Universitatea Politehnica București

Facultatea de Electronică, Telecomunicații și Tehnologia Informației

**Accesul la resurse SO WIN32**

Grupa: 431A

**Pâsoi Mihai**

**Jircu Andrei**

**Aflorei Victor**

Coordonator științific:

Conf. Dr. Ing. Ștefan Stăncescu

Cuprins

[1.Introducere in programarea in Windows (Pâsoi Mihai 431A) 3](#_Toc357455711)

[1.1.1 Programarea in Win32 3](#_Toc357455712)

[1.1.2 Deosebiri între programele DOS și Windows. 3](#_Toc357455713)

[1.2 Componentele Windows 4](#_Toc357455714)

[1.3 Identificatori Windows 4](#_Toc357455715)

[1.4 Funcţii 5](#_Toc357455716)

[1.5 Crearea, alterarea si eliminarea ferestrelor 6](#_Toc357455717)

[2. Gestionarea memoriei în Windows (Jircu Andrei 431A) 7](#_Toc357455718)

[2.1 Modelul de memorie Win32 7](#_Toc357455719)

[2.2 Memoria globală și locală 7](#_Toc357455720)

[2.3 Memoria virtuală 8](#_Toc357455721)

[2.3.1 Eliberarea memoriei virtuale 9](#_Toc357455722)

[2.3.2 Blocurile de memorie virtuală 10](#_Toc357455723)

[2.4 Memoria heap 10](#_Toc357455724)

[2.4.1 Alocarea unui bloc de memorie în memoria heap globală 11](#_Toc357455725)

[2.4.2 Crearea unui heap într-un proces 11](#_Toc357455726)

[2.4.3 Testarea dimenisunii memoriei alocate din heap 11](#_Toc357455727)

[2.5 Descărcarea unui bloc de memorie alocată 12](#_Toc357455728)

[2.6 Testarea memoriei calculatorului 12](#_Toc357455729)

[3.Procese si fire de execuție (Aflorei Victor 431A) 13](#_Toc357455730)

[3.1 Procesele 13](#_Toc357455731)

[3.2 Crearea proceselor 14](#_Toc357455732)

[3.3 Terminarea proceselor 15](#_Toc357455733)

[3.4 Extinderea prin procese copil 16](#_Toc357455734)

[3.5 Firele de execuție 16](#_Toc357455735)

[3.6. Determinarea identificatorului ID al unui fir sau proces 17](#_Toc357455736)

[3.7. Nivelurile de prioritate 17](#_Toc357455737)

[3.8. Obtinerea contextului unui fir 18](#_Toc357455738)

4.Concluzii .................................................................................................................................................................18

5.Bibliografie .............................................................................................................................................................19

# 1.Introducere în programarea Windows (Pâsoi Mihai 431A)

## 1.1.1 Programarea in Win32

Un sistem de operare este definit de propriul API (Application Programming Interface). Această interfață conține toate funcțiile pe care programatorul le poate folosi pentru realizarea unor aplicații caracteristice sistemului de operare utilizat. În cazul sistemelor Windows, API folosește o arhitectură de program specifică.

Pe parcursul timpului, Windows API a suferit modificări, cea mai importantă fiind cea din timpul schimbării de la arhitectura pe 16 biţi la cea pe 32 de biţi. API-ul pentru versiunile de Windows pe 16 biţi (de la Windows 1.0 la Windows 3.1) este denumit Win16. Cel pentru versiunile Windows pe 32 de biţi (Windows 95 / 98 / NT) este cunoscut ca Win32. Multe dintre apelările de funcţii au rămas la fel în urma tranziţiei de la Win16 la Win32, dar unele au suferit modificări.

Toate versiunile de Windows pe 32 de biţi suportă Win32 API pentru aplicaţiile curente şi sunt compatibile cu Win16 API pentru a asigura compatibilitatea cu aplicaţii mai vechi. Se pot observa deosebiri între modurile în care se realizează compatiblitatea între sistemele Windows NT şi Windows 95 / 98. La Windows NT, apelările de funcţii Win16 sunt convertite în apelări Win32. pentru a fi procesate de sistemul de operare.

La Windows 95 /98, procesul este invers. Apelările de funcţii Win32 sunt convertite în apelări Win16 pentru a fi procesate de sistemul de operare.

Windows NT şi Windows 98 sunt amândouă sisteme de operare ce suportă Win32 API. Deşi fiecare sistem prezintă anumite caracteristici individuale, asemănările dintre cele două este majoră, astfel încât se pot scrie programe care să funcţioneze sub amândoua sistemele. [1][2]

## 1.1.2 Deosebiri între programele DOS și Windows.

Între programele Windows şi programele DOS sunt diferenţe semnificative. Prima deosebire vizibilă ar fi modul în care se generează ieşirea. Un program DOS va genera ieşirea „Jamsa’s C/C++ Programmer’s Bible” în felul următor:

Jamsa’s C/C++ Programmer’s Bible

C:\>

Un program Windows ce creează o casetă de mesaj va genera o ieşire precum cea din fig.1:

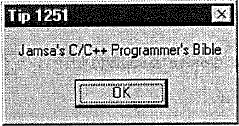


Figura 2

Alte deosebiri dintre programarea Windows şi DOS includ faptul că Windows este un sistem de operare multitasking, astfel incăt programele vor partaja spaţiul în memorie cu cel puţin un alt program. De asemenea, Windows prelucrează informaţii bazate pe text în mod diferit faţă de DOS.

