Universitatea Politehnică Bucureşti

Facultatea de Electronică, Telecomunicaţii şi Tehnologia Informaţiei

**Sisteme de Operare**

Funcţiile principale Win32 API

**Profesor coordonator:** prof. univ. dr. ing. Ştefan Stăncescu

**Student:** Tudor Ștefan-Eugen

**Grupa:** 433A (CTI)

Anul universitar

2014-2015

**Cuprins**

1.Introducere.................................................................................2

2.Funcții specifice........................................................................4

3.Compatibilitatea cu compilatoarele...........................................5

4.Funcționarea unui program Windows.......................................6

5.Concluzii.................................................................................10

6.Bibliografie..............................................................................11

**1.Introducere**

În mod normal, un program Windows este bazat pe evenimente. Programul principal așteaptă ca un eveniment să se întample, iar apoi apelează o procedură care să se ocupe de el(handler). Astfel de evenimente pot fi apăsarea tastelor, mișcarea mouse-ului sau apăsarea unui buton al mouse-ului.

Handler-ul, procedura care este apelată ca urmare a apariției evenimentelor, actualizează ecranul și starea internă a programului.

La sistemul de operare Windows, apelarea bibliotecilor este decuplată de apelarea de sistem. Microsoft a creat un set de proceduri numite Win32 Application Program Interface – pe scurt, Win32 API – care sunt folosite pentru a folosi anumite servicii de sistem. Această interfața este compatibilă, parțial la versiunile mai vechi, cu toate sistemele de operare Windows începând cu Windows 95. Decuplarea dintre apelul bibliotecilor și apelul de sistem permite schimbarea apelurilor de sistem în timp făra a face programele existente inutilizabile.

Win32 API se bazează în primul rând pe limbajul de programare C, ceea ce înseamnă ca funcțiile și structurile de date sunt descrise în acest limbaj în versiunile recente ale documentației. Totuși, nu este neaparată nevoie de un compilator C, Win32 API putând fi folosit de orice compilator sau asembler compatibil cu structurile de date de nivel mic.

 Numărul de apeluri de Win32 API este foarte mare, în număr de mii. Chiar dacă multe astfel de apeluri sunt de sistem, un număr substanțial de mare sunt din spațiul utilizatorului. Ca o consecință, în Windows este imposibil de spus care este un apel de sistem(efectuat de kernel) și care este un simplu apel de bibliotecă din spațiul utilizatorului. Ținănd cont de actualizarile suferite de Windows, ceea ce la o versiune este un apel de sistem, la altă versiune poate fi un apel din spațiul utilizatorului. [[1]](#footnote-2)

*Figura 1: Arhitectura Windows[[2]](#footnote-3)*



*Figura 2: Straturi ale Win32 API [[3]](#footnote-4)*



Se observă împarțirea dintre modul Utilizator și modul Kernel:

* modul utilizator conține: aplicațiile, DLL-urile sistemului Windows, Ntdll.dll.
* modul Kernel conține: apelurile de sistem, driverele de sistem.

Aceste două moduri comunică printr-o interfața numită System-Call. Această interfață asigură posibilitatea unui program de a cere accesul la un serviciu Kernel-ului unui sistem de operare.

*Figura 3:Compatibilitatea programelor bazate pe Win32 la diferite versiuni de Windows[[4]](#footnote-5)*



**2.Funcții specifice**

Funcționalitatea dată de Windows API poate fi grupată în 8 categorii:

