www.mohandesyar.com

عنـــــــوان

آشنایی با برنامه نویسی اسمبلی ویندوز

پژوهش و نگارش : وحید نصیری

بهار ۱۳۸۴

قسمت دوم

نگارش ۱

مقدمه :

بهترین روش فراگیری یک زبان برنامه نویسی، سیستم : "مطالعه – برنامه نویسی – مطالعه" است. به همین جهت در قسمت دوم آشنایی با برنامه نویسی اسمبلی ویندوز ، در ابتدا یک مثال کامل و قابل اجرا ارائه شده و سپس نکات مهم کدنویسی آن مورد بررسی قرار می گیرند.تمامی کدهای ذکر شده در متن ، ضمیمه این مجموعه می‌باشند.

مثال ۱ – پنجره‌ها در ویندوز

برنامه زیر را در نظر بگیرید:

```
.486
.model flat, stdcall
option casemap:none

includelib \masm32\lib\kernel32.lib
includelib \masm32\lib\user32.lib
includelib \masm32\lib\gdi32.lib

include \masm32\include\windows.inc
include \masm32\include\kernel32.inc
include \masm32\include\user32.inc
include \masm32\include\gdi32.inc
WinMain PROTO STDCALL   :DWORD, :DWORD, :DWORD, :DWORD
WndProc PROTO STDCALL   :DWORD, :DWORD, :DWORD, :DWORD

.data?
hInstance       dd      ?


.data
ClassName       db      "FirstWindowClass",0
AppName         db      "FirstWindow",0

.code
start:

    invoke  GetModuleHandle, NULL
    mov     hInstance, eax
    invoke  WinMain, hInstance, NULL, NULL, SW_SHOWNORMAL
    invoke  ExitProcess, NULL

WinMain proc hInst:HINSTANCE,hPrevInst:HINSTANCE,CmdLine:LPSTR,CmdShow:DWORD
    LOCAL wc:WNDCLASSEX
    LOCAL hwnd:DWORD
    LOCAL msg:MSG
    mov     wc.cbSize,SIZEOF WNDCLASSEX
    mov     wc.style,   CS_HREDRAW or CS_VREDRAW
    mov     wc.lpfnWndProc, OFFSET WndProc
```

```
    mov     wc.cbClsExtra,NULL
    mov     wc.cbWndExtra,NULL
    push    hInst
    pop     wc.hInstance
    mov     wc.hbrBackground,COLOR_WINDOW
    mov     wc.lpszMenuName,NULL
    mov     wc.lpszClassName,OFFSET ClassName
    invoke  LoadIcon,NULL,IDI_APPLICATION
    mov     wc.hIcon,   eax
    mov     wc.hIconSm, eax
    invoke  LoadCursor,NULL,IDC_ARROW
    mov     wc.hCursor,eax
    invoke  RegisterClassEx,    addr wc

    INVOKE CreateWindowEx,NULL,ADDR ClassName,ADDR AppName,\
           WS_OVERLAPPEDWINDOW-WS_SIZEBOX-WS_MAXIMIZEBOX,CW_USEDEFAULT,\
           CW_USEDEFAULT,400,300,NULL,NULL,\
           hInst,NULL
    mov   hwnd,eax
    invoke ShowWindow, hwnd,SW_SHOWNORMAL
    invoke UpdateWindow, hwnd
    .WHILE TRUE
        invoke GetMessage, ADDR msg,NULL,0,0
        .BREAK .IF (!eax)
                invoke TranslateMessage, ADDR msg
                invoke DispatchMessage, ADDR msg
    .ENDW
    mov     eax,msg.wParam
    ret
WinMain endp

WndProc proc hWnd:DWORD, uMsg:DWORD, wParam:DWORD, lParam:DWORD
mov eax, uMsg
.IF eax==WM_CREATE
    invoke  MessageBox, NULL, ADDR AppName, ADDR AppName, NULL
.ELSEIF eax==WM_DESTROY
    invoke PostQuitMessage, NULL
.ELSE
    invoke DefWindowProc, hWnd, uMsg, wParam, lParam
.ENDIF
ret
WndProc endp
end start
```

در ادامه قصد بررسی کامل کد فوق ، که پنجره‌ای را نمایش می‌دهد ، داریم.

به سادگی می توان حدس زد که چرا به windows ، ویندوز می گویند! تحت ویندوز (در حالت برنامه‌های رومیزی) دو نوع برنامه در حالت‌های Console و GUI قابل تصور هستند.برنامه‌های حالت console شبیه به برنامه‌های داس در یک پنجره داس مانند اجرا می‌شوند (هر چند ۳۲ بیتی هستند). اما اکثر برنامه‌های مورد استفاده دارای GUI (graphical user interface) و یا بقولی"رابط گرافیکی کاربر" ، جهت برهم‌کنش با کاربر هستند. این رابط گرافیکی توسط پنجره‌ها (windows) ایجاد می‌شوند.تقریبا هر چیزی را که در ویندوز مشاهده می کنید از یک پنجره تشکیل شده است.

برای ایجاد یک پنجره ، مراحل زیر باید صورت گیرد:

- دریافت دستگیره وهله برنامه ایجاد شده (instance handle). در ویندوز های ۳۲ بیتی این عدد در حقیقت آدرس خطی برنامه در حافظه است.
- دریافت دستورات خط فرمان (اجباری نیست. تنها در صورتیکه برنامه شما این مورد را نیاز داشته باشد باید پردازش شود. اینکار با استفاده از تابع GetCommandLine قابل انجام است.)
- رجیستر کردن کلاس پنجره
- ایجاد پنجره
- نمایش پنجره بر روی صفحه میزکاری
- به روز رسانی پنجره
- ایجاد حلقه ای بی پایان جهت پردازش پیغام های رسیده به برنامه
- پردازش پیغام های رسیده توسط رویه پنجره
- خاتمه بخشیدن به اجرای برنامه ، درصورتیکه کاربر پنجره را ببندد.

کلاس‌های ایجاد پنجره :

هر پنجره دارای یک کلاس است. برای پنجره‌ی والد می‌توان کلاسی دلخواه را ایجاد کرد اما برای کنترل هایی که بر روی فرم والد ایجاد می شوند ، از کلاسهای استانداردی مانند EDIT ، STATIC و غیره می‌توان استفاده نمود.

ساختارها:

کلاس پنجره با کمک تابع RegisterClassEx رجیستر می‌شود.این تابع به صورت زیر تعریف می گردد:

```
ATOM RegisterClassEx(
CONST WNDCLASSEX *lpwcx // address of structure with class data
);
```

lpwcx: Points to a WNDCLASSEX structure. You must fill the structure with the appropriate class attributes before passing it to the function.

تنها پارامتر مورد استفاده درآن یک اشاره‌گر به ساختار WNDCLASSEX می‌باشد.

در ادامه مرور مجددی خواهیم داشت بر نحوه تعریف و استفاده از ساختارها:

یک ساختار مجموعه ای از متغیرها (داده‌ها) است.با رهنمود STRUCT به شکل زیر قابل تعریف است:

```
SOMESTRUCTURE   STRUCT
   dword1 dd ?
   dword2 dd ?
   some_word dw ?
   abyte db ?
   anotherbyte db ?
SOMESTRUCTURE   ENDS
```

برای مقدار دهی به آن در قسمت داده های کد داریم:

```
Initializedstructure  SOMESTRUCTURE <100,200,10,'A',90h>
```

و یا اگر بخواهیم آنرا در قسمت داده‌های مقدار دهی اولیه نشده تعریف کنیم:

```
UnInitializedstructure  SOMESTRUCTURE  <>
```

در این حالت صرفا حافظه ی مورد نیاز پیش بینی شده و متغیرهای ساختار با صفر مقدار دهی می‌شوند.

پس از تعریف ساختار برای استفاده از آن به صورت زیر عمل می‌شود:

```
Mov  eax, Initializedstructure.some_word
; eax will hold 10 now
Inc  UnInitializedstructure.dword1
; the dword1 of the structure is increased by one
```

نحوه ذخیره شدن اعضای ساختار فوق در حافظه به شکل زیر است:

موقعیت در حافظه	محتوا
offset of Initializedstructure	100 (dword, 4 bytes)
offset of Initializedstructure + 4	200 (dword, 4 bytes)
offset of Initializedstructure + 8	10 (word, 2 bytes)
offset of Initializedstructure + 10	65 or 'A' (1 byte)
offset of Initializedstructure + 11	90h (1 byte)

ساختار WNDCLASSEX :

با توجه به MSDN ، تعریف این ساختار استاندارد در ویندوز به شکل زیر است:

```
typedef struct _WNDCLASSEX { // wc
   UINT cbSize;
   UINT style;
   WNDPROC lpfnWndProc;
   int cbClsExtra;
   int cbWndExtra;
   HANDLE hInstance; HICON hIcon;
   HCURSOR hCursor; HBRUSH hbrBackground;
   LPCTSTR lpszMenuName; LPCTSTR lpszClassName;
   HICON hIconSm;
} WNDCLASSEX;
```

توضیحاتی در مورد اعضای آن :

عضو	توضیح
cbSize	اندازه ساختار WNDCLASSEX. به صورت زیر محاسبه می شود: mov ws.cbSize, **SIZEOF** WNDCLASSEX
style	سبک پنجره را تعیین می کند (برای مثال داشتن میله‌های لغزنده و غیره).
lpfnWndProc	اشاره‌گری به تابع پنجره . (در ادامه بیشتر توضیح داده خواهد شد) lpfn = long pointer to function تحت ویندوز اشاره‌گرهای دور"far" و نزدیک "near" وجود ندارند (بدلیل مدل معماری تخت). در اینجا تنها اشاره‌گر وجود دارد.
cbClsExtra	بایت‌های اضافی که نیاز است تخصیص داده شود. (برای ما اهمیتی ندارد)
cbWndExtra	بایت‌های اضافی که نیاز است برای نمونه ایجاد شده کلاس پنجره ، تخصیص داده شود. (برای ما اهمیتی ندارد)
hInstance	دستگیره‌ای به وهله ایجاد شده از برنامه. با استفاده از تابع GetModuleHandle بدست می‌آید.
hIcon	دستگیره‌ای به منبع آیکون مورد استفاده برای پنجره.
hCursor	دستگیره‌ای به منبع مکان نما مورد استفاده برای پنجره.
hbrBackground	دستگیره‌ای به برسی که برای نقاشی پنجره بکار می‌رود! و یا یکی از ثوابت زیر: COLOR_WINDOW, COLOR_BTNFACE , COLOR_BACKGROUND
lpszMenuName	اشاره‌گری به رشته‌ای مختوم به نال که نام منبع منوی مورد استفاده برای پنجره را مشخص می‌کند.همچنین ID یک منبع نیز می‌تواند باشد.
lpszClassName	اشاره‌گری به رشته‌ای مختوم به نال که نام کلاس پنجره را مشخص می‌کند.
hIconSm	دستگیره‌ای به منبع آیکونی کوچک که مرتبط است به کلاس پنجره.

رجیستر کردن یک کلاس:

پس از تعاریف اولیه کد ، مانند اضافه کردن کتابخانه‌های مورد نیاز که پیشتر در مورد آنها کاملا بحث گردید ، نوبت به رجیستر کردن کلاس پنجره در رویه WinMain است.

توابع مورد استفاده مانند RegisterWindowClassEx, CreateWindowEx در کتابخانه user32.dll و تابع ExitProcess در کتابخانه kernel32.dll واقع شده است. بنابراین به سادگی می‌توان تشخیص داد که کتابخانه‌هایی مانند user32.lib و فایل inc مربوطه باید به برنامه ضمیمه شوند.

در این رویه ،کلاس پنجره ایجاد و مقدار دهی اولیه می‌شود.

```
WinMain proc hInst:DWORD, hPrevInst:DWORD, CmdLine:DWORD, CmdShow:DWORD

ret
WinMain endp
```

از این رویه هیچگاه مستقیما استفاده نخواهد شد و صرفا روشی است متداول جهت آغاز برنامه‌های ویندوز.++VC پارامترهای این تابع را به صورت خودکار مقدار دهی می کند، اما در اینجا باید اینکار را به صورت دستی انجام دهیم.
hInst و یا (Instance handle (= module handle ، دستگیره ای به وهله ایجاد شده از برنامه است. CmdShow ، پرچمی است که نحوه نمایش پنجره را نمایش می‌دهد (اطلاعات بیشتر در این مورد را در MSDN درباره ShowWindow می‌توان یافت).

در ابتدای کد برنامه (پیش از تابع WinMain) داریم:

```
invoke  GetModuleHandle, NULL
mov     hInstance, eax
invoke  WinMain, hInstance, NULL, NULL, SW_SHOWNORMAL
invoke  ExitProcess, NULL
```

تابع getmodulehandle ، دستگیره وهله ایجاد شده از برنامه را بر می‌گرداند. همانطور که پیشتر نیز ذکر شد رویه استاندارد توابع API ویندوز ، این است که خروجی خود را در رجیستر eax قرار می‌دهند (خط دوم در کد فوق).
سپس تابع WinMain با مقادیر پیش فرض و همچنین مورد نیاز آن فراخوانی شده و در آخر کد خاتمه می‌یابد.
همانطور که دقت کردید پارامترهای دوم و سوم این تابع با Null مقدار دهی شده اند.پارامتر دوم مساوی hPrevInst است. در ویندوزهای ۳۲ بیتی وهله‌ی پیشینی وجود ندارد.بنابراین مقدار این آرگومان همواره مساوی صفر است و هر برنامه در فضای آدرس خود تنها می‌باشد. این آرگومان به ارث رسیده از ویندوزهای ۱۶ بیتی است. زمانی که تمام وهله‌های برنامه‌ها در یک فضای آدرس اجرا می‌شدند.پارامتر سوم هم مربوط به دریافت دستورات خط فرمان است. در حالت کلی تعریف این تابع به صورت زیر است:

```
WinMain proc Inst:HINSTANCE, hPrevInst:HINSTANCE, CmdLine:LPSTR, CmdShow:DWORD
```

در بدنه رویه WinMain ، ابتدا اعضای ساختار WNDCLASSEX مقدار دهی اولیه شده و کلاس پنجره رجیستر می‌شود.تنها یادآوری یک نکته در مورد آن حائز اهمیت است:

```
Push  hInst
pop    wc.hInstance
```

چرا ما در اینجا از دستور زیر استفاده نکردیم؟

```
Mov  wc.hInstance, hInst
```

زیرا دستورالعمل Mov امکان انتقال حافظه را از یک مکان به مکانی دیگر ، به صورت مستقیم مهیا نمی کند. بنابراین داده ابتدا بر روی پشته قرار داده شده و سپس در مقصد بازیابی می‌شود.

رهنمود LOCAL ، جهت تخصیص حافظه از پشته ، جهت متغیرهای محلی بکار می‌رود. تعریف این متغیرها باید بلافاصله پس از تعریف PROC ، باشد.بنابراین LOCAL wc:WNDCLASSEX ، به MASM می گوید که از پشته به اندازه ساختار WNDCLASSEX ، جهت متغیر wc ، حافظه اختصاص دهد. سپس استفاده از wc در کد به سادگی امکان پذیر بوده و نیازی به نگرانی در مورد اعمال پشته در اینجا نیست.

فقط باید دقت داشت که متغیرهای محلی پس از اتمام کار رویه تخریب می‌شوند و خارج از رویه قابل دسترسی نیستند.

در ادامه ، پنجره نیاز به یک آیکون و مکان‌نما دارد. در اینجا تابع LoadIcon دو پارامتر می‌پذیرد. پارامتر اول ، module handle بوده و پارامتر دوم ، اشاره‌گری است به رشته ای که مساوی نام منبع آیکون است (و یا یک ID). اگر پارامتر اول را مساوی Null در نظر بگیریم ، پارامتر دوم را می‌توان از آیکون های استاندارد انتخاب کرد.

```
invoke LoadIcon, NULL, IDI_APPLICATION
mov  wc.hIcon, eax
mov  wc.hIconSm, eax

invoke LoadCursor,NULL,IDC_ARROW
mov  wc.hCursor,eax
```

و در پایان کدهای رجیستر کردن کلاس پنجره داریم:

```
Invoke  RegisterClassEx, ADDR wc
```

ایجاد پنجره :

پس از ثبت کلاس پنجره می‌توان از آن استفاده کرد.

```
HWND CreateWindowEx(
   DWORD dwExStyle, // extended window style
   LPCTSTR lpClassName, // pointer to registered class name
   LPCTSTR lpWindowName, // pointer to window name
   DWORD dwStyle, // window style
   int x, // horizontal position of window
   int y, // vertical position of window
   int nWidth, // window width
   int nHeight, // window height
   HWND hWndParent, // handle to parent or owner window
   HMENU hMenu, // handle to menu, or child-window identifier
   HINSTANCE hInstance, // handle to application instance
   LPVOID lpParam // pointer to window-creation data
);
```

نحوه فراخوانی آن در برنامه :

```
.data
AppName "FirstWindow",0
.code
INVOKE CreateWindowEx,NULL,ADDR ClassName,ADDR AppName,\
WS_OVERLAPPEDWINDOW,CW_USEDEFAULT,\
CW_USEDEFAULT,400,300,NULL,NULL,\
hInst,NULL
mov hwnd, eax
invoke ShowWindow, hwnd, SW_SHOWNORMAL
invoke UpdateWindow, hwnd
```

اگر نیاز باشد تا یک سطر از کد در دو یا چند سطر ادامه یابد از "\" در انتهای سطر استفاده خواهد شد.

از خروجی تابع CreateWindowEx (دستگیره ای به پنجره ایجاد شده) ، جهت نمایش آن استفاده خواهیم کرد.

حلقه پیغام‌ها:

نحوه ردوبدل کردن اطلاعات بین پنجره، برنامه و سایر برنامه‌های ویندوز با استفاده از پیغام‌ها صورت می‌گیرد.هر پنجره دارای حلقه‌ای بی پایان جهت بررسی پیغام‌های رسیده به آن می‌باشد.در صورتیکه پیغامی به پنجره برسد ، در این حلقه بررسی شده و به تابع DispatchMessage ارسال می‌شود.این تابع ، رویه پنجره شما را فراخوانی می‌کند.

اگر تابع GetMessage ، پیغام WM_QUIT را دریافت کند (که به معنای درخواست خاتمه بخشیدن به برنامه است)، مقدار صفر را بازگشت داده و حلقه خاتمه می‌یابد. تابع TranslateMessage ، فشرده شدن کلیدها را به پیغام‌ها ترجمه می‌کند.

این حلقه در تمامی برنامه‌ها مورد استفاده قرار می‌گیرد.

رویه پنجره:

پیغام ، به رویه پنجره فرستاده می‌شود. رویه پنجره باید همواره به این شکل باشد:

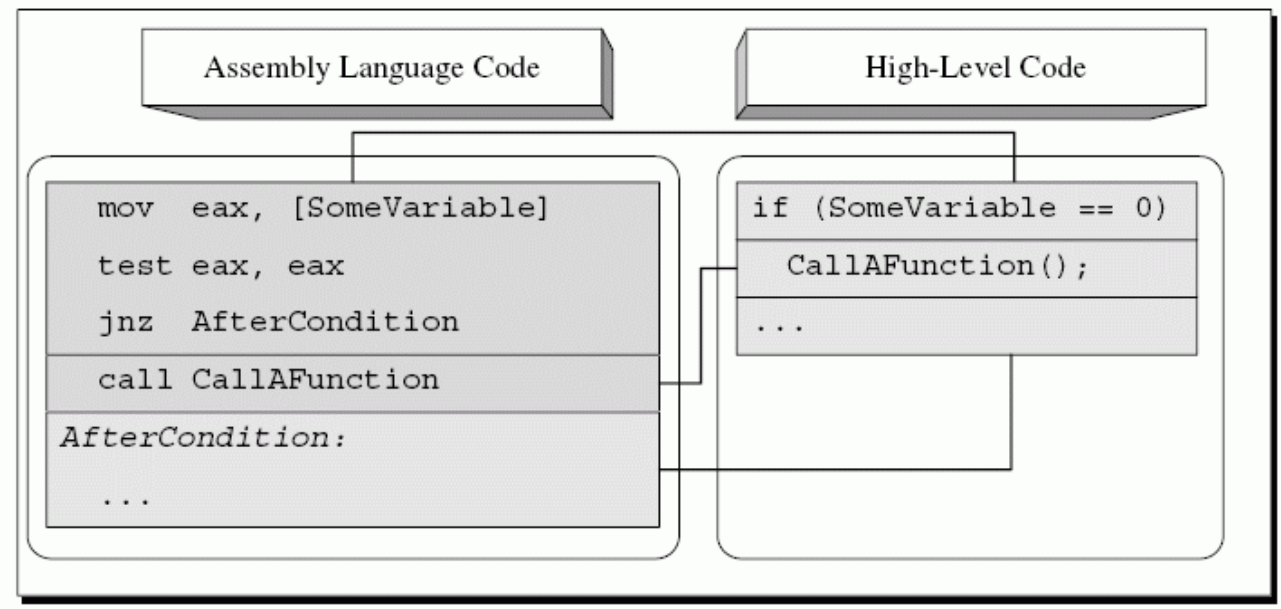
```
WndProc PROTO STDCALL :DWORD, :DWORD, :DWORD, :DWORD

.code

WndProc proc hWnd:DWORD, uMsg:DWORD, wParam:DWORD, lParam:DWORD

   mov eax, uMsg
   .IF eax==XXXX
   .ELSEIF eax==XXXX
   .ELSE
      invoke DefWindowProc, hWnd, uMsg, wParam, lParam
   .ENDIF

ret
WndProc endp
```

پیغامهایی که پنجره آنها را پردازش نخواهد کرد باید به تابع DefWindowProc فرستاده شوند.

در پایان ، بررسی دیس اسمبلی فایل exe ایجاد شده (برای مثال توسط OllyDbg) بسیار جالب بوده و در مورد درک نحوه ترجمه و مقدار دهی رجیسترها و پشته و غیره در برنامه بسیار مؤثر است (و نیازی به تکرار آن در اینجا نمی‌باشد).

تا اینجا شاید اینطور به نظر آید که صرفا ایجاد و نمایش یک پنجره ساده بسیار پیچیده و طولانی است. باید دقت داشت که قسمت عمده‌ای از کد فوق صرفا یک قالب (template) می‌باشد که در تمامی برنامه‌های مشابه تکرار خواهد گردید.

مقایسه کد زبان سی و کد اسمبلی یک شرط ساده یک طرفه



از آنجائیکه اکثر برنامه نویسان قبل از آشنایی با زبان اسمبلی ، با زبانهای سطح بالاتری مانند C آشنا می شوند ، دانستن معادلهای دستورات این زبانها به زبان اسمبلی ضروری می‌نماید.بنابراین در ادامه پس از هر مثال یک نمونه از این موارد نیز بررسی خواهند گردید.

در شکل فوق ، شرط ساده زیر به زبان اسمبلی نمایش داده شده است:

```
if (SomeVariable == 0)
        CallAFunction();
```

همانطور که در قسمت قبل نیز ذکر شد، از دستورالعمل TEST برای مقایسه تساوی با صفر EAX استفاده می‌شود.اینکار با انجام عملیات بیتی AND بر روی eax و تنظیم پرچم نهایی (ZF) صورت می گیرد.در ادامه اگر eax مساوی صفر نباشد، پرش صورت خواهد گرفت.

مثال ۲- ترسیم متن

متن در ویندوز نوعی شیء GUI است.هر کاراکتر از تعداد زیادی نقطه (pixel) تشکیل شده است.به همین جهت از واژه‌ی ترسیم بجای نوشتن استفاده گردید.در حالت معمول ، ترسیم متن در ناحیه جاری برنامه انجام می شود (هر چند ترسیم متن خارج از ناحیه برنامه نیز میسر است.)

ترسیم متن در ویندوز با داس به شدت متفاوت است. تحت داس ابعاد صفحه نمایش را در حالت معمول می توان ۸۰ در ۲۵ در نظر گرفت.اما در ویندوز صفحه نمایش بین برنامه‌های مختلف به اشتراک گذاشته شده و باید قوانینی را درنظر گرفت تا برنامه‌ها، متون مربوط به خود را بر روی یکدیگر جای نویسی نکنند.

به همین جهت ویندوز ناحیه ترسیمی هر برنامه را به ابعاد پنجره‌های در حال نمایش آن محدود می‌کند. اندازه‌ی پنجره‌ها توسط کاربران قابل تغییر است. بنابراین باید به صورت پویا به این تغییر اندازه عکس العمل نشان داد.
قبل از ترسیم متن بر روی پنجره باید از ویندوز کسب اجازه شود! بله، شما همانند داس کنترل کامل و مطلقی بر روی صفحه نمایش ندارید. در این حالت شما از ویندوز در مورد ترسیم بر روی محدوده مجاز پنجره خود درخواست مجوز می‌کنید. سپس ویندوز اندازه پنجره ، فونت ، رنگ و سایر خواص GDI را مشخص کرده و دستگیره ای به آنرا به برنامه شما بر می‌گرداند (handle to device context). سپس از device context (بافت ابزار) بعنوان مجوزی برای ترسیم بر روی پنجره می‌توان استفاده کرد.

Device context چیست؟ صرفا ساختاری داده ای نگهداری شده توسط ویندوز است. device context وابسته به ابزاری خاص مانند چاپگر و یا نمایشگر ویدیویی است. تعدادی از خواص موجود در بافت افزار ،گرافیکی هستند مانند رنگ، فونت و غیره. اینها همچنین دارای مقادیر پیش فرضی نیز هستند که در صورت نیاز می‌توان تغییرشان داد.با داشتن این مقادیر پیش فرض دیگر نیازی نیست تا به ازای هر فراخوانی توابع GDI ، تمام خواص آنها را نیز مقدار دهی کرد.

از بافت ابزار می‌توان بعنوان محیطی آماده شما برای استفاده شما توسط ویندوز یاد کرد. هنگامیکه برنامه ای نیاز دارد تا ترسیمی را انجام دهد ، باید در ابتدا دستگیره ای (handle) به بافت ابزار دریافت نماید.به صورت معمول از روش‌های زیر برای این مقصود می‌توان سود جست:

- فراخوانی **BeginPaint** در پاسخ به پیغام **WM_PAINT**
- فراخوانی **GetDC** در پاسخ به سایر پیغام‌ها
- فراخوانی **CreateDC** برای ایجاد بافت ابزاری دلخواه

در اینجا یک مورد را نباید فراموش کرد. پس از استفاده از دستگیره بافت ابزار ، باید آنرا به سیستم عامل بازگشت داد و عموما اینکار پس از پایان پردازش یک پیغام رسیده باید انجام شود. دستگیره ای را که به سبب یک درخواست دریافت کرده اید ، به جهت درخواستی دیگر بازگشت ندهید.

ویندوز توسط پیغام WM_PAINT به پنجره‌ای خاص اعلام می‌کند که اکنون زمان ترسیم مجدد ناحیه مربوط به خود است. ویندوز محتوای هیچ پنجره ای را ذخیره و بازیابی نمی‌کند. اما اگر موقعیتی رخ دهد که نیاز به ترسیم مجدد پنجره‌ای باشد، با پیغامی مناسب این مطلب را به پنجره اعلام خواهد کرد (برای مثال پوشانده شدن قسمتی از پنجره توسط پنجره‌ای دیگر) . این وظیفه‌ی پنجره است تا پس از دریافت این پیغام ، توسط کدی مناسب ترسیم مجدد ناحیه مربوطه را انجام دهد. پردازش پیغام WM_PAINT رسیده ، همانند مثال قبل باید در رویه پنجره صورت گیرد.

مفهوم دیگری که باید با آن آشنا شد ، invalid rectangle است.مستطیل غیرمجاز ، کوچکترین ناحیه‌ای در پنجره است که نیاز به ترسیم مجدد دارد. اگر ویندوز چنین ناحیه ای را تشخیص دهد ، پیغام WM_PAINT را به پنجره ارسال می‌کند. در جهت پاسخ دهی به پیغام WM_PAINT ، ساختار paintstruct که مختصات مستطیل غیرمجاز را در خود دارد قابل دریافت است.

برای Validate کردن این ناحیه invalid می‌توان تابع BeginPaint را فراخوانی کرد.اگر این اعتباردهی صورت نگیرد حداقل باید توابع DefWindowProc و یا ValidateRect فراخوانی شوند.در غیراینصورت ویندوز مرتبا پیغام WM_PAINT را برای پنجره ارسال خواهد کرد.

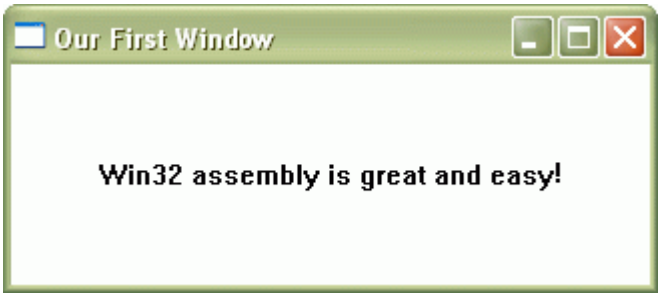
جهت پاسخ دهی به پیغام WM_PAINT به صورت زیر می‌توان عمل کرد:

- ابتدا دستگیره بافت ابزار توسط **BeginPaint** باید دریافت شود.
- ترسیم ناحیه کلاینت
- بازگشت دادن دستگیره بافت افزار به سیستم عامل توسط فراخوانی تابع **EndPaint** .

لازم به ذکر است که اعتباردهی صریح مستطیل غیرمجاز ضرورتی ندارد. اینکار به صورت خودکار توسط فراخوانی BeginPaint قابل انجام است.بین دو فراخوانی BeginPaint-Endpaint می‌توان از انواع توابع GDI جهت ترسیم بر روی پنجره استفاده کرد.تقریبا تمام آنها جهت کارکرد نیاز به دستگیره‌ای به بافت ابزار دارند.

محتوای برنامه :

در این برنامه قصد داریم عبارت Win32 assembly is great and easy! را در میانه پنجره ترسیم نماییم.