Programele Windows nu sunt la fel de liniare ca programele DOS, programele Windows fiind construite în contextul răspunsului la acţiunile utilizatorului şi al mesajelor sistemului.[3]

## 1.2 Componentele Windows

Blocul de construcţie Windows este constituit din una sau mai multe ferestre sau casete de dialog. Fiecare obiect dintr-o fereastră este la rândul lui o fereastră de alt tip. În general, Windows îşi construieşte o fereastră standard din şapte componente de bază. Aceastea sunt:

1. Window Frame (cadru fereastră): este baza pe care se află toate componentele dintr-o fereastră.
2. Title Bar (bara de titlu): oferă utilizatorului informaţii despre program. Aceasta extinde dimensiunea ferestrei de-a lungul laturii de sus, identifică ce arată fereastra şi permite utilizatorului să execute numeroare operaţii cu fereastra, fiind punctul de control pentru deplasarea ferestrei şi este locul de plasare a meniului sistem .
3. Butoanele Minimize, Maximize, Restore şi Close Window: sunt situate in bara de titlu. Ele permit modificarea dimensiunii ferestrei şi închiderea ei.
4. Client Area (suprafaţa client): este secţiunea ferestrei unde se desfăşoară majoritatea acţiunilor.
5. Scroll Bar (bara de defilare): permite utilizatorului să navigheze de la dreapta la stânga şi de sus în jos în interiorul ferestrei.
6. Menu Bar (bara de meniu): este o componentă a majorităţii ferestrelor principale. În general nu este existentă în ferestre copil (child windows). Majoritatea programelor Windows au în componenţa barei de meniu opţiunile File şi Help. Programele mai complexe pot avea chiar si peste 10 opţiuni, fiecare cu mai multe opţiuni.
7. Status Bar (bara de stare): oferă utilizatorului informaţii caracteristice cu detalii despre poziţia curentă în program.[4]

Componentele sunt prezentare în cadrul unei ferestre în figura 2.

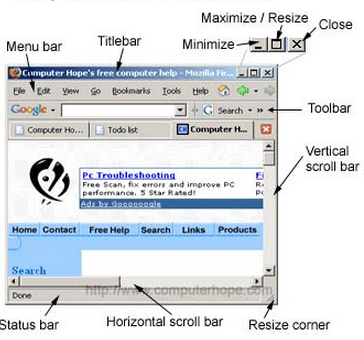


Figura 2

## 1.3 Identificatori Windows

În windows se folosesc identificatori pentru a menţine informaţii despre fişiere. Un identificator folosit în mod curent este indentificatorul de fereastră. Acesta este folosit pentru a obţine sau menţine informaţii despre ferestrele din program sau alte obiecte din sistem. Windows testează indentificatorul de fereastră din lista proprie de identificatori valizi de ferestre şi apoi trimite un mesaj sau execută o acţiune în fereastra corespunzătoare.

Împreună cu identificatorul de fereastră, în total sunt 12 tipuri de identificatori. Spre exemplu, deoarece Windows este un sistem de operate multi-tasking, este posibil să avem mai multe copii sau instanţe ale programului, care rulează în acelaşi timp. Windows păstrează, pentru fiecare instanţă, un singur număr pe care îl stochează într-o tabelă de identificatori de tip HINSTANCE. [2]

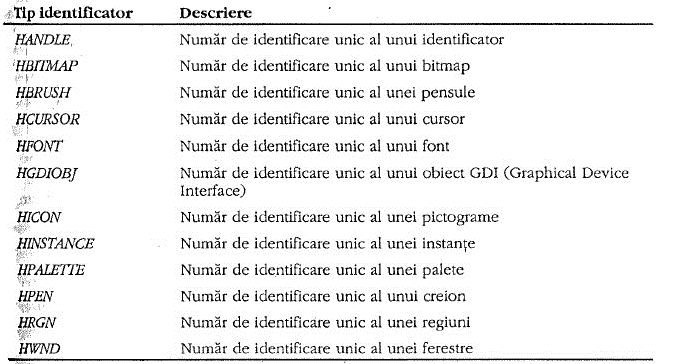


Figura 3 Lista de identificatori

## 1.4 Funcţii

În această secţiune vor fi prezentate unele din funcţiile necesare pentru crearea unui program Windows.

* Funcţiile callback : este o funcţie căreia i se transmite adresa unei a treia funcţii care revine cu informaţii. În cadrul programelor se întâlnesc des funcţii de tip callback împreună cu funcţii API specifice, precum *EnumFontFamilies* şi *EnumWindows*. Aceste funcţii sunt esenţiale în Windows întrucât programul trebuie să gestioneze multe dintre acţiunile lor prin API.
* Funcţia WinMain: returnează o valoare *int* la fel ca multe alte programe în C++. Este asemănătoare cu funţia *main*, antetele lor executând prelucrări similare.
* Funcţia CreateWindow: este funcţia utilizată pentru crearea efectivă a ferestrei. Cei 11 parametri ai funcţiei CreateWindow sunt: lpsyClassName, lpsyWindowName, dwStyle, x, y, nWidth, nHeight, hwndParent, hmenu, hinst, lpvParam. Multe dintre ferestre prezintă de obicei caracteristici comune.
* Funcţie ShowWindow: este funcţia cu care face fereastra vizibilă utilizatorului. Ex:

BOOL ShowWindow (HWND hWnd, int nCmdShow);

Această funcţie returnează valoarea *true* sau *false* pe care programul trebuie să o testeze pentru a determina reuşita sau eşecul.

