# 3. Controlul ferestrelor Windows

Uşurința lucrului cu aplicațiile-fereastră din sistemul de operare Windows, a dus la o răspândire vertiginoasă a acestuia. Windows nu a adus numai ferestre uşor de utilizat cu o interfață prietenoasă ci o dată cu API-ul Win32 şi *Application Wizard*-ul din Visual Studio s-a creat şi posibilitatea creării rapide a aplicațiilor bazate pe ferestre.

Dezvoltatorii de softuri pentru Windows nu numai că au posibilitatea manipulării controalelor puse la dispoziție de mediu ci pot chiar rescrie acestea pentru a corespunde cerințelor. Această caracteristică a sistemului Windows nu a fost neglijată de aplicațiile malițioase, care pot astfel imita ferestrele de login, preluând parola utilizatorilor legitimi, care este apoi transmisă prin rețea destinatarului.

În acest context, înțelegerea uşurinței cu care se pot dezvolta asemenea aplicații poate fi considerată ca fiind o cerință de bază pentru cei care studiază domeniul vast al *Securității Informației*.

Uşurința manipulării ferestrelor Windows va fi ilustrată în cadrul acestui capitol prin realizarea unei aplicații care preia controlul total asupra spațiului de lucru până când se introduce o parolă validă într-o căsuță de editare.

## 3.1 Punctul de pornire: aplicații MFC bazate pe ferestre dialog

Pentru început, vom crea o aplicație MFC bazată pe ferestre dialog. Ştergem butonul *Cancel*, întrucât nu avem nevoie de el, şi adaugăm un control de editare. Pentru controlul de editare alegem ID-ul `IDC_EDIT_PWD` şi ataşăm o variabilă membru de tip `CString` (de ex. `m_szPwd`). După aceasta, ne asigurăm că mesajul `WM_INITDIALOG` are ataşat o funcție membru (i.e. `OnInitDialog`).

Fereastra care trebuie să rezulte la rularea aplicației, este ilustrată în Figura 3.1.


**Figura 3.1** Fereastră dialog de introducere a unei parole

## 4.2 Controlul închiderii ferestrelor

În acest stadiu de început, fereastra creată încă nu poate obliga utilizatorul să facă nimic. Acesta o poate închide, o poate dezactiva, fără ca fereastra să poată „riposta”. Dar cu puțin cod în plus, vom vedea că această fereastră poate deveni o interfață ce nu va

permite accesarea spațiului de lucru fără introducerea unei parole corecte. Prevenirea închiderii ferestrei create este primul pas.

O fereastră poate fi închisă prin următoarele metode:

- Închidere directă: Alt+F4, apăsarea butonului ‘X’;
- Închidere indirectă: apăsarea butonului ESC sau a butonului ENTER;
- Închidere forțată: dintr-un vizualizator de procese cum este Task Manager, activat prin combinația de taste CTRL+Alt+Del.

**Închiderea directă** poate fi evitată prin suprascrierea metodei `OnClose`, sau prin ataşarea unei metode mesajului `WM_CLOSE` şi ştergerea apelului metodei `OnClose` din clasa de bază:

```
void CTestTotalControlDlg::OnClose()
{
      //CDialog::OnClose();
}
```

**Închiderea indirectă** se poate interzice prin suprascrierea metodelor `OnCancel` şi `OnOK` care sunt apelate la apăsarea tastelor ESC respectiv ENTER şi ştergerea apelului metodelor `OnCancel` şi `OnOK` din clasa de bază:

```
void CTestTotalControlDlg::OnCancel()
{
     //CDialog::OnCancel();
}

void CTestTotalControlDlg::OnOK()
{
     //CDialog::OnOK();
}
```

**Închiderea forțată** este puțin mai greu de realizat întrucât acest mecanism a fost introdus special pentru evitarea acestor situații în care aplicațiile pot prelua controlul total asupra spațiului de lucru. În funcție de sistemul de operare utilizat, pot fi implementate diferite mecanisme care să permită totuşi prevenirea închiderii forțate.