شکل ۱- ترسیم متن

```
.386
.model flat,stdcall
option casemap:none

WinMain proto :DWORD,:DWORD,:DWORD,:DWORD

include \masm32\include\windows.inc
include \masm32\include\user32.inc
includelib \masm32\lib\user32.lib
include \masm32\include\kernel32.inc
includelib \masm32\lib\kernel32.lib

.DATA
ClassName db "SimpleWinClass",0
AppName  db "Our First Window",0
OurText  db "Win32 assembly is great and easy!",0

.DATA?
hInstance HINSTANCE ?
CommandLine LPSTR ?

.CODE
start:
    invoke GetModuleHandle, NULL
    mov    hInstance,eax
    invoke GetCommandLine
    mov CommandLine,eax
    invoke WinMain, hInstance,NULL,CommandLine, SW_SHOWDEFAULT
    invoke ExitProcess,eax

WinMain proc hInst:HINSTANCE, hPrevInst:HINSTANCE, CmdLine:LPSTR, CmdShow:DWORD
    LOCAL wc:WNDCLASSEX
    LOCAL msg:MSG
    LOCAL hwnd:HWND
    mov   wc.cbSize,SIZEOF WNDCLASSEX
    mov   wc.style, CS_HREDRAW or CS_VREDRAW
    mov   wc.lpfnWndProc, OFFSET WndProc
```

```
    mov   wc.cbClsExtra,NULL
    mov   wc.cbWndExtra,NULL
    push  hInst
    pop   wc.hInstance
    mov   wc.hbrBackground,COLOR_WINDOW+1
    mov   wc.lpszMenuName,NULL
    mov   wc.lpszClassName,OFFSET ClassName
    invoke LoadIcon,NULL,IDI_APPLICATION
    mov   wc.hIcon,eax
    mov   wc.hIconSm,eax
    invoke LoadCursor,NULL,IDC_ARROW
    mov   wc.hCursor,eax
    invoke RegisterClassEx, addr wc
    invoke CreateWindowEx,NULL,ADDR ClassName,ADDR AppName,\
          WS_OVERLAPPEDWINDOW,CW_USEDEFAULT,\
          CW_USEDEFAULT,CW_USEDEFAULT,CW_USEDEFAULT,NULL,NULL,\
          hInst,NULL
    mov   hwnd,eax
    invoke ShowWindow, hwnd,SW_SHOWNORMAL
    invoke UpdateWindow, hwnd
       .WHILE TRUE
             invoke GetMessage, ADDR msg,NULL,0,0
             .BREAK .IF (!eax)
             invoke TranslateMessage, ADDR msg
             invoke DispatchMessage, ADDR msg
       .ENDW
       mov     eax,msg.wParam
       ret
WinMain endp

WndProc proc hWnd:HWND, uMsg:UINT, wParam:WPARAM, lParam:LPARAM
    LOCAL hdc:HDC
    LOCAL ps:PAINTSTRUCT
    LOCAL rect:RECT
    .IF uMsg==WM_DESTROY
       invoke PostQuitMessage,NULL
    .ELSEIF uMsg==WM_PAINT
       invoke BeginPaint,hWnd, ADDR ps
       mov    hdc,eax
       invoke GetClientRect,hWnd, ADDR rect
       invoke DrawText, hdc,ADDR OurText,-1, ADDR rect, \
             DT_SINGLELINE or DT_CENTER or DT_VCENTER
       invoke EndPaint,hWnd, ADDR ps
    .ELSE
       invoke DefWindowProc,hWnd,uMsg,wParam,lParam
       ret
    .ENDIF
    xor   eax, eax
    ret
WndProc endp
end start
```

بررسی کد فوق:

عمده برنامه فوق همانند مثال قبلی است. دراینجا بیشتر بر روی تفاوت‌ها و نکات مهم تمرکز خواهد گردید.

```
LOCAL hdc:HDC
LOCAL ps:PAINTSTRUCT
LOCAL rect:RECT
```

اینها متغیرهای محلی هستند که در قسمت WM_PAINT ، توسط توابع GDI بکارگرفته خواهند شد.دستگیره بافت ابزار دریافت شده از تابع BeginPaint در متغیر hdc ذخیره می‌گردد.ps ازنوع PAINTSTRUCT است.بطور معمول از ps مستقیما استفاده نمی‌شود. این متغیر به تابع BeginPaint ارسال شده و توسط ویندوز به صورت خودکار پر می‌گردد. در آخر ps به تابع EndPaint جهت خاتمه ترسیم ارسال می‌شود.rect از نوع ساختار RECT بوده و به صورت زیر تعریف می‌شود:

```
RECT Struct
   left        LONG ?
   top         LONG ?
   right       LONG ?
   bottom      LONG ?
RECT ends
```

یادآوری:

مبدا مختصات در اینجا گوشه سمت چپ بالا می‌باشد. بنابراین y=10 در پایین y=0 قرار می‌گیرد.

```
invoke BeginPaint,hWnd, ADDR ps
mov    hdc,eax
invoke GetClientRect,hWnd, ADDR rect
invoke DrawText, hdc,ADDR OurText,-1, ADDR rect, \
       DT_SINGLELINE or DT_CENTER or DT_VCENTER
invoke EndPaint,hWnd, ADDR ps
```

در جهت پاسخگویی به پیغام WM_PAINT ، تابع BeginPaint با دو پارامتر فراخوانی می‌شود.پارامتر اول دستگیره ای است به پنجره مورد نظر و پارامتر دوم متغیر ps مقدار دهی اولیه نشده می‌باشد.پس از فراخوانی موفقیت آمیز این تابع ، خروجی آن به صورت متداول در رجیستر eax قرار می‌گیرد (و معادل است با دستگیره‌ای به بافت ابزار). در ادامه برای دریافت ابعاد کلاینت مورد نظر برای ترسیم بر روی آن ، تابع GetClientRect فراخوانی می‌گردد.این ابعاد در متغیر rect قرار گرفته و توسط تابع DrawText بعنوان یکی از پارامترها مورد استفاده قرار می‌گیرد. تابع DrawText به صورت زیر تعریف می‌شود:

```
DrawText  proto hdc:HDC, lpString:DWORD, nCount:DWORD, lpRect:DWORD, uFormat:DWORD
```

تابع فوق متن دلخواهی را در مستطیل دریافت شده با خواص تعریف شده در دستگیره بافت ابزار مورد استفاده مانند رنگ، فونت و غیره ، ترسیم می کند. توضیحات پارامترهای آن به شرح زیر هستند :

hdc : دستگیره‌ای به بافت ابزار (handle to device context)

lpString : اشاره گری به رشته‌ای که باید توسط تابع در مستطیل تعریف شده ترسیم شود.

nCount : تعداد کاراکترهای نمایش یافته. اگر رشته مختوم به نال باشد باید این مقدار را مساوی 1- قرار داد.

lpRect : اشاره گری به مستطیلی است که قرار است ترسیم در آن انجام شود.لازم به ذکر است که این مستطیل از نوع clipping rectangle می‌باشد.بدین معنا که ترسیم در خارج از محدوده‌ی آن میسر نیست.

uFormat : بیانگر نحوه نمایش متن است. اگر نیاز به ترکیب آنها باشد از or استفاده می گردد. سه مقدار برای آن ممکن است:

DT_SINGLELINE : بیانگر ترسیم در یک خط می‌باشد.

DT_CENTER : متن را در وسط صفحه ترسیم می کند (به صورت افقی).

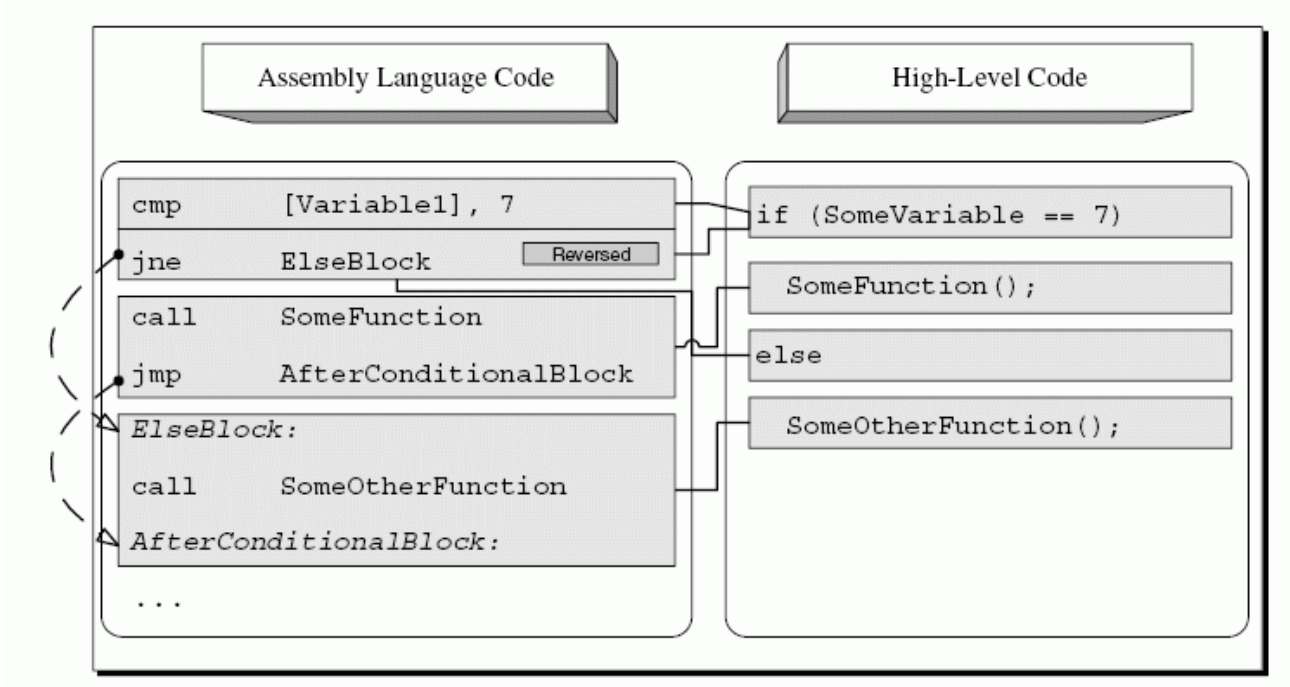
DT_VCENTER : متن را در وسط صفحه ترسیم می کند (به صورت عمودی و باید با DT_SINGLELINE بکار رود).

پس از پایان ترسیم ، باید تابع EndPaint جهت بازگرداندن منابع تخصیص یافته به سیستم عامل ، فراخوانی شود. تمام!

مرور نکات مهم برنامه فوق:

- فراخوانی BeginPaint-EndPaint در جهت پاسخگویی به پیغام WM_PAINT .
- هر نوع ترسیمی را می توان در بین BeginPaint و EndPaint انجام داد.
- اگر مایل به ترسیم ناحیه کلاینت مطابق سایر درخواست‌ها باشید به یکی از دو روش زیر می‌توان عمل کرد:
  - فراخوانی GetDC-ReleaseDC و انجام ترسیمات بین این دو فراخوانی.
  - فراخوانی InvalidateRect و UpdateWindow جهت غیرمعتبرکردن کل ناحیه کلاینت و وادار کردن ویندوز تا پیغام WM_PAINT را به برنامه ارسال کند.سپس انجام ترسیم در قسمت WM_PAINT قابل انجام است.

مقایسه کد زبان سی و کد اسمبلی یک شرط دو طرفه



در این نوع شرط ها درصورت برآورده نشدن شرط اصلی ، راه حل دیگر ارائه شده ، انتخاب خواهد شد. در کد اسمبلی معادل (شکل فوق)، پرش‌های شرطی و غیر شرطی پس از مقایسه انجام شده، قابل ملاحظه هستند.

مثال ۳- ترسیم متن به روشی دیگر

در این مثال قصد داریم تا در مورد خواص متن ترسیم شده مانند رنگ ، قلم انتخابی و اندازه آن ، بیشتر بحث کنیم.

مقدمه:

سیستم رنگ ویندوز بر مبنای مقادیر RGB است (R=red, G=Green, B=Blue). هر مؤلفه ی آن بازه ای از صفر تا ۲۵۵ دارد (یک بایت) . برای مثال رنگ قرمز خالص معادل 255,0,0 است و رنگ سفید خالص مساوی است با 255,255,255 . همانطور که ملاحظه می‌فرمایید کار کردن با سیستم رنگ RGB ساده نیست.

برای تنظیم رنگ متن و رنگ زمینه آن ، توابع SetTextColor و SetBkColor مهیا هستند.تمام آنها نیاز به hdc و مقدار ۳۲ بیتی RGB دارد. ساختار RGB به صورت زیر تعریف شده است:

```
RGB_value struct
   unused   db 0
   blue      db ?
   green     db ?
   red       db ?
RGB_value ends
```

لازم به ذکر است که بایت اول ، بکار نرفته و مساوی صفر باید قرار گیرد. جهت سهولت کار با این ساختار بهتر است ماکروی زیر تعریف شود:

```
RGB macro red,green,blue
   xor   eax,eax
   mov  ah,blue
   shl   eax,8
   mov  ah,green
   mov  al,red
endm
```

بهتر است این ماکرو در یک فایل inc قرار گرفته و به برنامه ملحق شود.

برای ایجاد قلم می‌توان از CreateFont ویا CreateFontIndirect استفاده کرد. تابع CreateFontIndirect تنها یک پارامتر ورودی داشته و آن اشاره‌گری به ساختار LOGFONT می‌باشد.این تابع قابلیت انعطاف بیشتری داشته و برای حالتی که نیاز است در برنامه به دفعات ، قلم تعویض شود مناسب خواهد بود. از آنجائیکه ما در برنامه تنها یک قلم را ایجاد خواهیم کرد از تابع CreateFont استفاده می گردد.پس از فراخوانی این تابع ، خروجی آن دستگیره ای به قلم مورد استفاده جهت بکارگیری در زمینه ابزار ، می‌باشد.پس از آن هر تابع متنی که مورد استفاده قرار گیرد از این فونت که همراه زمینه ابزار شده است ، بهره می‌جوید.

کد برنامه:

شکل ۲- ترسیم متن

```
.386
.model flat,stdcall
option casemap:none

WinMain proto :DWORD,:DWORD,:DWORD,:DWORD

include \masm32\include\windows.inc
include \masm32\include\user32.inc
include \masm32\include\kernel32.inc
include \masm32\include\gdi32.inc
includelib \masm32\lib\user32.lib
includelib \masm32\lib\kernel32.lib
includelib \masm32\lib\gdi32.lib

RGB macro red,green,blue
        xor eax,eax
        mov ah,blue
        shl eax,8
        mov ah,green
        mov al,red
endm

.data
ClassName db "SimpleWinClass",0
AppName  db "Our First Window",0
TestString  db "Win32 assembly is great and easy!",0
FontName db "script",0

.data?
hInstance HINSTANCE ?
CommandLine LPSTR ?

.code
 start:
    invoke GetModuleHandle, NULL
    mov    hInstance,eax
    invoke GetCommandLine
    mov CommandLine,eax
    invoke WinMain, hInstance,NULL,CommandLine, SW_SHOWDEFAULT
    invoke ExitProcess,eax
```

```
WinMain proc hInst:HINSTANCE,hPrevInst:HINSTANCE,CmdLine:LPSTR,CmdShow:DWORD
   LOCAL wc:WNDCLASSEX
   LOCAL msg:MSG
   LOCAL hwnd:HWND
   mov   wc.cbSize,SIZEOF WNDCLASSEX
   mov   wc.style, CS_HREDRAW or CS_VREDRAW
   mov   wc.lpfnWndProc, OFFSET WndProc
   mov   wc.cbClsExtra,NULL
   mov   wc.cbWndExtra,NULL
   push  hInst
   pop   wc.hInstance
   mov   wc.hbrBackground,COLOR_WINDOW+1
   mov   wc.lpszMenuName,NULL
   mov   wc.lpszClassName,OFFSET ClassName
   invoke LoadIcon,NULL,IDI_APPLICATION
   mov   wc.hIcon,eax
   mov   wc.hIconSm,eax
   invoke LoadCursor,NULL,IDC_ARROW
   mov   wc.hCursor,eax
   invoke RegisterClassEx, addr wc
   invoke CreateWindowEx,NULL,ADDR ClassName,ADDR AppName,\
         WS_OVERLAPPEDWINDOW,CW_USEDEFAULT,\
         CW_USEDEFAULT,CW_USEDEFAULT,CW_USEDEFAULT,NULL,NULL,\
         hInst,NULL
   mov   hwnd,eax
   invoke ShowWindow, hwnd,SW_SHOWNORMAL
   invoke UpdateWindow, hwnd
   .WHILE TRUE
            invoke GetMessage, ADDR msg,NULL,0,0
            .BREAK .IF (!eax)
            invoke TranslateMessage, ADDR msg
            invoke DispatchMessage, ADDR msg
   .ENDW
   mov     eax,msg.wParam
   ret
WinMain endp

WndProc proc hWnd:HWND, uMsg:UINT, wParam:WPARAM, lParam:LPARAM
   LOCAL hdc:HDC
   LOCAL ps:PAINTSTRUCT
   LOCAL hfont:HFONT

   .IF uMsg==WM_DESTROY
      invoke PostQuitMessage,NULL
   .ELSEIF uMsg==WM_PAINT
      invoke BeginPaint,hWnd, ADDR ps
      mov    hdc,eax
      invoke CreateFont,24,16,0,0,400,0,0,0,OEM_CHARSET,\
                        OUT_DEFAULT_PRECIS,CLIP_DEFAULT_PRECIS,\
                        DEFAULT_QUALITY,DEFAULT_PITCH or FF_SCRIPT,\
                        ADDR FontName
      invoke SelectObject, hdc, eax
      mov    hfont,eax
      RGB    200,200,50
      invoke SetTextColor,hdc,eax
      RGB    0,0,255
      invoke SetBkColor,hdc,eax
      invoke TextOut,hdc,0,0,ADDR TestString,SIZEOF TestString
      invoke SelectObject,hdc, hfont
      invoke EndPaint,hWnd, ADDR ps
```

```
   .ELSE
      invoke DefWindowProc,hWnd,uMsg,wParam,lParam
      ret
   .ENDIF
   xor    eax,eax
   ret
WndProc endp

end start
```

بررسی کد فوق:

```
      invoke CreateFont,24,16,0,0,400,0,0,0,OEM_CHARSET,\
                        OUT_DEFAULT_PRECIS,CLIP_DEFAULT_PRECIS,\
                        DEFAULT_QUALITY,DEFAULT_PITCH or FF_SCRIPT,\
                        ADDR FontName
```

تابع CreateFont ، قلمی منطقی را که مطابق پارامترهای آن درخواست شده، ایجاد می کند. توضیحات پارامترهـای آن به شرح زیر هستند:

```
CreateFont proto nHeight:DWORD,\
                 nWidth:DWORD,\
                 nEscapement:DWORD,\
                 nOrientation:DWORD,\
                 nWeight:DWORD,\
                 cItalic:DWORD,\
                 cUnderline:DWORD,\
                 cStrikeOut:DWORD,\
                 cCharSet:DWORD,\
                 cOutputPrecision:DWORD,\
                 cClipPrecision:DWORD,\
                 cQuality:DWORD,\
                 cPitchAndFamily:DWORD,\
                 lpFacename:DWORD
```

nHeight : بیانگر ارتفاع حروف است. اگر مساوی صفر قرار گیرد، ارتفاع پیش فرض استفاده خواهد شد.

nWidth : عرض دلخواه حروف.اگر مساوی صفر قرار گیرد، ویندوز طول و عـرض حـروف را متناسـب بـا هـم تنظیم می کند.

nEscapement : زاویه ترسیم کاراکتر بعدی را نسبت به قبلی بر حسب درجه مشخص مـی کنـد. حالـت معمـول صفر است.

nOrientation : درجه چرخش ترسیم حروف را بر حسب درجه مشخص می کند.

nWeight : ضخامت هر کاراکتر را مشخص می کند.مقادیر زیر برای این پارامتر مجاز هستند:

```
FW_DONTCARE     equ  0
FW_THIN         equ  100
FW_EXTRALIGHT   equ  200
FW_ULTRALIGHT   equ  200
FW_LIGHT        equ  300
FW_NORMAL       equ  400
FW_REGULAR      equ  400
FW_MEDIUM       equ  500
FW_SEMIBOLD     equ  600
FW_DEMIBOLD     equ  600
FW_BOLD         equ  700
FW_EXTRABOLD    equ  800
FW_ULTRABOLD    equ  800
FW_HEAVY        equ  900
FW_BLACK        equ  900
```

cItalic : برای حالت معمول صفر باید وارد شود.برای حالت مایل هر عددی غیر از صفر باید مشخص شود.

cUnderline : برای حالت معمول صفر باید وارد شود.

cStrikeOut : برای حالت معمول صفر باید وارد شود.

cCharSet : اگر مساوی OEM_CHARSET قرار گیرد ویندوز حالت وابسته به سیستم عامل را انتخاب خواهد کرد.

cOutputPrecision : دقت ترسیم فونت را مشخص می کند مقدار OUT_DEFAULT_PRECIS حالت پیش فرض را انتخاب خواهد کرد.

cClipPrecision : دقت برش را مشخص می کند (برای حالتی که قسمتی از فونت خارج مستطیل ترسیمی قرار گیرد) . CLIP_DEFAULT_PRECIS حالت پیش فرض را مشخص می کند.

cQuality : کیفیت خروجی را مشخص می کند. سه مقدار , DEFAULT_QUALITY, PROOF_QUALITY DRAFT_QUALITY را برای آن می توان مشخص کرد.بر این مبنا میزان دقتی که GDI باید بخرج دهد تا کیفیت متن ترسیمی مطابق قلم واقعی باشد، مشخص می‌شود.

cPitchAndFamily : تراکم حروف و خانواده ترسیمی آن. برای ترکیب حالتهای مختلف آنها می‌توان از or استفاده کرد.

lpFacename : رشته ای مختوم به نال معرف نام قلم مورد استفاده.

بدیهی است که برای توضیحات بیشتر باید به Win32 API reference مراجعه کرد (مانند MSDN). در ادامه کد داریم:

```
invoke SelectObject, hdc, eax
mov    hfont,eax
```

پس از دریافت دستگیره فونت منطقی ایجاد شده، تابع SelectObject آنرا به بافت ابزار، جهت استفاده سایر توابع GDI انتساب می‌دهد.

```
RGB     200,200,50
invoke  SetTextColor,hdc,eax
RGB     0,0,255
invoke  SetBkColor,hdc,eax
```

از ماکرو RGB که پیشتر آنرا ایجاد کردیم جهت تنظیم رنگ دو تابع فوق استفاده شده است. در ادامه :

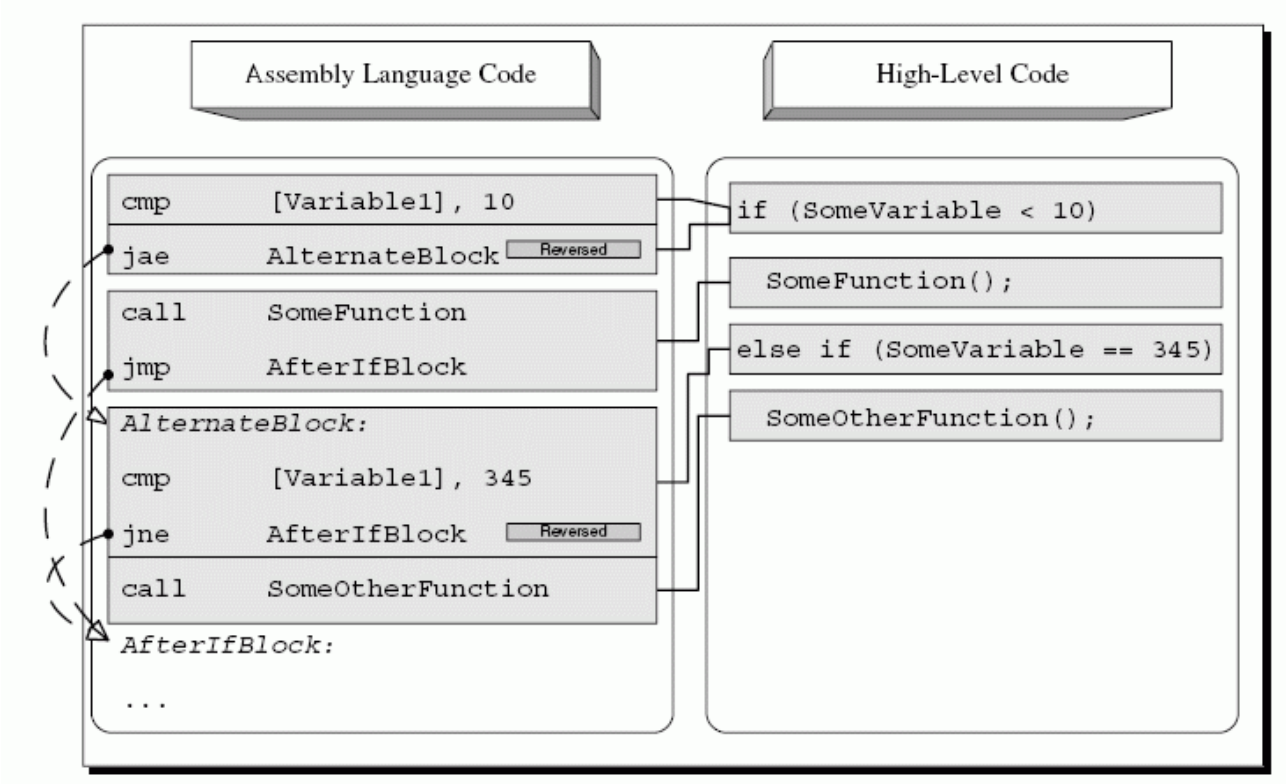
```
invoke TextOut,hdc,0,0,ADDR TestString,SIZEOF TestString
```

از تابع TextOut برای ترسیم متنی با مشخصاتی تنظیم شده (فونت و رنگ آن) استفاده گردیده است. پس از آن:

```
invoke  SelectObject,hdc, hfont
```

پس از انجام عملیات ترسیم متن ، باید اشیاء تخصیص داده شده مجددا به سیستم عامل بازگشت داده شوند.

مقایسه کد زبان سی و کد اسمبلی شرط‌های چندگانه



در تصویر فوق ، نحوه پیاده سازی else-if زبانهای سطوح بالاتر به زبان اسمبلی را به کمک پرش‌های شرطی و غیرشرطی ، می‌توان مشاهده کرد.

مثال ۴– دریافت ورودی از صفحه کلید

از آنجائیکه بطور معمول تنها یک صفحه کلید به همراه هر رایانه وجود دارد ، تمام برنامه‌های ویندوز باید آنرا بین خویش به اشتراک بگذارند. ویندوز عهده دار مسؤولیت ارسال اطلاعات کلیدهای فشرده شده بر روی صفحه کلید ، به پنجره‌ای است که تمرکز ورودی را در اختیار دارد. نوار عنوان پنجره‌ای که تمرکز را در اختیار دارد ، رنگی متفاوت با سایر پنجره‌ها دارد.

دو نوع اصلی پیغام‌های صفحه کلید وجود دارند. برای مثال اگر به صفحه کلید بعنوان مجموعه ای از کلیدها نگاه شود ، در این حالت با فشرده شدن کلیدی ، ویندوز پیغام WM_KEYDOWN را به پنجره‌ای که تمرکز را در اختیار دارد ارسال خواهد کرد، تا آنرا از فشرده شدن کلید مطلع کند. همچنین با رها شدن کلید ، پیغام WM_KEYUP به پنجره ارسال می گردد. و یا اگر به صفحه کلید بعنوان دستگاه ورود حروف نگاه شود ، با فشردن شدن هر کلیدی، پیغام WM_CHAR به پنجره دارای تمرکز فرستاده می‌شود تا آنرا برای مثال از فشرده شدن کلید "a" مطلع سازد.

در حقیقت پیغام‌های WM_KEYDOWN و WM_KEYUP توسط تابع TranslateMessage به پیغام WM_CHAR ترجمه می‌شوند. رویه پنجره مختار است که هر سه پیغام را پردازش کند و یا تنها یکی از آنها را. در اغلب حالات از پیغام‌های WM_KEYDOWN و WM_KEYUP می‌توان صرفنظر کرد زیرا همانطور که عنوان شد به WM_CHAR ترجمه می‌شوند. ما در این مثال بر روی WM_CHAR متمرکز خواهیم شد.

کد برنامه :

```
.386
.model flat,stdcall
option casemap:none

WinMain proto :DWORD,:DWORD,:DWORD,:DWORD

include \masm32\include\windows.inc
include \masm32\include\user32.inc
include \masm32\include\kernel32.inc
include \masm32\include\gdi32.inc
includelib \masm32\lib\user32.lib
includelib \masm32\lib\kernel32.lib
includelib \masm32\lib\gdi32.lib

.data
ClassName db "SimpleWinClass",0
AppName  db "Our First Window",0
char WPARAM 20h                 ; the character the program receives from keyboard
```

```
.data?
hInstance HINSTANCE ?
CommandLine LPSTR ?

.code
start:
   invoke GetModuleHandle, NULL
   mov    hInstance,eax
   invoke GetCommandLine
   mov CommandLine,eax
   invoke WinMain, hInstance,NULL,CommandLine, SW_SHOWDEFAULT
   invoke ExitProcess,eax

WinMain proc hInst:HINSTANCE,hPrevInst:HINSTANCE,CmdLine:LPSTR,CmdShow:DWORD
   LOCAL wc:WNDCLASSEX
   LOCAL msg:MSG
   LOCAL hwnd:HWND
   mov   wc.cbSize,SIZEOF WNDCLASSEX
   mov   wc.style, CS_HREDRAW or CS_VREDRAW
   mov   wc.lpfnWndProc, OFFSET WndProc
   mov   wc.cbClsExtra,NULL
   mov   wc.cbWndExtra,NULL
   push  hInst
   pop   wc.hInstance
   mov   wc.hbrBackground,COLOR_WINDOW+1
   mov   wc.lpszMenuName,NULL
   mov   wc.lpszClassName,OFFSET ClassName
   invoke LoadIcon,NULL,IDI_APPLICATION
   mov   wc.hIcon,eax
   mov   wc.hIconSm,eax
   invoke LoadCursor,NULL,IDC_ARROW
   mov   wc.hCursor,eax
   invoke RegisterClassEx, addr wc
   invoke CreateWindowEx,NULL,ADDR ClassName,ADDR AppName,\
         WS_OVERLAPPEDWINDOW,CW_USEDEFAULT,\
         CW_USEDEFAULT,CW_USEDEFAULT,CW_USEDEFAULT,NULL,NULL,\
         hInst,NULL
   mov   hwnd,eax
   invoke ShowWindow, hwnd,SW_SHOWNORMAL
   invoke UpdateWindow, hwnd
   .WHILE TRUE
            invoke GetMessage, ADDR msg,NULL,0,0
            .BREAK .IF (!eax)
            invoke TranslateMessage, ADDR msg
            invoke DispatchMessage, ADDR msg
      .ENDW
   mov     eax,msg.wParam
   ret
WinMain endp

WndProc proc hWnd:HWND, uMsg:UINT, wParam:WPARAM, lParam:LPARAM
   LOCAL hdc:HDC
   LOCAL ps:PAINTSTRUCT

   .IF uMsg==WM_DESTROY
      invoke PostQuitMessage,NULL
   .ELSEIF uMsg==WM_CHAR
      push wParam
      pop  char
      invoke InvalidateRect, hWnd,NULL,TRUE
```

```
   .ELSEIF uMsg==WM_PAINT
      invoke BeginPaint,hWnd, ADDR ps
      mov    hdc,eax
      invoke TextOut,hdc,0,0,ADDR char,1
      invoke EndPaint,hWnd, ADDR ps
   .ELSE
      invoke DefWindowProc,hWnd,uMsg,wParam,lParam
      ret
   .ENDIF
   xor    eax,eax
   ret
WndProc endp
end start
```

بررسی کد فوق:

```
char WPARAM 20h                ; the character the program receives from keyboard
```

این متغیری است که حرف دریافت شده از صفحه کلید را در خود ذخیره می‌کند. از آنجائیکه این کاراکتر به پارامتر WPARAM رویه پنجره ارسال می‌شود، جهت سهولت، نوع آنرا WPARAM انتخاب کردیم.همچنین مقدار اولیه 20h که معادل با" فاصله" است را به آن نسبت داده ایم ، زیرا در اولین باری که پنجره ناحیه کلاینت خود را به روز در می‌آورد ، ورودی از طرف صفحه کلید وجود ندارد.به همین جهت قصد داریم در ابتدا صرفا یک فاصله را نمایش دهیم. در ادامه داریم:

```
   .ELSEIF uMsg==WM_CHAR
      push wParam
      pop  char
      invoke InvalidateRect, hWnd,NULL,TRUE
```

جهت پردازش پیغام WM_CHAR رسیده ، قطعه کد فوق به رویه پنجره اضافه گردیده است. کاراکتر در متغیر char قرارگرفته و سپس تابع InvalidateRect فراخوانی می‌شود. همانطور که در مثال قبل نیز ذکر گردید ، با فراخوانی این تابع ، مستطیل پنجره غیرمعتبر شده و ویندوز وادار خواهد گردید تا پیغام WM_PAINT را به رویه پنجره ارسال کند. تعریف این تابع به صورت زیر است:

```
InvalidateRect  proto  hWnd:HWND,\
                       lpRect:DWORD,\
                       bErase:DWORD
```

lpRect اشاره‌گری به مستطیلی است که غیرمجاز اعلام می‌گردد. اگر بجای آن Null وارد شود کل ناحیه مربوطه غیرمعتبر اعلام خواهد شد.

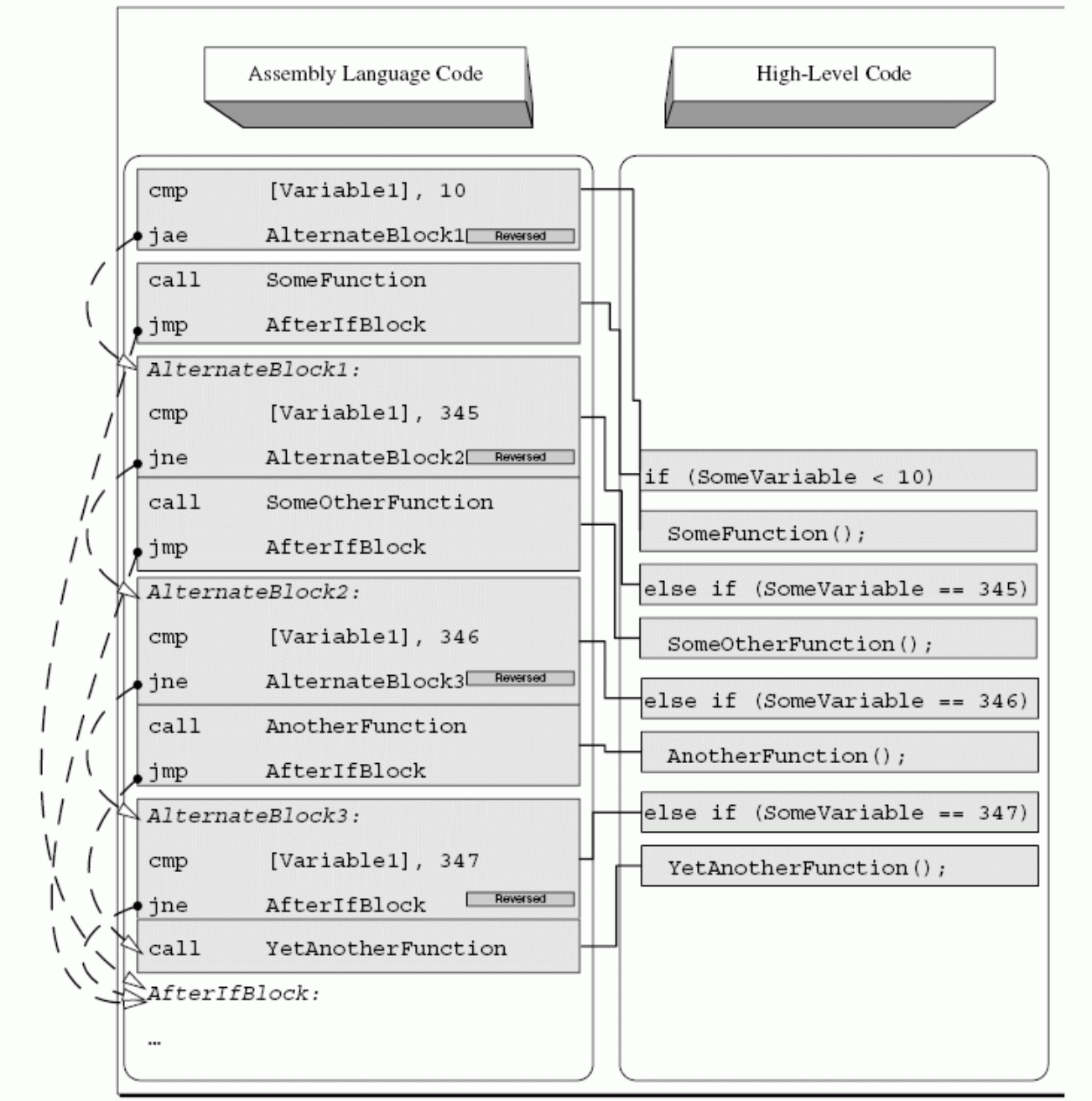
bErase به ویندوز اعلام می کند که آیا نیاز است زمینه نیز پاک گردد یا خیر. اگر این پرچم به TRUE تنظیم شود ویندوز با فراخوانی تابع BeginPaint ، زمینه مستطیل درخواستی را پاک خواهد کرد.