* Funcţia RegisterClass: înregistrează în Windows stilul ferestrei creat înainte de a fi utilizat (se înregistrează o clasă definită de utilizator)

ATOM RegisterClass (CONST WNDCLASS \*lpwc);

* Funcţia RegisterClassEx : este similară funcţiei anterioare, dar în plus permite înregistrarea cu o pictogramă pe care o va plasa în bara de titlu în fiecare instanţă a clasei înregistrate.

ATOM RegisterClass (CONST WNDCLASSES \*lpwcx);[5][6]

## 1.5 Crearea, alterarea si eliminarea ferestrelor

Crearea unei ferestre este o operaţie relativ simplă. În primă fază trebuie stabilite componentele şi aspectul ferestrei şi apoi se utilizează funcţia Win32 API CreateWindows pentru crearea efectivă a ferestrei. În funcţie de parametrii introduşi, fereastra va lua o formă caracteristică.

Unul dintre parametrii transmişi funcţiei Createwindow de fiecare dată când este invocată este parametrul DWORD dwStyle. Se iniţializează valoarea parametrului folosind o secvenţă SAU ăe bit care combină o serie de constante cu stilurile ferestrei.

Ex: WS\_BORDER (creează o fereastră cu un chenar subţire) ; WS\_CHILD (creează o fereastră copil)

În mod similar se poate folosi funcţia CreateWindowExt pentru a crea ferestre cu stiluri extinse cum ar fi ferestre cu suprapunere, pop-up şi ferestre copil.

Pentru distrugerea ferestrei, se apelează funcţia API DestroyWindow. Aceasta va şterge fereastra transmisă în singurul ei parametru.

BOOL DestroyWindow (HWND hWnd);

Funcţia DestroyWindow distruge meniul ferestrei şi eliberează firul din coada de mesaje. Această funcţie este de obicei folosită pentru a răspunde unei comenzi de ieşire din cadrul programului. [2]

# 2. Gestionarea memoriei în Windows (Jircu Andrei 431A)

## 2.1 Modelul de memorie Win32

Pentru modelul de memorie Win32 procesele (programele) au propriul spațiu de adresă virtual

(pe 32 biți) ce poate ajunge până la valoarea de 4 GB.

Această memorie de 4 GB este împărțită în două blocuri de memorie. Primul bloc de 2 GB

(0x00000000 – 0x7FFFFFFF) este pusă la dispoziția utilizatorului, iar al II- lea bloc de memorie

(0x80000000- 0xFFFFFFFF) conține nucleul sistemului de operare.

Procesele Win32 sunt încărcate în adresa de bază a aplicațiilor Win32 ( 0x00400000) ce face parte din intervalul de memorie disponibil utilizatorului.

Nucleul sistemului de operare este cea mai importantă “componentă“ a sistemului de operare, ce gestionează pentru fiecare proces în parte o pagină de mapare. Rolul paginilor de mapare este de a converti adrese virtuale în adrese fizice. Spre deosebire de mediul Win16, unde gestionarea memoriei locale se face în cadrul spațiului de adresă al procesului și alocarea globală se face în afara spațiului de adresă al procesului; Win32 alocă ambele tipuri de memorie în cadrul spațiului de adresă al procesului.[7]

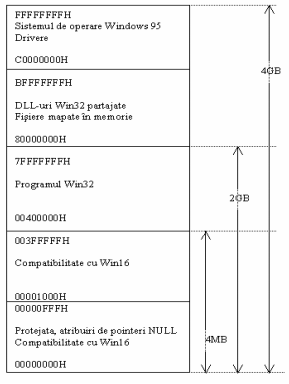


Figura 4 Organizarea memoriei în Win32 pentru Windows 95

## 2.2 Memoria globală și locală

Un proces folosește funcțiile *GlobalAlloc* și *LocalAlloc* pentru a aloca blocuri de memorie de diferite dimensiuni reprezentate pe 32 de biți. Obiectele de memorie alocate de program pentru aceste două funcții pot fi obiecte fixe sau mobile. Aceste funcții blochează memoria pentru ca obiectele să nu mai poată fi mutate sau înlăturate.

