executată întotdeauna când procesul sau firul de execuţie apelant se lansează sau îşi termină execuţia. De asemenea, această funcţie se execută atunci când DLL este încărcat sau descărcat prin intermediul funcţiilor LoadLibrary() şi FreeLibrary().

Există două metode de *link*-editare a unui DLL în cadrul proiectului nostru (DLL creat de noi sau obținut dintr-o sursă terță): implicită și explicita. Astfel:

1. **Link-editarea implicita (realizată între codul programului dezvoltat și cel al DLL pe care dorim sa-l utilizăm):**
   a. Pentru a realiza această *link*-editare avem nevoie de:
      - Un fişier header, *.h, ce va conține declarațiile funcțiilor sau claselor exportate din interiorul DLL-ului - de aici apare şi necesitatea creării unui astfel de fişier;
      - Un fişier de tip librărie, *.LIB – acest fişier va fi creat în momentul în care DLL este construit:
      - Fişierul DLL "*.dll" rezultat în urma compilării: **Build → Build Solutions** sau prin apăsarea combinaţiei Ctrl + Shift + B.

      Ultimele 2 fişiere le veţi găsi în directoarele **Build** sau **Release** în cadrul proiectului creat la început (funcţie de opţiunea **Solution Configuration** aleasă);
   b. Se *link*-editează proiectul cu fişierul ".lib" pus in directorul de lucru al programului. Atunci când vom rula programul fişierul ".dll" trebuie pus în directorul de lucru al programului sau într-un director aflat în căile de căutare implicită;
   c. *Link*-editatare se poate realiza prin:
      - Includerea #pragma comment(lib, "DLLTutorial.lib") la începutul fişierului în C sau C++ în cadrul căruia apelăm funcția/funcțiile interne DLL-ului – unde DLLTutorial este numele lib-ului pe care îl puneți în directorul programului;
      - Sau din *project setings*.
   d. De asemenea deoarece apelăm o funcție a cărui cod nu există în proiectul nostru (se află în interiorul unui DLL) la începutul fişierului în care o apelăm trebuie să includem fişierul *header* (ceva gen: #include "DLLTutorial.h" – unde DLLTutorial.h este fişier header

asociat DLL-ului, vezi subpunctul "*a*" anterior) ce conține declarația funcției/funcțiilor ce sunt interne DLL-ului.

2. **Link-editarea explicită:**
   a. Aveți nevoie doar de fișierul DLL și de *header*-ul asociat cu acesta.
   b. Pașii ce trebuie executați:
      - Apelați funcția `LoadLibrary()` pentru a încărca DLL-ul și pentru a obține un identificator unic al lui (*handle*)
      - Apelați funcția `GetProcAddress()` pentru a obține un pointer pe o anumită funcție exportată
      - Apelați funcția `FreeLibrary()` atunci când nu veți mai apela o altă funcție internă DLL-ului, acesta nu mai este utilizat și ocupă resurse în mod inutil.

În final, după parcurgerea tuturor etapelor necesare realizării tuturor cerințelor impuse în partea de început a laboratorului veți avea un număr de 3 fișiere. Primul dintre fișiere este aplicația, realizată cu suportul MFC-ului, care prin intermediul interfeței grafice ne ajută să realizăm cele două operații matematice propuse inițial (introducerea datelor, obținerea și afișarea rezultatului pe GUI). Celelalte două fișiere sunt cele două DLL-uri de suport în care sunt implementate cele două funcții matematice.

Atenţie! În momentul în care testați funcționalitatea finală a programului aveți grijă să puneți în directorul asociat aplicației (directorul ce se găsește în folderul ***Program Files*** de pe placa de dezvoltare) și cele 2 DLL-uri. Transferați-le pe aceste 2 DLL cu ajutorul unui USB *stick* de pe calculatorul *host* pe sistemul de dezvoltare. Mediul de dezvoltare Visual Studio nu va face această operaţie pentru dumneavoastră.

6.5. Temă

1. În interiorul oricărui DLL, din cele 2 existente, realizaţi operaţia de înmulţire cu 10 prin intermediul unei funcţii interne unui alt DLL independent de cele 2 dezvoltate anterior.
2. Apelaţi funcţiile interne celor 2 DLL-uri prin intermediul unei *link*-editări explicite.