انجام عملیات ترسیم کاراکتر بر روی پنجره را می‌توان در پاسخ به پیغام WM_CHAR نیز انجام داد (با فراخوانی GetDC و ReleaseDC). اما از آنجائیکه این قسمت از کد تنها زمانی فراخوانی می‌شود که کلیدی فشرده شده باشد ، هنگام نیاز به ترسیم مجدد کل ناحیه کلاینت پنجره ، این امر رخ نخواهد داد. بنابراین بهتر است ترسیم کاراکتر در پاسخ به پیغام WM_PAINT صورت گیرد. در ادامه داریم:

```
invoke TextOut,hdc,0,0,ADDR char,1
```

زمانیکه تابع InvalidateRect فراخوانی می‌شود ، پیغام WM_PAINT به پنجره ارسال می گردد. بنابراین کد قرار گرفته در قسمت یاد شده اجرا می گردد . با فشردن هر کلیدی ، کاراکتر فشرده شده در سمت چپ ، بالای پنجره ترسیم می گردد. همچنین با تغییر سایز پنجره و یا به حداقل و یا حداکثر رساندن آن نیز ، کاراکتر بر روی صفحه حفظ می گردد زیرا کد ترسیمی آن در قسمت مربوط به WM_PAINT قرار گرفته است.

مقایسه کد زبان سی و کد اسمبلی شرط‌های ترکیبی



برنامه‌های واقعی عموما از شرط‌های ترکیبی تشکیل می‌گردند. همانطور که در شکل نیز مشخص است تبدیل این کد به زبان اسمبلی نسبتا پیچیده می‌باشد.

مثال ۵- دریافت ورودی از طریق ماوس

در این برنامه قصد داریم در محل کلیک چپ ماوس، متنی را در ناحیه کلاینت پنجره ترسیم نماییم.

ویندوز فعالیت‌های ماوس (مانند حرکت ماوس ، کلیک‌های راست و چپ و دوبار کلیک کردن) را ردیابی کرده و پیغام‌هایی را متناسب با آنها به پنجره مربوطه ارسال می‌کند. بر خلاف پیغام‌های صفحه کلید که صرفا به پنجره ای ارسال می‌گردند که تمرکز را در اختیار دارد، پیغام‌های مربوط به ماوس به تمامی پنجره‌هایی که اشاره‌گر آن بر روی آنها است ، فرستاده می‌شود (خواه این پنجره فعال باشد یا خیر). حتی این پیغامها می‌توانند مربوط به ناحیه خارج از مستطیل کلاینت پنجره نیز باشند . خوشبختانه در اکثر اوقات می‌توان صرفا پیغامهای مرتبط به ناحیه کلاینت پنجره را دریافت و پردازش کرد و از مابقی پیغامها صرفنظر نمود.

به ازای هر دکمه ی ماوس ، دو پیغام وجود دارد:
WM_LBUTTONDOWN , WM_RBUTTONDOWN and WM_LBUTTONUP , WM_RBUTTONUP

اگر ماوسی دارای سه دکمه بود ، پیغامهای زیر نیز وجود دارند:
WM_MBUTTONDOWN and WM_MBUTTONUP

هنگامیکه ماوس از روی ناحیه کلاینت عبور می‌کند پیغام زیر از طرف ویندوز صادر می‌شود:
WM_MOUSEMOVE

یک پنجره تنها هنگامی پیغام‌های مربوط به دوبار کلیک شدن ماوس (WM_LBUTTONDBCLK or WM_RBUTTONDBCLK) را دریافت می‌کند که کلاس پنجره دارای پرچم حالت CS_DBLCLKS باشد.در غیراینصورت صرفا پیغامهای فشرده شدن ویا رها شدن دکمه‌های ماوس را دریافت می‌کند.

برای تمامی پیغامهای رسیده، پارامتر lParam حاوی مختصات ماوس می‌باشد.در این پارامتر، low word آن حاوی مختصات x و high word آن حاوی مختصات y است (نسبت به گوشه سمت چپ بالا). همچنین پارامتر wParam بیانگر حالات دکمه‌های ماوس و همچنین دکمه‌های shift و ctrl است.

کد برنامه :

```
.386
.model flat,stdcall
option casemap:none

WinMain proto :DWORD,:DWORD,:DWORD,:DWORD

include \masm32\include\windows.inc
include \masm32\include\user32.inc
include \masm32\include\kernel32.inc
include \masm32\include\gdi32.inc
includelib \masm32\lib\user32.lib
includelib \masm32\lib\kernel32.lib
includelib \masm32\lib\gdi32.lib

.data
ClassName db "SimpleWinClass",0
AppName  db "Our First Window",0
MouseClick db 0         ; 0=no click yet

.data?
hInstance HINSTANCE ?
CommandLine LPSTR ?
hitpoint POINT <>

.code
start:
   invoke GetModuleHandle, NULL
   mov    hInstance,eax
   invoke GetCommandLine
   mov CommandLine,eax
   invoke WinMain, hInstance,NULL,CommandLine, SW_SHOWDEFAULT
   invoke ExitProcess,eax

WinMain proc hInst:HINSTANCE,hPrevInst:HINSTANCE,CmdLine:LPSTR,CmdShow:DWORD
   LOCAL wc:WNDCLASSEX
   LOCAL msg:MSG
   LOCAL hwnd:HWND
   mov   wc.cbSize,SIZEOF WNDCLASSEX
   mov   wc.style, CS_HREDRAW or CS_VREDRAW
   mov   wc.lpfnWndProc, OFFSET WndProc
   mov   wc.cbClsExtra,NULL
   mov   wc.cbWndExtra,NULL
   push  hInst
   pop   wc.hInstance
   mov   wc.hbrBackground,COLOR_WINDOW+1
   mov   wc.lpszMenuName,NULL
   mov   wc.lpszClassName,OFFSET ClassName
   invoke LoadIcon,NULL,IDI_APPLICATION
   mov   wc.hIcon,eax
   mov   wc.hIconSm,eax
   invoke LoadCursor,NULL,IDC_ARROW
   mov   wc.hCursor,eax
   invoke RegisterClassEx, addr wc
   invoke CreateWindowEx,NULL,ADDR ClassName,ADDR AppName,\
         WS_OVERLAPPEDWINDOW,CW_USEDEFAULT,\
         CW_USEDEFAULT,CW_USEDEFAULT,CW_USEDEFAULT,NULL,NULL,\
```

```
        hInst,NULL
    mov   hwnd,eax
    invoke ShowWindow, hwnd,SW_SHOWNORMAL
    invoke UpdateWindow, hwnd
    .WHILE TRUE
            invoke GetMessage, ADDR msg,NULL,0,0
            .BREAK .IF (!eax)
            invoke DispatchMessage, ADDR msg
    .ENDW
    mov     eax,msg.wParam
    ret
WinMain endp

WndProc proc hWnd:HWND, uMsg:UINT, wParam:WPARAM, lParam:LPARAM
    LOCAL hdc:HDC
    LOCAL ps:PAINTSTRUCT

    .IF uMsg==WM_DESTROY
        invoke PostQuitMessage,NULL
    .ELSEIF uMsg==WM_LBUTTONDOWN
        mov eax,lParam
        and eax,0FFFFh
        mov hitpoint.x,eax
        mov eax,lParam
        shr eax,16
        mov hitpoint.y,eax
        mov MouseClick,TRUE
        invoke InvalidateRect,hWnd,NULL,TRUE
    .ELSEIF uMsg==WM_PAINT
        invoke BeginPaint,hWnd, ADDR ps
        mov    hdc,eax
        .IF MouseClick
            invoke lstrlen,ADDR AppName
            invoke TextOut,hdc,hitpoint.x,hitpoint.y,ADDR AppName,eax
        .ENDIF
        invoke EndPaint,hWnd, ADDR ps
    .ELSE
        invoke DefWindowProc,hWnd,uMsg,wParam,lParam
        ret
    .ENDIF
    xor    eax,eax
    ret
WndProc endp
end start
```

بررسی کد فوق:

```
.ELSEIF uMsg==WM_LBUTTONDOWN
        mov eax,lParam
        and eax,0FFFFh
        mov hitpoint.x,eax
        mov eax,lParam
        shr eax,16
        mov hitpoint.y,eax
```

```
mov MouseClick,TRUE
invoke InvalidateRect,hWnd,NULL,TRUE
```

رویه پنجره ، منتظر فشردن شدن دکمه چپ ماوس می‌ماند. هنگامیکه پیغام WM_LBUTTONDOWN دریافت شود، پارامتر lParam حاوی مختصات محل کلیک در مستطیل ناحیه کلاینت پنجره است . این مختصات در ساختار زیر ذخیره می‌شود:

```
POINT STRUCT
   x   dd ?
   y   dd ?
POINT ENDS
```

و همچنین پرچم MouseClick را به TRUE تنظیم می کند (بدین معنا که حداقل یک کلیک چپ ماوس بر روی ناحیه کلاینت پنجره انجام شده است، از آن در قسمت ترسیمی برنامه استفاده خواهیم کرد). در ادامه داریم:

```
Mov  eax,lParam
and  eax,0FFFFh
mov  hitpoint.x,eax
```

از آنجائیکه مختص x ، مساوی low word پارامتر lParam است و اعضای ساختار POINT ، ۳۲ بیتی هستند ، نیاز است تا پیش از ذخیره سازی در hitpoint.x ، high word آن، صفر شود (توسط AND). پس از آن:

```
shr  eax,16
mov hitpoint.y,eax
```

از آنجائیکه مختص y ، مساوی high word پارامتر lParam است، باید آنرا توسط عملیات شیفت در low word مربوط به eax ، قبل از ذخیره سازی در hitpoint.y ، قرار داد.

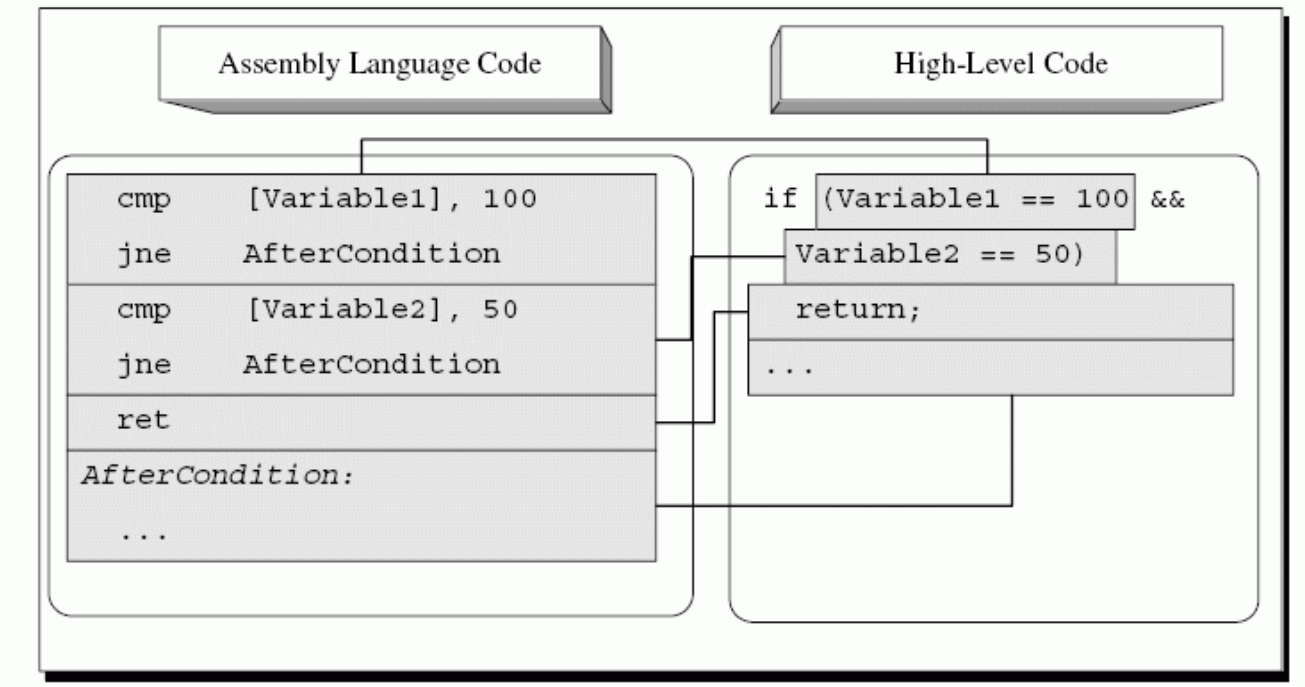
در قسمت پردازش پیغام‌های رسیده WM_PAINT (که فراخوانی تابع InvalidateRect سبب فراخوانی اجباری آن می گردد) ، متنی در محل کلیک ماوس ترسیم می گردد.

```
.IF MouseClick
   invoke  lstrlen,ADDR AppName
   invoke  TextOut, hdc, hitpoint.x, hitpoint.y, ADDR AppName, eax
.ENDIF
```

در قسمت WM_PAINT ، ابتدا پرچم MouseClick بررسی شده و در صورت TRUE بودن آن ، عملیات ترسیم صورت می گیرد.

از تابع lstrlen جهت دریافت طول رشته مورد استفاده در تابع TextOut ، استفاده می‌شود.

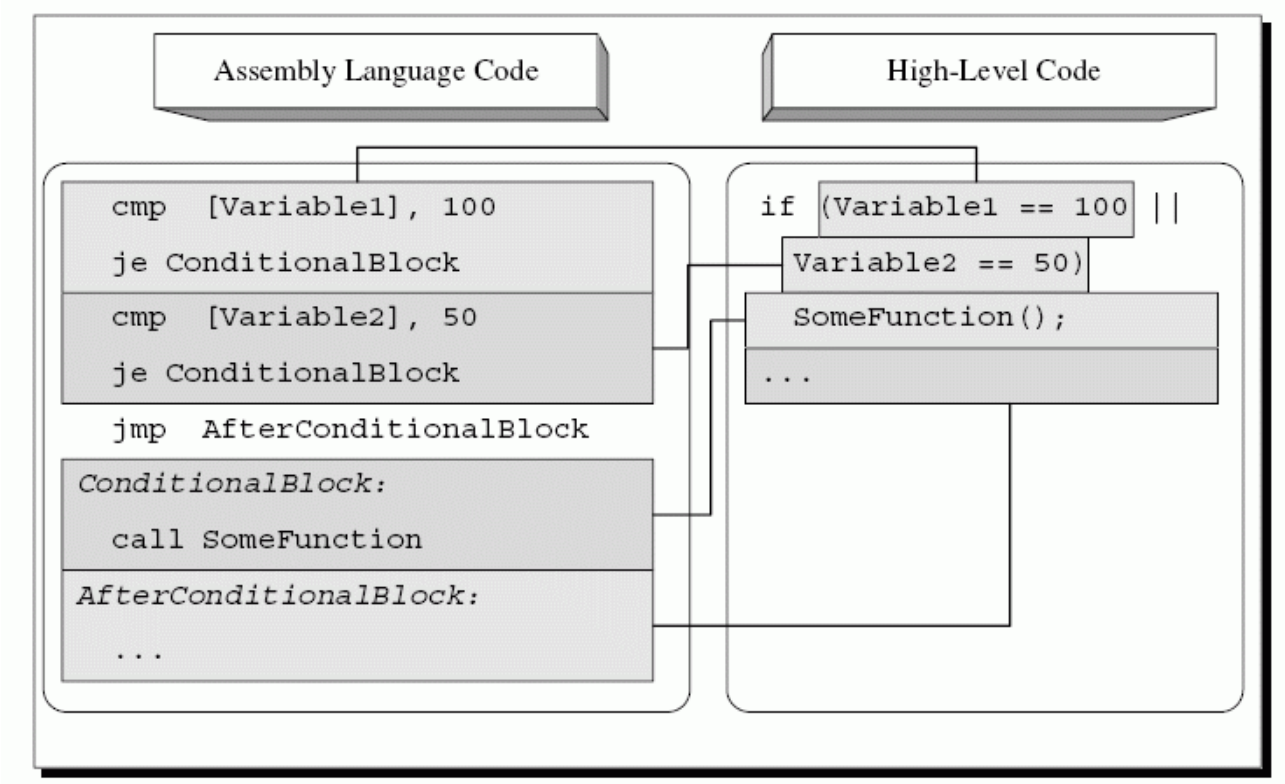
مقایسه کد زبان سی و کد اسمبلی شرط‌هایی با عملگرهای منطقی



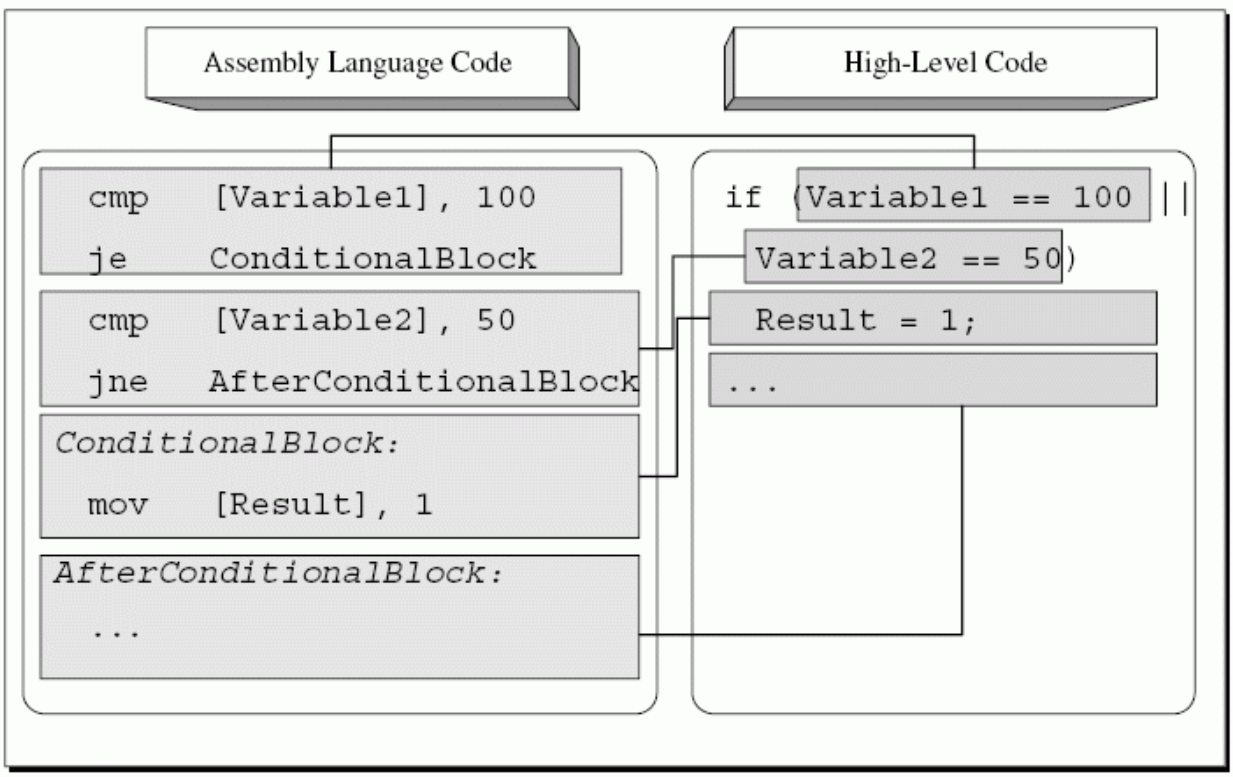
زبانهای سطح بالاتر با کمک عملگرهای منطقی ، امکان بیان شرط های ترکیبی را در قالب یک شرط میسر می‌کنند. دو عملگر بکار رفته در این نوع شروط عمدتا AND و OR هستند (این نوع عملگرها را با عملگرهای بیتی اشتباه نکنید، برای مثال در زبان C ، این نوع AND با && مشخص می‌شود).

کد اسمبلی معادل ، همانگونه که در شکل نیز مشخص شده است ، از دو پرش شرطی به یک برچسب تشکیل گردیده است.

نوع دیگر این نوع شرط‌ها ، استفاده از OR است (شکل بعد). این عملگر در زبان سی با || مشخص می‌شود.برای پیاده سازی این نوع شرط ها به زبان اسمبلی ، به ازای هر شرط یک پرش شرطی و در نهایت استفاده از یک پرش غیرشرطی لازم است.



کامپایلرهایی مانند VC++ روش‌های بهینه تر و سریعتری مانند شکل زیر را برای این منظور ارائه داده اند:



مثال ۶- کار با منوها

در این برنامه طرز اضافه کردن منوها به پنجره برنامه را خواهیم آموخت.

منو یکی از مهمترین اجزای یک برنامه است و لیستی را از توانایی‌های یک برنامه ، ارائه می‌دهد. توسط آن کاربر در ابتدای کار نیازی نخواهد داشت تا حتما راهنمای همراه برنامه را به دقت مطالعه نماید. از آنجائیکه هدف منوها استفاده سریع و ساده از برنامه توسط کاربر است ، باید در هنگام ایجاد آنها استانداردهایی را در این باره مراعات نمود.دو آیتم اولیه در منوی برنامه باید File و Edit بوده و آخرین آیتم ، مربوط به راهنمای برنامه باشد.سایر آیتم های خود را بین edit و راهنمای برنامه قرار دهید.

منو ، نوعی منبع (resource) محسوب می‌شود. انواع مختلفی از منابع مانند dialog box, string table, icon, bitmap, menu و غیره موجود هستند.

منابع عموما در فایلی جداگانه و با پسوند rc ایجاد می‌شوند. سپس این فایل و سورس برنامه در حین مرحله link ، ترکیب می‌شوند. فایل اجرایی نهایی ، حاوی کد و ریسورس‌ها خواهد بود.

فایل ریسورس را توسط هر نوع ویرایشگر متنی می‌توان آماده کرد. البته بدیهی است که آماده سازی دستی آنها کاری وقت گیر و طاقت فرسا است. به همراه کامپایلرهایی مانند Visual C++, Borland C++ و غیره، برنامه‌های resource editor نیز موجود هستند.

یک فایل منبع منو به صورت زیر تعریف می‌شود (شبیه به تعریف یک ساختار در زبان C):

```
MyMenu  MENU
{
   [menu list here]
}
```

بجای استفاده از براکت‌ها می‌توان از begin و end نیز استفاده کرد (جهت سهولت کار برنامه نویسان پاسکال!).

لیست منوها با عبارات MENUITEM و یا POPUP (منوی جهنده) مشخص می‌شوند. روش تعریف آن نیز به صورت زیر است:

```
MENUITEM   "&text", ID [,options]
```

لازم به ذکر است که قرار گرفتن & پیش از هر حرفی ، سبب قرار گرفتن یک underline برای آن می‌گردد.از ID جهت مشخص کردن آیتمی که بر روی آن کلیک شده است (هنگامیکه پیغام مربوط به آن ، به برنامه می رسد) ، استفاده می‌شود. بنابراین ID بکارگرفته شده باید منحصربفرد باشد. مواردی که به صورت options مشخص شده اند اختیاری بوده و به شرح زیر هستند :

- **GRAYED** : آیتم هایی که به این صورت مشخص شوند به صورت غیرفعال و خاکستری نمایش داده می‌شوند. این آیتم ها سبب ارسال پیغام WM_COMMAND نمی گردند.
- **INACTIVE** : این آیتم به صورت معمول نمایش داده شده و غیرفعال خواهد بود (سبب ارسال پیغام WM_COMMAND نمی‌شود.)
- **MENUBREAK** : این آیتم و موارد بعدی ، در خطی جدید ظاهر خواهند شد.
- **HELP** : این آیتم و موارد پس از آن راست چین خواهند شد.

برای ترکیب حالت های مختلف فوق می توان از or استفاده کرد. تنها دقت داشته باشید که **INACTIVE** and **GRAYED** را نمی‌توان با هم بکار برد.

موارد **POPUP** قالب زیر را دارند:

```
  POPUP  "&text" [,options]
{
  [menu list]
}
```

با اینکار منویی جهنده ، نمایانگر لیستی از آیتم‌ها پدید می‌آید.

نوع ویژه ای از MENUITEM با نام MENUITEM SEPARATOR نیز وجود دارد (که سبب ترسیم خطی افقی در لیست منوها می گردد).

مرحله بعد ، استفاده از فایل منبع تعریف شده در برنامه می‌باشد. دو راه برای انتساب منو به برنامه وجود دارد:

الف) اگر منویی به نام FirstMenu داشته باشید، آنرا می‌توان به عضو lpszMenuName ، مربوط به ساختار WNDCLASSEX انتساب داد.

```
.DATA

        MenuName  db "FirstMenu",0

…….....
…….....
.CODE

        …….....
        mov   wc.lpszMenuName, OFFSET MenuName
        …….....
```

ب) ارسال آن به پارامتر menu handle تابع CreateWindowEx :

```
.DATA

        MenuName  db "FirstMenu",0
        hMenu HMENU ?

.....................
.....................
.CODE

        .....................
        invoke LoadMenu, hInst, OFFSET MenuName
        mov   hMenu, eax
        invoke CreateWindowEx,NULL,OFFSET ClsName,\
                OFFSET Caption, WS_OVERLAPPEDWINDOW,\
                CW_USEDEFAULT,CW_USEDEFAULT,\
                CW_USEDEFAULT,CW_USEDEFAULT,\
                NULL,\
               hMenu,\
                hInst,\
                NULL\
        .....................
```

سوال: تفاوت این دو روش چیست؟

در حالت اول ، منوی ایجاد شده ، منوی پیش فرض پنجره خواهد شد.هر پنجره ای که بر مبنای این کلاس ایجاد شود منویی یکسان خواهد داشت.

اگر می‌خواهید که هر پنجره ایجاد شده از یک کلاس پنجره ، منوهایی متفاوت داشته باشد باید از روش دوم استفاده کنید.

در ادامه لازم به ذکر است که با انتخاب هر گزینه ای از منو، پیغام WM_COMMAND توسط ویندوز به رویه پنجره برنامه ارسال می گردد.در این حالت low word مربوط به wParam حاوی ID گزینه انتخابی است.

کد برنامه :

```
.386
.model flat,stdcall
option casemap:none

WinMain proto :DWORD,:DWORD,:DWORD,:DWORD

include \masm32\include\windows.inc
include \masm32\include\user32.inc
include \masm32\include\kernel32.inc
includelib \masm32\lib\user32.lib
includelib \masm32\lib\kernel32.lib

.data
ClassName db "SimpleWinClass",0
AppName  db "Our First Window",0
MenuName db "FirstMenu",0             ; The name of our menu in the resource file.
Test_string db "You selected Test menu item",0
Hello_string db "Hello, my friend",0
Goodbye_string db "See you again, bye",0

.data?
hInstance HINSTANCE ?
CommandLine LPSTR ?

.const
IDM_TEST equ 1                ; Menu IDs
IDM_HELLO equ 2
IDM_GOODBYE equ 3
IDM_EXIT equ 4

.code
start:
   invoke GetModuleHandle, NULL
   mov    hInstance,eax
   invoke GetCommandLine
   mov CommandLine,eax
   invoke WinMain, hInstance,NULL,CommandLine, SW_SHOWDEFAULT
   invoke ExitProcess,eax

WinMain proc hInst:HINSTANCE,hPrevInst:HINSTANCE,CmdLine:LPSTR,CmdShow:DWORD
   LOCAL wc:WNDCLASSEX
   LOCAL msg:MSG
   LOCAL hwnd:HWND
   mov   wc.cbSize,SIZEOF WNDCLASSEX
   mov   wc.style, CS_HREDRAW or CS_VREDRAW
   mov   wc.lpfnWndProc, OFFSET WndProc
   mov   wc.cbClsExtra,NULL
   mov   wc.cbWndExtra,NULL
   push  hInst
   pop   wc.hInstance
   mov   wc.hbrBackground,COLOR_WINDOW+1
   mov   wc.lpszMenuName,OFFSET MenuName       ; Put our menu name here
   mov   wc.lpszClassName,OFFSET ClassName
   invoke LoadIcon,NULL,IDI_APPLICATION
```

```
    mov   wc.hIcon,eax
    mov   wc.hIconSm,eax
    invoke LoadCursor,NULL,IDC_ARROW
    mov   wc.hCursor,eax
    invoke RegisterClassEx, addr wc
    invoke CreateWindowEx,NULL,ADDR ClassName,ADDR AppName,\
          WS_OVERLAPPEDWINDOW,CW_USEDEFAULT,\
          CW_USEDEFAULT,CW_USEDEFAULT,CW_USEDEFAULT,NULL,NULL,\
          hInst,NULL
    mov   hwnd,eax
    invoke ShowWindow, hwnd,SW_SHOWNORMAL
    invoke UpdateWindow, hwnd
    .WHILE TRUE
              invoke GetMessage, ADDR msg,NULL,0,0
              .BREAK .IF (!eax)
              invoke DispatchMessage, ADDR msg
    .ENDW
    mov     eax,msg.wParam
    ret
WinMain endp

WndProc proc hWnd:HWND, uMsg:UINT, wParam:WPARAM, lParam:LPARAM
    .IF uMsg==WM_DESTROY
       invoke PostQuitMessage,NULL
    .ELSEIF uMsg==WM_COMMAND
       mov eax,wParam
       .IF ax==IDM_TEST
          invoke MessageBox,NULL,ADDR Test_string,OFFSET AppName,MB_OK
       .ELSEIF ax==IDM_HELLO
          invoke MessageBox, NULL,ADDR Hello_string, OFFSET AppName,MB_OK
       .ELSEIF ax==IDM_GOODBYE
          invoke MessageBox,NULL,ADDR Goodbye_string, OFFSET AppName, MB_OK
       .ELSE
          invoke DestroyWindow,hWnd
       .ENDIF
    .ELSE
       invoke DefWindowProc,hWnd,uMsg,wParam,lParam
       ret
    .ENDIF
    xor    eax,eax
    ret
WndProc endp
end start


******************************************************************************************************************************

                                                        Menu.rc
******************************************************************************************************************************

#define IDM_TEST 1
#define IDM_HELLO 2
#define IDM_GOODBYE 3
#define IDM_EXIT 4

FirstMenu MENU
{
 POPUP "&PopUp"
       {
        MENUITEM "&Say Hello",IDM_HELLO
```

```
      MENUITEM "Say &GoodBye", IDM_GOODBYE
      MENUITEM SEPARATOR
      MENUITEM "E&xit",IDM_EXIT
     }
 MENUITEM "&Test", IDM_TEST
}
```

بررسی کد فوق:

در ابتدا بررسی فایل منبع آن :

```
#define  IDM_TEST          1          /* equal to IDM_TEST equ 1*/
#define  IDM_HELLO         2
#define  IDM_GOODBYE       3
#define  IDM_EXIT          4
```

در اینجا ID های مورد استفاده برای تعاریف گزینه‌های مختلف تعریف شده اند. بدیهی است که از هر عددی می‌توان استفاده کرد تا زمانیکه در این مجموعه منحصربفرد باشند. در ادامه :

```
FirstMenu MENU
```

تعریف منو با واژه کلیدی مربوطه. سپس :

```
 POPUP "&PopUp"
      {
       MENUITEM "&Say Hello",IDM_HELLO
       MENUITEM "Say &GoodBye", IDM_GOODBYE
       MENUITEM SEPARATOR
       MENUITEM "E&xit",IDM_EXIT
      }
```

در بالا منویی جهنده با عضو سوم جدا کننده تعریف شده است.در ادامه :

```
 MENUITEM  "&Test", IDM_TEST
```

نوار منو را در منوی اصلی تعریف می‌کند.