* Servicii de bază (Base Services) – asigură acces la resursele fundamentale disponibile unui sistem Windows, unde sunt incluse: fișiere, procese, fire de execuție, rezolvarea erorilor - aceste funcții se găsesc în kernel.exe, krnl286.exe sau krnl386(pentru Windows 16-bit) și în kernel32.dll (pentru Windows 32-bit).
* Servicii avansate (Advanced Services) – asigură acces la funcționalități adiționale kernel-ului, ca: registrii Windows, închiderea/repornirea sistemului, pornirea/oprirea/creerea unui serviciu Windows, administrarea conturilor utilizatorilor – aceste funcții se găsesc în advapi32.dll (pentru Windows 32-bit).
* Interfața grafică (Graphics Device Interface) – asigură funcționalitate pentru ieșirea de continut vizual la monitoare, imprimante, pe lânga alte dispozitive de ieșire – se găsesc în gdi.exe (Windows 16-bit) sau gdi32.dll (Windows 32-bit) pentru modul utilizator. Pentru modul kernel se găsesc în win32k.sys, care comunică direct cu driver-ul grafic.
* Interfața utilizatorului (User Interface) – asigură funcționalități cu ajutorul cărora se pot crea și administra ferestre și majoritatea controlului de bază, cum ar fi: butoanele și bara de defilare, primirea de informație de intrare de la mouse și tastatură – se găsește în user.exe(Windows 16-bit) și user32.dll(Windows 32-bit). Începând cu Windows XP, comenzile de bază se găsesc în comctl32.dll, împreună cu comenzile comune.
* Biblioteca funcțiilor de dialog în fereastră (Common Dialog Box Library) – asigură aplicațiilor accesul la ferestrele standard de dialog pentru a deschide și salva fișiere, pentru alegerea de culoare și font – biblioteca se găsește în commdlg.dll(Windows 16-bit) și în comdlg32.dll(Windows 32-bit) – intră în categoria de Interfata Utilizatorului în cadrul API-ului.
* Biblioteca de comenzi comune (Common Control Library) – asigură aplicațiilor accesul la funcții avansate de control puse la dispoziție de sistemul de operare – sunt incluse: bara de stare, bara de progres, bara de unelte etc. Aceste biblioteci se găsesc într-un fișier dll numit commctrl.dll(Windows 16-bit) și comctl32(Windows 32-bit) – intră în categoria de Interfața Utilizatorului în cadrul API-ului.
* Windows Shell – asigură aplicațiilor funcționalitatea oferită de shell-ul sistemului de operare, cât și posibilitatea de a îl modifica și îmbunătăți – componentele se găsesc în shell.dll(Windows 16-bit) și shell32.dll(Windows 32-bit). Funcțiile Shell Lightweight se găsesc în shlwapi.dll și intră în categoria de Interfața Utilizatorului în cadrul API-ului.
* Servicii de rețea (Network Services) – oferă acces la numeroasele funcții de rețea ale sistemului de operare – se găsește în netapi32.dll (Windows 32-bit). [[5]](#footnote-6)

**3.Compatibilitatea cu compilatoarele**

Pentru a dezvolta aplicații care folosesc Windows API, un compilator trebuie să poată folosi DLL-urile (Dynamic Link Library) specifice Microsoft. Compilatorul trebuie de asemenea să se ocupe de fișierele header care expun numele intern al funcțiilor API, sau să furnizeze el însuși asemenea fișiere.

Pentru anumite aplicații, compilatorul trebuie să poată folosi fișiere IDL(Interface Definition Language). Puse la un loc, aceste necesități(compilatoarele, uneltele, bibliotecile și fișierele header) sunt cunoscute ca Kitul de dezvoltare software Microsoft(Microsoft Platform SDK).

Suport special pentru compilatoare Windows este necesar și pentru mecanismul SEH(Structured Exception Handling). Acest sistem are două funcționalități: prin intermediul lui kernel-ul notifică aplicația de anumite condiții excepționale, cum ar fi umplerea stivei și asigură un substrat pe care pot fi implementate anumite metode de manipulare a excepțiilor.

SEH se bazează pe introducerea manipulatorilor de excepții în stiva, apoi adaugarea acestora într-o listă legată(linked list) stocată în memoria locală a firului de execuție. Când o excepție este aruncată, kernelul și bibliotecile de bază derulează stivă care rulează manipulatori și îi filtrează pe măsură ce îi descoperă. În final, orice excepție care nu este tratată de aplicație în sine, va fi rezolvată prin manipulatorul standard de tratare a excepțiilor – backstop handler. Acesta produce apariția unei ferestre dialog de tip eroare.