Funcțiile memoriei globale și locale sunt :

* GlobalReAlloc și LocalReAlloc ce au rolul de a modifica dimensiunea blocurilor de memorie;
* GlobalFree și LocalFree eliberează memoria alocată de GlobalAlloc și LocalAlloc ;
* GlobalDiscard și LocalDiscard pentru înlăturarea obiectelor de memorie ;
* GlobalFlags și LocalFlags pentru a afla informații despre un anumit obiect de memorie ; [8]

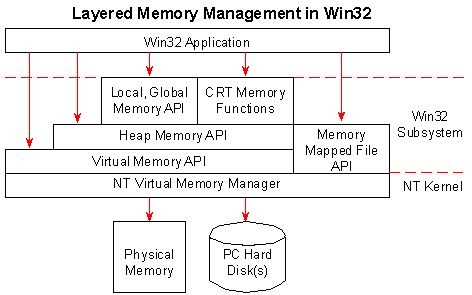


Figura 5 Distribuirea memoriei pe straturi în Win32

## 2.3 Memoria virtuală

Memoria virtuală reprezintă o zonă de stocare intermediară ce o folosește un proces în cazul în care necesită mai multă memorie RAM decât se găsește pe placa de bază. Datorită modului în care sistemul de operare Windows gestionează memoria virtuală, spațiul de adresă virtuală pentru fiecare proces în parte este mult mai mare decât adresa fizică totală disponibilă tuturor proceselor. [9]

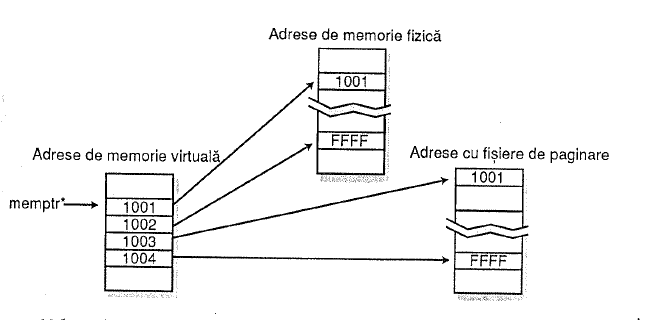


Figura 6 Utilizarea memoriei virtuale pentru a accesa memoria fizică

### 2.3.1 Eliberarea memoriei virtuale

Alocarea și eliberarea memoriei virtuale se face cu ajutorul funcțiilor : *VirtualAlloc* și *VirtualFree.*

Implementarea funcției *VirtualFree* :

BOOL VirtualFree (

LPVOID lpAddress,

DWORD dwSize,

DWORD dwFreeType

) ;

* lpAdress – indică adresa de bază a regiunii de pagini pentru eliberare
* dwSize – specific dimensiunea în octeți a regiunii care va fi eliberată
* dwFreeType – tipul operației de eliberare a memoriei

Dacă parametrul dwFreeType include indicatorul MEM\_RELEASE, parametrul dwSize trebuie să fie zero. Regiunea de pagini afectate va include toate paginile ce conțin unul sau mai mulți octeți în intervalul cuprins între parametrul lpAddress și dwSize.[9]

Exemplu de rezervare a unui interval de adrese utilizând funcția VirtualAlloc :

/\* Rezervare a unui interval de 10 MB de adrese \*/

lpBase = VirtualAlloc (NULL,

10485760,

MEM\_RESERVE,

PAGE\_NOACCESS);

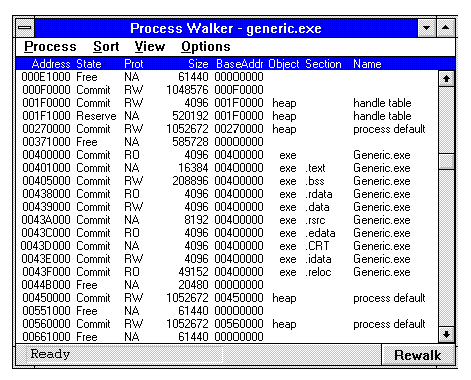


Figura 7 Heap inițial pentru fiecare proces

### 2.3.2 Blocurile de memorie virtuală

Spațiul de 2 GB de adrese folosite intr-un proces este de trei tipuri :

* Liber (free)
* Rezervat (reserved) pentru o viitoare utilizare
* Angajat (committed) pentru o memorie specific.

În figura de mai jos este reprezentat un proces ce conține cele trei regiuni de adrese : libere, rezervate și angajate.

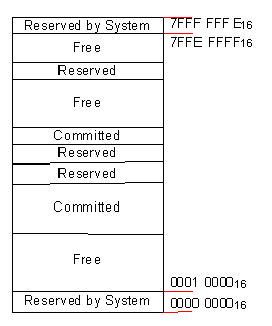


Figura 8 Proces ce conține cele trei regiuni de adrese

Rezervarea adreselor reprezintă o operație rapidă, complet independentă. Chiar dacă rezervăm 1 GB sau o dimensiune mai mică de adrese, procesul de rezervare este relativ rapid. Nu este deloc surprinzător, deoarece nu sunt alocate resurse pe parcursul acestei operații.

## 2.4 Memoria heap

Memoria heap reprezintă memoria disponibilă aplicației noastre. Dimensiunea maximă a memoriei heap este de 2 GB. Funcțiile Win32 de heap permit proceselor să creeze un “heap privat “ , ce reprezintă un bloc de una sau mai multe pagini în spațiul de adresă al procesului.

Funcțiile memoriei heap sunt :

* HeapCreate – produce un heap de o anumită dimensiune ;
* HeapAlloc și HeapFree – alocă și eliberează memorie din heap ;
* HeapDestroy – distrugerea memoriei heap ; [10]

### 2.4.1 Alocarea unui bloc de memorie în memoria heap globală

Vom utiliza funcția *GlobalAlloc* pentru a aloca memorie din memoria heap globală. Această funcție alocă numărul de octeți specificați din heap.

