Meniuri

- permit lansarea uşoară a unor comenzi
- orice fereastră poate avea asociat un meniu
- de regulă, meniurile sunt asociate ferestrei principale a aplicaţiei
- elemente ale unui meniu
  - comenzi
  - submeniuri
  - separatori

Exemplu de definire

```
IDM_MENU1 MENU
BEGIN
   POPUP "&File"
   BEGIN
      MENUITEM "&Open", ID_MYOPEN
      MENUITEM "&Save", ID_MYSAVE
      MENUITEM "E&xit", ID_MYEXIT
   END
END
```

Utilizarea meniurilor

- cum putem indica asocierea unui meniu cu o fereastră?

1. la stabilirea stilului ferestrei

```
wincl.lpszMenuName = menuName;
```

2. apelul unei funcţii specializate

```
HMENU LoadMenu(HINSTANCE
   hInstance, LPCTSTR
   lpMenuName);
```

Acceleratori

- taste sau combinaţii de taste care permit accesarea rapidă a unor comenzi
- de regulă asociate elementelor din meniuri
  - dar nu obligatoriu
- pot fi
  - taste comune
  - taste comune împreună cu Alt, Ctrl, Shift (sau combinaţii ale acestora)

Exemplu de definire

```
IDR_ACCELERATOR1 ACCELERATORS
BEGIN
   "^O", ID_MYOPEN, ASCII, NOINVERT
   "^S", ID_MYSAVE, ASCII, NOINVERT
   "^X", ID_MYEXIT, ASCII, NOINVERT
END
```

Utilizarea acceleratorilor

2 etape

1. încărcarea resursei (tabel de acceleratori)
   - înainte de intrarea în bucla de mesaje a ferestrei
2. prelucrarea mesajelor primite de fereastră pentru identificarea acceleratorilor folosiţi
   - în bucla de mesaje a ferestrei

- în ambele etape sunt apelate funcţii specializate

Exemplu tipic de utilizare

```
hAcc = LoadAccelerators(hInstance,
  MAKEINTRESOURCE(IDC_ACC1));
while(GetMessage(&msg,NULL,0,0))
  if(!TranslateAccelerator(msg.hwnd,
  hAcc,&msg)) {
     TranslateMessage(&msg);
     DispatchMessage(&msg);
     }
```

Preluarea mesajelor

- cum este notificată aplicaţia de declanşarea unei acţiuni prin intermediul meniurilor sau acceleratorilor?
- prin bucla de mesaje a ferestrei
  - la fel ca orice alt eveniment Windows
- este utilizat tot mesajul WM_COMMAND
  - la fel ca în cazul dialogurilor

Exemplu

```
case WM_COMMAND:
    switch(LOWORD(wParam)) {
      case ID_MYOPEN:
        // ...
        break;
      case ID_MYSAVE:
        // ...
        break;
      // ...
    }
```

IV. Afişarea în Windows

Contexte de dispozitiv

- DC (*device context*)
- obiecte ale sistemului Windows prin care se pot accesa interfeţele cu utilizatorul
  - ecranul/fereastra
  - imprimanta etc.
- tipul de *handle* asociat - HDC
- orice operaţie cu fereastra necesită obţinerea unui *handle* către aceasta

Afişarea ferestrei (1)

- se poate produce
  - de câte ori decide Windows că este necesar
    - minimizarea ferestrei sau acoperirea de către alta
    - readucerea ferestrei în prim-plan
    - redimensionarea ferestrei
    - etc.
  - la cererea explicită a programului
    - prin apelul funcţiei `InvalidateRect`

Afişarea ferestrei (2)

- în toate aceste cazuri, Windows trimite ferestrei mesajul WM_PAINT
  - prin tratarea sa decidem ce se afişează în fereastră
- trebuie să obţinem un *handle* către contextul de dispozitiv al ferestrei
  - iar la final să îl eliberăm
  - când nu mai este necesar