Astfel, pe Windows 95, 98, Me, deoarece administratorul de procese are doar rolul administrării aplicațiilor care rulează şi nu pentru administrarea utilizatorilor, următoarea secvență de cod va bloca afişarea lui:

```
BOOL old;
SystemParametersInfo( SPI_SCREENSAVERRUNNING, TRUE, &old, 0 );
```

Funcția API `SystemParametersInfo` permite interogarea şi setarea parametrilor sistemului (e.g. wallpaper, mouse, protector de ecran). Parametrul `SPI_SCREENSAVERRUNNING` poate activa/dezactiva protectorul de ecran, ceea ce nu implică rularea unei asemenea aplicații ci doar activarea/dezactivarea unor funcționalități, printre care se numără şi combinația de taste CTRL+Alt+Del. În variabila `old` se stochează starea anterioară a protectorului de ecran. Prin utilizarea acestei funcții sistemul

de operare va avea impresia că rulează un protector de ecran, motiv pentru care nu va activa fereastra de administrare a proceselor.

O dată cu sistemele NT, protejarea utilizatorilor a făcut ca Task Managerul, activat prin combinația de taste CTRL+Alt+Del, să permită şi logarea-relogarea utilizatorilor. Această funcționalitate a fost adăugată pentru a nu permite aplicațiilor malițioase să imite fereastra de logare, profitând astfel de naivitatea utilizatorilor care îşi introduc parola ori de câte ori l-i se cere.

În sistemele NT, combinația de taste CTRL+Alt+Del poate activa Task Managerul pentru administrarea proceselor sau versiunea de Login pentru logare, re-logare în funcție de setările din conturile utilizator:

- În „Control Panel" se alege „User Accounts";
- Se selectează „Change the way users log on or off";
- Se selectează „Use the welcome screen for fast and easy logon" (NT) sau „Use the Welcome screen" (XP).

Astfel, apăsarea combinației CTRL+Alt+Del va determina activarea doar a ferestrei Task Manager, fără activarea ferestrei de Login. Fereastra Task Manager poate fi tratată ca o fereastră uzuală, asupra căreia pot fi aplicate toate funcțiile aferente, inclusiv cele de ascundere:

```
HWND hWnd=::FindWindow( NULL, "Windows Task Manager" );
if ( hWnd != NULL )
{
      ::ShowWindow( hWnd, SW_HIDE );
}
```

În secvența de cod prezentată anterior, funcția `FindWindow`, apelată din domeniul global de funcții prin utilizarea operatorului rezoluție `::`, va returna un handle-ul ferestrei Task Manager. Primul parametru al acestei funcții este clasa de apartenență a ferestrei (e.g. Dialog, Window) iar al doilea parametru este numele ferestrei. Întrucât cunoaştem denumirea ferestrei, primul parametru va fi completat cu `NULL`.

Dacă fereastra este găsită, ea va fi ascunsă prin apelul funcției `ShowWindow` cu parametrul `SW_HIDE`.

Întrucât nu se cunoaşte momentul în care utilizatorul încearcă activarea Task Managerului, putem folosi un **Timer** (i.e. Temporizator) cu ajutorul căruia la intervale regulate căutăm fereastra Task Manager şi o ascundem.

Un **Timer** are ataşat un interval de timp şi o funcție care se apelează la scurgerea timpului configurat. Astfel, dacă alegem timpul de activare al timerului la 100 milisecunde, vom putea verifica dintr-o funcție apelată automat, dacă fereastra Task Manager este activă sau nu, la intervale de 100 milisecunde.

Activarea Timerului se poate realiza din funcția `OnInitDialog`, în momentul inițializării ferestrei dialog:

```
BOOL CTestTotalControlDlg::OnInitDialog()
{
	CDialog::OnInitDialog();

	...
	SetTimer( 1, 100, NULL );
}
```

Primul parametru al funcției `SetTimer` este un ID transmis funcției apelate la activarea timerului prin care se va putea diferenția timerul care a apelat funcția. Această facilitate este necesară întrucât mai multe timere pot avea atașate aceeași funcție de apel.