در ادامه کد برنامه را بررسی خواهیم کرد:

```
MenuName db "FirstMenu",0              ; The name of our menu in the resource file.
Test_string db "You selected Test menu item",0
Hello_string db "Hello, my friend",0
Goodbye_string db "See you again, bye",0
```

از آنجائیکه در یک فایل منبع می توان چندین منو را تعریف کرد ، باید نام منوی مورد استفاده را دقیقا مشخص نمود. سپس پیغامهای مورد استفاده هنگامیکه کاربر بر روی گزینه‌ای خاص کلیک می‌کند، نیز تعریف شده‌اند . در ادامه:

```
IDM_TEST equ 1                ; Menu IDs
IDM_HELLO equ 2
IDM_GOODBYE equ 3
IDM_EXIT equ 4
```

تعریف ID های گزینه‌ها جهت استفاده در رویه پنجره. این مقادیر باید با مقدارهای تعریف شده در فایل منبع دقیقا یکسان باشد. سپس :

```
.ELSEIF uMsg==WM_COMMAND
   mov eax,wParam
   .IF ax==IDM_TEST
      invoke MessageBox,NULL,ADDR Test_string,OFFSET AppName,MB_OK
   .ELSEIF ax==IDM_HELLO
      invoke MessageBox, NULL,ADDR Hello_string, OFFSET AppName,MB_OK
   .ELSEIF ax==IDM_GOODBYE
      invoke MessageBox,NULL,ADDR Goodbye_string, OFFSET AppName, MB_OK
   .ELSE
      invoke DestroyWindow,hWnd
   .ENDIF
```

در رویه پنجره پیغامهای WM_COMMAND را پردازش خواهیم کرد. همانطور که پیشتر نیز ذکر گردید در این حالت low word مربوط به wParam حاوی ID گزینه انتخابی است (یعنی مقدار قرار گرفته در ax).

اگر کاربر گزینه خروج را انتخاب کرده باشد ، تابع DestroyWindow سبب بسته شدن پنجره می‌شود.

در آخر اگر بخواهیم از روش دوم نمایش منو استفاده کنیم ، کد برنامه باید به صورت زیر اصلاح شود:

```
.data?
hInstance HINSTANCE ?
CommandLine LPSTR ?
hMenu HMENU ?                 ; handle of our menu
```

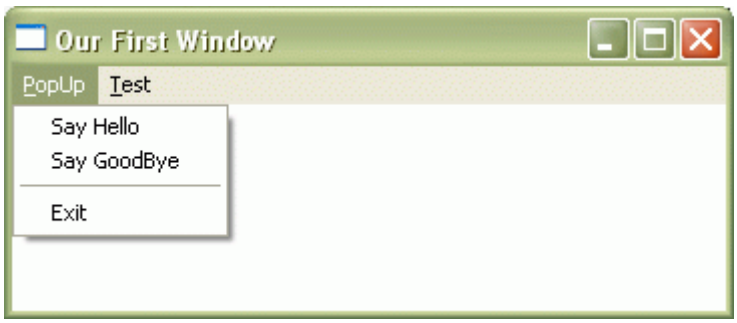
```
   invoke LoadMenu, hInst, OFFSET MenuName
   mov    hMenu,eax
   INVOKE CreateWindowEx,NULL,ADDR ClassName,ADDR AppName,\
     WS_OVERLAPPEDWINDOW,CW_USEDEFAULT,\
     CW_USEDEFAULT,CW_USEDEFAULT,CW_USEDEFAULT,NULL,hMenu,\
     hInst,NULL
```

متغیری از نوع HMENU دستگیره‌ای به منوی ما ایجاد خواهد کرد. پیش از فراخوانی CreateWindowEx ، تابع LoadMenu ، دستگیره لازم را برای استفاده در تابع CreateWindowEx ، برخواهد گرداند.

نحوه کامپایل فایل ریسورس و الحاق آن به فایل اجرایی نهایی:

برای مثال فایل asm برنامه به نام p7.asm و فایل منبع آن به نام p7.rc در آدرس C:\masm32\prog قرار گرفته اند.با اجرای خطوط زیر در خط فرمان ، فایل ریسورس کامپایل شده و نهایتا فایل اجرایی برنامه ساخته می‌شود:

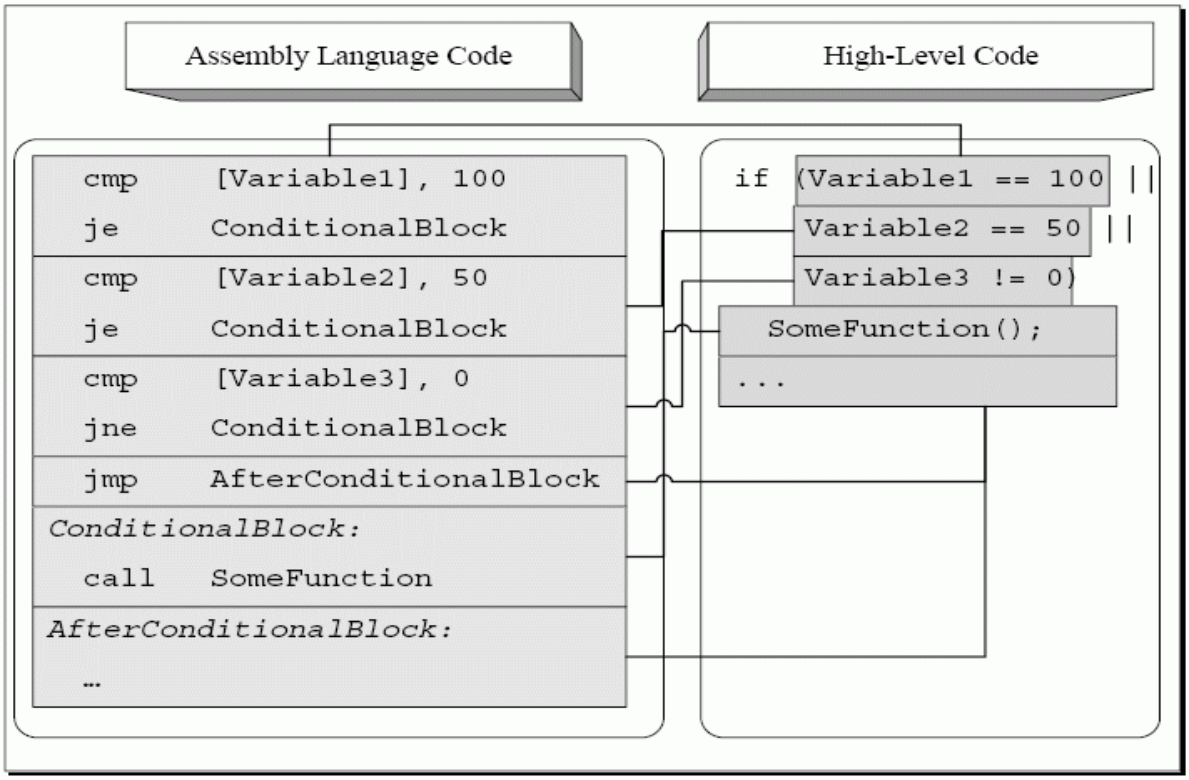
```
C:\masm32\bin\ml.exe /c /coff /Cp p7.asm
C:\masm32\bin\rc.exe p7.rc
C:\masm32\bin\link.exe /SUBSYSTEM:WINDOWS /LIBPATH:C:/masm32/prog/ p7.obj    p7.res
pause
```



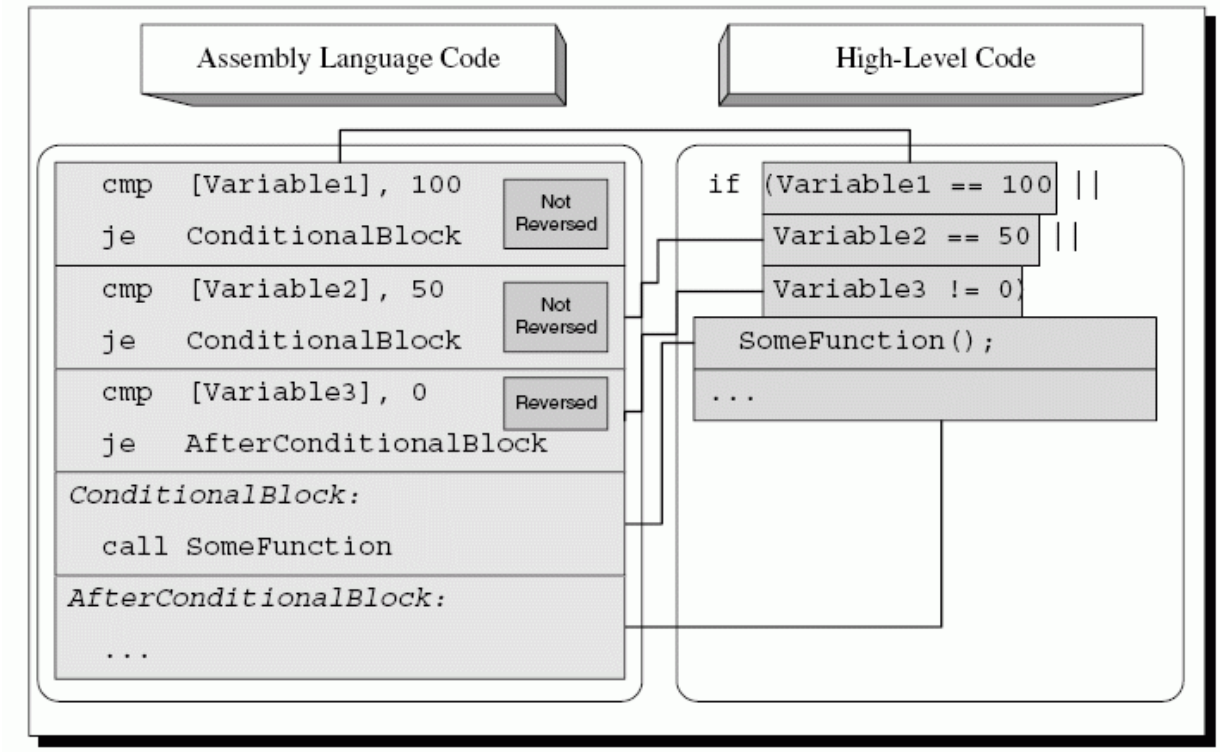
شکل ۳– نمایی از اجرای برنامه کار با منوها

بهتر است جهت سهولت اجرای دستورهای فوق ، فایلی با پسوند bat حاوی چهار خط فوق ایجاد کنید. پس از اجرای آن ، مراحل یکی پس از دیگری اجرا خواهند شد.

مقایسه کد زبان سی و کد اسمبلی یک شرط با بیش از دو عملگر منطقی



حالت سریعتر و بهینه تر را نیز در شکل زیر می‌توان مشاهده نمود :



مثال ۷- کار با Child Window Controls

ویندوز کلاس‌های متعددی از پنجره‌های از پیش تعریف شده را جهت استفاده در برنامه‌های ما تعریف کرده است. در اغلب اوقات از آنها بعنوان کامپوننت‌های dialog box استفاده می‌شود ، به همین جهت child window controls نام گرفته‌اند.یک child window پیغام‌های ماوس و صفحه کلید مربوط به خود را پردازش کرده و هنگامیکه حالت آنها تغییر کرد پنجره‌ی والد را باخبر می‌سازد.

مثالهایی از پنجره‌های از پیش تعریف شده شامل دکمه‌ها ، listbox ، checkbox ، radio button و غیره می‌باشد.برای استفاده از این کنترل‌ها نیاز است تا در ابتدا آنها را بوسیله توابع CreateWindow or CreateWindowEx ایجاد کرد.لازم به ذکر است که نیازی به رجیستر کردن کلاس این پنجره ها وجود ندارد. زیرا این امر توسط ویندوز پیش تر صورت گرفته است. پارامتر نام کلاس در اینجا "باید" نامی از پیش تعریف شده انتخاب شود. برای مثال اگر می‌خواهید دکمه‌ای را ایجاد کنید باید button را بعنوان پارامتر نام کلاس پنجره ، در تابع CreateWindowEx معرفی نمایید. سایر پارامترهای مورد نیاز ، دستگیره‌ای به پنجره‌ی والد و ID کنترل می‌باشد. این ID باید در بین سایر کنترل‌هایی که بر روی فرم قرار می‌گیرند یکتا باشد.

هنگامیکه کنترلی بر روی پنجره والد ایجاد می‌شود پیغام WM_CREATE را به آن ارسال خواهد کرد.کنترل سایر پیغام‌های خود را بوسیله WM_COMMAND به پنجره والد ارسال می‌کند. ID کنترل توسط low word مربوط به wParam، کد باخبرسازی توسط high word مربوط به wParam و دستگیره به پنجره توسط lParam در این حین ارسال می‌گردد. هر کنترل دارای کدباخبری سازی متفاوتی است و بهتر است در این مورد به مرجع API ویندوز مراجعه کرد.

همچنین پنجره والد نیز می‌تواند به کنترل‌ها ، پیغام‌های لازم را توسط فراخوانی تابع SendMessage ارسال کند.

کد برنامه :

در این مثال پنجره‌ای را ایجاد خواهیم کرد که دارای یک کنترل ادیت و یک دکمه است. هنگامیکه بر روی دکمه کلیک می‌شود یک message box حاوی متن نوشته در کنترل ادیت نمایش داده خواهد شد. همچنین از منوها نیز در این برنامه استفاده می‌گردد.

```
.386
.model flat,stdcall
option casemap:none

WinMain proto :DWORD,:DWORD,:DWORD,:DWORD
```

```
include \masm32\include\windows.inc
include \masm32\include\user32.inc
include \masm32\include\kernel32.inc
includelib \masm32\lib\user32.lib
includelib \masm32\lib\kernel32.lib

.data
ClassName db "SimpleWinClass",0
AppName  db "Our First Window",0
MenuName db "FirstMenu",0
ButtonClassName db "button",0
ButtonText db "My First Button",0
EditClassName db "edit",0
TestString db "Wow! I'm in an edit box now",0

.data?
hInstance HINSTANCE ?
CommandLine LPSTR ?
hwndButton HWND ?
hwndEdit HWND ?
buffer db 512 dup(?)                ; buffer to store the text retrieved from the edit box

.const
ButtonID equ 1                      ; The control ID of the button control
EditID equ 2                        ; The control ID of the edit control
IDM_HELLO equ 1
IDM_CLEAR equ 2
IDM_GETTEXT equ 3
IDM_EXIT equ 4

.code
start:
   invoke GetModuleHandle, NULL
   mov    hInstance,eax
   invoke GetCommandLine
   mov CommandLine,eax
   invoke WinMain, hInstance,NULL,CommandLine, SW_SHOWDEFAULT
   invoke ExitProcess,eax

WinMain proc hInst:HINSTANCE,hPrevInst:HINSTANCE,CmdLine:LPSTR,CmdShow:DWORD
   LOCAL wc:WNDCLASSEX
   LOCAL msg:MSG
   LOCAL hwnd:HWND
   mov   wc.cbSize,SIZEOF WNDCLASSEX
   mov   wc.style, CS_HREDRAW or CS_VREDRAW
   mov   wc.lpfnWndProc, OFFSET WndProc
   mov   wc.cbClsExtra,NULL
   mov   wc.cbWndExtra,NULL
   push  hInst
   pop   wc.hInstance
   mov   wc.hbrBackground,COLOR_BTNFACE+1
   mov   wc.lpszMenuName,OFFSET MenuName
   mov   wc.lpszClassName,OFFSET ClassName
   invoke LoadIcon,NULL,IDI_APPLICATION
   mov   wc.hIcon,eax
   mov   wc.hIconSm,eax
   invoke LoadCursor,NULL,IDC_ARROW
   mov   wc.hCursor,eax
   invoke RegisterClassEx, addr wc
   invoke CreateWindowEx,WS_EX_CLIENTEDGE,ADDR ClassName, \
```

```
                ADDR AppName, WS_OVERLAPPEDWINDOW,\
                CW_USEDEFAULT, CW_USEDEFAULT,\
                300,200,NULL,NULL, hInst,NULL
   mov   hwnd,eax
   invoke ShowWindow, hwnd,SW_SHOWNORMAL
   invoke UpdateWindow, hwnd
   .WHILE TRUE
      invoke GetMessage, ADDR msg,NULL,0,0
      .BREAK .IF (!eax)
      invoke TranslateMessage, ADDR msg
      invoke DispatchMessage, ADDR msg
   .ENDW
   mov     eax,msg.wParam
   ret
WinMain endp

WndProc proc hWnd:HWND, uMsg:UINT, wParam:WPARAM, lParam:LPARAM
   .IF uMsg==WM_DESTROY
      invoke PostQuitMessage,NULL
   .ELSEIF uMsg==WM_CREATE
      invoke CreateWindowEx,WS_EX_CLIENTEDGE, ADDR EditClassName,NULL,\
                WS_CHILD or WS_VISIBLE or WS_BORDER or ES_LEFT or\
                ES_AUTOHSCROLL,\
                50,35,200,25,hWnd,8,hInstance,NULL
      mov  hwndEdit,eax
      invoke SetFocus, hwndEdit
      invoke CreateWindowEx,NULL, ADDR ButtonClassName,ADDR ButtonText,\
                WS_CHILD or WS_VISIBLE or BS_DEFPUSHBUTTON,\
                75,70,140,25,hWnd,ButtonID,hInstance,NULL
      mov  hwndButton,eax
   .ELSEIF uMsg==WM_COMMAND
      mov eax,wParam
      .IF lParam==0
         .IF ax==IDM_HELLO
            invoke SetWindowText,hwndEdit,ADDR TestString
         .ELSEIF ax==IDM_CLEAR
            invoke SetWindowText,hwndEdit,NULL
         .ELSEIF  ax==IDM_GETTEXT
            invoke GetWindowText,hwndEdit,ADDR buffer,512
            invoke MessageBox,NULL,ADDR buffer,ADDR AppName,MB_OK
         .ELSE
            invoke DestroyWindow,hWnd
         .ENDIF
      .ELSE
         .IF ax==ButtonID
            shr eax,16
            .IF ax==BN_CLICKED
               invoke SendMessage,hWnd,WM_COMMAND,IDM_GETTEXT,0
            .ENDIF
         .ENDIF
      .ENDIF
   .ELSE
      invoke DefWindowProc,hWnd,uMsg,wParam,lParam
      ret
   .ENDIF
    xor    eax,eax
   ret
WndProc endp
end start
```

فایل منبع مورد استفاده :

```
#define IDM_HELLO 1

#define IDM_CLEAR 2

#define IDM_GETTEXT 3

#define IDM_EXIT 4


FirstMenu MENU

{

 POPUP "&Test Controls"

      {

       MENUITEM "Say Hello",IDM_HELLO

       MENUITEM "Clear Edit Box",IDM_CLEAR

       MENUITEM "Get Text", IDM_GETTEXT

       MENUITEM SEPARATOR

       MENUITEM "E&xit",IDM_EXIT

      }

}
```

بررسی کد فوق:

```
    .ELSEIF uMsg==WM_CREATE
        invoke CreateWindowEx,WS_EX_CLIENTEDGE, \
                    ADDR EditClassName,NULL,\
                    WS_CHILD or WS_VISIBLE or WS_BORDER or ES_LEFT\
                    or ES_AUTOHSCROLL,\
                    50,35,200,25,hWnd,EditID,hInstance,NULL
        mov  hwndEdit,eax
        invoke SetFocus, hwndEdit
        invoke CreateWindowEx,NULL, ADDR ButtonClassName,\
                    ADDR ButtonText,\
                    WS_CHILD or WS_VISIBLE or BS_DEFPUSHBUTTON,\
                    75,70,140,25,hWnd,ButtonID,hInstance,NULL
        mov  hwndButton,eax
```

کنترل‌ها در حین پردازش پیغام WM_CREATE ایجاد می‌شوند.

فراخوانی تابع CreateWindowEx با حالت WS_EX_CLIENTEDGE سبب نمایش سطحی فرورفته می‌گردد. نام کلاس هر کنترل از پیش تعریف شده بوده و معادل edit برای ادیت باکس و button برای دکمه می‌باشد.

هر کنترل علاوه بر حالتهای ممکن برای پنجره والد ، دارای حالتهای مخصوص به خود نیز می باشد. برای مثال حالتهای اضافی یک دکمه با _BS و حالتهای ادیت باکس با _ES شروع می‌شوند. برای مطالعه بیشتر این موارد می‌توان به MSDN مراجعه کرد.

لازم به ذکر است که ID کنترل بجای دستگیره منو ، بکار گرفته شده است. این مورد مشکلی را ایجاد نخواهد کرد زیرا کنترل‌ها نمی‌توانند دارای منو باشند.

بعد از ایجاد هر کنترلی ، می‌توان دستگیره آنرا برای استفاده‌های آتی در یک متغیر ذخیره کرد.

از تابع SetFocus جهت انتقال تمرکز به ادیت باکس استفاده گردید. در این حالت بلافاصله پس از نمایش پنجره ، می‌توان به سادگی از ادیت باکسی دارای فوکوس و انتخاب شده ، کمک گرفت.

همانطورکه ذکر شد، هر کنترل توسط پیغام WM_COMMAND با پنجره‌ی والد ارتباط برقرار می‌کند.

```
.ELSEIF uMsg==WM_COMMAND
   mov eax,wParam
   .IF lParam==0
```

از مثالهای قبل بخاطر داریم که منوها نیز از پیغام WM_COMMAND جهت بر قراری ارتباط با پنجره‌ی والد کمک می‌گیرند.اما چگونه می‌توان بین این دو مورد تمیز قائل شد؟

پاسخ در جدول زیر مشخص شده است:

	Low word of wParam	High word of wParam	lParam
Menu	Menu ID	0	0
Control	Control ID	Notification code	Child Window Handle

بنابراین اگر lParam مساوی صفر بود پیغام از طرف منو صادر شده است. از wParam نمی‌توان استفاده کرد زیرا احتمال صفر بودن کد باخبرسازی کنترل ، وجود دارد. در ادامه :

```
.IF ax==IDM_HELLO
   invoke SetWindowText,hwndEdit,ADDR TestString
.ELSEIF ax==IDM_CLEAR
   invoke SetWindowText,hwndEdit,NULL
.ELSEIF  ax==IDM_GETTEXT
   invoke GetWindowText,hwndEdit,ADDR buffer,512
   invoke MessageBox,NULL,ADDR buffer,ADDR AppName,MB_OK
```

برای قرار دادن رشته‌های متنی درون ادیت باکس از تابع SetWindowText می‌توان استفاده کرد و اگر نیاز بود تا محتوای آنرا خالی نمود ، می توان از پارامتر null بعنوان ورودی کمک گرفت.

تابع SetWindowText یکی از توابع با کاربرد عمومی API ویندوز محسوب می شود. از آن برای تغییر عنوان یک پنجره و یا متن روی یک دکمه می توان استفاده کرد.

برای دریافت متن درون یک ادیت باکس می‌توان از تابع GetWindowText کمک گرفت. در ادامه :

```
.IF ax==ButtonID
        shr eax,16
        .IF ax==BN_CLICKED
          invoke SendMessage,hWnd,WM_COMMAND,IDM_GETTEXT,0
        .ENDIF
    .ENDIF
```

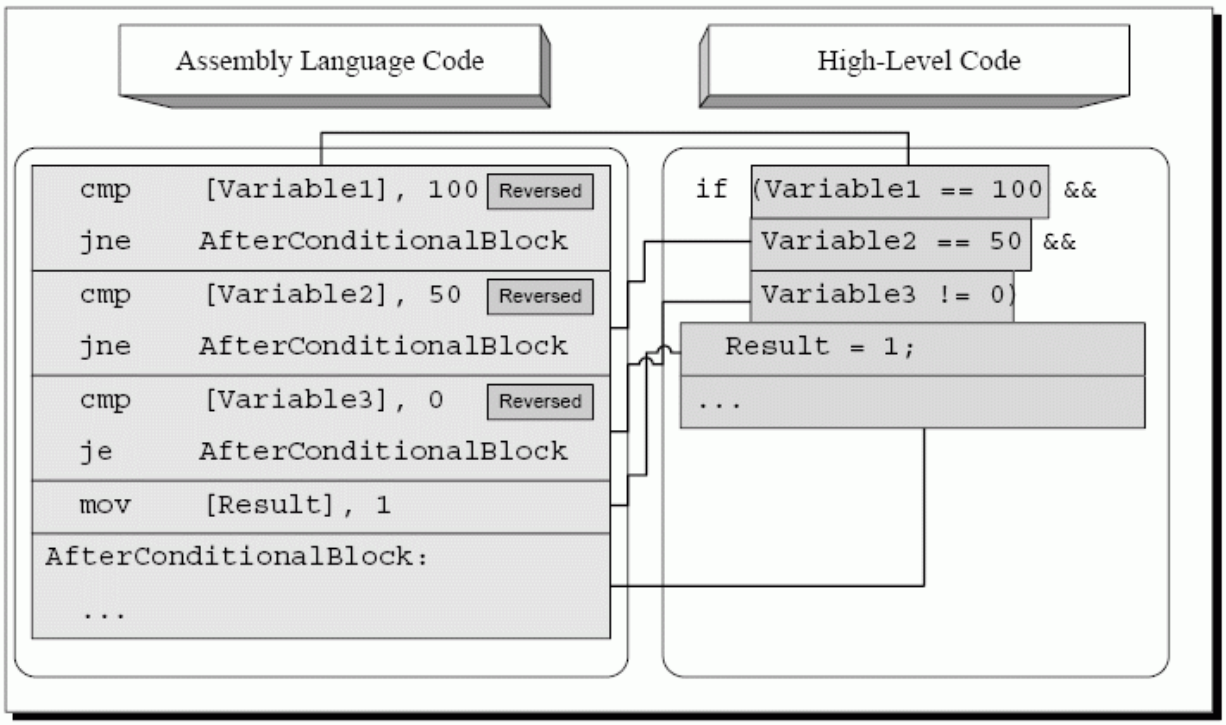
کد بالا هنگامیکه بر روی دکمه برنامه کلیک شود اجرا خواهد شد. در ابتدا low word مربوط به wParam با ID دکمه مقایسه می‌شود. سپس high word مربوط به wParam با کد باخبر سازی دکمه (BN_CLICKED) ، مقایسه می گردد.

شکل ۴– تصویری از اجرای برنامه کار با کنترل‌ها

برای اینکه از دوباره نویسی کد در این قسمت اجتناب شود ، بهتر است با پیغامی همانند پیغام منوی مربوطه که به پنجره ارسال می‌شود، عملیات نمایش Message Box را انجام داد. با کمک تابع SendMessage اینکار انجام پذیر است.

در پایان لازم به ذکر است که اگر تابع TranslateMessage فراخوانی نشود ، عملیات دریافت ورودی از صفحه کلید و ترجمه آن به متن قابل استفاده در ادیت باکس انجام نخواهد شد.

مقایسه کد زبان سی و کد اسمبلی یک شرط با ترکیبات مختلفی از عملگرهای منطقی



در تصویر فوق نحوه پیاده سازی یک شرط مرکب از چندین and را ملاحظه می‌نمایید. کد اسمبلی ارائه شده بهینه سازی شده نیز می‌باشد.

مثال ۸- استفاده از دیالوگ باکس‌ها بعنوان پنجره اصلی

اگر به مثال قبلی بیشتر دقت کرده باشید حتما متوجه شده اید که امکان تغییر تمرکز (فوکوس) ورودی ، از یک کنترل به کنترل دیگر توسط فشردن کلید tab میسر نیست. تنها راه انتخاب یک کنترل در این حالت ، انتخاب کنترل توسط کلیک ماوس است.

پیش از توضیحات بیشتر نیاز است تا با دیالوگ باکس‌ها بیشتر آشنا شویم. دیالوگ باکس چیزی بیشتر از یک پنجره متداول جهت کاربر با کنترل‌ها نیست. همچنین ویندوز برای کار با دیالوگ باکس‌ها از یک مدیر داخلی استفاده می‌کند. به این صورت دیگر نیازی نیست تا برنامه نویس درگیر مدیریت انتقال تمرکز به سایر کنترل‌ها و امثال آن شود.

تعریف یک دیالوگ باکس توسط فایلهای منبع (ریسورس) انجام می‌شود. بهتر است برای تعریف اینگونه فایلهای ریسورس از ابزارهای بصری مهیا در بسیاری از کامپایلرها استفاده شود. لازم به ذکر است که تمامی ریسورس‌های یک برنامه در آخر در یک فایل منبع قرار می‌گیرند.

دو نوع کلی دیالوگ باکس وجود دارد : modal and modeless .در حالت modeless (برخلاف حالت modal) می‌توان تمرکز ورودی را به سایر پنجره ها نیز انتقال داد (برای مثال پنجره جستجو در MS-Word). پنجره های modal نیز دارای دو حالت هستند : application modal and system modal . در حالت application modal ، امکان تغییر تمرکز ورودی به سایر پنجره‌های یک برنامه نیست، ولی پنجره‌های سایر برنامه‌ها را به سادگی می‌توان انتخاب کرد. اما در حالت system modal امکان انتخاب سایر پنجره‌های برنامه های دیگر نیز نیست. بنابراین ابتدا باید عملیات و پاسخ دهی به این پنجره انجام و تمام شود و سپس می‌توان به سایر برنامه‌ها پرداخت.

دیالوگ باکس‌هایی با حالت modeless با فراخوانی تابع CreateDialogParam ایجاد شده و دیالوگ باکس‌هایی با حالت مودال توسط تابع DialogBoxParam ایجاد می‌گردند. اگر به قالب دیالوگ باکس مودال ، حالت DS_SYSMODAL نیز اضافه شود ، دایالوگ باکس از نوع system modal خواهد شد.

برای تبادل پیغام با کنترل‌های قرار گرفته بر روی دیالوگ باکس‌ها از تابع SendDlgItemMessage باید استفاده کرد.تعریف آن به صورت زیر است:

```
SendDlgItemMessage proto hwndDlg:DWORD,\
                         idControl:DWORD,\
                         uMsg:DWORD,\
                         wParam:DWORD,\
                         lParam:DWORD
```

برای مثال جهت دریافت متن از ادیت کنترل به صورت زیر می توان عمل کرد:

```
call SendDlgItemMessage, hDlg, ID_EDITBOX, WM_GETTEXT, 256, ADDR text_buffer
```

برای یافتن لیست پیغامهای مجاز آن باید به MSDN مراجعه کرد.

همچنین ویندوز برای کار با یک سری از کنترل ها توابع ویژه ای نیز دارد. برای مثال GetDlgItemText و CheckDlgButton و غیره جهت کار با دکمه ها و تکست باکس‌ها. این نوع توابع ، برنامه نویسی را ساده تر کرده و برنامه نویس را از بکارگیری جزئیات بیشتری مانند lParam و غیره ، بی نیاز می کند. بنابراین تنها در حالتی از SendDlgItemMessage استفاده نمایید که توابع مخصوص کنترل وجود نداشته باشند.

ویندوز پیغام‌های مربوط به دیالوگ باکس را به یک تابع callback که فرمت زیر را دارد ارسال می کند:

```
DlgProc  proto hDlg:DWORD ,\
               iMsg:DWORD ,\
               wParam:DWORD ,\
               lParam:DWORD
```

رویه دیالوگ باکس همانند رویه پنجره است ، اما خروجی آن بجای LRESULT ، TRUE و یا FALSE می‌باشد.رویه حقیقی و اصلی دیالوگ باکس توسط خود ویندوز اداره می‌شود. این رویه ، تابع callback ما را هنگامیکه پیغامی به آن برسد با خبر خواهد کرد. این رویه اگر پیغام رسیده را پردازش می نماید باید خروجی TRUE را در EAX قرار داده و اگر قصد پردازش آنرا ندارد باید مقدار FALSE را در EAX بعنوان خروجی تابع قرار دهد.

از آنجائیکه رویه دیالوگ باکس تعریف شده در بالا رویه حقیقی آن نیست، پیغامهایی را که پردازش نمی‌نماید به DefWindowProc ارسال نمی کند.

دو کاربرد مجزا برای دیالوگ باکس وجود دارد. استفاده از آن بعنوان پنجره اصلی برنامه و یا استفاده از آن بعنوان وسیله ورود اطلاعات.

پیاده سازی دیالوگ باکس به صورت پنجره اصلی برنامه به دو شکل ممکن است:

۱. می‌توان از قالب دیالوگ باکس بعنوان قالب کلاسی که با فراخوانی RegisterClassEx رجیستر می‌شود، استفاده کرد. در این حالت دیالوگ باکس بعنوان یک پنجره معمول رفتار خواهد کرد. در این حالت دیالوگ باکس پیغام ها را از طریق رویه پنجره مشخص شده توسط پارامتر lpfnWndProc ، دریافت می کند (نه از طریق رویه دیالوگ باکس). مزیت این روش این است که ویندوز هنگام ایجاد دیالوگ باکس ، کنترل های قرار گرفته بر روی آنرا نیز ایجاد می کند. همچنین ویندوز اعمالی مانند انتقال تمرکز ورودی توسط tab و غیره را مدیریت خواهد کرد. اینجا دیالوگ باکس بدون ایجاد پنجره والد ایجاد می‌شود.