Implementarea funcției *GlobalAlloc :*

HGLOBAL GlobalAlloc (

UINT uFlags,

DWORD dwBytes

* uFlags – specifică modul de alocare a memoriei.În cazul in care valoarea este zero, valoarea implicită este *GMEM\_FIXED* ( alocă memorie fixă; valoarea returnată este un pointer la blocul de memorie )
* dwBytes – conține numărul de octeți de alocat. Dacă parametrul este zero, iar parametrul uFlags specifică valoarea *GMEM\_MOVEABLE* ( alocă memorie mobilă;acest indicator nu poate fi combinat cu *GMEM\_FIXED*; valoarea returnată este identificatorul obiectului de memorie ) funcția va returna un identificator al unui obiect de memorie ce poate fi înlăturat.

În cazul în care memoria heap nu conține suficient spațiu liber, *GloablAlloc* returnează NULL.[9]

### 2.4.2 Crearea unui heap într-un proces

Crearea unui heap într-un proces se face cu ajutorul funcției *HeapCreate.*

Implementarea funcției *HeapCreate* :

HANDLE HeapCreate (

DWORD flOptions,

DWORD dwInitialSize,

DWORD dwMaximumSize

);

* flOptions – specifică atributele opționale ale noului heap.
* dwInitialSize – dimensiune inițială a zonei heap. Determină cantitatea inițială de stocare fizică alocată pentru heap.
* dwMaximumSize – dimensiune maximă a zonei heap [2]

### 2.4.3 Testarea dimenisunii memoriei alocate din heap

Pentru testarea dimensiunii memoriei alocate din heap vom folosi funcția *HeapSize.*

Funcția *HeapSize* returnează dimensiunea în octeți a blocului de memorie alocat din heap.

Implementarea funcției *HeapSize* :

DWORD HeapSize (

HANDLE hHeap,

DWORD dwFlags,

LPCVOID lpMem

) ;

* hHeap – specifică zona heap în care se înlătură blocul de memorie
* dwFlags – specifică mai multe aspecte controlabile ale accesării blocului de memorie
* lpMem – indică blocul de memorie a cărui dimensiune se va obține [2]

## 2.5 Descărcarea unui bloc de memorie alocată

Pentru descărcarea unui bloc de memorie alocată vom folosi funcția *GlobalDiscard.* Pentru a descărca blocul de memorie anterior vom folosi indicatorul GMEM\_DISCARDABLE ce alocă memorie descărcabilă ( memorie ce nu este fixată la o anumită adresă în spațiul de adresă virtuală al procesului).

Implementarea funcției *GlobalDiscard* :

HGLOBAL GlobalDiscard (

HGLOBAL hglbMem

) ;

* hglbMem – identifică obiectul de memorie globală de descărcat. Va afișa valoarea NULL, în cazul în care funcția nu reușește.
* Funcția *GlobalDiscard* descarcă numai obiecte globale alocate de procesul apelant cu valoarea GMEM\_DISCARDABLE. [2]

## 2.6 Testarea memoriei calculatorului

Pentru afla informații despre cantitatea de memorie liberă disponibilă vom folosi funcția *GlobalMemoryStatus.* Această funcțiereturnează informații despre memoria fizică și memoria virtuală.

Implementarea funcției *GlobalMemoryStatus* :

VOID GlobalMemoryStatus (

LPMEMORYSTATUS lpBuffer

* lpBuffer indică o structură LPMEMORYSTATUS în care este returnată informația despre disponibilitatea memoriei curente.
* Structura MEMORYSTATUS conține informația despre disponibilitatea memoriei curente.[11]

# 3.Procese si fire de execuție (Aflorei Victor 431A)

## 3.1 Procesele

Una din cele mai importante caracteristici ale sistemului Windows este suportul multitasking. Noțiunea de multitasking reprezintă rularea simultană în memorie a mai multor procese.

Un proces poate fi definit ca obiectul care deține toate resursele unei aplicații. În Windows un proces poate crea unul sau mai multe fire de execuție. Un fir de execuție reprezintă o cale independentă de execuție în cadrul unui proces cu care firul partajează spațiul de adresă, codul și datele globale. Fiecare fir de execuție posedă propriul set de registre, propria stivă, propriile mecanisme de intrare, inclusiv o coadă privată de mesaje.Windows 95 și Windows NT ordonează firele în funcție de prioritățile alocate fiecăruia.[2],[12]

Un proces poate cuprinde alocări de memorie globală, pagini virtuale și așa mai departe.

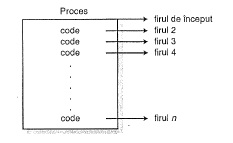


Figura 9 Relațiile dintre fire și procese

În măsura în care procesele au nevoie de memorie, Windows alocă memorie virtuală pentru fiecare proces. În cazul în care avem un calculator pe care rulează simultan mai multe procese este impotantă indentificarea procesului activ curent. Această indentificare este importantă deoarece sistemul de operare va atașa automat prioritate înaltă majorității solicitărilor CPU exercitate de procesul activ. Pentru accelerarea execuției procesului activ curent Windows alocă memorie fizică suplimentară.