Al doilea parametru reprezintă intervalul de timp ales, iar al treilea parametru este adresa unei funcții care se apelează la expirarea timpului. Dacă ultimul parametru este `NULL`, atunci sistemul va pune în coada de mesaje a aplicației un mesaj `WM_TIMER`. În corpul funcției de tratare a acestui mesaj se poate implemena codul de căutare și ascundere. De exemplu, o funcție `OnTimer` atașată mesajului `WM_TIMER` poate avea următoarea definiție:

```
void CTestTotalControlDlg::OnTimer(UINT nIDEvent)
{
 if ( nIDEvent == 1 )
 {
	HWND hWnd=::FindWindow( NULL, "Windows Task Manager" );
	if ( hWnd != NULL )
	{
		::ShowWindow( hWnd, SW_HIDE );
	}
 }
 CDialog::OnTimer(nIDEvent);
}
```

## 3.3 Maximizarea ferestrelor

Dacă dorim ca fereastra noastră să acopere tot ecranul pentru a nu permite alegerea și execuția altor aplicații, atunci va trebui să o redimensionăm. Pentru aceasta, vom determina mai întâi dimensiunea spațiului de lucru, reprezentând dimensiunea ecranului - dimensiunea barei de start:

```
CRect rWorkArea;
SystemParametersInfo( SPI_GETWORKAREA, 0, &rWorkArea, 0 );
```

După aceasta, interogăm dimensiunea barei de start. În cazul în care nu găsim bara de start, considerăm că are o lungime de 600 pixeli și o lățime de 100 de pixeli.

```
CRect rShellArea;
HWND hTrayWnd = ::FindWindow( "Shell_TrayWnd", NULL );
if ( hTrayWnd != NULL )
{
	::GetWindowRect( hTrayWnd, &rShellArea );
}
else
{
```

```
        rShellArea = CRect( 0, 0, 600, 100 );
}
```

Având aceste informații, putem modifica noua poziție și dimensiune a ferestrei:

```
MoveWindow( 0, 0,
            rWorkArea.Width(),
            rWorkArea.Height() + rShellArea.Height() );
```

## 3.4 Ferestre „Top-Most”

Dacă adunăm toate elementele descrise până acum și le implementăm într-o aplicație fereastră, aplicația respectivă are proprietatea că nu poate fi închisă și fereastra aferentă acoperă tot ecranul. Cu toate acestea, utilizatorul mai poate folosi combinația de taste Alt+Tab pentru a selecta o altă aplicație și implicit, activarea barei de start.

Întrucât există o serie de aplicații, în afară de Task Manager, care ar putea închide fereastra creată cum ar fi aplicația „Process Viewer”, instalată o dată cu Visual Studio 6, nu trebuie să acordăm posibilitatea utilizatorului să ativeze un alt program.

Aceasta se poate realiza folosind o fereastră care este plasată deasupra celorlalte ferestre. Așa cum este de altfel și cazul ferestrei „Task Managerului”, pentru a realiza acest lucru, trebuie setat atributul „Top-Most” al ferestrei:

```
SetWindowPos( &CWnd::wndTopMost,
              0, 0, 0, 0,
              SWP_NOMOVE | SWP_NOSIZE );
```

Primul parametru al funcției `SetWindowPos` este cel care ne interesează, întrucât aceasta plasează fereastra deasupra celorlalte ferestre. Următorii 4 parametrii se referă la poziția (*x*, *y*) și dimensiunea (*width*, *height*) ferestrei. Ultimul parametru permite specificarea atributelor modificate. După cum se poate observa, funcția nu va muta fereastra și nu o va redimensiona, fiind date doar atributele `SWP_NOMOVE` și `SWP_NOSIZE`.

De fapt, pentru setarea dimensiunilor ferestrei și a proprietății de „Top-Most”, se poate folosi un singur apel:

```
SetWindowPos( &CWnd::wndTopMost,                    //Top-Most
              0,                                    //x
              0,                                    //y
              workRect.Width(),                     //width
              workRect.Height()+shellRect.Height(), //height
              0 );                                  //set all
```

Pentru ca efectul funcției să fie cel dorit, să nu poată apărea nici o altă fereastră cu această proprietate, această funcție ar trebui apelată în cadrul funcției `OnTimer`.

Totodată, cu toate că fereastra aceasta este „top-most”, prin mișcarea ei se va putea totuși activa o altă fereastră aflată în spatele ei. Din această cauză, trebuie ascunsă bara de titlu prin dezactivarea opțiunii Resursa_Dialog/Properties/Styles/Title bar.