کد برنامه (مربوط به روش اول):

```
.386
.model flat,stdcall
option casemap:none
WinMain proto :DWORD,:DWORD,:DWORD,:DWORD
include \masm32\include\windows.inc
include \masm32\include\user32.inc
include \masm32\include\kernel32.inc
includelib \masm32\lib\user32.lib
includelib \masm32\lib\kernel32.lib

.data
ClassName db "DLGCLASS",0
MenuName db "MyMenu",0
DlgName db "MyDialog",0
AppName db "Our First Dialog Box",0
TestString db "Wow! I'm in an edit box now",0

.data?
hInstance HINSTANCE ?
CommandLine LPSTR ?
buffer db 512 dup(?)

.const
IDC_EDIT       equ 3000
IDC_BUTTON     equ 3001
IDC_EXIT       equ 3002
IDM_GETTEXT    equ 32000
IDM_CLEAR      equ 32001
IDM_EXIT       equ 32002

.code
start:
   invoke GetModuleHandle, NULL
   mov    hInstance,eax
   invoke GetCommandLine
   mov CommandLine,eax
   invoke WinMain, hInstance,NULL,CommandLine, SW_SHOWDEFAULT
   invoke ExitProcess,eax

WinMain proc hInst:HINSTANCE,hPrevInst:HINSTANCE,CmdLine:LPSTR,CmdShow:DWORD
   LOCAL wc:WNDCLASSEX
   LOCAL msg:MSG
   LOCAL hDlg:HWND
   mov   wc.cbSize,SIZEOF WNDCLASSEX
   mov   wc.style, CS_HREDRAW or CS_VREDRAW
   mov   wc.lpfnWndProc, OFFSET WndProc
   mov   wc.cbClsExtra,NULL
   mov   wc.cbWndExtra,DLGWINDOWEXTRA
   push  hInst
   pop   wc.hInstance
   mov   wc.hbrBackground,COLOR_BTNFACE+1
   mov   wc.lpszMenuName,OFFSET MenuName
   mov   wc.lpszClassName,OFFSET ClassName
   invoke LoadIcon,NULL,IDI_APPLICATION
   mov   wc.hIcon,eax
   mov   wc.hIconSm,eax
```

```
    invoke LoadCursor,NULL,IDC_ARROW
    mov   wc.hCursor,eax
    invoke RegisterClassEx, addr wc
    invoke CreateDialogParam,hInstance,ADDR DlgName,NULL,NULL,NULL
    mov   hDlg,eax
    invoke ShowWindow, hDlg,SW_SHOWNORMAL
    invoke UpdateWindow, hDlg
    invoke GetDlgItem,hDlg,IDC_EDIT
    invoke SetFocus,eax
    .WHILE TRUE
       invoke GetMessage, ADDR msg,NULL,0,0
       .BREAK .IF (!eax)
      invoke IsDialogMessage, hDlg, ADDR msg
       .IF eax ==FALSE
          invoke TranslateMessage, ADDR msg
          invoke DispatchMessage, ADDR msg
       .ENDIF
    .ENDW
    mov     eax,msg.wParam
    ret
WinMain endp

WndProc proc hWnd:HWND, uMsg:UINT, wParam:WPARAM, lParam:LPARAM
    .IF uMsg==WM_DESTROY
       invoke PostQuitMessage,NULL
    .ELSEIF uMsg==WM_COMMAND
       mov eax,wParam
       .IF lParam==0
          .IF ax==IDM_GETTEXT
             invoke GetDlgItemText,hWnd,IDC_EDIT,ADDR buffer,512
             invoke MessageBox,NULL,ADDR buffer,ADDR AppName,MB_OK
          .ELSEIF ax==IDM_CLEAR
             invoke SetDlgItemText,hWnd,IDC_EDIT,NULL
          .ELSE
             invoke DestroyWindow,hWnd
          .ENDIF
       .ELSE
          mov edx,wParam
          shr edx,16
          .IF dx==BN_CLICKED
             .IF ax==IDC_BUTTON
                invoke SetDlgItemText,hWnd,IDC_EDIT,ADDR TestString
             .ELSEIF ax==IDC_EXIT
                invoke SendMessage,hWnd,WM_COMMAND,IDM_EXIT,0
             .ENDIF
          .ENDIF
       .ENDIF
    .ELSE
       invoke DefWindowProc,hWnd,uMsg,wParam,lParam
       ret
    .ENDIF
    xor    eax,eax
    ret
WndProc endp
end start
```

محتویات فایل منبع برنامه فوق:

```
#include "resource.h"

#define IDC_EDIT                3000
#define IDC_BUTTON              3001
#define IDC_EXIT                3002

#define IDM_GETTEXT             32000
#define IDM_CLEAR               32001
#define IDM_EXIT                32003


MyDialog DIALOG 10, 10, 205, 60
STYLE 0x0004 | DS_CENTER | WS_CAPTION | WS_MINIMIZEBOX |
WS_SYSMENU | WS_VISIBLE | WS_OVERLAPPED | DS_MODALFRAME | DS_3DLOOK
CAPTION "Our First Dialog Box"
CLASS "DLGCLASS"
BEGIN
   EDITTEXT        IDC_EDIT,   15,17,111,13, ES_AUTOHSCROLL | ES_LEFT
   DEFPUSHBUTTON   "Say Hello", IDC_BUTTON,    141,10,52,13
   PUSHBUTTON      "E&xit", IDC_EXIT,  141,26,52,13, WS_GROUP
END


MyMenu  MENU
BEGIN
   POPUP "Test Controls"
   BEGIN
      MENUITEM "Get Text", IDM_GETTEXT
      MENUITEM "Clear Text", IDM_CLEAR
      MENUITEM "", , 0x0800 /*MFT_SEPARATOR*/
      MENUITEM "E&xit", IDM_EXIT
   END
END
```

بررسی کد فوق (مربوط به روش اول):

این مثال نحوه ثبت قالب دیالوگ را بعنوان کلاس پنجره و ایجاد پنجره ای متداول را نمایش می‌دهد. این روش برنامه نویسی را ساده تر می نماید زیرا ایجاد کنترل‌ها به صورت خودکار انجام می‌شود.

```
MyDialog DIALOG 10, 10, 205, 60
```

در اینجا نام دیالوگ (MyDialog) به همراه کلمه کلیدی DIALOG ، تعریف شده است. سایر پارامترها مساوی مختصات گوشه بالا سمت چپ فرم و طول و عرض آن در واحد دیالوگ باکس است (با pixel یکی نیست). در ادامه:

```
STYLE 0x0004 | DS_CENTER | WS_CAPTION | WS_MINIMIZEBOX |
WS_SYSMENU | WS_VISIBLE | WS_OVERLAPPED | DS_MODALFRAME | DS_3DLOOK
```

حالت دیالوگ باکس را تعریف می کند.

```
CAPTION "Our First Dialog Box"
```

متنی است که در نوار عنوان پنجره نمایش داده می‌شود.

```
CLASS "DLGCLASS"
```

این خط از کد بسیار مهم می‌باشد.به کمک کلمه کلیدی class ، از قالب دیالوگ باکس بعنوان کلاس پنجره می‌توان استفاده کرد. کلمه ای که پس از آن می‌آید نام کلاس است. در ادامه :

```
BEGIN
   EDITTEXT        IDC_EDIT,   15,17,111,13, ES_AUTOHSCROLL | ES_LEFT
   DEFPUSHBUTTON   "Say Hello", IDC_BUTTON,    141,10,52,13
   PUSHBUTTON      "E&xit", IDC_EXIT,  141,26,52,13
END
```

قطعه کد فوق کنترل‌های تعریف شده بر روی پنجره را بین begin و end تعریف می کند. نحوه تعریف عمومی آن به صورت زیر است:

```
control-type   "text"   ,controlID, x, y, width, height [,styles]
```

قسمت جالب توجه این کد، ساختار کلاس می‌باشد :

```
mov   wc.cbWndExtra,DLGWINDOWEXTRA
mov   wc.lpszClassName,OFFSET ClassName
```

به صورت معمول عضو cbWndExtra ، نال در نظر گرفته می‌شود.اما اگر قالب یک دیالوگ باکس را بخواهیم بعنوان یک کلاس پنجره رجیستر کنیم ، باید این عضو را با DLGWINDOWEXTRA مقدار دهی کنیم. در ادامه باید دقت داشت که نام کلاس باید با نام تعریف شده آن توسط کلمه کلیدی class یکی باشد. سایر اعضای این ساختار مانند مثالهای قبل مقدار دهی خواهند شد.

پس از پر کردن ساختار کلاس ، آنرا با فراخوانی تابع RegisterClassEx رجیستر می کنیم.در ادامه :

```
invoke CreateDialogParam,hInstance,ADDR DlgName,NULL,NULL,NULL
```

پس از رجیستر کردن کلاس پنجره ، دیالوگ باکس خود را ایجاد می کنیم. تابع فوق دیالوگ باکسی modeless را ایجاد می کند.این تابع ۵ آرگومان را می‌پذیرد ، اما پر کردن دو پارامتر اول آن کفایت می کند. پارامتر اول دستگیره ای به وهله برنامه بوده و پارامتر دوم اشاره‌گری است به نام قالب دیالوگ باکس (و نه به نام کلاس).

در این لحظه ، دیالوگ باکس و کنترل‌های آن توسط ویندوز ایجاد شده و رویه پنجره ، پیغام WM_CREATE را طبق معمول دریافت می کنند. در ادامه:

```
invoke GetDlgItem,hDlg,IDC_EDIT
invoke SetFocus,eax
```

پس از ایجاد ادیت باکس ، تمرکز ورودی را به آن انتقال می‌دهیم.

اگر این قطعه کد را در قسمت WM_CREATE قرار دهیم، فراخوانی تابع GetDlgItem با شکست مواجه خواهد شد (زیرا هنوز ایجاد نشده است). بنابراین آنرا پس از فراخوانی UpdateWindow قرار داده‌ایم. تابع GetDlgItem ، ID کنترل را دریافت کرده و دستگیره پنجره وابسته به کنترل را باز می گرداند. (این روشی است که با دانستن ID یک کنترل می‌توان دستگیره پنجره مربوطه را بازگشت داد)

```
invoke IsDialogMessage, hDlg, ADDR msg
.IF eax ==FALSE
   invoke TranslateMessage, ADDR msg
   invoke DispatchMessage, ADDR msg
.ENDIF
```

به کمک فراخوانی تابع IsDialogMessage ، به مدیریت درونی دیالوگ باکس اجازه می دهیم تا رخ داد های صفحه کلید را برای ما پردازش کند. بازگشتی این تابع TRUE است.

همچنین یکی دیگر از تفاوت های این برنامه با برنامه‌های قبلی ، استفاده از تابع GetDlgItemText جهت دریافت رشته ورودی است. استفاده از این تابع هنگام کار با دیالوگ باکس ها سبب سادگی کار می‌شود.

در مثال دومی از این دست ، دیالوگ باکسی به صورت modal ایجاد خواهیم کرد. در این حالت حلقه ی پیغامها و یا رویه پنجره را نخواهید یافت. زیرا نیازی به آنها نیست!

کد برنامه (مربوط به روش دوم):

```
.386
.model flat,stdcall
option casemap:none

DlgProc proto :DWORD,:DWORD,:DWORD,:DWORD

include \masm32\include\windows.inc
include \masm32\include\user32.inc
include \masm32\include\kernel32.inc
includelib \masm32\lib\user32.lib
includelib \masm32\lib\kernel32.lib

.data
DlgName db "MyDialog",0
AppName db "Our Second Dialog Box",0
TestString db "Wow! I'm in an edit box now",0

.data?
hInstance HINSTANCE ?
CommandLine LPSTR ?
buffer db 512 dup(?)

.const
IDC_EDIT        equ 3000
IDC_BUTTON      equ 3001
IDC_EXIT        equ 3002
IDM_GETTEXT     equ 32000
IDM_CLEAR       equ 32001
IDM_EXIT        equ 32002


.code
start:
   invoke GetModuleHandle, NULL
   mov    hInstance,eax
   invoke DialogBoxParam, hInstance, ADDR DlgName,NULL, addr DlgProc, NULL
   invoke ExitProcess,eax

DlgProc proc hWnd:HWND, uMsg:UINT, wParam:WPARAM, lParam:LPARAM
   .IF uMsg==WM_INITDIALOG
      invoke GetDlgItem, hWnd,IDC_EDIT
      invoke SetFocus,eax
   .ELSEIF uMsg==WM_CLOSE
      invoke SendMessage,hWnd,WM_COMMAND,IDM_EXIT,0
   .ELSEIF uMsg==WM_COMMAND
      mov eax,wParam
      .IF lParam==0
         .IF ax==IDM_GETTEXT
            invoke GetDlgItemText,hWnd,IDC_EDIT,ADDR buffer,512
            invoke MessageBox,NULL,ADDR buffer,ADDR AppName,MB_OK
         .ELSEIF ax==IDM_CLEAR
            invoke SetDlgItemText,hWnd,IDC_EDIT,NULL
         .ELSEIF ax==IDM_EXIT
            invoke EndDialog, hWnd,NULL
         .ENDIF
```

```
      .ELSE
         mov edx,wParam
         shr edx,16
         .if dx==BN_CLICKED
            .IF ax==IDC_BUTTON
               invoke SetDlgItemText,hWnd,IDC_EDIT,ADDR TestString
            .ELSEIF ax==IDC_EXIT
               invoke SendMessage,hWnd,WM_COMMAND,IDM_EXIT,0
            .ENDIF
         .ENDIF
      .ENDIF
   .ELSE
      mov eax,FALSE
      ret
   .ENDIF
   mov eax,TRUE
   ret
DlgProc endp
end start
```

dialog.rc (part 2)

```
#include "resource.h"

#define IDC_EDIT                3000
#define IDC_BUTTON              3001
#define IDC_EXIT                3002

#define IDR_MENU1               3003

#define IDM_GETTEXT             32000
#define IDM_CLEAR               32001
#define IDM_EXIT                32003


MyDialog DIALOG 10, 10, 205, 60
STYLE 0x0004 | DS_CENTER | WS_CAPTION | WS_MINIMIZEBOX |
WS_SYSMENU | WS_VISIBLE | WS_OVERLAPPED | DS_MODALFRAME | DS_3DLOOK
CAPTION "Our Second Dialog Box"
MENU IDR_MENU1
BEGIN
   EDITTEXT        IDC_EDIT,   15,17,111,13, ES_AUTOHSCROLL | ES_LEFT
   DEFPUSHBUTTON   "Say Hello", IDC_BUTTON,    141,10,52,13
   PUSHBUTTON      "E&xit", IDC_EXIT,  141,26,52,13
END


IDR_MENU1  MENU
BEGIN
   POPUP "Test Controls"
   BEGIN
      MENUITEM "Get Text", IDM_GETTEXT
      MENUITEM "Clear Text", IDM_CLEAR
      MENUITEM "", , 0x0800 /*MFT_SEPARATOR*/
      MENUITEM "E&xit", IDM_EXIT
   END
END
```

بررسی کد فوق (مربوط به روش دوم):

```
DlgProc proto :DWORD,:DWORD,:DWORD,:DWORD
```

در خط فوق تعریف رویه DlgProc که در کد زیر با اپراتور addr مورد استفاده قرار می‌گیرد ، مشخص شده است:

```
invoke  DialogBoxParam, hInstance, ADDR DlgName,NULL, addr  DlgProc, NULL
```

تابع فوق دیالوگ باکسی مودال را ایجاد خواهد کرد. این تابع تا زمانی که دیالوگ باکس تخریب نشود به پایان نخواهد رسید. در ادامه :

```
.IF uMsg==WM_INITDIALOG
   invoke GetDlgItem, hWnd,IDC_EDIT
   invoke SetFocus,eax
.ELSEIF uMsg==WM_CLOSE
   invoke SendMessage,hWnd,WM_COMMAND,IDM_EXIT,0
```

رویه دیالوگ باکس شبیه به رویه پنجره بوده با این تفاوت که پیغام WM_CREATE را دریافت نخواهد کرد. اولین پیغامی را که دریافت خواهد کرد WM_INITDIALOG می‌باشد.کدهای آغازین برنامه را در این قسمت می‌توان قرار داد. لازم به ذکر است ، در صورتیکه اینجا پیغامی پردازش شود باید خروجی رویه که در EAX قرار می‌گیرد با TRUE مقدار دهی شود.

هنگامیکه پیغام WM_CLOSE به دیالوگ باکس فرستاده می‌شود ، مدیریت درونی دیالوگ باکس، پیغام WM_DESTROY را برای رویه دیالوگ باکس ارسال نمی‌کند. بنابراین برای عکس العمل نشان دادن به بسته شدن دیالوگ باکس توسط کاربر، باید پیغام WM_CLOSE را پردازش کرد.در اینجا عمل کلیک شدن بر روی گزینه خروج منو را شبیه سازی کرده ایم.

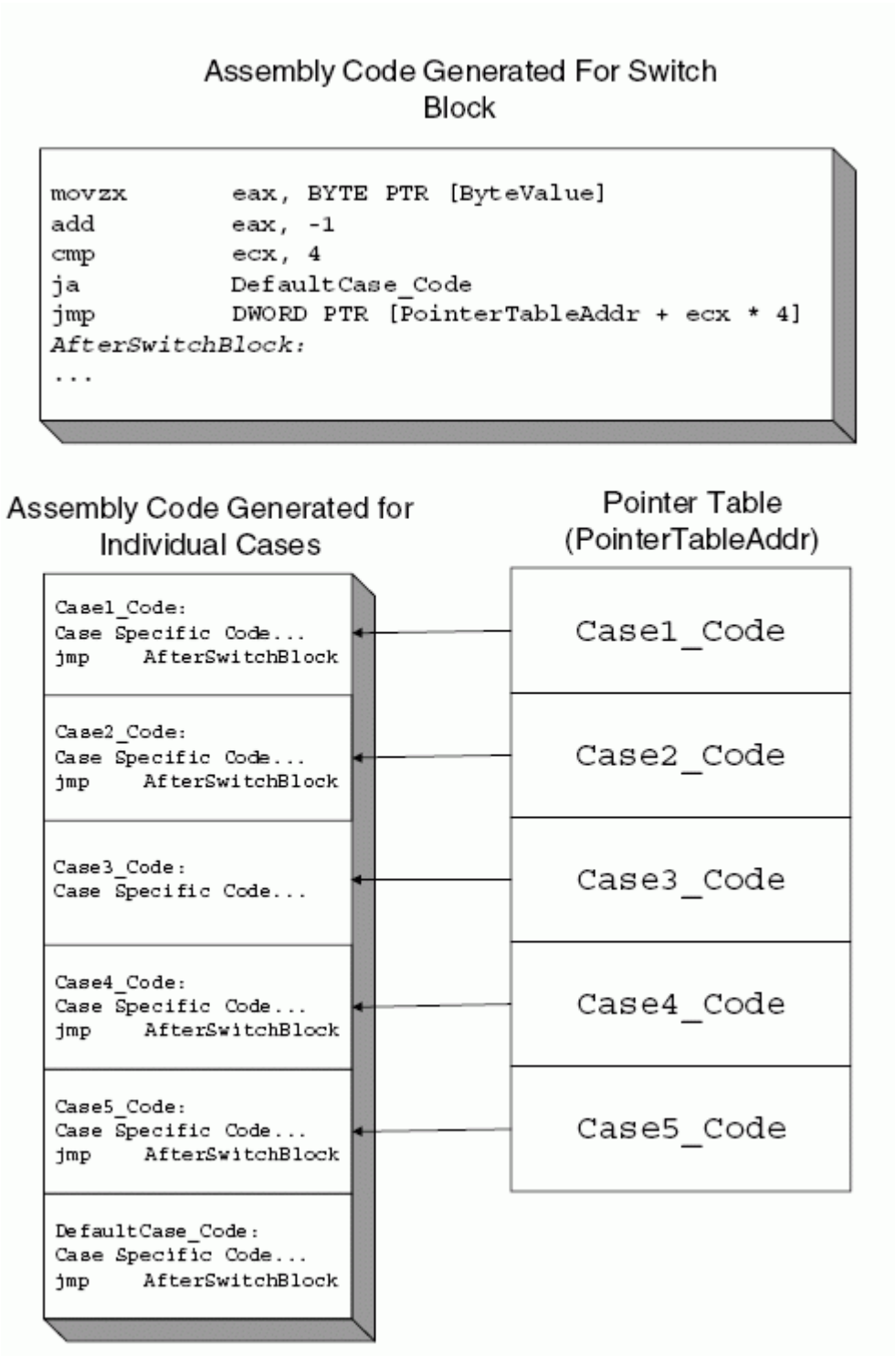
تنها راه تخریب دیالوگ باکس فراخوانی EndDialog است.فراخوانی این تابع سریعا سبب تخریب دیالوگ باکس نمی‌شود. این تابع تنها پرچم مربوطه را در مدیریت درونی دیالوگ باکس به TRUE تنظیم می‌کند و باقیمانده کد یکی پس از دیگری اجرا خواهند شد.

بررسی فایل منبع این مثال:

بجای بکارگیری رشته بعنوان نام منو ، از مقدار IDR_MENU1 استفاده شده است. اینکار برای الحاق کردن منو به دیالوگ باکس توسط DialogBoxParam لازم است. لازم به ذکر است که در قالب دیالوگ باکس ، از کلمه کلیدی MENU به همراه آی دی مربوطه استفاده می‌شود.

با ارسال پیغام WM_SETICON به دیالوگ باکس در حین WM_INITDIALOG ، می‌توان آیکون آنرا تنظیم کرد.

مقایسه کد زبان سی و کد اسمبلی یک Switch



کد اسمبلی فوق جهت switch زیر ارائه گردید:

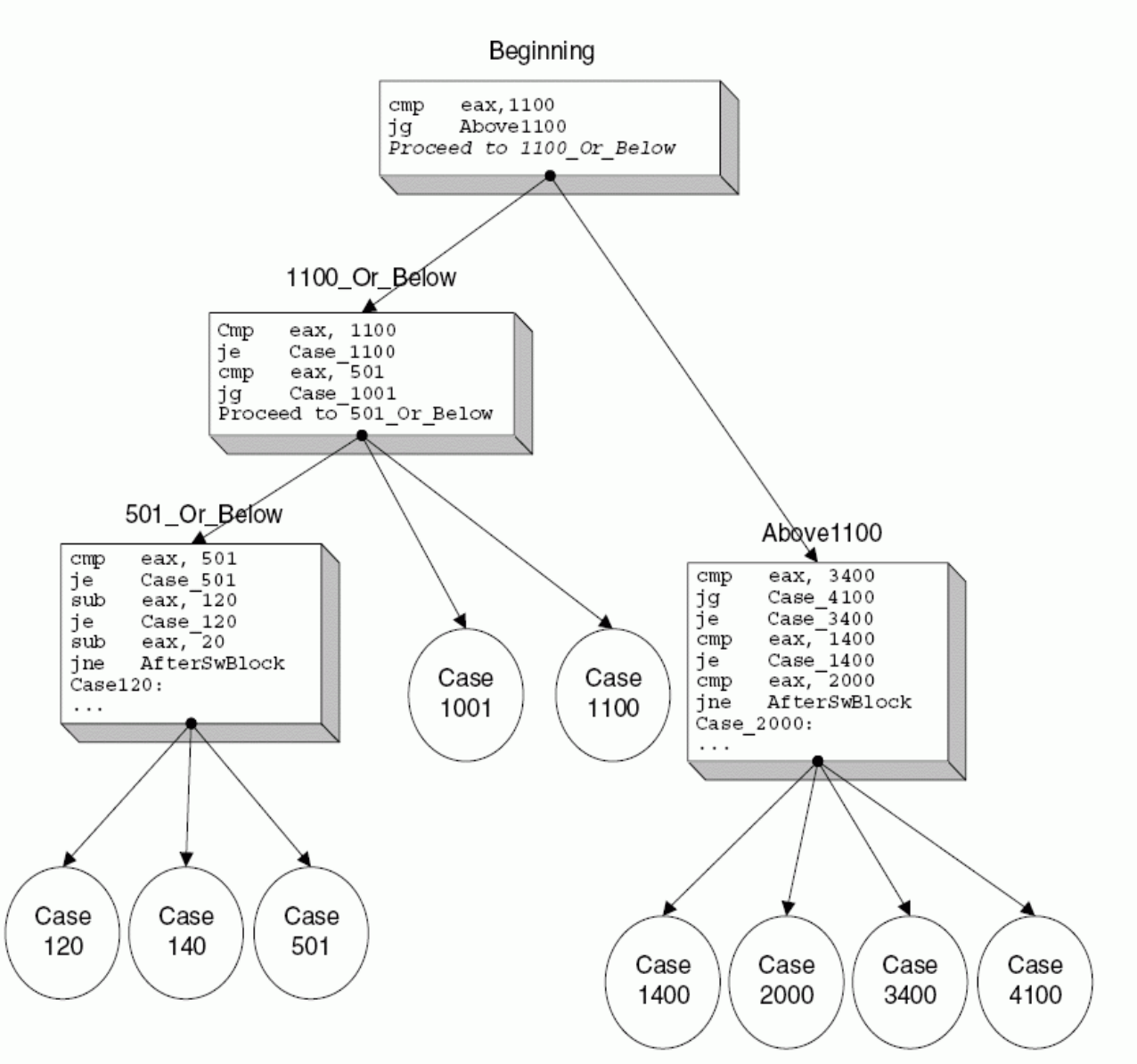
Original Source Code

```
Switch (ByteValue)
{
      case  1:
            Case Specific Code...
            break;
      case  2:
            Case Specific Code...
            break;
      case  3:
            Case Specific Code...
      case  4:
            Case Specific Code...
            break;
      case  5:
            Case Specific Code...
            break;
      default:
            Case Specific Code...
            break;
};
```

هنگامیکه رفتارهای مختلفی را از مقادیر مختلف یک عملگر انتظار داریم ، از سوئیچ استفاده می کنیم. سوئیچ درحقیقت جدولی از مقادیر برای یک عملگر و واکنش‌های مناسب به این مقادیر می‌باشد. همچنین امکان ارائه یک واکنش برای چندین مقدار نیز میسر است.

یکی از بهینه ترین روش‌ها برای پیاده سازی سوئیچ ها ، ایجاد جدولی از اشاره‌گرها است.هر کدام از قطعات یک سوئیچ به صورت جداگانه کامپایل شده و سپس اشاره‌گری به هر کدام از این قطعات در یک جدول ذخیره می‌شوند.هنگام نیاز به اجرای قطعه کد قرار گرفته در یکی از شرط‌های سوئیچ ، عملگر مربوطه بعنوان اندیسی در این جدول اشاره‌گرها رفتار کرده و پردازشگر به سادگی به قطعه کد مربوطه پرش خواهد کرد.

شکل زیر دیدگاه یک کامپایلر را از یک سوئیچ و نحوه ی اجرای آنرا بیان می کند:



این درخت ، پیاده سازی سوئیچ زیر می‌باشد:

```
switch (Value)
{
	case 120:
		Code...
		break;
	case 140:
		Code...
		break;
	case 501:
		Code...
		break;
	case 1001:
		Code...
```

```
                break;
        case 1100:
                Code...
                break;
        case 1400:
                Code...
                break;
        case 2000:
                Code...
                break;
        case 3400:
                Code...
                break;
        case 4100:
                Code...
                break;
};
```

مثال ۹– استفاده از دیالوگ باکس‌ها (موارد تکمیلی)

در این مثال نحوه استفاده از common dialog boxes را خواهیم آموخت.

ویندوز یک سری پنجره‌های از پیش تعریف شده جهت استفاده در برنامه های شما دارد. هدف از ارائه آنها نیز استاندارد کردن رابط‌های کاربر برنامه‌ها می‌باشد. این پنجره‌ها شامل موارد زیر هستند:

file, print, color, font, and search dialog boxes

این دیالوگ باکس‌های استاندارد در comdlg32.dll قرار دارند. به همین جهت برای استفاده از آنها نیاز است تا comdlg32.lib به برنامه الحاق شود.

برای فراخوانی این دیالوگ باکس‌های استاندارد می‌توان از توابع مخصوص اینکار استفاده کرد مانند GetOpenFileName برای نمایش پنجره گشودن فایلها ، برای نمایش دیالوگ باکس ذخیره بعنوان ... از GetSaveFileName و غیره. هر کدام از این توابع برای کار نیاز به اشاره گری به ساختاری ویژه دارند (که در MSDN قابل مطالعه هستند).

در این مثال قصد نمایش open file dialog را داریم. تعریف تابع GetOpenFileName به صورت زیر است:

```
GetOpenFileName proto lpofn:DWORD
```

تنها پارامتر این تابع اشاره گری است به ساختار OPENFILENAME . اگر خروجی این تابع TRUE باشد یعنی کاربر فایلی را انتخاب کرده است ، در غیر اینصورت خیر. تعریف این ساختار به صورت زیر است:

```
OPENFILENAME  STRUCT
  lStructSize DWORD  ?
  hwndOwner HWND  ?
  hInstance HINSTANCE ?
  lpstrFilter LPCSTR  ?
  lpstrCustomFilter LPSTR  ?
  nMaxCustFilter DWORD  ?
  nFilterIndex DWORD  ?
  lpstrFile LPSTR  ?
  nMaxFile DWORD  ?
  lpstrFileTitle LPSTR  ?
  nMaxFileTitle DWORD  ?
  lpstrInitialDir LPCSTR  ?
  lpstrTitle LPCSTR  ?
  Flags  DWORD  ?
  nFileOffset WORD  ?
  nFileExtension WORD  ?
  lpstrDefExt LPCSTR  ?
  lCustData LPARAM  ?
  lpfnHook DWORD  ?
  lpTemplateName LPCSTR  ?
OPENFILENAME  ENDS
```

توضیحات اعضای مهم آن به شرح زیر است:

lStructSize	اندازه ساختار OPENFILENAME به بایت.
hwndOwner	دستگیره ای به پنجره open file dialog box .
hInstance	دستگیره ای به وهله برنامه ای که open file dialog box را ایجاد می کند.
lpstrFilter	فیلتری که برای نمایش فایلها مورد استفاده قرار می گیرد (معرفی شده توسط رشته‌های مختوم به نال). برای مثال: FilterString db "All Files (*.*)",0, "*.*",0 db "Text Files (*.txt)",0,"*.txt",0,0 لازم به ذکر است که تنها الگوی معرفی شده در دومین زوج تعریف گردیده توسط ویندوز برای اعمال فیلتر مورد استفاده قرار می گیرد. همچنین یک صفر اضافی هم در انتهای تعریف رشته فیلتر باید قرار گیرد تا خاتمه ی کل آنرا مشخص کند.
nFilterIndex	تعیین می کند که کدام زوج معرفی شده بعنوان فیلتر باید در آغاز نمایش دیالوگ باکس بعنوان پیش فرض انتخاب شوند. این اندیس از یک شروع می‌شود.
lpstrFile	اشاره‌گر به بافری است که جهت ذخیره کردن نام و مسیر فایل انتخاب شده توسط کاربر ، بکار می‌رود. حداقل طول آن باید 260 bytes باشد.
nMaxFile	اندازه بافر lpstrFile است.
lpstrTitle	اشاره‌گری به عنوان دیالوگ باکس نمایش داده شده.
Flags	ویژگی ها و حالات دیالوگ باکس را تعیین می کند.
nFileOffset	پس از اینکه کاربری فایلی را جهت گشودن انتخاب کرد ، این عضو حاوی اندیس اولین کاراکتر نام فایل می‌شود. برای مثال اگر نام فایل به همراه مسیر کامل آن c:\windows\system\lz32.dll باشد، این عضو مساوی ۱۸ خواهد شد.
nFileExtension	پس از اینکه کاربری فایلی را جهت گشودن انتخاب کرد ، این عضو حاوی اندیس اولین کاراکتر پسوند فایل می‌شود.

در مثال این قسمت پس از نمایش دیالوگ باکس استاندارد گشودن فایل و انتخاب فایل توسط کاربر ، نام فایل به همراه مسیر و پس آن توسط یک message box نمایش داده خواهد شد.