În figura de ma jos este reprezentat un model de memorie pentru trei aplicații simultane, în care aplicația activă curentă este *first\_app*.

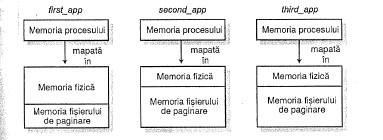


Figura 10 Exemplu de model de memorie pentru trei aplicații care se execută simultan

În cazul în care utilizatorul face activ programul second\_app, Windows va realoca memorie fizică astfel încât va elibera mai mult spațiu pentru executarea programului second\_app.

## 3.2 Crearea proceselor

În sistemele de operare complexe este necesară o modalitate de a crea şi termina procesele în funcţie de necesităţile din timpul funcţionării.

Principalele evenimente care determină crearea proceselor sunt:

* Inițializarea sistemului;
* Execuția unui apel de sistem pentru crearea unui proces de către un proces în execuție;
* O cerere a utilizatorului pentru crearea unui nou proces;
* Inițierea unui lucrări în fundal (batch job).[13]

În mod normal când un sistem de operare pornește sunt create mai multe procese.

Aceste procese pot fi împărțite în 2 categorii:

* Procese de prim plan (foreground processes) – procese care interacționează cu

utilizatorii umani și efectuează operații destinate acestora;

* Procese de fundal (background processes) – procese care nu sunt asociate cu un

utilizator uman. Un exemplu de proces de fundal poate fi un proces desemnat să accepte poșta electronică.

Pe lângă procesele create la momentul pornirii sistemului, se pot crea și procese noi. Crearea proceselor noi este importantă atunci când sarcina de îndeplinit poate fi descompusă în mai multe procese înrudite care interacționează dar sunt independente.

Principala funcție utilizată în programarea Windows este funcția *CreateProcess.* Această funcție produce un proces nou și firul său de execuție primar. Firul de execuție primar este denumit frecvent proces copil. *CreateProcess* permite procesului părinte (adică procesul care apelează funcția *CreateProcess*) să specifice mediul de operare a noului proces (modul său de lucru, modul în care va apărea implicit pe ecran și prioritatea procesului). Linia de comandă și conținutul ei se transmite către procesul copil.[2]

Pe lângă crearea unui proces, funcția *CreateProcess* produce un nou obiect fir de execuție. Noul proces și identificatorii noului fir sunt creați cu drepturi depline de acces. Dacă nu este furnizat un descriptor de securitate, fiecare identificator poate fi utilizat în orice funcție care reclamă un identificator de obiect de acel tip. Dacă funcția oferă un descriptor de securitate, programul efectuează o testare de acces asupra tuturor utilizărilor ulterioare ale indentificatorului înainte ca accesul să fie permis. Dacă testarea de acces nu permite accesul, procesul solicitant nu va fi capabil să utilizeze identificatorul pentru a avea acces la fir.

Windows atribuie procesului un identificator pe 32 de biți. Identificatorul este valid până la terminarea procesului și pentru identificarea procesului sau specificarea funcției *OpenProcess* utilizată la deschiderea unui identificator la proces. De-asemenea un identificator de 32 de biți este alocat și pentru firul de execuție care este valid până la terminarea firului și poate fi utilizat pentru identificarea unică a firului în sistem.

Microsoft specifică că un proces se poate termina prin utilizarea funcției *ExitProcess*, funcție care anuntă toate bibliotecile cu legare dinamică (DLL) atașate procesului despre terminarea sa iminentă. Sunt și mijloace de a termina un proces care nu anunța bibliotecile DLL atașate.

În cadrul unui proces sunt utilizate funcțiile *ExitProcess, ExitThread, CreateThread, CreateRemoteThread* împreună cu procesul care începe să ruleze. La un moment dat numai unul din aceste evenimente poate să aibă loc în același spațiu de adresă. Pentru a putea fi îndeplinite acestea sunt necesare următoarele restricții:

* Pe parcursul rutinelor de pornire a procesului și de inițializare a bibliotecilor DLL, pot

fi create noi fire de execuție, dar ele nu încep execuția până când Windows nu încheie inițializarea bibliotecii DLL pentru proces;

* Numai un singur fir într-un proces poate fi present la un moment dat, în inițializarea

unui DLL sau a unei rutine detașate;

* Funcția *ExitProcess* nu se returnează decât când nu mai există fire în inițializările de

DLL sau în rutinele detașate.[14]

Procesul creat rămâne în sistem până când toate firele din cadrul procesului s-au încheiat și toți identificatorii de process și oricare din firele sale s-au închis în urma apelurilor la funcția *CloseHandle.* Atât identificatorii pentru process, cât și cei pentru firele principale trebuie închise prin apeluri la funcția *CloseHandle.* După ce este creat procesul, dacă acești identificatori nu mai sunt necesari, este bine să fie imediat inchiși.

Când se încheie ultimul fir din proces, sunt lansate următoarele evenimente:

- Toate obiectele deschise de către proces sunt implicit închise.

- Starea de terminare a procesului se schimbă de la valoarea sa inițială STILL\_ACTIVE la starea de terminare a ultimului fir.