 În ziua de azi, MinGW și Cygwin oferă un asemenea mediu bazat pe GNU Compiler Collection, folosind o colecție de fișiere header de sine stătatoare pentru a face simplă legătura cu DLL-urile Win32. [[6]](#footnote-7)

*Figura 4: Crearea unui program folosind un compilator standard Win32 [[7]](#footnote-8)*



**4.Funcționarea unui program Windows**

Când o fereastră este creată, parametrii specifică dacă aceasta poate fi mutată, redimensionată, sau dacă poate fi derulată de către utilizator. Fereastră principala a majorității programelor poate fi mutată, redimensionată, faptă care are consecințe enorme în scrierea programelor Windows. În privat, programele trebuie sa fie anunțate de schimbări în dimensiuni ale ferestrelor și trebuie să fie pregătite să redeseneze conținutul în orice moment al rulării.

 Prin urmare, programele Windows sunt orientate pe mesaje. Acțiunile utilizatorilor care includ tastatura și mouse-ul sunt reținute de Windows și convertite în mesaje programului care deține fereastra adresată. Fiecare program are o coadă în care sunt trimise mesajele care se referă la ferestrele lui. Bucla principală a programului constă în scoaterea urmatorului mesaj din coadă și procesarea acestuia prin apelarea unei proceduri interne pentru tipul respectiv de mesaj. În unele cazuri, aceste proceduri pot fi apelate de însuși Windows, trecând peste coada de mesaje.

 În exemplul de mai jos, este prezentată structura scheletului unui program principal pentru Windows.

 Programul începe prin includerea unui fișier header, care conține macro-uri(text care conține o secvență de apeluri sau rutine), tipuri de date, constanțe, prototipuri de funcții și alte informații necesare de programele Windows.

 Urmează declararea numelui și parametrilor. Macro-ul WINAPI este o instrucțiune adresată compilatorului, făcându-l pe acesta să folosească un anumit parametru. Primul parametru, h, este o instanța de manipulator(handler) care este folosită pentru a identifica programul în cadrul întregului sistem. În mare parte, Win32 este orientat pe obiecte, ceea ce înseamnă că sistemul conține obiecte(programe, fișiere, ferestre) care au stare și cod asociat, numite metode, care operează în acea stare. Obiectele folosesc manipulatori, iar în acest caz, h identifică programul. Al doilea parametru este prezent numai pentru compatibilitatea cu versiunile anterioare. Al treilea parametru, szCmd, este un șir de caractere care conține linia de comandă care a pornit programul, chiar daca nu a fost pornit dintr-o linie de comandă. Al patrulea parametru, iCmdShow, specifică daca fereastra inițială ar trebui să ocupe întregul ecran, doar o parte din ecran sau nicio parte din ecran(doar în bara de procese).

 Fiecare fereastră trebuie sa aibă asociată o clasă obiect care sa ii definească proprietățile. În cazul nostru, acea clasă este wndclass. Un obiect de tipul WNDCLASS are 10 toate fiind inițializate într-un program normal. Cel mai important câmp este IpfnWndProc, care este un pointer pe 32 de biti către funcția care se ocupă de mesajele direcționate către fereastră.

 După inițializarea wndclass, RegisterClass este apelată pentru a o trece sistemului. Următoarea apelare, CreateWindow, alocă memorie pentru structurile de date ale ferestrei și returnează un manipulator pentru a îl referenția mai tarziu. Programul mai face două apeluri, pentru a pune conturul ferestrei pe ecran și pentru a o umple complet.

 Bucla principală a programului constă în primirea unui mesaj, efectuarea anumitor operații asupra lui, iar apoi trimiterea acestuia înapoi la Windows. [[8]](#footnote-9)

*Figura 5: Exemplu de schelet al unui program main*

#include <windows.h>

int WINAP WinMain(HINSTANCE h, HINSTANCE, hprev, char \*szCmd, int iCmdShow)

{

 WNDCLASS wndclass; /\* class object for this window \*/

 MSG msg; /\* incoming messages are stored here \*/

HWND hwnd; /\* handle (pointer) to the window object \*/

/\* Initialize wndclass\*/

wndclass.lpfnWndProc = WndProc; /\* tells which procedure to call \*/

wndclass.lpszClassName = ”Program name”; /\* text to the title bar \*/

wndclass.hIcon = LoadIcon(NULL, IDI\_APPLICATION); /\* load program icon \*/

wndclass.hCursor = LoadCursor(NULL, IDC\_ARROW); /\* load mouse cursor \*/

RegisterClass(&wndclass); /\* tell Windows about wndclass \*/

hwnd = CreateWindow(...) /\* allocate storage for the window \*/

ShowWindow(hwnd, iCmdShow); /\* display the window on the screen \*/

UpdateWindow(hwnd); /\* tell the window to paint itself \*/

while(GetMessage(&msg, NULL, 0, 0)){ /\* get message from queue \*/

 TranslateMessage(&msg); /\* translate the message \*/

 DispatchMessage(&msg); /\* send msg to the appropiate procedure \*/

}

return(msg.wParam);