کد برنامه:

```
.386
.model flat,stdcall
option casemap:none
WinMain proto :DWORD,:DWORD,:DWORD,:DWORD
include \masm32\include\windows.inc
include \masm32\include\user32.inc
include \masm32\include\kernel32.inc
include \masm32\include\comdlg32.inc
includelib \masm32\lib\user32.lib
includelib \masm32\lib\kernel32.lib
includelib \masm32\lib\comdlg32.lib

.const
IDM_OPEN equ 1
IDM_EXIT equ 2
MAXSIZE equ 260
OUTPUTSIZE equ 512

.data
ClassName db "SimpleWinClass",0
AppName  db "Our Main Window",0
MenuName db "FirstMenu",0
ofn   OPENFILENAME <>
FilterString db "All Files",0,"*.*",0
          db "Text Files",0,"*.txt",0,0
buffer db MAXSIZE dup(0)
OurTitle db "-=Our First Open File Dialog Box=-: Choose the file to open",0
FullPathName db "The Full Filename with Path is: ",0
FullName  db "The Filename is: ",0
ExtensionName db "The Extension is: ",0
OutputString db OUTPUTSIZE dup(0)
CrLf db 0Dh,0Ah,0

.data?
hInstance HINSTANCE ?
CommandLine LPSTR ?

.code
start:
   invoke GetModuleHandle, NULL
   mov    hInstance,eax
   invoke GetCommandLine
   mov CommandLine,eax
   invoke WinMain, hInstance,NULL,CommandLine, SW_SHOWDEFAULT
   invoke ExitProcess,eax

WinMain proc hInst:HINSTANCE,hPrevInst:HINSTANCE,CmdLine:LPSTR,CmdShow:DWORD
   LOCAL wc:WNDCLASSEX
   LOCAL msg:MSG
   LOCAL hwnd:HWND
   mov   wc.cbSize,SIZEOF WNDCLASSEX
   mov   wc.style, CS_HREDRAW or CS_VREDRAW
   mov   wc.lpfnWndProc, OFFSET WndProc
   mov   wc.cbClsExtra,NULL
   mov   wc.cbWndExtra,NULL
```

```
    push  hInst
    pop   wc.hInstance
    mov   wc.hbrBackground,COLOR_WINDOW+1
    mov   wc.lpszMenuName,OFFSET MenuName
    mov   wc.lpszClassName,OFFSET ClassName
    invoke LoadIcon,NULL,IDI_APPLICATION
    mov   wc.hIcon,eax
    mov   wc.hIconSm,eax
    invoke LoadCursor,NULL,IDC_ARROW
    mov   wc.hCursor,eax
    invoke RegisterClassEx, addr wc
    invoke CreateWindowEx,WS_EX_CLIENTEDGE,ADDR ClassName,ADDR AppName,\
           WS_OVERLAPPEDWINDOW,CW_USEDEFAULT,\
           CW_USEDEFAULT,300,200,NULL,NULL,\
           hInst,NULL
    mov   hwnd,eax
    invoke ShowWindow, hwnd,SW_SHOWNORMAL
    invoke UpdateWindow, hwnd
    .WHILE TRUE
       invoke GetMessage, ADDR msg,NULL,0,0
       .BREAK .IF (!eax)
       invoke TranslateMessage, ADDR msg
       invoke DispatchMessage, ADDR msg
    .ENDW
    mov     eax,msg.wParam
    ret
WinMain endp

WndProc proc hWnd:HWND, uMsg:UINT, wParam:WPARAM, lParam:LPARAM
    .IF uMsg==WM_DESTROY
       invoke PostQuitMessage,NULL
    .ELSEIF uMsg==WM_COMMAND
       mov eax,wParam
       .if ax==IDM_OPEN
          mov ofn.lStructSize,SIZEOF ofn
          push hWnd
          pop  ofn.hwndOwner
          push hInstance
          pop  ofn.hInstance
          mov  ofn.lpstrFilter, OFFSET FilterString
          mov  ofn.lpstrFile, OFFSET buffer
          mov  ofn.nMaxFile,MAXSIZE
          mov  ofn.Flags, OFN_FILEMUSTEXIST or \
             OFN_PATHMUSTEXIST or OFN_LONGNAMES or\
             OFN_EXPLORER or OFN_HIDEREADONLY
          mov  ofn.lpstrTitle, OFFSET OurTitle
          invoke GetOpenFileName, ADDR ofn
          .if eax==TRUE
             invoke lstrcat,offset OutputString,OFFSET FullPathName
             invoke lstrcat,offset OutputString,ofn.lpstrFile
             invoke lstrcat,offset OutputString,offset CrLf
             invoke lstrcat,offset OutputString,offset FullName
             mov  eax,ofn.lpstrFile
             push ebx
             xor  ebx,ebx
             mov  bx,ofn.nFileOffset
             add  eax,ebx
             pop  ebx
             invoke lstrcat,offset OutputString,eax
             invoke lstrcat,offset OutputString,offset CrLf
             invoke lstrcat,offset OutputString,offset ExtensionName
```

```
            mov  eax,ofn.lpstrFile
            push ebx
            xor ebx,ebx
            mov  bx,ofn.nFileExtension
            add eax,ebx
            pop ebx
            invoke lstrcat,offset OutputString,eax
            invoke MessageBox,hWnd,OFFSET OutputString,ADDR AppName,MB_OK
            invoke RtlZeroMemory,offset OutputString,OUTPUTSIZE
         .endif
      .else
         invoke DestroyWindow, hWnd
      .endif
   .ELSE
      invoke DefWindowProc,hWnd,uMsg,wParam,lParam
      ret
   .ENDIF
   xor    eax,eax
   ret
WndProc endp
 end start
```

بررسی کد فوق :

```
        Mov  ofn.lStructSize,SIZEOF ofn
        push hWnd
        pop   ofn.hwndOwner
        push hInstance
        pop   ofn.hInstance
```

در اینجا اعضای ساختار ofn مقدار دهی شده‌اند. در ادامه :

```
mov  ofn.lpstrFilter, OFFSET FilterString
```

FilterString نام فیلتری است که به صورت زیر تعریف می‌شود:

```
FilterString  db "All Files",0,"*.*",0
              db "Text Files",0,"*.txt",0,0
```

در ادامه بافر و اندازه مورد نیاز آن ، جهت قرار گیری نام و مسیر فایل تعریف می‌شود :

```
        mov  ofn.lpstrFile, OFFSET buffer
        mov  ofn.nMaxFile,MAXSIZE
```

سپس ویژگی‌های دیالوگ باکس تعریف می‌شوند (نامهای انتخابی اکثر آنها بسیار واضح است. برای توضیحات بیشتر می‌توان به MSDN مراجعه کرد) :

```
mov  ofn.Flags, OFN_FILEMUSTEXIST or \
                OFN_PATHMUSTEXIST or OFN_LONGNAMES or\
                OFN_EXPLORER or OFN_HIDEREADONLY
```

در ادامه : مشخص کردن عنوان دیالوگ باکس:

```
mov  ofn.lpstrTitle, OFFSET OurTitle
```

سپس فراخوانی تابع نمایشی:

```
Invoke  GetOpenFileName, ADDR ofn
```

که اشاره‌گری به ساختار تعریف شده را بعنوان آرگومان ورودی می‌گیرد.در این زمان دیالوگ باکس نمایش داده می‌شود.تابع تا زمانیکه کاربر فایلی را انتخاب نکند و یا روی دکمه‌ی cancel کلیک ننماید ، چیزی را بر نمی‌گرداند.این تابع در صورتی که کاربر فایلی را انتخاب نمایید ، TRUE را بعنوان خروجی در eax قرار می‌دهد.در غیراینصورت خروجی FALSE خواهد بود. در ادامه :

```
.if eax==TRUE
   invoke lstrcat,offset OutputString,OFFSET FullPathName
   invoke lstrcat,offset OutputString,ofn.lpstrFile
   invoke lstrcat,offset OutputString,offset CrLf
   invoke lstrcat,offset OutputString,offset FullName
```

پس از اینکه کاربر فایلی را انتخاب کرد ، رشته خروجی را برای نمایش در message box آماده می‌کنیم. ابتدا قطعه‌ای از حافظه را برای OutputString آماده کرده و سپس از تابع lstrcat برای اتصال رشته ها به هم استفاده می‌کنیم.

برای قرار دادن خطوط در چندین خط مختلف می توان از **CrLf** (carriage return-line feed) استفاده کرد. سپس:

```
mov   eax,ofn.lpstrFile
push  ebx
xor   ebx,ebx
mov   bx,ofn.nFileOffset
add   eax,ebx
pop   ebx
invoke  lstrcat,offset OutputString,eax
```

خطوط فوق نیاز به توضیحات بیشتری دارند. nFileOffset حاوی اندیسی در ofn.lpstrFile است. اندازه اولی WORD و اندازه دومی DWORD است و امکان افزودن آنها به یکدیگر موجود نیست.بنابراین nFileOffset را باید در low word مربوط به ebx قرار داد و سپس آنرا به مقدار lpstrFile افزود.در ادامه :

```
invoke MessageBox,hWnd,OFFSET OutputString,ADDR AppName,MB_OK
```

پیغام را توسط message box نمایش خواهیم داد. سپس :

```
invoke RtlZeroMemory,offset OutputString,OUTPUTSIZE
```

قبل از مقدار دهی OutputString با رشته ای دیگر، باید محتوای آنرا پاک کرد. به همین جهت از تابع RtlZeroMemory استفاده گردیده است.

مقایسه کد زبان سی و کد اسمبلی یک حلقه

نمایش حلقه در زبان سی	پیاده سازی حلقه در زبان اسمبلی

c = 0;
while (c < 1000)
{
array[c] = c;
c++;
}
	mov ecx, DWORD PTR [array]
	xor eax, eax
	LoopStart:
	mov DWORD PTR [ecx+eax*4], eax
	add eax, 1
	cmp eax, 1000
	jl LoopStart

پیاده سازی یک حلقه در زبان اسمبلی همانند استفاده از دستورات شرطی می‌باشد ، با این تفاوت که مرتبا قطعه کدی را تا زمان صادق بودن شرط آن، اجرا می کند.

حلقه به همراه یک break در آن :

نمایش حلقه در زبان سی	پیاده سازی حلقه در زبان اسمبلی

do
{
if (array[c])
break;
 	
array[c] = c;
c++;
} while (c < 1000);
	mov eax, DWORD PTR [c]
	mov ecx, DWORD PTR [array]
	LoopStart:
	cmp DWORD PTR [ecx+eax*4], 0
	jne AfterLoop
	mov DWORD PTR [ecx+eax*4], eax
	add eax, 1
	cmp eax, 1000
	jl LoopStart
	AfterLoop:

مثال ۱۰– مدیریت حافظه و کار با فایلها

در این قسمت طرز کار ابتدایی با فایلها و مدیریت حافظه را خواهیم آموخت. همچنین از common dialog boxes نیز استفاده خواهد شد.

یادآوری :

از دید یک برنامه ، مدیریت حافظه در ویندوز ساده و واضح می‌باشد. هر پروسه در ویندوز ، فضای آدرس دهی حافظه‌ای معادل ۴ گیگابایت را دارا است. مدل حافظه در این حالت ، به مدل حافظه‌ی تخت معروف است. در این حالت، تمام رجیسترهای سگمنت (یا انتخاب‌گرها) ، به یک آدرس آغازین اشاره می‌کنند و آفست ها ۳۲ بیت هستند. بنابراین هر برنامه در فضای آدرس دهی حافظه خود ، به هر آدرسی بدون نیاز به تغییر مقدار انتخابگرها می‌تواند دسترسی پیدا کند. این مدل ، مدیریت حافظه را بسیار ساده می‌کند.

در اینجا دیگر اشاره‌گرهای "near" or "far" وجود ندارند. تحت Win16 دو نوع عمده از توابع مدیریت حافظه وجود دارند: Global and Local .نوع های Global با حافظه‌های تخصیص داده شده در سایر سگمنت ها سروکار داشتند بنابراین از نوع far بودند. نوع های Local با local heap پردازشگر سروکار داشتند بنابراین از نوع near بودند. تحت Win32 این دو نوع یکی هستند. بنابراین فراخوانی GlobalAlloc or LocalAlloc یک نتیجه را بر می‌گردانند.

مراحل تخصیص و بکارگیری حافظه به شرح زیر هستند:

- تخصیص قطعه‌ای از حافظه با فراخوانی تابع GlobalAlloc . این تابع دستگیره‌ای را به قطعه حافظه تخصیص داده شده بر می‌گرداند.
- قفل کردن این قطعه تخصیص داده شود با فراخوانی تابع GlobalLock . این تابع دستگیره ای به قطعه حافظه مورد نظر را دریافت و سپس اشاره گری به آنرا بر می‌گرداند.
- رهاکردن قفل قطعه حافظه با فراخوانی تابع GlobalUnlock . این تابع اشاره‌گر ایجاد شده به قطعه حافظه را غیرمعتبر می‌کند.
- آزاد سازی حافظه تخصیص داده شده توسط فراخوانی تابع GlobalFree . این تابع دستگیره ای را به قطعه حافظه مورد نظر ، دریافت می‌کند.

همانطور که ذکر گردید نگارش local این توابع نیز اعمال یکسانی را تحت ویندوزهای ۳۲ بیتی انجام می دهند.

همچنین روش فوق با بکارگیری پرچم GMEM_FIXED در تابع GlobalAlloc ، ساده تر می گردد. در این حالت خروجی تابع اشاره‌گری به قطعه حافظه تخصیص داده شده خواهد. در این حالت دیگر نیازی به فراخوانی دو تابع قفل کردن قطعه حافظه و رها کردن قفل مربوطه نیست.

البته در این برنامه از روش سنتی اینکار بدون بکارگیری پرچم فوق جهت آشنایی بیشتر ، استفاده خواهیم کرد.

استفاده از فایلها تحت Win32 شبیه داس می باشد. مراحل مورد نیاز یکی هستند و تنها باید وقفه ها را به توابع API تبدیل کرد. این مراحل به شرح زیر هستند:

۱. گشودن و یا ایجاد فایل توسط فراخوانی تابع CreateFile . این تابع چندین منظوره می باشد. از آن برای گشودن فایلها ، پورتها و غیره می‌توان استفاده کرد. پس از اجرای موفقیت آمیز ، دستگیره ای به فایل و یا وسیله مورد نظر برمی‌گرداند. از این دستگیره برای انجام عملیات روی فایل و یا وسیله مربوطه کمک می‌گیریم. برای انتقال اشاره گر به فایل می توان از تابع SetFilePointer استفاده کرد.
۲. انجام عملیات خواندن و یا نوشتن با فراخوانی توابع ReadFile or WriteFile .این توابع قطعه ای از حافظه را به فایل و یا از فایل انتقال می‌دهند . بنابراین باید قطعه تخصیص داده شده از حافظه ، به اندازه کافی برای نگهداری داده‌ها بزرگ باشد.
۳. بستن فایل با فراخوانی CloseHandle .

کد برنامه:

در این برنامه قصد گشودن یک فایل متنی و سپس نمایش محتویات آن در یک ادیت باکس به همراه قابلیت ویرایش و ذخیره سازی متن تغییر کرده در آن را داریم.

```
.386
.model flat,stdcall
option casemap:none
WinMain proto :DWORD,:DWORD,:DWORD,:DWORD
include \masm32\include\windows.inc
include \masm32\include\user32.inc
include \masm32\include\kernel32.inc
include \masm32\include\comdlg32.inc
includelib \masm32\lib\user32.lib
includelib \masm32\lib\kernel32.lib
includelib \masm32\lib\comdlg32.lib

.const
IDM_OPEN equ 1
IDM_SAVE equ 2
```

```
IDM_EXIT equ 3
MAXSIZE equ 260
MEMSIZE equ 65535

EditID equ 1                    ; ID of the edit control

.data
ClassName db "Win32ASMEditClass",0
AppName  db "Win32 ASM Edit",0
EditClass db "edit",0
MenuName db "FirstMenu",0
ofn   OPENFILENAME <>
FilterString db "All Files",0,"*.*",0
          db "Text Files",0,"*.txt",0,0
buffer db MAXSIZE dup(0)

.data?
hInstance HINSTANCE ?
CommandLine LPSTR ?
hwndEdit HWND ?                 ; Handle to the edit control
hFile HANDLE ?                  ; File handle
hMemory HANDLE ?                ;handle to the allocated memory block
pMemory DWORD ?                 ;pointer to the allocated memory block
SizeReadWrite DWORD ?           ; number of bytes actually read or write

.code
start:
   invoke GetModuleHandle, NULL
   mov    hInstance,eax
   invoke GetCommandLine
   mov CommandLine,eax
   invoke WinMain, hInstance,NULL,CommandLine, SW_SHOWDEFAULT
   invoke ExitProcess,eax

WinMain proc hInst:HINSTANCE,hPrevInst:HINSTANCE,CmdLine:LPSTR,CmdShow:SDWORD
   LOCAL wc:WNDCLASSEX
   LOCAL msg:MSG
   LOCAL hwnd:HWND
   mov   wc.cbSize,SIZEOF WNDCLASSEX
   mov   wc.style, CS_HREDRAW or CS_VREDRAW
   mov   wc.lpfnWndProc, OFFSET WndProc
   mov   wc.cbClsExtra,NULL
   mov   wc.cbWndExtra,NULL
   push  hInst
   pop   wc.hInstance
   mov   wc.hbrBackground,COLOR_WINDOW+1
   mov   wc.lpszMenuName,OFFSET MenuName
   mov   wc.lpszClassName,OFFSET ClassName
   invoke LoadIcon,NULL,IDI_APPLICATION
   mov   wc.hIcon,eax
   mov   wc.hIconSm,eax
   invoke LoadCursor,NULL,IDC_ARROW
   mov   wc.hCursor,eax
   invoke RegisterClassEx, addr wc
   invoke CreateWindowEx,WS_EX_CLIENTEDGE,ADDR ClassName,ADDR AppName,\
         WS_OVERLAPPEDWINDOW,CW_USEDEFAULT,\
         CW_USEDEFAULT,300,200,NULL,NULL,\
         hInst,NULL
   mov   hwnd,eax
   invoke ShowWindow, hwnd,SW_SHOWNORMAL
```

```
    invoke UpdateWindow, hwnd
    .WHILE TRUE
        invoke GetMessage, ADDR msg,NULL,0,0
        .BREAK .IF (!eax)
        invoke TranslateMessage, ADDR msg
        invoke DispatchMessage, ADDR msg
    .ENDW
    mov     eax,msg.wParam
    ret
WinMain endp

WndProc proc uses ebx hWnd:HWND, uMsg:UINT, wParam:WPARAM, lParam:LPARAM
    .IF uMsg==WM_CREATE
        invoke CreateWindowEx,NULL,ADDR EditClass,NULL,\
                WS_VISIBLE or WS_CHILD or ES_LEFT or ES_MULTILINE or\
                ES_AUTOHSCROLL or ES_AUTOVSCROLL,0,\
                0,0,0,hWnd,EditID,\
                hInstance,NULL
        mov hwndEdit,eax
        invoke SetFocus,hwndEdit
;=============================================
;       Initialize the members of OPENFILENAME structure
;=============================================
        mov ofn.lStructSize,SIZEOF ofn
        push hWnd
        pop  ofn.hWndOwner
        push hInstance
        pop  ofn.hInstance
        mov  ofn.lpstrFilter, OFFSET FilterString
        mov  ofn.lpstrFile, OFFSET buffer
        mov  ofn.nMaxFile,MAXSIZE
    .ELSEIF uMsg==WM_SIZE
        mov eax,lParam
        mov edx,eax
        shr edx,16
        and eax,0ffffh
        invoke MoveWindow,hwndEdit,0,0,eax,edx,TRUE
    .ELSEIF uMsg==WM_DESTROY
        invoke PostQuitMessage,NULL
    .ELSEIF uMsg==WM_COMMAND
        mov eax,wParam
        .if lParam==0
            .if ax==IDM_OPEN
                mov  ofn.Flags, OFN_FILEMUSTEXIST or \
                            OFN_PATHMUSTEXIST or OFN_LONGNAMES or\
                            OFN_EXPLORER or OFN_HIDEREADONLY
                invoke GetOpenFileName, ADDR ofn
                .if eax==TRUE
                    invoke CreateFile,ADDR buffer,\
                            GENERIC_READ or GENERIC_WRITE ,\
                            FILE_SHARE_READ or FILE_SHARE_WRITE,\
                            NULL,OPEN_EXISTING,FILE_ATTRIBUTE_ARCHIVE,\
                            NULL
                    mov hFile,eax
                    invoke GlobalAlloc,GMEM_MOVEABLE or GMEM_ZEROINIT,MEMSIZE
                    mov  hMemory,eax
                    invoke GlobalLock,hMemory
                    mov  pMemory,eax
                    invoke ReadFile,hFile,pMemory,MEMSIZE-1,ADDR SizeReadWrite,NULL
                    invoke SendMessage,hwndEdit,WM_SETTEXT,NULL,pMemory
                    invoke CloseHandle,hFile
```

```
            invoke GlobalUnlock,pMemory
            invoke GlobalFree,hMemory
          .endif
          invoke SetFocus,hwndEdit
        .elseif ax==IDM_SAVE
          mov ofn.Flags,OFN_LONGNAMES or\
                OFN_EXPLORER or OFN_HIDEREADONLY
          invoke GetSaveFileName, ADDR ofn
            .if eax==TRUE
              invoke CreateFile,ADDR buffer,\
                          GENERIC_READ or GENERIC_WRITE ,\
                          FILE_SHARE_READ or FILE_SHARE_WRITE,\
                          NULL,CREATE_NEW,FILE_ATTRIBUTE_ARCHIVE,\
                          NULL
              mov hFile,eax
              invoke GlobalAlloc,GMEM_MOVEABLE or GMEM_ZEROINIT,MEMSIZE
              mov  hMemory,eax
              invoke GlobalLock,hMemory
              mov  pMemory,eax
              invoke SendMessage,hwndEdit,WM_GETTEXT,MEMSIZE-1,pMemory
              invoke WriteFile,hFile,pMemory,eax,ADDR SizeReadWrite,NULL
              invoke CloseHandle,hFile
              invoke GlobalUnlock,pMemory
              invoke GlobalFree,hMemory
            .endif
            invoke SetFocus,hwndEdit
          .else
            invoke DestroyWindow, hWnd
          .endif
        .endif
      .ELSE
        invoke DefWindowProc,hWnd,uMsg,wParam,lParam
        ret
.ENDIF
xor    eax,eax
ret
WndProc endp
end start
```

بررسی کد فوق:

```
    invoke CreateWindowEx,NULL,ADDR EditClass,NULL,\
            WS_VISIBLE or WS_CHILD or ES_LEFT or ES_MULTILINE or\
            ES_AUTOHSCROLL or ES_AUTOVSCROLL,0,\
            0,0,0,hWnd,EditID,\
            hInstance,NULL
     mov hwndEdit,eax
```

ما در قسمت WM_CREATE ، یک کنترل تکست باکس را ایجاد خواهیم کرد. لازم به ذکر است که مقادیر x, y, width,height آن با اوی صفر وارد شده اند زیرا در ادامه با تغییر اندازه آن ، کل پنجره را توسط آن خواهیم پوشاند.

همچنین با بکارگیری حالت WS_VISIBLE برای تکست باکس ، دیگر نیازی به فراخوانی تابع ShowWindow جهت نمایش آن نخواهیم داشت. از این نکته برای پنجره والد نیز می‌توان استفاده کرد. در ادامه :

```
;=============================================
;       Initialize the members of OPENFILENAME structure
;=============================================
      mov ofn.lStructSize,SIZEOF ofn
      push hWnd
      pop  ofn.hWndOwner
      push hInstance
      pop  ofn.hInstance
      mov  ofn.lpstrFilter, OFFSET FilterString
      mov  ofn.lpstrFile, OFFSET buffer
      mov  ofn.nMaxFile,MAXSIZE
```

در ادامه اعضای ofn را مقدار دهی اولیه خواهیم کرد.از آنجائیکه از این ساختار در نمایش save as dialog box نیز استفاده خواهیم کرد ، تنها مقادیر مشترک بین توابع GetOpenFileName و GetSaveFileName مقدار دهی شدند.

قسمت WM_CREATE مکان بسیار مناسبی برای مقدار دهی اولیه متغیرها می‌باشد. سپس :

```
.ELSEIF uMsg==WM_SIZE
   mov eax,lParam
   mov edx,eax
   shr edx,16
   and eax,0ffffh
   invoke MoveWindow,hwndEdit,0,0,eax,edx,TRUE
```

هنگامیکه اندازه ناحیه کلاینت پنجره برنامه تغییر کند ، پیغام WM_SIZE صادر خواهد شد.همچنین این پیغام در ابتدای ایجاد پنجره نیز دریافت می‌شود. برای دریافت این پیغام ، کلاس پنجره باید دارای حالتهای CS_VREDRAW and CS_HREDRAW باشد. ما از این پیغام جهت تغییر اندازه ادیت باکس برنامه به اندازه ناحیه کلاینت پنجره کمک می گیریم. در ابتدا باید طول و عرض جاری پنجره را بدست آورد.مقدار high word مربوط به lParam مساوی طول و low word آن معادل عرض پنجره است. سپس از این اطلاعات در تابع MoveWindow جهت تغییر اندازه ادیت باکس کمک می گیریم. در ادامه:

```
.if ax==IDM_OPEN
mov  ofn.Flags, OFN_FILEMUSTEXIST or \
            OFN_PATHMUSTEXIST or OFN_LONGNAMES or\
            OFN_EXPLORER or OFN_HIDEREADONLY
invoke GetOpenFileName, ADDR ofn
```

هنگامیکه کاربر از منوی برنامه گزینه گشودن را انتخاب کند ، دیالوگ باکس OpenFile را نمایش می‌دهیم. سپس :

```
.if eax==TRUE
invoke CreateFile,ADDR buffer,\
        GENERIC_READ or GENERIC_WRITE ,\
        FILE_SHARE_READ or FILE_SHARE_WRITE,\
        NULL,OPEN_EXISTING,FILE_ATTRIBUTE_ARCHIVE,\
        NULL
mov hFile,eax
```

پس از انتخاب فایل ، از تابع CreateFile ، جهت گشودن آن استفاده خواهیم کرد. همچنین در اینجا مشخص می‌کنیم که فایل را جهت خواندن و نوشتن در آن گشوده ایم . پس از گشودن فایل ، دستگیره ای به آن را در یک متغیر عمومی جهت استفاده های بعدی قرار می‌دهیم. تعریف این تابع به صورت زیر است:

```
CreateFile proto lpFileName:DWORD,\
        dwDesiredAccess:DWORD,\
        dwShareMode:DWORD,\
        lpSecurityAttributes:DWORD,\
        dwCreationDistribution:DWORD\,
        dwFlagsAndAttributes:DWORD\,
        hTemplateFile:DWORD
```

dwDesiredAccess : مقدار صفر آن به معنای سطح دسترسی خواندن و نوشتن است . اگر مساوی GENERIC_READ قرار گیرد ، فایل صرفا جهت خواندن ، باز می‌شود و اگر مساوی GENERIC_WRITE قرار گیرد، فایل برای نوشتن در آن ، باز می‌شود.

dwShareMode : دسترسی و یا عدم دسترسی سایر پروسه های فعال را در مورد عملیات بر روی فایل گشوده شده ، مشخص می‌کند. اگر مساوی صفر قرار گیرد این فایل با سایر پروسه ها به اشتراک گذاشته نمی‌شود. اگر مساوی FILE_SHARE_READ قرار گیرد، به سایر پروسه ها اجازه خواندن از فایل گشوده داده خواهد شد و اگر مساوی FILE_SHARE_WRITE قرار گیرد دیگر پروسه‌ها اجازه نوشتن در فایل گشوده شده در برنامه را می یابند.

lpSecurityAttributes : تحت ویندوزهای ۳۲ بیتی اهمیتی ندارد.

dwCreationDistribution:عملیاتی را که تابع در صورت وجود و یا عدم وجود فایل ذکر شده در lpFileName باید انجام دهد ، مشخص می کند. این موارد به شرح زیر هستند:

CREATE_NEW : فایل جدیدی را ایجاد خواهد کرد. اگر فایل از پیش وجود داشته باشد تابع با یک خطا متوقف می شود.

CREATE_ALWAYS : فایل جدیدی را ایجاد خواهد کرد.اگر این فایل از پیش وجود داشته باشد ، جای‌گزینی می گردد.

OPEN_EXISTING : فایل را می گشاید. اگر فایل از پیش وجود نداشته باشد تابع با یک خطا متوقف می شود.

OPEN_ALWAYS : فایل را در صورت وجود می گشاید.اگر فایل موجود نباشد و حالت CREATE_NEW پیشتر مشخص شده باشد ، فایل جدیدی ایجاد خواهد شد.
TRUNCATE_EXISTING : فایل را می گشاید. سپس کل محتویات آنرا تخلیه کرده و اندازه فایل صفر بایت می‌شود.سطح دسترسی در این حالت باید GENERIC_WRITE باشد. اگر فایل موجود نباشد تابع با یک خطا متوقف خواهد شد.

dwFlagsAndAttributes : بیانگر خواص فایل مانند آرشیو ، عادی، فقط خواندنی و غیره می باشد.

در ادامه :

```
invoke  GlobalAlloc,GMEM_MOVEABLE or GMEM_ZEROINIT,MEMSIZE
mov   hMemory,eax
invoke  GlobalLock,hMemory
mov   pMemory,eax
```

هنگامیکه فایلی گشوده شد ، قطعه ای از حافظه را برای استفاد در توابع ReadFile and WriteFile تخصیص می دهیم. با مشخص کردن GMEM_MOVEABLE ، به ویندوز اجازه انتقال قطعه حافظه مورد استفاده جهت یکپارچگی بیشتر حافظه را می‌دهیم. با تنظیم پرچم GMEM_ZEROINIT ، قطعه حافظه تخصیص داده شده در ابتدا با صفر پر می‌شود.
پس از به پایان رسیدن کار تابع GlobalAlloc ، رجیستر eax حاوی دستگیره ای به قطعه حافظه تخصیص داده شده خواهد بود. سپس با فراخوانی GlobalLock ، اشاره گری به قطعه حافظه بدست خواهد آمد.سپس:

```
invoke ReadFile,hFile,pMemory,MEMSIZE-1,ADDR SizeReadWrite,NULL
invoke SendMessage,hwndEdit,WM_SETTEXT,NULL,pMemory
```

هنگامیکه قطعه حافظه آماده شد ، از تابع ReadFile جهت خواندن اطلاعات از فایل استفاده می کنیم. زمانیکه فایل گشوده می‌شود ، اشاره‌گر آن در آفست صفر قرار دارد. بنابراین آماده خواندن اولین بایت فایل به بعد هستیم. اولین پارامتر تابع ReadFile ، دستگیره ای به فایل مورد نظر جهت خواندن است. دومین پارامتر اشاره گری است به قطعه ای از حافظه که داده ها را در خود ذخیره خواهد کرد. پارامتر بعدی تعداد بایتهایی است که باید از فایل خوانده شوند.چهارمین آرگومان ، آدرس متغیری است که تعداد بایتهای واقعی خوانده شده از فایل درون آن قرار می گیرد.
پس از پرکردن قطعه حافظه از داده‌ها ، توسط ارسال پیغام WM_SETTEXT ، ادیت باکس را مقدار دهی خواهیم کرد. دراینجا lParam حاوی اشاره گری است به قطعه حافظه مورد استفاده. پس از این ، ادیت باکس داده های خوانده شده را نمایش می‌دهد.

```
invoke CloseHandle,hFile
invoke GlobalUnlock,pMemory
```

```
        invoke GlobalFree,hMemory
    .endif
```

با توجه به اینکه قصد ذخیره سازی داده های ویرایش شده در ادیت باکس را در فایلی دیگر داریم ، بنابراین فایل مورد نظر را پس از خواندن ، خواهیم بست و منابع تخصیص یافته را نیز آزاد می کنیم. البته می‌توان حافظه اختصاص یافته را در اینجا آزاد نکرد و از آن در قسمت ذخیره سازی داده‌ها مجددا استفاده کرد. در ادامه:

```
invoke SetFocus,hwndEdit
```

پس از نمایش پنجره ، تمرکز ورودی به ادیت باکس منتقل خواهد شد.

هنگامیکه ذخیره کردن فایل ، پس از انتخاب گزینه مربوطه از منو، دیالوگ باکس "ذخیره کردن بعنوان..." نمایش داده خواهد شد:

```
mov ofn.Flags,OFN_LONGNAMES or\
                OFN_EXPLORER or OFN_HIDEREADONLY
```

از آنجائیکه قصد ایجاد فایلی جدید را داریم ، پرچم‌های OFN_FILEMUSTEXIST and OFN_PATHMUSTEXIST را بکار نخواهیم برد (زیرا اجازه ایجاد فایلی که وجود ندارد را نخواهد داد) و همچنین پارامتر dwCreationDistribution تابع CreateFile باید به CREATE_NEW تنظیم شود. باقیمانده کد تفاوتی با آنچه که گذشت ندارد. تنها تفاوت در قسمت زیر است:

```
invoke SendMessage,hwndEdit,WM_GETTEXT,MEMSIZE-1,pMemory
invoke WriteFile,hFile,pMemory,eax,ADDR SizeReadWrite,NULL
```

از پیغام WM_GETTEXT جهت دریافت اطلاعات از ادیت باکس و انتقال آن به حافظه استفاده شده است. خروجی تابع که در eax قرار گرفته ، اندازه داده ها درون بافر است. پس از قرار گرفتن داده‌ها درحافظه ، آنها را در فایل خواهیم نوشت.

مثال ۱۱- Memory Mapped Files

اگر مثال برنامه قبل را بررسی کرده باشید احتمالا نقایصی را در آن یافته اید. اگر حجم فایل مورد نظر برای خواندن، از اندازه قطعه حافظه تخصیص یافته شده برای آن بیشتر باشد، و یا اگر رشته ای را که بدنبال آن هستید ، نیمه اول آن در انتهای قطعه حافظه قرار گرفته باشد ، چکار باید کرد ؟

پاسخ سنتی به سوال اول این است که باید به صورت مداوم تا انتهای فایل ، داده ها را یکی پس از دیگری خواند و مورد استفاده قرار داد. پاسخ سوال دوم این است که باید برای حالت خاص انتهای فایل، موارد لازم را در نظر داشت.