Obţinerea *handle*-ului

```
HDC BeginPaint(HWND hwnd,
  LPPAINTSTRUCT lpPaint);
```

- `hwnd` - *handle* către fereastră
- `lpPaint` - pointer către o structură cu informaţii despre fereastră
  - tipul structurii - PAINTSTRUCT
  - structura este completată de către funcţie

Eliberarea *handle*-ului

```
BOOL EndPaint(HWND hwnd, CONST
  PAINTSTRUCT *lpPaint);
```

- parametrii - aceiaşi ca la `BeginPaint`
- structura indicată de `lpPaint` trebuie să fie cea completată de `BeginPaint`

Forţarea redesenării ferestrei

```
BOOL InvalidateRect(HWND hwnd,
  CONST RECT* lpRect, BOOL
  bErase);
```

- `hwnd` - *handle* către fereastră
- `lpRect` - pointer către un dreptunghi conţinând coordonatele zonei de redesenat
  - NULL - întreaga fereastră
- `bErase` - indică dacă fundalul trebuie şters sau nu la redesenare

Funcţii de desenare

- odată obţinut handle-ul către contextul de dispozitiv, putem utiliza funcţiile de desenare
  - forme geometrice
  - text
  - manipularea caracteristicilor contextului de dispozitiv

Exemple

```
BOOL Rectangle(HDC hdc, int
  nLeftRect, int nTopRect, int
  nRightRect, int BottomRect);
BOOL Ellipse(HDC hdc, int
  nLeftRect, int nTopRect, int
  nRightRect, int BottomRect);
int DrawText(HDC hDC, LPCTSTR
  lpString, int nCount, LPRECT
  lpRect, UINT uFormat);
```

Elementele unui DC

- un context de dispozitiv are o serie de elemente
  - bitmap
  - font
  - pen
  - brush
- fiecare poate fi modificat dacă este necesar

Modificarea elementelor unui DC

- trebuie creat un element de acelaşi tip, având caracteristicile dorite
- apoi elementul curent din DC este înlocuit cu elementul nou creat
  - cu ajutorul metodei `SelectObject`
  - aceasta returnează un *handle* către obiectul deselectat din DC, pentru a-l putea refolosi ulterior

Utilizarea de elemente noi

- predefinite
  - pot fi obţinute cu ajutorul funcţiei `GetStockObject`
- create prin program
  - cu ajutorul unor funcţii specializate
  - exemple: `CreateBitmap, CreateFont, CreateSolidBrush, CreatePen`
  - după utilizare, trebuie distruse prin apelul funcţiei `DeleteObject`

Afişare fără WM_PAINT (1)

- putem utiliza funcţiile de afişare şi în alte situaţii decât ca răspuns la WM_PAINT?
- da
- dar efectul lor poate dispărea la următoarea redesenare
  - deci la primirea unui nou mesaj WM_PAINT
- pot fi utile şi pentru alte sarcini decât afişarea propriu-zisă

Afişare fără WM_PAINT (2)

- handle-ul către DC este obţinut/eliberat cu ajutorul altor funcţii

```
HDC GetDC(HWND hwnd);
int ReleaseDC(HWND hwnd, HDC
  hDC);
```

- în rest, utilizarea este similară

DC-uri în memorie (1)

- un context de dispozitiv este un obiect Windows, stocat în memorie
- deci putem defini DC-uri care nu au legătură directă cu un dispozitiv fizic
- exemplu: menţinem o imagine complexă într-un DC din memorie
  - când este necesar, o copiem în DC-ul ferestrei
  - avantaj: afişarea este mult mai rapidă

DC-uri în memorie (2)

- nu toate DC-urile sunt identice
- trebuie ca DC-ul din memorie să fie compatibil cu DC-ul ferestrei
- deci trebuie creat printr-o funcţie specializată

```
HDC CreateCompatibleDC(HDC
  hdc);
```