}

long CALLBACK WndProc(HWND hwnd, UINT message, UINT wParam, long lParam)

{

 /\* Declarations go here. \*/

 switch(message){

 case WM\_CREATE: ...; return ...; /\* create window \*/

 case WM\_PAINT: ...; return ...; /\* create window \*/

 case WM\_DESTROY: ...; return ...; /\* create window \*/

}

return(DefWindowProc(hwnd, message, wParam, lParam)); /\* default \*/

}

Dupa programul principal urmează procedura WndProc, care se ocupă de diferite mesaje ce pot fi trimise ferestrei. CALLBACK este folosit pentru a specifica secvența de apelare pentru parametrii. Primul parametru este manipulatorul pe care sa îl folosească fereastra. Al doilea parametru este tipul mesajului. Al treilea și al patrulea parametru pot fi folosiți pentru a aduce informații suplimentare în cazul în care este nevoie.

WM\_CREATE si WM\_DESTROY sunt tipuri de mesaje care sunt trimise la începutul, respectiv sfârșitul, unui program. Dau posibilitatea programului să aloce memorie pentru structuri de date iar apoi să le returneze.

Al treilea tip de mesaj, WM\_PAINT, este o instrucțiune adresată programului, ca acesta sa umple fereastră. Nu este apelată numai la început, când este desenată fereastra, ci și pe întreg parcursul execuției programului. Spre deosebire de sistemele de operare bazate pe text, în Windows, un program nu poate presupune ca orice desenează pe ecran va rămâne acolo până când este șters. Alte ferestre pot fi trase peste aceasta, meniurile pot fi trase în jos peste fereastră, ferestrele dialog și sfaturile pentru unelte o pot acoperi parțial. Când sunt înlăturate, fereastra care a fost acoperită trebuie să fie redesenată. Windows transmite unui program nevoia de a redesena o fereastră printr-un mesaj de tip WM\_PAINT. Se transmit de asemenea și informații cu privire la ce parte a ferestrei a fost acoperită, în caz că este mai ușor de redesenat acea parte a ferestrei, față de a redesena întreagă fereastră.

Windows îi poate transmite unui program o instrucțiune prin două modalități: prima este plasarea unui mesaj în coada de mesaje a acestuia. Această metodă este folosită pentru date de intrare de la tastatură, mouse sau de la cronometre care au expirat. A două metodă presupune apelarea directa de către Windows a WndProc. Această metodă este folosită pentru restul de evenimente. Din moment ce Windows este notificat când un mesaj este procesat în întregime, poate aștepta transmiterea unui nou apel până când cel actual este terminat.

În cazul în care apare un mesaj neașteptat și pentru a evita un comportament întâmplător, programul ar trebui să apeleze DefWindowProc la sfârșitul lui WndProc pentru a lăsa manipulatorul standard să rezolve aceste cazuri.[[9]](#footnote-10)

De desenatul pe ecran se ocupă un pachet care conține sute de proceduri puse la un loc pentru a forma GDI (Graphics Device Interface). Acesta se poate ocupa de text și de toate tipurile de grafică și este conceput astfel încât sa fie independent de platforma și dispozitiv. Înainte ca un program să deseneze într-o fereastră, trebuie să capete contextul de dispozitiv, care este o structură de date internă care conține atribute ale ferestrei, cum ar fi fontul curent, culoarea textului, culoarea de background etc. Majoritatea apelurilor GDI folosesc contextul de dispozitiv, fie pentru a desena sau pentru a seta sau a lua proprietățile.