- Windows stabilește obiectul fir al firului principal pe starea de semnalizare, satisfăcând orice fir aflat în așteptarea obiectului.

- Windows stabilește obiectul de proces pe starea de semnalizare, satisfăcând orice fir aflat în așteptarea obiectului.

În plus față de funcția *CreateProcess,* Win 32 are aproximativ alte 100 de funcții pentru gestiunea și sincronizarea proceselor.

## 3.3 Terminarea proceselor

După crearea unui proces, acesta începe să se execute și își efectuează sarcina. Terminare unui proces este determinată de obicei de una din următoarele condiții:

* Ieșire normală (voluntară);
* Ieșire cu eroare (voluntară);
* Eroare fatală (involuntară);
* Terminare de către alt process (involuntară).[13]

Majoritatea proceselor se termină pentru că și-au îndeplinit sarcina.

Funcția folosită pentru încheierea executării unui process este *ExitProcess.* Funcția încheie un

proces împreună cu toate firele sale și returnează controlul către locația apelantă.

Pentru închiderea unui process trebuie întotdeauna apelată funcția *ExitProcess* care oferă o încheiere corectă a procesului.

Funcția *ExitProcess* termină efectiv procesul curent după ce toate bibliotecile DLL atașate au executat valoarea de terminare a precesului. Terminarea procesului are ca efect următoarele:

* Sunt închiși toți identificatorii de obiect deschiși de proces.
* Toate firele din cadrul preocesului își termină execuția.
* Starea obiectului din process devine semnalizată, satisfăcând toate firele care s-au aflat

în așteptarea terminării firelor procesului ce se încheie.

* Starea de terminare a procesului se schimbă din valoarea STILL\_ACTIVE la valoarea

de ieșire a procesului.[14]

Terminarea unui proces nu are ca efect terminarea proceselor copil. Terminarea unui process nu duce în mod necesar la eliminarea din sistemul de operare a obiectului process. Un proces este șters atunci când ultimul identificator la process este închis.

## 3.4 Extinderea prin procese copil

În aplicațiile complexe sunt întâlnite situații în care un anumit bloc de cod să execute o operație. Pentru a efectua această operație programul apelează o anumită funcție însă funcțiile sunt executate în serie, deci codul nu poate continua executarea până ce o funcție nu își încheie execuția. O altă metodă de executare a unui alt bloc de cod este crearea unui nou fir în cadrul procesului care preia execuția. Firele multiple pot cauza probleme de sincronizare când un fir trebuie să utilizeze rezultatele noului fir.

O soluție mai bună este generarea de procese noi numite *procese copil.* Operarea cu procese permite programelor să continue executarea operațiilor procesului copil pe o anumită problemă sau permite programelor să-și suspende execuția până ce procesul copil operează asupra unei probleme.

## 3.5 Firele de execuție

Un fir de execuție este o cale de execuție într-un proces. Fiecare proces Win32 conține unul sau mai multe fire de execuție. Când este inițializată o nouă instanță a unui process sistemul de operare creează un nou fir primar pentru procesul respectiv. Când programul este încărcat în Windows atunci are loc și pornirea firului pimar. Firul va apela funcția *WinMain* și își va continua execuția până când funcția *WinMain* își încetează prelucrarea iar programul apelează *ExitProcess* pentru închiderea sa.

Pentru cele mai multe aplicații, firul primar creat de sistemul de operare este singurul fir cerut de aplicație. Procesele pot crea fire suplimentare pentru a-și ușura operarea. Firele suplimentare sunt create pentru a folosi la maxim și cât mai eficient timpul de prelucrare al unității centrale. Când sunt create firele suplimentare sunt trimise sistemului de operare cereri suplimentare pentru alocări de timp CPU. Firele suplimentare permit programelor să efectueze mai eficient prelucrări de fundal, să efectueze calcule extinse și activități bazate pe timp și evenimente.

Sistemul va prelucra unele fire mai des decât altele, atât în cadrul unui process, cât și în procese diferite în funcție de prioritatea lor. [2],[12],[15]

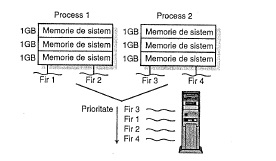


Figura 11 Modelul logic al firului de process

## 3.6. Determinarea identificatorului ID al unui fir sau proces

Pentru firul sau procesul curent programele au nevoie de identificatori temporari sau pseudo-handle. Interfața Win32 API furnizează două funcții*: GetCurrentThreadId* și *GetCurrentProcessId*, care permit programelor să obțină o valoare unică *DWORD* utilizată de sistemul de operare pentru reprezentarea internă a firelor și proceselor.

Prototipurile funcțiilor sunt:

DWORD GetCurrentThreadId (void);

DWORD GetCurrentProcessId(void);

Funcția *GetCurrentThreadId* returnează identificatorul de fir al firului apelant, care este valoarea returnată de firul apelant. Până la terminarea firului, identificatorul de fir identifică în mod unic firul în sistem.[2]

Funcția *GetCurrentProcessId* returnează identificatorul de proces al procesului apelant. Funcția nu are parametri. Valoarea returnată este identificatorul de proces al procesului apelant. Până la terminarea firului, identificatorul de proces identifică în mod unic procesul în sistem.

