# 24. Tétel

Mintapéldán keresztül mutassa be egy egyszerű WinAPI ablakozó alkalmazás felépítését!

## Embedded Windows

### Fajtái

* Windows 7, 8 Embedded Standard
  + PC alapú eszközökre
  + ATM
  + Játékgépek
  + Pénztárgépek
  + POS terminálok
* Windows Embedded Compact – ami CE ☺
  + PDA-k
  + GPS-ek
  + Ipari automatizálási eszközök
  + valósidejű, komponens orientált, multitaskosz OS
  + preemptív ütemezés
  + felépítés:
    - HW
    - Kernel (Nk.exe)
      * OAM – OEM Abstraction Layer
      * kernel.dll
    - Win32 CE API (felhasználói réteg)

### fejlesztés

* + verziótól függ
  + natív alkalmazások
    - Win32 CE API-cal lehet fordítani bármilyen kódot szinte (C, C++, VB, VBA)
  + .NET Compact Framework (MC++, C#, VB.NET)

### Saját HW-re illesztés

* + esközfüggő rendszerkompoensekből álló csomagra van szükség
  + BSP – Boarad Support Package
    - booloader
    - OAL
  + driverek

## Win32 CE API

### API – Application Programming Interface

* + a rendszert alkotó dll-ek publikus függvényei
  + az alapfunkciók a Coredll.dll-ben avnnak
  + kiegészítő funkciók külön dll-ekben
  + konvenciók: Hungarian Notation

### Változótípusok: a nyelvfüggetlenség érdekében saját típusokat definiál

* + UINT (unsigned int)
  + LONG (long)
  + WPARAM (word parameter (UINT))
  + LPARAM (long parameter (LONG))
  + HANDLE
  + LRESULT
  + HWND
  + WORD
  + DWORD

### Ütemezés:

* + processzek – szálak
  + egy processzhez tartozó összes szál ugyanazt a virtuális címtartományt használja 🡪 a szálak elérik egymás adatait
  + időszelet – quantum – 100 ms, de ez állítható szálanként
  + 256 prioritási szint – default: 215

### Szinkronizációs objektumok

* + Kritikus szakasz – műveletek atomivá tétele
  + Mutex – kölcsönös kizárás erőforrások védelmére konkurens hozzáférés ellen
  + Szemafor – erőforrásokhoz való hozzáférés szabályozása
  + Esemény – események bekövetkezésének várása, és ezek jelzése
  + Várakozási sor – szálak közöti szinkronizált adattovábbítás

## Fejlesztés Windows CE-re

Asztali PC-s verziótól csak kicsit tér el

### Ablakozó alkalmazás felépítése API-val

* natív módon futó alkalmazások
  + az OS API-ját felhasználva készítünk közvetlenül futtatható alkalmazásokat
  + nem OOP, mert az OS API-ja sem az (leginkább C)
  + MFC, ATL, STL
    - olyan kiegszítő szolgáltatások, amelyek interfészelik az OOP progit az API-hoz
    - C++
    - VBA
* entry point: WinMain
* az applikáció létrehoz néhány ablakot
  + minden ablakhoz tartozik egy ablakkezelő fv.: CALLBACK fv.
  + nem a mi programunk hívja, hanem a Windows
* egy tipikus Windows program részei:
  + Init
    - ablak osztály regisztrálása
    - változók inicializállása
  + főablak létrehozása
  + üzenetkezelő ciklus
    - kiolvassa a várakozási sorból az üzeneteket, és továbbítja a Windowsnnak
    - a Windows ezekkel a paraméterekkel hívja meg az ablakkezelő f-t
  + főablak kezelő fv-e (Window procedure)
    - üzenetekre való reagálás
  + Kilépés előtti takarítás 🡪 kilépés

### Menedzselt alkalmazásfejlesztés CE-re

* Windows API-ban a fejlesztés bonyolult
* szoftverfejlesztő cégek: az idő pénz 🡪 támogatják a menedzselt alkalmazásfejlesztést
* natív fejlesztés: teljes mértékben ki tudjuk használni az OS API-ja által nyújtot lehetőségeket
* menedzselt fejlesztés (.NET)
  + kell futtató környezet (CLR – Common Language Runtime)
  + köztes nyelv:
    - MSIL – Microsoft Intermediate Language
    - a fejlesztés alatt csak egy köztes kódot használunk
    - a tényleges fordítás
      * telepítéskor
      * első futtatáskor – JIT – Just in Time
    - ☺ - némileg platformfüggetlen
* .NET-es alkalmazások minimálisan maradnak el a WinAPI-soktól
* nyilvánvalóan csak felhasználói programok születhetnek (OAL, service-k csak natív módban)

#include ….

// globális változók deklarálása

WinMain(…) {

// lokális változók deklarálása

InitProgram();

// ablakosztály regisztrálása

WNDCLASSEX wcex;

…

wcex.lpfnWndProc = WndProc;

RegisterClassEx(&wcex);

// ablak létrehozása

hWnd = CreateWindow(…);

// ablak megjelenítése

ShowWindow(…);

UpdateWindow(hWnd);

// Üzenetkezelő ciklus

while (GetMessage(&msg, NULL, NULL, NULL)) {

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return msg.wParam

}

// ABLAKKEZELŐ FV.

LONG CALLBACK WndProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam) {

// lokális változók deklarálása

// egy switch-case szerkezetben megvizsgáljuk milyen üzenetet

// kapott a függvény paraméterként és az üzenettől függően csinálunk

// vmit

switch (msg) {

case WM\_CREATE:

...

return 0;

case WM\_PAINT:

DrawSomething();

return 0;

case WM\_COMMAND:

switch (wParam) {

case IDM\_FILENEW:

// pl. egy menüelem azonosító

DoSomething();

break;

}

return 0;

case WM\_DESTROY:

FreeEverything(); // pl. memória felszabadítása

PostQuitMessage(0);

return 0;

}

return DefWindowProc(hwnd, msg, wParam, lParam);