شاید بتوان قطعه بسیار بزرگی از حافظه را برای ذخیره کردن محتویات کل یک فایل تخصیص داد ، اما به یک چنین برنامه‌هایی، resource hog گفته می‌شود. File mapping (نگاشت فایل) راه حلی برای این موضوع است. با کمک File mapping می‌توان تصور کرد که کل فایل هم اکنون بدرون حافظه بارگذاری شده است و با استفاده از یک اشاره گر حافظه می‌توان اطلاعاتی را از آن خواند و یا در آن ذخیره کرد. دیگر نیازی به استفاده از توابع مدیریت حافظه API ویندوز نخواهد بود. همچنین از File mapping برای به اشتراک گذاشتن داده‌ها بین پروسه های مختلف نیز می‌توان استفاده کرد. با کمک File mapping ، فایل واقعی دیگر درگیر نخواهد بود و این حالت شبیه به قطعه حافظه رزرو شده‌ای است که تمام پروسه‌ها می‌توانند آنرا ببینند. البته در این حالت برای به اشتراک گذاشتن داده ها، باید به نکات مربوط به تردها و همزمانی بین آنها دقت داشت ، درغیر اینصورت برنامه در مدت کوتاهی از کار خواهد افتاد.

در این مثال در مورد ایجاد shared memory region (ناحیه حافظه به اشتراک گذاشته شده) توسط File mapping ، بحث نخواهیم کرد. بلکه هدف ما ، آشنایی با نحوه نگاشت فایل در حافظه است. برای مثال یک PE loader ، از روش نگاشت فایل ، برای بارگذاری فایلهای اجرایی به درون حافظه استفاده می‌کند. این روش بسیار مناسب است زیرا تنها قسمت مورد نیاز از فایل خوانده خواهد شد.

روش نگاشت فایل‌ها محدودیت‌های خود را نیز دارا است. هنگامیکه یک فایل به درون حافظه نگاشت می‌گردد ، اندازه آن در حین عملیات نباید تغییر کند. بنابراین روش نگاشت فایلها برای کار با فایلهای فقط خواندنی و یا عملیاتی که بر روی حجم فایل تاثیری نمی‌گذارند، بسیار ایده‌آل است.

البته در حالتهایی که اندازه فایل تغییر می کنند نیز می‌توان از این روش استفاده نمود و بدیهی است که در این حالت‌ها باید اندازه جدید را محاسبه نمود و فایل نگاشت شده بدرون حافظه را بر اساس اندازه جدید ایجاد کرد.

برای استفاده از روش نگاشت فایل‌ها بدرون حافظه ، مراحل زیر باید صورت گیرند:

- با فراخوانی تابع **CreateFile** فایل مورد نظر برای نگاشت ، گشوده می‌شود.
- فراخوانی **CreateFileMapping** با استفاده از دستگیره بدست آمده از مرحله قبل. این تابع شیء نگاشت فایل را از فایل گشوده شده در قسمت قبل ، ایجاد می‌نماید.

- فراخوانی **MapViewOfFile** جهت نگاشت کل و یا قسمتی انتخابی از فایل بدورن حافظه. این تابع اشاره گری را به ابتدای ناحیه نگاشت شده باز می گرداند.
- استفاده از اشاره گر فوق جهت خواندن و یا نوشتن داده ها در فایل
- فراخوانی **UnmapViewOfFile** برای پایان دادن به نگاشت فایل
- با کمک تابع **CloseHandle** ، فایل نگاشت شده بسته می‌شود.
- فراخوانی **CloseHandle** جهت بستن فایل اصلی باز شده توسط تابع CreateFile .

کد برنامه:

```
.386
.model flat,stdcall
WinMain proto :DWORD,:DWORD,:DWORD,:DWORD
include \masm32\include\windows.inc
include \masm32\include\user32.inc
include \masm32\include\kernel32.inc
include \masm32\include\comdlg32.inc
includelib \masm32\lib\user32.lib
includelib \masm32\lib\kernel32.lib
includelib \masm32\lib\comdlg32.lib

.const
IDM_OPEN equ 1
IDM_SAVE equ 2
IDM_EXIT equ 3
MAXSIZE equ 260

.data
ClassName db "Win32ASMFileMappingClass",0
AppName  db "Win32 ASM File Mapping Example",0
MenuName db "FirstMenu",0
ofn   OPENFILENAME <>
FilterString db "All Files",0,"*.*",0
          db "Text Files",0,"*.txt",0,0
buffer db MAXSIZE dup(0)
hMapFile HANDLE 0                    ; Handle to the memory mapped file, must be
                                     ;initialized with 0 because we also use it as
                                     ;a flag in WM_DESTROY section too

.data?
hInstance HINSTANCE ?
CommandLine LPSTR ?
hFileRead HANDLE ?                   ; Handle to the source file
hFileWrite HANDLE ?                  ; Handle to the output file
hMenu HANDLE ?
pMemory DWORD ?                      ; pointer to the data in the source file
SizeWritten DWORD ?                  ; number of bytes actually written by WriteFile

.code
start:
      invoke GetModuleHandle, NULL
```

```
    mov    hInstance,eax
    invoke GetCommandLine
    mov CommandLine,eax
    invoke WinMain, hInstance,NULL,CommandLine, SW_SHOWDEFAULT
    invoke ExitProcess,eax

WinMain proc hInst:HINSTANCE,hPrevInst:HINSTANCE,CmdLine:LPSTR,CmdShow:DWORD
  LOCAL wc:WNDCLASSEX
  LOCAL msg:MSG
  LOCAL hwnd:HWND
  mov   wc.cbSize,SIZEOF WNDCLASSEX
  mov   wc.style, CS_HREDRAW or CS_VREDRAW
  mov   wc.lpfnWndProc, OFFSET WndProc
  mov   wc.cbClsExtra,NULL
  mov   wc.cbWndExtra,NULL
  push  hInst
  pop   wc.hInstance
  mov   wc.hbrBackground,COLOR_WINDOW+1
  mov   wc.lpszMenuName,OFFSET MenuName
  mov   wc.lpszClassName,OFFSET ClassName
  invoke LoadIcon,NULL,IDI_APPLICATION
  mov   wc.hIcon,eax
  mov   wc.hIconSm,eax
  invoke LoadCursor,NULL,IDC_ARROW
  mov   wc.hCursor,eax
  invoke RegisterClassEx, addr wc
  invoke CreateWindowEx,WS_EX_CLIENTEDGE,ADDR ClassName,\
           ADDR AppName, WS_OVERLAPPEDWINDOW,CW_USEDEFAULT,\
          CW_USEDEFAULT,300,200,NULL,NULL,\
  hInst,NULL
  mov   hwnd,eax
  invoke ShowWindow, hwnd,SW_SHOWNORMAL
  invoke UpdateWindow, hwnd
  .WHILE TRUE
    invoke GetMessage, ADDR msg,NULL,0,0
    .BREAK .IF (!eax)
    invoke TranslateMessage, ADDR msg
    invoke DispatchMessage, ADDR msg
  .ENDW
  mov    eax,msg.wParam
  ret
WinMain endp

WndProc proc hWnd:HWND, uMsg:UINT, wParam:WPARAM, lParam:LPARAM
  .IF uMsg==WM_CREATE
    invoke GetMenu,hWnd                  ;Obtain the menu handle
    mov  hMenu,eax
    mov ofn.lStructSize,SIZEOF ofn
    push hWnd
    pop  ofn.hWndOwner
    push hInstance
    pop  ofn.hInstance
    mov  ofn.lpstrFilter, OFFSET FilterString
    mov  ofn.lpstrFile, OFFSET buffer
    mov  ofn.nMaxFile,MAXSIZE
  .ELSEIF uMsg==WM_DESTROY
    .if hMapFile!=0
      call CloseMapFile
    .endif
    invoke PostQuitMessage,NULL
```

```
   .ELSEIF uMsg==WM_COMMAND
      mov eax,wParam
      .if lParam==0
         .if ax==IDM_OPEN
            mov  ofn.Flags, OFN_FILEMUSTEXIST or \
                        OFN_PATHMUSTEXIST or OFN_LONGNAMES or\
                        OFN_EXPLORER or OFN_HIDEREADONLY
                        invoke GetOpenFileName, ADDR ofn
            .if eax==TRUE
               invoke CreateFile,ADDR buffer,\
                                 GENERIC_READ ,\
                                 0,\
                                 NULL,OPEN_EXISTING,FILE_ATTRIBUTE_ARCHIVE,\
                                 NULL
               mov hFileRead,eax
               invoke CreateFileMapping,hFileRead,NULL,PAGE_READONLY,0,0,NULL
               mov     hMapFile,eax
               mov     eax,OFFSET buffer
               movzx  edx,ofn.nFileOffset
               add     eax,edx
               invoke SetWindowText,hWnd,eax
               invoke EnableMenuItem,hMenu,IDM_OPEN,MF_GRAYED
               invoke EnableMenuItem,hMenu,IDM_SAVE,MF_ENABLED
            .endif
         .elseif ax==IDM_SAVE
            mov ofn.Flags,OFN_LONGNAMES or\
                        OFN_EXPLORER or OFN_HIDEREADONLY
            invoke GetSaveFileName, ADDR ofn
            .if eax==TRUE
               invoke CreateFile,ADDR buffer,\
                                 GENERIC_READ or GENERIC_WRITE ,\
                                 FILE_SHARE_READ or FILE_SHARE_WRITE,\
                                 NULL,CREATE_NEW,FILE_ATTRIBUTE_ARCHIVE,\
                                 NULL
               mov hFileWrite,eax
               invoke MapViewOfFile,hMapFile,FILE_MAP_READ,0,0,0
               mov pMemory,eax
               invoke GetFileSize,hFileRead,NULL
               invoke WriteFile,hFileWrite,pMemory,eax,ADDR SizeWritten,NULL
               invoke UnmapViewOfFile,pMemory
               call   CloseMapFile
               invoke CloseHandle,hFileWrite
               invoke SetWindowText,hWnd,ADDR AppName
               invoke EnableMenuItem,hMenu,IDM_OPEN,MF_ENABLED
               invoke EnableMenuItem,hMenu,IDM_SAVE,MF_GRAYED
            .endif
         .else
            invoke DestroyWindow, hWnd
         .endif
      .endif
   .ELSE
      invoke DefWindowProc,hWnd,uMsg,wParam,lParam
      ret
   .ENDIF
   xor    eax,eax
   ret
WndProc endp

CloseMapFile PROC
      invoke CloseHandle,hMapFile
      mov    hMapFile,0
```

```
    invoke CloseHandle,hFileRead
    ret
CloseMapFile endp

end start
```

بررسی کد فوق:

```
invoke CreateFile,ADDR buffer,\
                        GENERIC_READ ,\
                        0,\
                        NULL,OPEN_EXISTING,FILE_ATTRIBUTE_ARCHIVE,\
                        NULL
```

هنگامیکه کاربر فایلی را توسط دیالوگ باکس گشودن فایل انتخاب می کند ، سعی در بازکردن آن خواهیم کرد. از پرچم GENERIC_READ ، جهت گشودن فایل به صورت فقط خواندنی استفاده شده و با قرار دادن dwShareMode مساوی صفر ، به سایر پروسه ها اجازه‌ی تغییر فایل در حین انجام عملیات ، داده نمی‌شود.در ادامه:

```
invoke CreateFileMapping,hFileRead,NULL,PAGE_READONLY,0,0,NULL
```

سپس از فایل گشوده شده، توسط تابع CreateFileMapping ، فایلی نگاشت شده به حافظه، ایجاد می کنیم. تعریف این تابع به صورت زیر است:

```
CreateFileMapping proto hFile:DWORD,\
                        lpFileMappingAttributes:DWORD,\
                        flProtect:DWORD,\
                        dwMaximumSizeHigh:DWORD,\
                        dwMaximumSizeLow:DWORD,\
                        lpName:DWORD
```

لازم به ذکر است که تابع فوق کل فایل را بدرون حافظه نگاشت نمی کند.توسط پارامترهای dwMaximumSizeHigh and dwMaximumSizeLow ، قسمت مورد نیاز انتخاب و بدرون حافظه نگاشت می گردد.

اگر اندازه ای بزرگتر از اندازه فایل واقعی را انتخاب نمایید ، فایل واقعی به اندازه ذکر شده بسط داده خواهد شد. با قرار دادن صفر بعنوان هر دو پارامتر ذکر شده ، اندازه فایل واقعی و فایل نگاشت شده یکی خواهد گردید.

اگر پارامتر lpFileMappingAttributes مساوی NULL قرار گیرد ، فایل نگاشت شده با خواص امنیتی پیش فرض ویندوز ایجاد می‌شود.

پارامتر flProtect نوع محافظت مورد نیاز برای فایل مپ شده را تعیین می کند. برای مثال مقدار PAGE_READONLY سبب می‌شود تا فایل نگاشت شده به صورت فقط خواندنی قابل استفاده شود. لازم به ذکر است که این خاصیت با خاصیت ذکر شده در تابع CreateFile نباید متناقض باشد . در غیراینصورت فراخوانی تابع نگاشت فایل با شکست مواجه خواهد شد.

پارامتر lpName به نام فایل نگاشت شده در حافظه اشاره می کند. اگر می خواهید این فایل را با سایر پروسه ها به اشتراک بگذارید باید نامی برای آن مشخص نمایید. اما در مثال ما برنامه ایجاد شده تنها برنامه ای است که از فایل استفاده خواهد کرد بنابراین از این پارامتر صرفنظر خواهیم کرد. در ادامه:

```
mov     eax,OFFSET buffer
movzx  edx,ofn.nFileOffset
add     eax,edx
invoke SetWindowText,hWnd,eax
```

پس از انجام موفقیت آمیز مراحل قبل ، عنوان پنجره را به نام فایل باز شده تغییر می‌دهیم. نام و مسیر کامل فایل در بافر ذخیره خواهند شد. از آنجائیکه تنها نام فایل را در عنوان پنجره می‌خواهیم نمایش دهیم ، بنابراین باید مقدار nFileOffset ساختار OPENFILENAME را به آدرس بافر اضافه نماییم. سپس :

```
invoke EnableMenuItem,hMenu,IDM_OPEN,MF_GRAYED
invoke EnableMenuItem,hMenu,IDM_SAVE,MF_ENABLED
```

بعنوان اقدامی احتیاطی ، پس از گشودن فایل اولیه ، نمی خواهیم کاربر چندین فایل را با هم باز کند بنابراین منوی گشودن فایل را با کمک EnableMenuItem غیرفعال و منوی ذخیره کردن فایل را فعال می کنیم.و اگر کاربر قصد بستن برنامه را داشت به صورت زیر منابع تخصیص یافته را باید آزاد نمائیم:

```
.ELSEIF uMsg==WM_DESTROY
    .if hMapFile!=0
        call CloseMapFile
    .endif
    invoke PostQuitMessage,NULL
```

تابع CloseMapFile در دستورات فوق به صورت زیر تعریف شده است:

```
CloseMapFile PROC
    invoke CloseHandle,hMapFile
    mov    hMapFile,0
    invoke CloseHandle,hFileRead
    ret
CloseMapFile endp
```

اگر کاربر گزینه ذخیره سازی داده‌ها را از منو انتخاب کند ، پس از نمایش دیالوگ باکس استاندارد "ذخیره سازی بعنوان ...." ، از کاربر نام فایل جدیدی را برای ذخیره سازی داده ها دریافت می‌کنیم و در ادامه برای ذخیره سازی:

```
invoke MapViewOfFile,hMapFile,FILE_MAP_READ,0,0,0
mov pMemory,eax
```

بالافاصله پس از ایجاد فایل خروجی ، به کمک تابع MapViewOfFile ، قسمت دلخواهی از فایل نگاشت شده به حافظه را به حافظه نگاشت می‌کنیم. این تابع به صورت زیر تعریف می‌شود:

```
MapViewOfFile proto hFileMappingObject:DWORD,\
                    dwDesiredAccess:DWORD,\
                    dwFileOffsetHigh:DWORD,\
                    dwFileOffsetLow:DWORD,\
                    dwNumberOfBytesToMap:DWORD
```

توضیحات پارامترهای آن به صورت زیر است:

dwDesiredAccess : عملیاتی را که می خواهیم روی فایل انجام دهیم مشخص می کند. برای حالت فقط خواندنی از FILE_MAP_READ استفاده می‌گردد.

dwFileOffsetHigh and **dwFileOffsetLow** : آفست آغازین قسمتی از فایل را که می‌خواهیم بدرون حافظه نگاشت کنیم ، مشخص می‌کنند. در این مثال چون کل فایل را می خواهیم مورد استفاده قرار دهیم ، آنها را مساوی صفر قرار داده‌ایم .

dwNumberOfBytesToMap : تعداد بایتهایی را که باید بدرون حافظه نگاشت کرد مشخص می‌کنند. با مساوی صفر قرار دادن این آرگومان ، کل فایل خوانده می‌شود.

پس از فراخوانی MapViewOfFile ، قسمت دلخواه بدرون حافظه بارگذاری می‌شود.حاصل این فراخوانی اشاره‌گری به قطعه حافظه برای کار با آن خواهد بود. سپس :

```
invoke GetFileSize,hFileRead,NULL
```

اندازه فایل به این ترتیب بدست آمده و خروجی تابع در eax قرار می‌گیرد. اگر اندازه فایل از ۴ گیگابایت بیشتر باشد، DWORD high اندازه فایل در FileSizeHighWord ذخیره می‌گردد. از آنجائیکه قصد کار با اینگونه فایلهای حجیم را نداریم می توان از آن صرفنظر کرد. در ادامه:

```
invoke WriteFile,hFileWrite,pMemory,eax,ADDR SizeWritten,NULL
```

از آن برای نوشتن داده‌های نگاشت شده در حافظه ، بدرون فایل استفاده شد. سپس منابع تخصیص داده شده آزاد می‌گردند:

```
invoke UnmapViewOfFile,pMemory

call   CloseMapFile
invoke CloseHandle,hFileWrite
```

بازیابی عنوان اصلی پنجره:

```
invoke SetWindowText,hWnd,ADDR AppName
```

سپس گزینه ذخیره کردن منو را غیرفعال و گزینه گشودن فایل را فعال می کنیم.

```
invoke EnableMenuItem,hMenu,IDM_OPEN,MF_ENABLED
invoke EnableMenuItem,hMenu,IDM_SAVE,MF_GRAYED
```

مثال ۱۲- پروسه

پروسه (process) چیست؟

پروسه برنامه ای است در حال اجرا ، حاوی فضای آدرس دهی مجازی اختصاصی، یک ترد و همچنین شامل سایر منابع سیستمی که برای آن قابل مشاهده می باشند.

همانگونه که از تعریف نیز مشخص است ، یک پروسه شامل چندین شیء است : فضای آدرس دهی ، ماژول (های) اجرایی و هر چیزی که یک فایل اجرایی آنرا ایجاد و یا جهت استفاده باز می کند. هر پروسه باید حداقل دارای یک ترد باشد. ترد (thread) چیست؟ ترد در حقیقت صف اجرایی است. هنگامیکه ویندوز پروسه ای را ایجاد می کند ، به ازای هر پروسه تنها یک ترد را خلق خواهد کرد. این ترد از اجرای اولین دستورالعمل (اینستراکشن) ماژول شروع به کار خواهد کرد. اگر یک پروسه نیاز به تردهای بیشتری داشته باشد ، آنها را می تواند ایجاد نماید.

زمانیکه ویندوز پیغامی را مبتنی بر ایجاد یک ترد دریافت می کند ، فضای آدرس دهی اختصاصی برای این ترد ایجاد کرده و سپس فایل اجرایی را به این فضای آدرسی دهی نگاشت می‌نماید.سپس تردی اولیه برای این پروسه ایجاد می گردد.

تحت ویندوزهای ۳۲ بیتی ، می‌توان با فراخوانی تابع CreateProcess ، از درون برنامه ها ، پروسه ای جدید را ایجاد نمود. تعریف این تابع به صورت زیر است:

```
CreateProcess proto lpApplicationName:DWORD,\
                    lpCommandLine:DWORD,\
                    lpProcessAttributes:DWORD,\
                    lpThreadAttributes:DWORD,\
                    bInheritHandles:DWORD,\
                    dwCreationFlags:DWORD,\
                    lpEnvironment:DWORD,\
                    lpCurrentDirectory:DWORD,\
                    lpStartupInfo:DWORD,\
                    lpProcessInformation:DWORD
```

lpApplicationName : نام فایل اجرایی مورد نظر برای اجرا است (با و یا بدون مسیر فایل). اگر این پارامتر را مساوی NULL قرار دهیم، باید نام فایل اجرایی را در پارامتر lpCommandLine مشخص نمود.

lpCommandLine : آرگومان خط فرمانی که برنامه دریافت و پردازش می کند. برای مثال :
notepad.exe readme.txt

lpProcessAttributes and lpthreadAttributes : خواص امنیتی پروسه و ترد اولیه را مشخص می کند. اگر این دو پارامتر NULL وارد شوند ، خواص امنیتی پیش فرض ویندوز مورد استفاده قرار می گیرند.

bInheritHandles : پرچمی است درجهت تعیین اینکه آیا پروسه جدید تمام دستگیره‌های باز شده توسط پروسه فعلی را به ارث ببرد یا خیر.

dwCreationFlags : پرچم‌هایی که رفتار پروسه ایجاد شده را مشخص می‌کنند. برای مثال بلافاصله پس از ایجاد پروسه، معلق گردیده ، اصلاح شده و سپس مورد استفاده قرار گیرد. همچنین حق تقدم تردهای آنرا نیز می‌توان تعیین کرد و به صورت معمول NORMAL_PRIORITY_CLASS مورد استفاده قرار می‌گیرد.

lpEnvironment : اشاره گری است به قطعه محیطی. اگر مساوی NULL قرار گیرد ، این خواص از برنامه والد به ارث خواهند رسید.

lpCurrentDirectory : اشاره‌گری است به دایرکتوری جاری پروسه ای که ایجاد خواهد شد (child process). اگر می‌خواهید که این مورد را نیز از برنامه والد به ارث ببرد آنرا مساوی NULL قرار دهید.

lpStartupInfo : اشاره‌گری است به ساختار STARTUPINFO که بیانگر نحوه نمایش پنجره اصلی پروسه جدید است. اگر حالت ویژه ای مد نظر نیست ، می‌توان این پارامتر را با کمک تابع GetStartupInfo از مقادیر پیش فرض برنامه والد مقدار دهی کرد.

lpProcessInformation : اشاره گری است به ساختار PROCESS_INFORMATION که دریافت کننده ی اطلاعات شناسایی پروسه ایجاد شده است. این ساختار به شکل زیر تعریف می‌شود:

```
PROCESS_INFORMATION STRUCT
   hProcess        HANDLE ?        ; handle to the child process
   hThread         HANDLE ?        ; handle to the primary thread of the child process
   dwProcessId     DWORD ?         ; ID of the child process
   dwThreadId      DWORD ?         ; ID of the primary thread of the child process
PROCESS_INFORMATION ENDS
```

Process handle and process ID : دو مفهوم مجزا هستند. process ID ، یک ID منحصر بفرد از پروسه در سیستم عامل است. process handle ، مقداری است که توسط ویندوز برای استفاده سایر توابع API ویندوز بازگشت داده می‌شود و منحصربفرد نیست.

پس از فراخوانی CreateProcess ، پروسه ای جدید ایجاد شده و خروجی تابع بلافاصله بازگشت داده می‌شود. برای بررسی اینکه آیا پروسه ایجاد شده هنوز فعال است یا خیر می‌توان از تابع GetExitCodeProcess استفاده کرد. قالب این تابع به صورت زیر است:

```
GetExitCodeProcess proto hProcess:DWORD, lpExitCode:DWORD
```

اگر این فراخوانی موفقیت آمیز باشد ، وضعیت پروسه در پارامتر lpExitCode قرار می گیرد. اگر این خروجی مساوی **STILL_ACTIVE** باشد ، بدین معنا است که پروسه ایجاد شده هنوز در حال اجرا است.

با فراخوانی تابع TerminateProcess می‌توان پروسه را اجبار خاتمه داد. تعریف این تابع به شرح زیر است:

```
TerminateProcess proto hProcess:DWORD, uExitCode:DWORD
```

کد خروجی دلخواهی را اینجا می‌توان بکار برد ، اما روش صحیحی برای خاتمه یک پروسه نمی‌باشد ، زیرا dll های ضمیمه شده به پروسه را باخبر نمی‌سازد.

کد برنامه:

هنگامیکه گزینه create process از منو انتخاب می‌شود ، پروسه‌ای جدید خلق شده و برنامه msgbox.exe اجرا می‌شود.پس از ایجاد پروسه، توسط گزینه terminate process منو، می توان برنامه را (در صورتیکه در حال اجرا باشد) خاتمه بخشید.

```
.386
.model flat,stdcall
option casemap:none
WinMain proto :DWORD,:DWORD,:DWORD,:DWORD
include \masm32\include\windows.inc
include \masm32\include\user32.inc
include \masm32\include\kernel32.inc
includelib \masm32\lib\user32.lib
includelib \masm32\lib\kernel32.lib

.const
IDM_CREATE_PROCESS equ 1
IDM_TERMINATE equ 2
IDM_EXIT equ 3

.data
ClassName db "Win32ASMProcessClass",0
AppName  db "Win32 ASM Process Example",0
MenuName db "FirstMenu",0
processInfo PROCESS_INFORMATION <>
programname db "msgbox.exe",0

.data?
hInstance HINSTANCE ?
CommandLine LPSTR ?
hMenu HANDLE ?
ExitCode DWORD ?       ; contains the process exitcode status from GetExitCodeProcess call.
```

```
.code
start:
        invoke GetModuleHandle, NULL
        mov    hInstance,eax
        invoke GetCommandLine
        mov CommandLine,eax
        invoke WinMain, hInstance,NULL,CommandLine, SW_SHOWDEFAULT
        invoke ExitProcess,eax

WinMain proc hInst:HINSTANCE,hPrevInst:HINSTANCE,CmdLine:LPSTR,CmdShow:DWORD
    LOCAL wc:WNDCLASSEX
    LOCAL msg:MSG
    LOCAL hwnd:HWND
    mov   wc.cbSize,SIZEOF WNDCLASSEX
    mov   wc.style, CS_HREDRAW or CS_VREDRAW
    mov   wc.lpfnWndProc, OFFSET WndProc
    mov   wc.cbClsExtra,NULL
    mov   wc.cbWndExtra,NULL
    push  hInst
    pop   wc.hInstance
    mov   wc.hbrBackground,COLOR_WINDOW+1
    mov   wc.lpszMenuName,OFFSET MenuName
    mov   wc.lpszClassName,OFFSET ClassName
    invoke LoadIcon,NULL,IDI_APPLICATION
    mov   wc.hIcon,eax
    mov   wc.hIconSm,eax
    invoke LoadCursor,NULL,IDC_ARROW
    mov   wc.hCursor,eax
    invoke RegisterClassEx, addr wc
    invoke CreateWindowEx,WS_EX_CLIENTEDGE,ADDR ClassName,ADDR AppName,\
          WS_OVERLAPPEDWINDOW,CW_USEDEFAULT,\
          CW_USEDEFAULT,300,200,NULL,NULL,\
          hInst,NULL
    mov   hwnd,eax
    invoke ShowWindow, hwnd,SW_SHOWNORMAL
    invoke UpdateWindow, hwnd
    invoke GetMenu,hwnd
    mov  hMenu,eax
    .WHILE TRUE
              invoke GetMessage, ADDR msg,NULL,0,0
              .BREAK .IF (!eax)
              invoke TranslateMessage, ADDR msg
              invoke DispatchMessage, ADDR msg
    .ENDW
    mov     eax,msg.wParam
    ret
WinMain endp

WndProc proc hWnd:HWND, uMsg:UINT, wParam:WPARAM, lParam:LPARAM
    LOCAL startInfo:STARTUPINFO
    .IF uMsg==WM_DESTROY
       invoke PostQuitMessage,NULL
    .ELSEIF uMsg==WM_INITMENUPOPUP
       invoke GetExitCodeProcess,processInfo.hProcess,ADDR ExitCode
       .if eax==TRUE
          .if ExitCode==STILL_ACTIVE
             invoke EnableMenuItem,hMenu,IDM_CREATE_PROCESS,MF_GRAYED
             invoke EnableMenuItem,hMenu,IDM_TERMINATE,MF_ENABLED
          .else
             invoke EnableMenuItem,hMenu,IDM_CREATE_PROCESS,MF_ENABLED
```

```
            invoke EnableMenuItem,hMenu,IDM_TERMINATE,MF_GRAYED
         .endif
      .else
         invoke EnableMenuItem,hMenu,IDM_CREATE_PROCESS,MF_ENABLED
         invoke EnableMenuItem,hMenu,IDM_TERMINATE,MF_GRAYED
      .endif
   .ELSEIF uMsg==WM_COMMAND
      mov eax,wParam
      .if lParam==0
         .if ax==IDM_CREATE_PROCESS
            .if processInfo.hProcess!=0
               invoke CloseHandle,processInfo.hProcess
               mov processInfo.hProcess,0
            .endif
            invoke GetStartupInfo,ADDR startInfo
            invoke CreateProcess,ADDR programname,NULL,NULL,NULL,FALSE,\
                                 NORMAL_PRIORITY_CLASS,\
                                 NULL,NULL,ADDR startInfo,ADDR processInfo
            invoke CloseHandle,processInfo.hThread
         .elseif ax==IDM_TERMINATE
            invoke GetExitCodeProcess,processInfo.hProcess,ADDR ExitCode
            .if ExitCode==STILL_ACTIVE
               invoke TerminateProcess,processInfo.hProcess,0
            .endif
            invoke CloseHandle,processInfo.hProcess
            mov processInfo.hProcess,0
         .else
            invoke DestroyWindow,hWnd
         .endif
      .endif
   .ELSE
      invoke DefWindowProc,hWnd,uMsg,wParam,lParam
      ret
   .ENDIF
   xor    eax,eax
   ret
WndProc endp
end start
```

بررسی کد فوق:

برنامه پنجره اصلی را ایجاد کرده و سپس menu handle را برای استفاده آتی ذخیره می کند. هنگامیکه کاربر بر روی گزینه Process در منو کلیک می نماید ، پیغام WM_INITMENUPOPUP را پردازش خواهیم کرد.

```
   .ELSEIF uMsg==WM_INITMENUPOPUP
      invoke GetExitCodeProcess,processInfo.hProcess,ADDR ExitCode
      .if eax==TRUE
         .if ExitCode==STILL_ACTIVE
            invoke EnableMenuItem,hMenu,IDM_CREATE_PROCESS,MF_GRAYED
            invoke EnableMenuItem,hMenu,IDM_TERMINATE,MF_ENABLED
         .else
            invoke EnableMenuItem,hMenu,IDM_CREATE_PROCESS,MF_ENABLED
            invoke EnableMenuItem,hMenu,IDM_TERMINATE,MF_GRAYED
         .endif
      .else
```

```
    invoke EnableMenuItem,hMenu,IDM_CREATE_PROCESS,MF_ENABLED
    invoke EnableMenuItem,hMenu,IDM_TERMINATE,MF_GRAYED
  .endif
```

در ابتدا توسط تابع GetExitCodeProcess بررسی می شود که آیا پروسه جدید در حال اجرا است ؟ پارامترهای این تابع توسط تابع CreateProcess مقدار دهی شده‌اند. اگر خروجی تابع ذکر شده FALSE باشد بدین معنا است که پروسه جدید هنوز ایجاد نشده است. در ادامه:

```
  .if ax==IDM_CREATE_PROCESS
    .if processInfo.hProcess!=0
      invoke CloseHandle,processInfo.hProcess
      mov processInfo.hProcess,0
    .endif
    invoke GetStartupInfo,ADDR startInfo
    invoke CreateProcess,ADDR programname,NULL,NULL,NULL,FALSE,\
                NORMAL_PRIORITY_CLASS,\
                NULL,NULL,ADDR startInfo,ADDR processInfo
    invoke CloseHandle,processInfo.hThread
```

هنگامیکه کاربر گزینه process start را از منو انتخاب می‌نماید ، مقدار عضو hProcess ساختار PROCESS_INFORMATION را بررسی می‌نماییم. در ابتدای کار این مقدار مساوی صفر است زیرا این ساختار در قسمت data. برنامه تعریف شده است. اگر این مقدار مساوی صفر باشد یعنی پروسه جدید ایجاد شده خاتمه یافته ، اما هنوز process handle بسته نشده ، بنابراین اینکار را در ادامه انجام خواهیم داد.

با فراخوانی GetStartupInfo ، اعضای ساختار startupinfo مقدار دهی خواهند شد. در ادامه خروجی تابع CreateProcess بدلیل پیچیده نشدن برنامه بررسی نشد. در برنامه های دیگر حتما باید اینکار صورت گیرد.