## 3.5 Ascunderea şi afişarea barei de Start

Pentru ca utilizatorul să nu mai poată lansa în execuție aplicații accesibile din bara de start, aceasta trebuie ascunsă. Întrucât bara de start nu este altceva decât o fereastră cu o serie de sub-ferestre, funcțiile care se aplică ferestrelor în general, se aplică şi acesteia.

Astfel, fereastra, sau altfel zis bara de start, poate fi ascunsă prin secvența de cod:

```
HWND hTrayWnd = ::FindWindow( "Shell_TrayWnd", NULL );
if ( hTrayWnd != NULL )
{
      ::ShowWindow( hTrayWnd, SW_HIDE );  //ascunderea ferestrei
}
```

Afişarea barei se face similar:

```
HWND hTrayWnd = ::FindWindow( "Shell_TrayWnd", NULL );
if ( hTrayWnd != NULL )
{
      ::ShowWindow( hTrayWnd, SW_SHOW ); //afisarea ferestrei
}
```

## 3.6 Preluarea parolei din fereastra de control total

Pentru ca parolele introduse în controalele de tip „edit-box” să nu fie vizibile, se poate specifica în editorul de resurse, la proprietățile controlului, că acest control va fi folosit pentru preluarea unei parole (click dreapta pe control în editorul de resurse, Properties/Styles/Password). Stilul parolă poate fi ales şi printr-o funcție membră a clasei CEdit, `SetPasswordChar`, folosind următoarea secvență de cod, unde s-a considerat ID-ul `IDC_EDIT_PWD` pentru controlului de introducere a parolei:

```
CEdit* pwPassword = ( CEdit* )GetDlgItem( IDC_EDIT_PWD );
if ( pwPassword != NULL )
{
      pwPassword->SetPasswordChar( _T('*') );
}
```

Folosind funcția `SetPasswordChar`, se configurează de fapt caracterul care se afişează în locul caracterelor introduse. Dacă dorim să resetăm controlul, adică să nu se mai afişeze caracterul ‘*’ în locul caracterelor introduse, putem utiliza secvența următoare de cod:

```
CEdit* pwPassword = ( CEdit* )GetDlgItem( IDC_EDIT_PWD );
if ( pwPassword != NULL )
{
      pwPassword->SetPasswordChar( 0 );
}
```

Asfel, la apăsarea butonului OK, textul introdus trebuie transferat din interfață în variabila membru `m_szPwd` specificată la începutul acestui capitol. Transferul valorilor din variabile membru în controalele din interfață şi vice versa se realizează folosind

funcția `UpdateData( BOOL )`. Dacă parametrul dat acestei funcții este `TRUE`, atunci se face un transfer de la interfața grafică în variabilele membru atașate. Altfel, transferul se face de la variabilele membru, la interfață. Funcționarea acestui mecanism este ilustrat și în Figura 4.2.

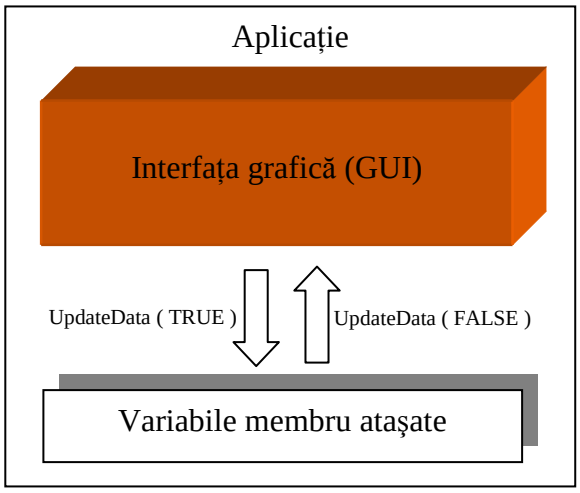


**Figura 4.2** Mecanismul transferului de date între interfața grafică și variabilele membru atașate controalelor din interfață

La apăsarea butonului OK, dacă parola introdusă este cea căutată, se afișează un mesaj și se închide aplicația:

```
void CTestTotalControlDlg::OnOK()
{
	// TODO: Add extra validation here
	UpdateData( TRUE );

	if ( m_szPwd == _T("parola de test!") )
	{
		AfxMessageBox( _T("Parola introdusa este corecta!"),
				MB_OK | MB_ICONINFORMATION );
		CDialog::OnOK();
	}
	else
	{
		AfxMessageBox( _T("Parola introdusa NU este corecta!"),
				MB_OK | MB_ICONEXCLAMATION );
	}
}
```

Funcția `AfxMessageBox` afișează mesaje într-o fereastră dialog. Tipul acestei ferestre și pictograma afișată se poate specifica printr-o combinație de parametrii, la fel ca și în cazul funcției `MessageBox`, prezentată în Capitolul 1.