## 3.7. Nivelurile de prioritate

Sistemul de operare Win32 planifică toate firele active pe baza nivelului curent de prioritate. Nivelurile de prioritate se extind de la valoarea 0 (cel mai scăzut nivel de prioritate) la valoarea 31 (cel mai înalt nivel de prioritate). Nivelul zero de prioritate este atribuit unui fir special al sistemului cunoscut cu numele de *fir de pagină zero.* Acest fir atribuie valoarea zero fiecărei pagini libere din sistem atunci când nu mai există alte fire care trebuie executate în sistem.[16]

Pentru determinarea nivelului de prioritate a unui fir sistemul utilizează doi pași. Primul pas este atribuirea unei *clase de prioritate* unui proces. Al doilea pas este atribuirea unui nivel de prioritate fiecărui fir deținut de proces.

În cazul în care un fir este cerut prima dată în cadrul procesului nivelul de prioritate al firului este similar cu cel al procesului.

Clasele de prioritate pentru procese sunt:

* HIGH\_PRIORITY\_CLASS – Indică un proces care efectuează activități urgente ce

solicită o executare imediată pentru a rula corect.

* IDLE\_PRIORITY\_CLASS – Indică un proces ale cărui fire rulează numai când

sistemul este inactiv și dau întâietate firelor oricărui proces care rulează într-o clasă de prioritate superioară.

* NORMAL\_PRIORITY\_CLASS – Indică un proces normal, fără nicio cerință specială

de programare a activității.

* REALTIME\_PRIORITY\_CLASS – Indică un proces care prezintă cea mai înaltă

prioritate posibilă.[2],[16]

## 

## 3.8. Obtinerea contextului unui fir

Windows stochează informațiile referitoare la fire într-o structură CONTEXT. În momentul în care programul manipulează mai multe fire din ce în ce mai frecvent avem nevoie de informații despre contextul firelor. Contextul unui fir specificat se obține cu ajutorul funcției *GetThreadContext.* Funcția se poate implementa astfel:

*BOOL GetThreadContext(*

*HANDLE hThread,*

*LPCONTEXT lpContext*

*);*

Valoarea membrului *ContextFlags* din această structură precizează care porțiune a contextului firului este preluată. Structura CONTEXT depinde în mare măsură de calculator. În mod curent, există stucturi CONTEXT definite pentru procesoare Intel, MIPS, Alpha și PowerPC.[2]

Funcția *GetThreadContext* va fi utilizată pentru preluarea contextului firului specificat. Funcția permite obținerea unui context selective bazat pe valoarea membrului *ContextFlags* al structurii CONTEXT.

# **4.** **Concluzii**

Funcțiile GloballAlloc și LocalAlloc sunt foarte uzuale în alocarea unor blocuri fixe de memorie ce pot fi numite pagini pentru utilizarea mecanismului interschimbării proceselor.Tehnica paginării este larg folosită și poate fi independentă de gestionarea memoriei.Există procesoare ce creează pagini în memoria fizică, sau, ca în cazul familiei Intel, în memoria virtuală. Cu ajutorul funcțiilor prezentate mai sus putem spune că diferitele operații, cum ar fi eliberarea memoriei virtuale, crearea unui heap într-un proces,descărcarea unui bloc de memorie alocată, testarea memoriei calculatorului etc, sunt foarte ușor de implementat.

În urma acestei prezentări, se pot observa avantajele folosirii programării cu WIN32 API în locul altor metode, avantaje precum control total asupra aplicațiilor, control total asupra fiecărei părți din GUI, evitarea prelucrării informațiilor adiționale, precum și dimensiunea redusă a aplicației compilate finale

# 5.Bibliografie

* [1] Programming Windows - Charles Petzold
* [2] Jamsa's C/C++ Programmer's Bible
* [3] <http://www.differencebetween.info/difference-between-dos-and-windows>
* [4] <http://www.screenio.com/gui_screenio/gs_htmlhelp_subweb/windows-stuff/windows-elements.htm>
* [5] <http://msdn.microsoft.com/en-us/library/vstudio/bb384843.aspx>
* [6] <http://www.functionx.com/win32/Lesson01c.htm>
* [7] <http://revistaie.ase.ro/content/17/pocatilu%2Bivan%2Bcazan.pdf>
* [8] <http://winapi.freetechsecrets.com/win32/WIN32Global_and_Local_Functions.htm>
* [9] <http://msdn.microsoft.com/en-us/library/ms810627.aspx>
* [10] <http://www.maxi-pedia.com/what+is+heap+and+stack>
* [11] Windows 95 System programming SECRETS - Matt Pietrek
* [12] <http://msdn.microsoft.com/en-us/library/windows/desktop/ms681917(v=vs.85).aspx>
* [13] <http://webhost.uoradea.ro/cpopescu/so/Cursul3.pdf>
* [14] <http://msdn.microsoft.com/en-us/library/windows/desktop/ms686722(v=vs.85).aspx>
* [15] <https://en.wikipedia.org/wiki/Thread_(computing)>
* [16] <http://msdn.microsoft.com/en-us/library/windows/desktop/ms685100(v=vs.85).aspx>