Există numeroase posibilități pentru a căpăta contextul dispozitivului. Un exemplu este:

Exemplu de preluare a contextului de dispozitiv:

hdc = GetDC(hwnd);

TextOut(hdc, x, y, psText, iLength);

ReleaseDC(hwnd, hdc);

Prima afirmație preia un manipulator pentru un context de dispozitiv, hdc. A doua afirmație folosește contextul dispozitivului pentru a scrie o linie de text pe ecran, specificând coordonatele (x, y) de unde începe sirul, un pointer la șir și lungimea acestuia. Al treilea apel decuplează contextul dispozitivului pentru a arăta că programul a terminat de desenat pentru moment. De menționat este că ReleaseDC conține informații redundante (folosirea lui hdc specifică în mod unic o fereastră). Folosirea de informație redundantă care nu are valoare efectivă este întâlnită în mod comun la Windows.

Pe scurt, un program Windows construiește, în mod normal, una sau mai multe ferestre cu o clasă obiect pentru fiecare. Asociată fiecărui program este o coadă de mesaj și un set de proceduri de manipulare. Comportamentul programului este determinat de evenimentele pe care le întâmpină, care sunt procesate de procedurile de manipulare. [[10]](#footnote-11)

**5.Concluzii**

Win32 pune la dispoziție interfețe cuprinzătoare cu mulți parametrii, de obicei cu trei sau patru moduri de realizare a aceluiași lucru. Asta înseamnă că Win32, deși pune la dispoziție un set de interfețe foarte bogat, introduce și o mare complexitate cauzată de stratificarea slabă, care amestecă funcțiile de nivel înalt cu funcțiile de nivel jos.

Win32 are apeluri pentru a crea și a administra atât procese cat și fire de execuție. Există de asemenea și numeroase apeluri care au ca scop comunicarea interproces, cum ar fi crearea, distrugerea și folosirea mutexe-urilor (concept de programare folosit pentru rezolvarea problemelor de multithreading), semafoare, evenimente, porturi de comunicație și alte obiecte IPC. [[11]](#footnote-12)

**6.Bibliografie**

1. <https://msdn.microsoft.com/en-us/library/172wfck9.aspx>
2. <http://en.wikipedia.org/wiki/Windows_API>
3. <https://msdn.microsoft.com>
4. <https://windowskernal.wordpress.com/>
5. <http://windowsitpro.com/>
6. http://www.testech-elect.com/ontime/rtt32.htm
7. Andrew S. Tanenbaum Modern Operating Systems, 3rd Edition 2008
8. Andrew S. Tanenbaum, Herbert Boss Modern Operating Systems, 4th Edition 2014
1. Introducere Win32 API - Andrew S. Tanenbaum Modern Operating Systems, 3rd Edition 2008, pag 58 [↑](#footnote-ref-2)
2. Figura 1 - https://windowskernal.wordpress.com/2011/08/21/windows-kernal [↑](#footnote-ref-3)
3. Figura 2 - http://windowsitpro.com/site-files/windowsitpro.com/files/archive/windowsitpro.com/content/content/46266/figure\_02.gif [↑](#footnote-ref-4)
4. Figura 3 - Andrew S. Tanenbaum Modern Operating Systems, 3rd Edition 2008, pag 813 [↑](#footnote-ref-5)
5. http://en.wikipedia.org/wiki/Windows\_API [↑](#footnote-ref-6)
6. Compatibilitatea Win32 cu diferite compilatoare - http://en.wikipedia.org/wiki/Windows\_API [↑](#footnote-ref-7)
7. Figura 4 - http://www.testech-elect.com/ontime/rtt32.htm [↑](#footnote-ref-8)
8. Functionarea unui program main Win32 - Andrew S. Tanenbaum Modern Operating Systems, 3rd Edition 2008, pag 405, 406, 407, 408 [↑](#footnote-ref-9)
9. Functionarea unui program main Win32 - Andrew S. Tanenbaum Modern Operating Systems, 3rd Edition 2008, pag 408 [↑](#footnote-ref-10)
10. Functionarea unui program Windows - Andrew S. Tanenbaum, Herbert Boss Modern Operating Systems, 4th Edition 2014, pag 411, Andrew S. Tanenbaum Modern Operating Systems, 3rd Edition 2008 pag 409 [↑](#footnote-ref-11)
11. Concluzii despre Win32 - Andrew S. Tanenbaum, Herbert Boss Modern Operating Systems, 4th Edition 2014, pag 872, 873 [↑](#footnote-ref-12)