Atenție! Păstrarea parolei în codul sursă, așa cum s-a realizat în exemplul anterior, nu este niciodată o practică bună fiindcă secvența de caractere este stocată în fișierul executabil în forma clară, fără să sufere vre-o modificare. Astfel, oricine poate descoperi parola prin simpla deschidere a executabilului cu un editor de texte.

De obicei aceste parole se păstrează în forma criptată sau sub forma *hash* (e.g. MD5, SHA-1) iar la testare mai întâi se calculează forma hash a parolei nou introduse care va reprezenta elementul de comparație cu parola stocată. Mai multe informații legate de acest subiect, în capitolele destinate funcțiilor *hash*.

## 3.7 Utilizarea ferestrelor speciale pentru preluarea parolelor

Cu toate că mecanismul utilizării unei funcții hash pentru stocarea parolei nu necesită stocarea acesteia în codul sursă, parola va fi stocată totuşi temporar în memoria asociată controlului fereastră „edit-box” la introducerea acesteia. Utilizarea acestor ferestre pentru introducerea parolelor nu este recomandată întrucât parolele nu sunt protejate, sunt păstrate în forma lor clară, iar memoria nu este ştearsă după distrugerea obiectului.

Sistemele bazate pe Windows NT pun la dispoziția utilizatorilor ferestre speciale pentru pentru introducerea parolelor. Acestea pot fi utilizate şi pentru autentificarea pe baza cardurilor electronice sau pe baza certificatelor. Crearea unei asemenea ferestre se realizează printr-un apel al funcției `CredUIPromptForCredentials`. Această funcție este declarată în *WinCred.h* şi este definită în *Credui.dll*. Pentru utilizarea ei, aplicațiile trebuie să includă fişierul *WinCred.h* şi să link-editeze aplicația cu *Credui.lib*.

Aceste ferestre sunt configurabile în totalitate, permit alegerea imaginii afişate, introducerea unui nume utilizator şi a unei parole sau doar a unei parole, etc. Fiind ferestre dialog modale, acestea nu permit activare a altor ferestre până la închiderea lor.

Un exemplu de utilizare a unei asemenea ferestre este următorul:

```
CREDUI_INFO info;
info.cbSize         = sizeof( CREDUI_INFO );
info.hwndParent     = GetSafeHwnd();
info.pszCaptionText = _T("Titlu fereastra introducere parola");
info.pszMessageText = _T("Messaj introducere parola");
info.hbmBanner      = NULL;

CString sTarget;
const DWORD dwErrorCode = 0;

_TCHAR pszUserName[ CREDUI_MAX_USERNAME_LENGTH + 1 ];
SecureZeroMemory( pszUserName, CREDUI_MAX_USERNAME_LENGTH + 1 );
_tcscpy_s( pszUserName, _T("Nume utilizator initial") );

_TCHAR pszPassword[ CREDUI_MAX_PASSWORD_LENGTH + 1 ];
SecureZeroMemory( pszPassword, CREDUI_MAX_PASSWORD_LENGTH + 1 );

BOOL bSaveChecked = false;

DWORD dwFlags = CREDUI_FLAGS_DO_NOT_PERSIST;

DWORD dwResult = ::CredUIPromptForCredentials(
                                    &info,
                                    sTarget,
                                    NULL,
                                    dwErrorCode,
                                    pszUserName,
                                    CREDUI_MAX_USERNAME_LENGTH,
```

```
                                        pszPassword,
                                        CREDUI_MAX_PASSWORD_LENGTH,
                                        &bSaveChecked,
                                        dwFlags );

  ... // Procesare parolă

// În final, memoria trebuie curățată
SecureZeroMemory( pszPassword, CREDUI_MAX_PASSWORD_LENGTH + 1 );
```

Primul parametru, `info`, asigură transmiterea informațiilor descriptive legate părintele ferestrei, titlul şi mesajul afişat. Totodată, prin această structură se poate transmite handle-ul unei imagini încărcate pentru a fi afişată în fereastra de introducere a parolei.

Al doilea parametru, `sTarget`, permite specificarea serverului de unde sunt încărcate informațiile de acces. Acest parametru va conține de regulă denumirea unui server, în cazul de față este şirul vid.

Următorul parametru, este unul rezervat pentru dezvoltări viitoare, valoarea lui trebuie să fie `NULL`.

Parametrul `dwErrorCode` asigură transferul stării anterioare a procesului de autentificare. De exemplu, fereastra poate fi afişată pentru o reintroducere a parolei după o autentificare fără succes care a eşuat cu un anumit cod de eroare.

Parametrul `pszUserName` reprezintă adresa de început a bufferului în care se va stoca denumirea utilizator introdusă. În exemplul dat, s-a stocat o valoare inițială în acest buffer, motiv pentru care fereastra va afişa deja un nume utilizator. Dacă nu se doreşte afişarea inițială a unui nume utilizator, se poate şterge apelul fucției `_tcscpy_s`. Pentru a preveni depăşirea dimensiunii bufferului la transferul valorii denumirii utilizatorului, parametrul următor reprezintă numărul de octeți ce pot fi stocați în buffer.

Următorul parametru, `pszPassword`, reprezintă adresa de început a bufferului în care se stochează parola introdusă. Similar parametrului `pszUserName`, parametrul următor a fost introdus pentru a preveni depăşirile de memorie şi reprezintă numărul octeților ce pot fi stocați în bufferul parolă.

Parametrul `bSaveChecked` stochează starea inițială a căsuței de salvare a parolei. La revenirea din funcție, acest parametru va conține starea căsuței de salvare. Căsuța de salvare poate fi ascunsă prin intermediul următorului parametru. În cazul în care căsuța nu este afişată, se consideră selectată.

Parametrul `dwFlags` permite definirea comportamentului ferestrei afişate. În exemplul dat, parametrul primeşte valoarea `CREDUI_FLAGS_DO_NOT_PERSIST`, prin care se asigură ascunderea căsuței de salvare a datelor introduse fără salvarea datelor de către serviciul de administrare a datelor sigure. Pe de altă parte, dacă se doreşte salvarea datelor fără afişarea căsuței de salvare se poate utiliza valoarea `CREDUI_FLAGS_PERSIST`. De fapt, acest parametru permite specificarea mai multor valori prin aplicarea operatorului sau pe biți. De exemplu, dacă se utilizează următoarea secvență, se va permite introducerea numai a cardurilor electronice de acces cu afişarea căsuței de salvare, fără salvarea datelor introduse:

```
DWORD dwFlags = CREDUI_FLAGS_DO_NOT_PERSIST |
                CREDUI_FLAGS_SHOW_SAVE_CHECK_BOX |
                CREDUI_FLAGS_REQUIRE_SMARTCARD;
```

Prin intermediul acestor opțiuni, datele introduce de utilizator pot fi totuşi salvate de aplicație, însă trebuie asigurată securitatea tuturor operațiilor ce implică manipularea lor.

În secvența de cod dată anterior s-a utilizat de fiecare dată `SecureZeroMemory` pentru a şterge *sigur* parola stocată în memoria aplicației. Prin utilizarea acestei funcții, față de funcțiile „clasice” `memset` sau `ZeroMemory`, se elimină eventualele optimizări effectuate de compilator care ar putea elimina codul de ştergere a memoriei.

**Temă.**

Cu toate că țelurile următoare ar trebui să fie suficiente pentru cele mai multe sisteme, există însă situații când trebuie luate o serie de alte măsuri pentru evitarea ocolirii aplicației de către utilizator. De exemplu, dacă se consideră o repornire a sistemului, aplicația va trebui repornită. Totodată, în cazul în care utilizatorul reuşeşte printr-o altă modalitate să închidă aplicația, administratorul ar trebui notificat de această situație.

Pentru ca totuşi, aplicația să nu poată fi oprită folosind un „Process Viewer”, sau alt program, se recomandă crearea aplicației sub forma unui serviciu pentru care utilizatorul normal să nu aibă drepturi de închidere.