بلافاصله پس از فراخوانی CreateProcess ، thread handle دریافت شده از ساختار processInfo ، بسته خواهد شد. بستن این دستگیره بدین معنا نیست که ترد خاتمه یافته ، بلکه به این معنا است که دیگر نمی‌خواهیم از دستگیره ترد در برنامه استفاده کنیم.در صورت عدم انجام اینکار نشتی منابع (resource leak) را در برنامه خواهیم داشت. سپس:

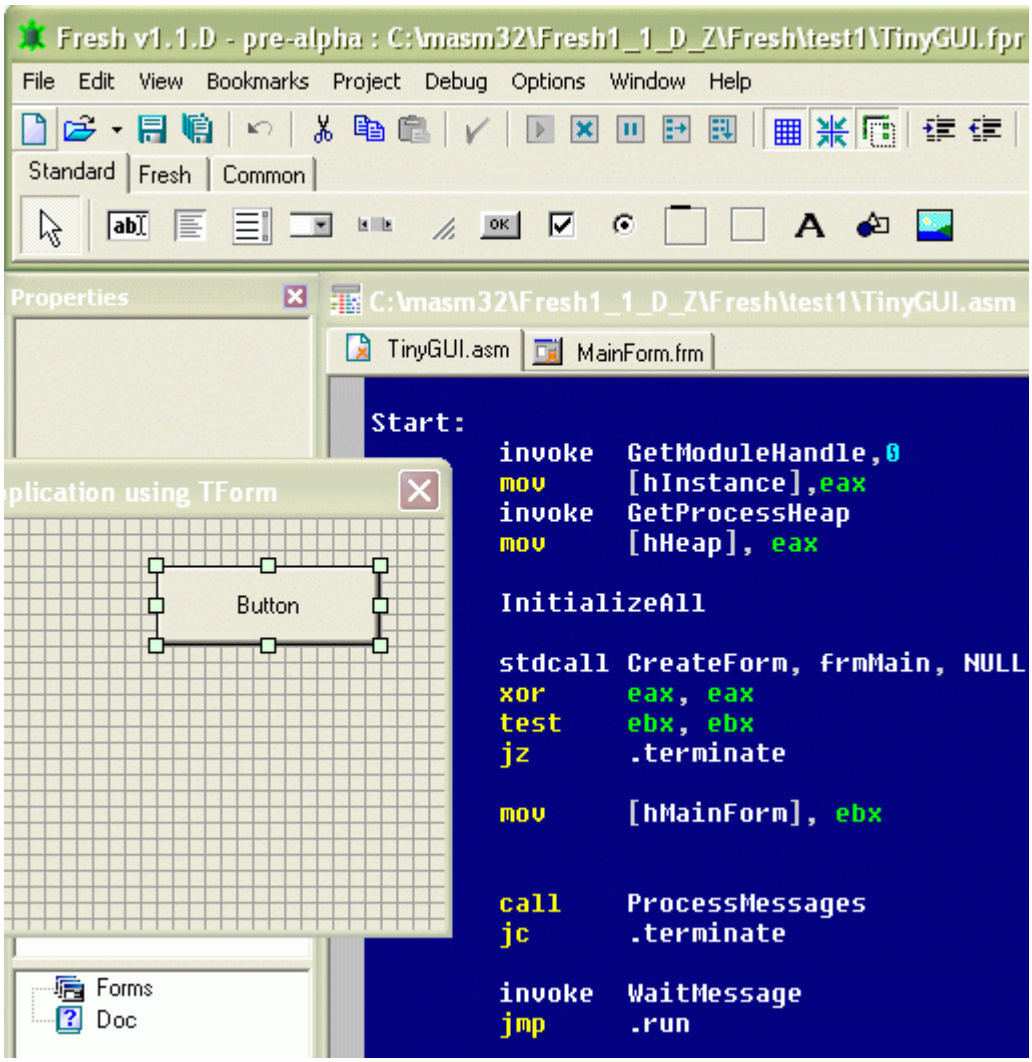
```
  .elseif ax==IDM_TERMINATE
    invoke GetExitCodeProcess,processInfo.hProcess,ADDR ExitCode
    .if ExitCode==STILL_ACTIVE
      invoke TerminateProcess,processInfo.hProcess,0
    .endif
    invoke CloseHandle,processInfo.hProcess
    mov processInfo.hProcess,0
```

هنگامیکه گزینه terminate process از منو انتخاب می‌شود ، ابتدا توسط تابع GetExitCodeProcess بررسی می‌شود که آیا پروسه هنوز فعال است یا خیر؟ در صورت مثبت بودن پاسخ ، با کمک TerminateProcess به پروسه خاتمه داده می‌شود. همچنین دستگیره این پروسه جدید نیز بسته می‌شود.

آشنایی مقدماتی با Fresh

اگر بدنبال ساده‌تر کردن کد نویسی اسمبلر هستید و دریافته‌اید که قسمت عمده ای از طراحی رابط کاربر برنامه‌ها همانند قالب یکسانی در اکثر برنامه‌ها تکرار می‌شوند، می‌توان از محیط‌های توسعه مجتمع (IDE) که به این منظور تهیه شده‌اند، استفاده کرد. برای نمونه می‌توان از Fresh یاد کرد. این برنامه از آدرس زیر قابل دریافت است:

http://fresh.flatassembler.net



شکل ۵ – نمایی از محیط Fresh
Flat assembler visual programming IDE

مثال ۱۳– برنامه نویسی چند ریسمانی (Multithreading Programming)

در مثال قبل آموختیم که هر پروسه حداقل از یک ترد تشکیل شده است که به آن primary thread نیز می‌گویند. همچنین یک برنامه نیز می‌تواند تردهایی را ایجاد نماید. از دیدگاه پیاده سازی ، یک ترد تابعی است که به صورت همزمان با برنامه اصلی اجرا می‌شود. بسته به نیاز می‌توان چندین وهله از یک تابع و یا چندین تابع مختلف را همزمان اجرا کرد.

تردها تحت یک پروسه اجرا می‌شوند بنابراین به تمام منابع یک پروسه مانند متغیرها و غیره دسترسی دارند. هر ترد پشته مخصوص خود را دارد ، بنابراین متغیرهای محلی آن ، صرفا مختص به آن خواهند بود.همچنین مجموعه رجیسترهای مورد استفاده برای یک ترد نیز مختص به آن بوده و در هنگام اجرای تردی دیگر ، ترد قبلی می تواند حالت قبلی خود را بخاطر آورده و در صورت نیاز آنرا مجددا از سر گیرد. این موارد به صورت درونی توسط ویندوز مدیریت می‌شوند. تردها را به دو گروه می‌توان تقسیم کرد:

۱. تردهای رابط کاربر: این نوع تردها پنجره مخصوص خود را ایجاد کرده و توانایی دریافت پیغام های ویندوز را دارند.
۲. ترد کاری : این نوع تردها پنجره مخصوص خود را ایجاد نکرده و توانایی دریافت پیغام های ویندوز را ندارند.از این نوع تردها برای انجام کارهایی در پس زمینه برنامه استفاده می‌شوند.

توصیه می‌شود در هنگام پیاده سازی برنامه‌های چندریسمانی این روند را دنبال کنید: اجازه دهید ترد اصلی برنامه کارهای مربوط به رابط کاربر را مدیریت کند و از این مبحث صرفا جهت ایجاد تردهای کاری استفاده نمایید. بنابراین ترد اصلی برنامه شبیه به مدیر و سایر تردها همانند دستیاران و افراد او رفتار خواهند کرد. مدیر کارها را به افراد تحت مدیریت خود محول کرده و تماس خود را با بیرون حفظ می‌نماید. افراد کارهای محوله را انجام داده و گزارش‌های لازم را به مدیر ارائه می‌دهند.بهتر است کارهای طولانی را در تردهای کاری انجام داد و مسؤولیت پاسخ دهی به پیغام‌های رسیده از بیرون را به ترد اصلی برنامه محول کرد. برای مثال اگر کارهای طولانی را به ترد اصلی برنامه محول نمایید ، تا پایان کار ترد ، برنامه نمی‌تواند به پیغام های رسیده از بیرون پاسخ لازم و مناسب را بدهد.

با فراخوانی تابع CreateThread یک ترد ایجاد خواهد شد. روش تعریف این تابع به شکل زیر است:

```
CreateThread proto lpThreadAttributes:DWORD,\
                   dwStackSize:DWORD,\
                   lpStartAddress:DWORD,\
                   lpParameter:DWORD,\
                   dwCreationFlags:DWORD,\
                   lpThreadId:DWORD
```

lpThreadAttributes : با مساوی NULL قرار دادن این پارامتر می توان از خواص امنیتی پیش فرض ویندوز در مورد ترد استفاده کرد.

dwStackSize : اندازه پشته ترد را مشخص می کند. با مساوی نال قرار دادن این پارامتر ، اندازه پشته با اندازه پشته ترد اصلی یکی خواهد شد.

lpStartAddress : آدرس تابع ترد است. تابع ترد ، تابعی است که کار ترد را انجام خواهد داد.

lpParameter : پارامتری که مایل هستید به تابع ترد ارسال کنید.

dwCreationFlags : اگر مساوی صفر قرار گیرد یعنی ترد بلافاصله پس از ایجاد باید اجرا شود. برخلاف آن CREATE_SUSPENDED است.

lpThreadId : تابع CreateThread آدرس id ترد ایجاد شده را در این پارامتر قرار می‌دهد.

اگر فراخوانی تابع CreateThread موفقیت آمیز باشد ، دستگیره (handle) ترد ایجاد شده بازگشت داده خواهد شد.همچنین تابع ترد ، کار خویش را بلافاصله در صورتیکه پرچم CREATE_SUSPENDED تنظیم نشده باشد ، شروع خواهد کرد.اگر این پرچم تنظیم شده باشد ، تابع ترد کار خود را تنها با فراخوانی تابع ResumeThread شروع می‌نماید.

اگر تابع ترد بوسیله دستورالعمل RET خاتمه یابد، ویندوز تابع ExitThread را جهت تابع ترد به صورت ضمنی فراخوانی می کند.امکان فراخوانی تابع ExitThread از درون تابع ترد نیز موجود است ، اما باید دقت داشت که کد خروجی ترد را می‌توان توسط تابع GetExitCodeThread بدست آورد.

اگر قصد خاتمه یک ترد را توسط تردی دیگر دارید ، با کمک تابع TerminateThread می‌توان اینکار را انجام داد. البته باید دقت داشت فراخوانی این تابع ، سبب خاتمه آنی ترد بدون فرصت دادن به آن جهت بازگرداندن منابع تخصیص‌داده شده به سیستم، می گردد.

حداقل سه روش برای تبادل اطلاعات بین تردها وجود دارد:

- استفاده از متغیرهای سراسری
- استفاده از پیغام‌های ویندوز
- رخدادها

تردها منابع پروسه‌ها را (شامل متغیرهای سراسری) بین خویش به اشتراک می گذارند.بنابراین تردها از متغیرهای سراسری جهت تبادل اطلاعات باهم استفاده می کنند.البته این مورد را باید با دقت کامل به همراه نکات مربوط به همزمانی تردها بکار گرفت.

فرض کنید دو ترد ، از یک ساختار با ۱۰ عضو ، به صورت مشترک استفاده می‌کنند. چه اتفاقی خواهد افتاد اگر هنگامیکه یک ترد در نیمه به روز رسانی مقادیر اعضای ساختار باشد و ناگهان ویندوز کنترل را از این ترد به تردی دیگر منتقل نماید؟ ترد دیگر با داده‌هایی ناقص روبرو خواهد شد. مدیریت برنامه‌های چند ریسمانی مشکل تر بوده و نیاز به دقت بیشتری دارند.

همچنین از پیغامهای ویندوز جهت تبادل اطلاعات بین تردها می‌توان کمک گرفت. اگر تردها از نوع تردهای رابط کاربر باشند می توان برای آنها با توجه به پیغامهای موجود ویندوز ، پیغامهای سفارشی طراحی کرد. برای مثال:

```
WM_MYCUSTOMMSG equ WM_USER+100h
```

اما اگر یکی از تردها ، تردکاری باشد دیگر از این روش نمی‌توان استفاده کرد. زیرا تردکاری پنجره مخصوص به خود را نداشته و از صف پیغامها بی‌بهره است. در این موارد از شمای زیر می‌توان استفاده کرد:

ترد رابط کاربر ⟸ متغیرهای سراسری ⟸ تردکاری

تردکاری ⟸ پیغامهای سفارشی ⟸ ترد رابط کاربر

و از این روش در برنامه این مثال استفاده خواهیم کرد.

آخرین روش تبادل اطلاعات بین تردها، استفاده از اشیاء رخدادها است. به این نوع اشیاء همانند نوعی پرچم نیز می‌توان نگریست. اگر شیء رخداد در حالت unsignalled باشد، ترد در دوره‌ی کمون به سر می‌برد. در این حالت قطعات زمانی CPU به ترد اختصاص داده نخواهد شد. اگر شیء رخداد در حالت signalled قرار گیرد، ویندوز ترد را به حالت نرمال درآورده و ترد کار عادی خود را آغاز خواهد کرد.

کد برنامه:

فایلهای همراه این مجموعه را دریافت و سپس برنامه thread1.exe را اجرا کنید. گزینه Savage Calculation را از منوی برنامه انتخاب کنید. اینکار سبب اجرای add eax,eax به تعداد 600,000,000 بار می‌گردد. لازم به ذکر است که در طول انجام این عملیات هیچگونه کاری با رابط کاربر برنامه نمی‌توان انجام داد. برای حل این مشکل می‌توان انجام

این محاسبات را به ترد کاری برنامه انتقال داد و به ترد اولیه برنامه اجازه پاسخ دهی به درخواست‌های رسیده به رابط کاربر را داد (هر چند اینکار کندتر از معمول قابل انجام خواهد شد اما بطور کامل از کار نمی‌افتد).

```
.386
.model flat,stdcall
option casemap:none
WinMain proto :DWORD,:DWORD,:DWORD,:DWORD
include \masm32\include\windows.inc
include \masm32\include\user32.inc
include \masm32\include\kernel32.inc
includelib \masm32\lib\user32.lib
includelib \masm32\lib\kernel32.lib

.const
IDM_CREATE_THREAD equ 1
IDM_EXIT equ 2
WM_FINISH equ WM_USER+100h

.data
ClassName db "Win32ASMThreadClass",0
AppName  db "Win32 ASM MultiThreading Example",0
MenuName db "FirstMenu",0
SuccessString db "The calculation is completed!",0

.data?
hInstance HINSTANCE ?
CommandLine LPSTR ?
hwnd HANDLE ?
ThreadID DWORD ?

.code
start:
   invoke GetModuleHandle, NULL
   mov    hInstance,eax
   invoke GetCommandLine
   mov CommandLine,eax
   invoke WinMain, hInstance,NULL,CommandLine, SW_SHOWDEFAULT
   invoke ExitProcess,eax

WinMain proc hInst:HINSTANCE,hPrevInst:HINSTANCE,CmdLine:LPSTR,CmdShow:DWORD
   LOCAL wc:WNDCLASSEX
   LOCAL msg:MSG
   mov   wc.cbSize,SIZEOF WNDCLASSEX
   mov   wc.style, CS_HREDRAW or CS_VREDRAW
   mov   wc.lpfnWndProc, OFFSET WndProc
   mov   wc.cbClsExtra,NULL
   mov   wc.cbWndExtra,NULL
   push  hInst
   pop   wc.hInstance
   mov   wc.hbrBackground,COLOR_WINDOW+1
   mov   wc.lpszMenuName,OFFSET MenuName
   mov   wc.lpszClassName,OFFSET ClassName
   invoke LoadIcon,NULL,IDI_APPLICATION
   mov   wc.hIcon,eax
   mov   wc.hIconSm,eax
```

```
    invoke LoadCursor,NULL,IDC_ARROW
    mov   wc.hCursor,eax
    invoke RegisterClassEx, addr wc
    invoke CreateWindowEx,WS_EX_CLIENTEDGE,ADDR ClassName,ADDR AppName,\
          WS_OVERLAPPEDWINDOW,CW_USEDEFAULT,\
          CW_USEDEFAULT,300,200,NULL,NULL,\
          hInst,NULL
    mov   hwnd,eax
    invoke ShowWindow, hwnd,SW_SHOWNORMAL
    invoke UpdateWindow, hwnd
    .WHILE TRUE
          invoke GetMessage, ADDR msg,NULL,0,0
          .BREAK .IF (!eax)
          invoke TranslateMessage, ADDR msg
          invoke DispatchMessage, ADDR msg
    .ENDW
    mov     eax,msg.wParam
    ret
WinMain endp

WndProc proc hWnd:HWND, uMsg:UINT, wParam:WPARAM, lParam:LPARAM
    .IF uMsg==WM_DESTROY
       invoke PostQuitMessage,NULL
    .ELSEIF uMsg==WM_COMMAND
       mov eax,wParam
       .if lParam==0
          .if ax==IDM_CREATE_THREAD
             mov  eax,OFFSET ThreadProc
             invoke CreateThread,NULL,NULL,eax,\
                                 0,\
                                 ADDR ThreadID
             invoke CloseHandle,eax
          .else
             invoke DestroyWindow,hWnd
          .endif
       .endif
    .ELSEIF uMsg==WM_FINISH
       invoke MessageBox,NULL,ADDR SuccessString,ADDR AppName,MB_OK
    .ELSE
       invoke DefWindowProc,hWnd,uMsg,wParam,lParam
       ret
    .ENDIF
    xor    eax,eax
    ret
WndProc endp

ThreadProc PROC USES ecx Param:DWORD
       mov  ecx,600000000
Loop1:
       add  eax,eax
       dec  ecx
       jz   Get_out
       jmp  Loop1
Get_out:
       invoke PostMessage,hwnd,WM_FINISH,NULL,NULL
       ret
ThreadProc ENDP

end start
```

بررسی کد فوق:

اگر کاربر از منوی برنامه گزینه Create Thread را انتخاب کند ، برنامه تردی را به شکل زیر ایجاد می کند:

```
.if ax==IDM_CREATE_THREAD
   mov  eax,OFFSET ThreadProc
   invoke CreateThread,NULL,NULL,eax,\
                  NULL,0,\
                  ADDR ThreadID
   invoke CloseHandle,eax
```

کد فوق ، تردی را جهت اجرای همزمان تابع ThreadProc با ترد اصلی برنامه ایجاد می کند.پس از اجرای موفقیت آمیز ترد ، تابع بلافاصله اجرا خواهد شد. همچنین از آنجائیکه در برنامه از دستگیره ترد استفاده نخواهد شد ، آنرا جهت جلوگیری از نشتی حافظه ، خواهیم بست (البته بستن آن به معنای خاتمه ترد نیست). در ادامه:

```
ThreadProc PROC USES ecx Param:DWORD
      mov  ecx,600000000
Loop1:
      add  eax,eax
      dec  ecx
      jz   Get_out
      jmp  Loop1
Get_out:
      invoke PostMessage,hwnd,WM_FINISH,NULL,NULL
      ret
ThreadProc ENDP
```

پس از پایان کار ترد ، پیغام WM_FINISH به پنجره اصلی برنامه ارسال می‌شود. این پیغام به صورت زیر تعریف شده‌است:

```
WM_FINISH equ WM_USER+100h
```

می‌توان چندین ترد را همزمان با انتخاب گزینه Create Thread از منو انتخاب کرد.

در این مثال تبادل داده‌ها بین تردها یک طرفه است (ارسال پیغام‌ها به پنجره برنامه). اگر می‌خواهید از پنجره به ترد کاری برنامه پیغام ها ارسال شود می توان به صورت زیر عمل کرد:

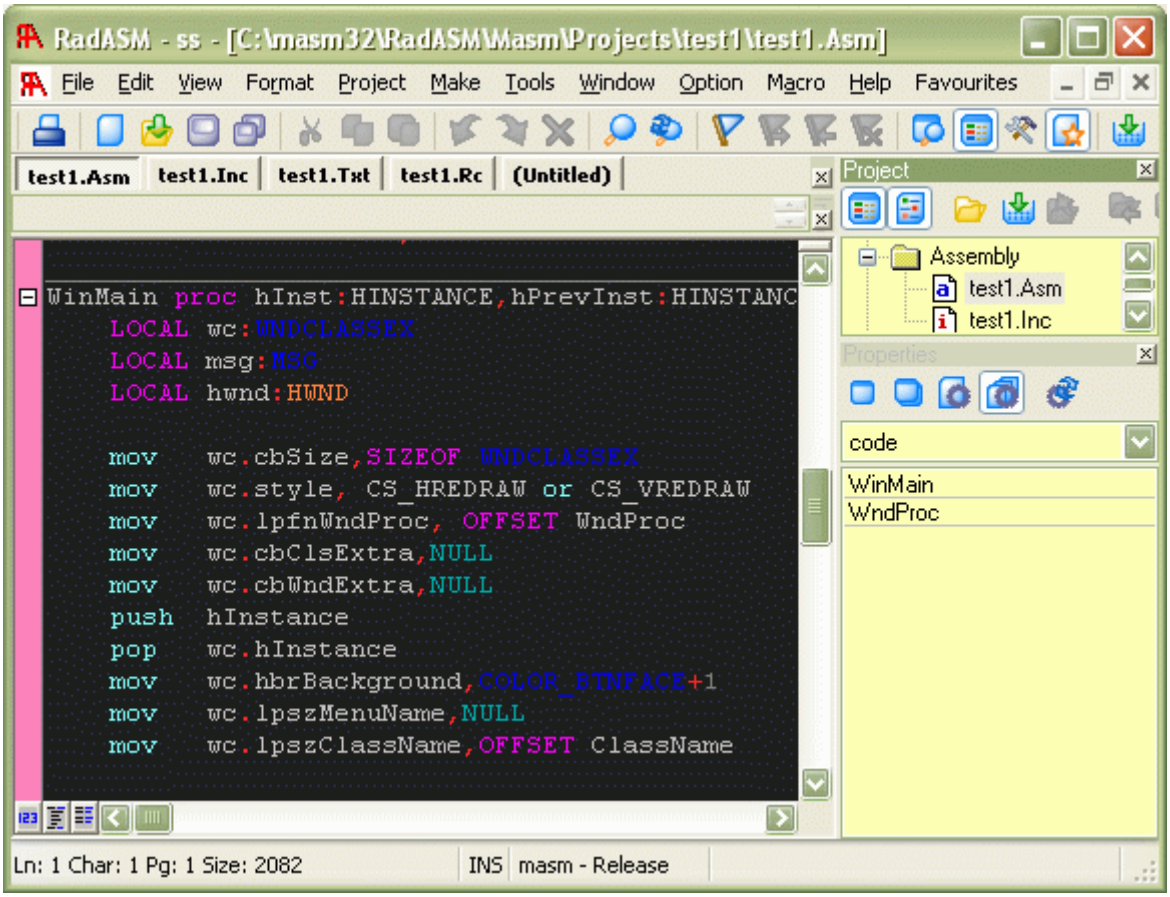
- اضافه کردن گزینه Kill Thread ، به منوی اصلی برنامه
- تعریف یک متغیر سراسری بعنوان یک پرچم . TRUE به معنای متوقف ساختن ترد و FALSE به معنای ادامه ترد است.
- اصلاح تابع ThreadProc به صورتیکه این پرچم را درون حلقه بررسی کند و در صورتیکه کاربر گزینه Kill Thread را انتخاب کرده بود ، با بررسی پرچم مربوطه ، ترد را خاتمه دهد.

آشنایی مقدماتی با RadASM

RadASM جهت سهولت کار با اسمبلرهای مختلفی همانند masm/tasm/fasm/nasm/goasm/hla طراحی شده است.

این برنامه از آدرس زیر قابل دریافت می‌باشد:

http://radasm.visualassembler.com



شکل ۶– نمایی از محیط RadASM

چندین مقاله آموزشی جهت کار با این IDE ، جهت عموم علاقمندان در آدرس زیر قرار دارند:

http://members.a1.net/ranmasaotome/main.html

مثال ۱۴– شیء رخداد (Event Object)

در مثال قبل تبادل اطلاعات بین تردها توسط پیغامها بررسی شد. دو روش دیگر شامل استفاده از متغیرهای عمومی و شیء رخداد باقی ماندند که در این مثال بررسی خواهند شد. شیء رخداد شبیه به یک سوئیچ است و دو حالت خاموش و روشن را می‌تواند داشته باشد.زمانیکه این سوئیچ روشن باشد به آن حالت signalled نیز گفته می‌شود. شیء رخداد ایجاد شده ، در کد ترد مربوطه قرار گرفته و ۲ حالت آن مورد بررسی قرار می‌گیرد. اگر شیء رخداد در حالت nonsignalled باشد ، تردی که منتظر وضعیت روشن آن است در حالت خواب می‌باشد.وقتی تردها در حالت انتظار باشند ، زمان کمی از CPU را به خود اختصاص خواهند داد.

یک شیء رخداد با استفاده از تابع CreateEvent مطابق تعریف زیر ، ایجاد می‌شود:.

```
CreateEvent proto lpEventAttributes:DWORD,\
                  bManualReset:DWORD,\
                  bInitialState:DWORD,\
                  lpName:DWORD
```

lpEventAttribute : اگر مساوی NULL قرار گیرد ویژگی‌های امنیتی پیش فرض مورد استفاده قرار می‌گیرند.

bManualReset : اگر مایل هستید پس از فراخوانی تابع WaitForSingleObject ، شیء ترد به حالت nonsignalled برگردانده شود، باید این پارامتر را مساوی FALSE قرار دهید. در غیراینصورت باید از تابع ResetEvent ، جهت به حالت اول بازگرداندن وضعیت شیء رخداد استفاده کنید.

bInitialState :با مساوی TRUE قرار دادن این پارامتر ، شیء رخداد تولیدی در حالت signalled ایجاد خواهد شد.

lpName : اشاره گری به رشته ASCIIZ (مختوم به نال) که نام شیء رخداد می‌باشد.این نام در هنگام فراخوانی OpenEvent بکار برده می‌شود.

اگر فراخوانی موفقیت آمیز باشد ، تابع ، دستگیره ای را به شیء رخداد ایجاد شده باز می‌گرداند ، در غیراینصورت خروجی نال خواهد بود. حالت شیء رخداد را با فراخوانی دو تابع SetEvent (به حالت signalled) و ResetEvent می‌توان تغییر داد.

هنگامیکه شیء رخداد ایجاد شد ، باید تابع WaitForSingleObject را در یک ترد جهت بررسی وضعیت آن قرار داد.این تابع به شکل زیر تعریف می‌گردد:

```
WaitForSingleObject proto hObject:DWORD, dwTimeout:DWORD
```

hObject : دستگیره ای به یکی از اشیاء همزمانی. شیء رخداد یک شیء همزمانی بشمار می‌آید.

dwTimeout : زمانی است (برحسب میلی ثانیه) که تابع تا رسیدن به وضعیت signalled صبر خواهد کرد. اگر این زمان سپری شد و هنوز وضعیت شیء nonsignalled بود ، تابع اجرا خواهد شد. اگر می‌خواهید مدت زمان این صبرکردن بی نهایت شود باید پارامتر ذکر شده را مساوی INFINITE قرار دهید.

کد برنامه:

این مثال پنجره ای را نمایش خواهد داد که منتظر انتخاب گزینه ای در منو توسط کاربر می باشد. اگر کاربر گزینه run thread را انتخاب نمود ، ترد محاسبات بسیار زمانبری را شروع خواهد کرد. پس از پایان ترد پیغامی به کاربر جهت اطلاع او نمایش داده می‌شود. هنگامیکه ترد در حال اجرا است کاربر امکان گزینه stop thread را خواهد داشت.

```
.386
.model flat,stdcall
option casemap:none
WinMain proto :DWORD,:DWORD,:DWORD,:DWORD
include \masm32\include\windows.inc
include \masm32\include\user32.inc
include \masm32\include\kernel32.inc
includelib \masm32\lib\user32.lib
includelib \masm32\lib\kernel32.lib

.const
IDM_START_THREAD equ 1
IDM_STOP_THREAD equ 2
IDM_EXIT equ 3
WM_FINISH equ WM_USER+100h

.data
ClassName db "Win32ASMEventClass",0
AppName  db "Win32 ASM Event Example",0
MenuName db "FirstMenu",0
SuccessString db "The calculation is completed!",0
StopString db "The thread is stopped",0
EventStop BOOL FALSE

.data?
hInstance HINSTANCE ?
CommandLine LPSTR ?
hwnd HANDLE ?
hMenu HANDLE ?
ThreadID DWORD ?
ExitCode DWORD ?
hEventStart HANDLE ?
```

```
.code
start:
    invoke GetModuleHandle, NULL
    mov    hInstance,eax
    invoke GetCommandLine
    mov CommandLine,eax
    invoke WinMain, hInstance,NULL,CommandLine, SW_SHOWDEFAULT
    invoke ExitProcess,eax

WinMain proc hInst:HINSTANCE,hPrevInst:HINSTANCE,CmdLine:LPSTR,CmdShow:DWORD
    LOCAL wc:WNDCLASSEX
    LOCAL msg:MSG
    mov   wc.cbSize,SIZEOF WNDCLASSEX
    mov   wc.style, CS_HREDRAW or CS_VREDRAW
    mov   wc.lpfnWndProc, OFFSET WndProc
    mov   wc.cbClsExtra,NULL
    mov   wc.cbWndExtra,NULL
    push  hInst
    pop   wc.hInstance
    mov   wc.hbrBackground,COLOR_WINDOW+1
    mov   wc.lpszMenuName,OFFSET MenuName
    mov   wc.lpszClassName,OFFSET ClassName
    invoke LoadIcon,NULL,IDI_APPLICATION
    mov   wc.hIcon,eax
    mov   wc.hIconSm,eax
    invoke LoadCursor,NULL,IDC_ARROW
    mov   wc.hCursor,eax
    invoke RegisterClassEx, addr wc
    invoke CreateWindowEx,WS_EX_CLIENTEDGE,ADDR ClassName,\
           ADDR  AppName,\
          WS_OVERLAPPEDWINDOW,CW_USEDEFAULT,\
          CW_USEDEFAULT,300,200,NULL,NULL,\
          hInst,NULL
    mov   hwnd,eax
    invoke ShowWindow, hwnd,SW_SHOWNORMAL
    invoke UpdateWindow, hwnd
    invoke GetMenu,hwnd
    mov  hMenu,eax
    .WHILE TRUE
           invoke GetMessage, ADDR msg,NULL,0,0
           .BREAK .IF (!eax)
           invoke TranslateMessage, ADDR msg
           invoke DispatchMessage, ADDR msg
    .ENDW
    mov     eax,msg.wParam
    ret
WinMain endp

WndProc proc hWnd:HWND, uMsg:UINT, wParam:WPARAM, lParam:LPARAM
    .IF uMsg==WM_CREATE
       invoke CreateEvent,NULL,FALSE,FALSE,NULL
       mov  hEventStart,eax
       mov  eax,OFFSET ThreadProc
       invoke CreateThread,NULL,NULL,eax,\
                           NULL,0,\
                           ADDR ThreadID
       invoke CloseHandle,eax
    .ELSEIF uMsg==WM_DESTROY
       invoke PostQuitMessage,NULL
    .ELSEIF uMsg==WM_COMMAND
```

```
        mov eax,wParam
        .if lParam==0
           .if ax==IDM_START_THREAD
              invoke SetEvent,hEventStart
              invoke EnableMenuItem,hMenu,IDM_START_THREAD,MF_GRAYED
              invoke EnableMenuItem,hMenu,IDM_STOP_THREAD,MF_ENABLED
           .elseif ax==IDM_STOP_THREAD
              mov  EventStop,TRUE
              invoke EnableMenuItem,hMenu,IDM_START_THREAD,MF_ENABLED
              invoke EnableMenuItem,hMenu,IDM_STOP_THREAD,MF_GRAYED
           .else
              invoke DestroyWindow,hWnd
           .endif
        .endif
     .ELSEIF uMsg==WM_FINISH
        invoke MessageBox,NULL,ADDR SuccessString,ADDR AppName,MB_OK
     .ELSE
        invoke DefWindowProc,hWnd,uMsg,wParam,lParam
        ret
.ENDIF
     xor    eax,eax
     ret
WndProc endp

ThreadProc PROC USES ecx Param:DWORD
        invoke WaitForSingleObject,hEventStart,INFINITE
        mov  ecx,600000000
        .WHILE ecx!=0
                .if EventStop!=TRUE
                        add  eax,eax
                        dec  ecx
                .else
                        invoke MessageBox,hwnd,ADDR StopString,ADDR AppName,MB_OK
                        mov  EventStop,FALSE
                        jmp ThreadProc
                .endif
        .ENDW
        invoke PostMessage,hwnd,WM_FINISH,NULL,NULL
        invoke EnableMenuItem,hMenu,IDM_START_THREAD,MF_ENABLED
        invoke EnableMenuItem,hMenu,IDM_STOP_THREAD,MF_GRAYED
        jmp   ThreadProc
        ret
ThreadProc ENDP
end start
```

بررسی کد فوق:

در این مثال روش دیگری در مورد پیاده سازی ترد ها مورد بررسی قرار گرفت.

```
.IF uMsg==WM_CREATE
        invoke CreateEvent,NULL,FALSE,FALSE,NULL
        mov  hEventStart,eax
        mov  eax,OFFSET ThreadProc
        invoke CreateThread,NULL,NULL,eax,\
                        NULL,0,\
```

```
                ADDR ThreadID
invoke CloseHandle,eax
```

در خلال پردازش پیغام WM_CREATE ، شیء رخداد و ترد ایجاد شدند.

شیء رخداد در حالت nonsignalled با به حالت اول درآمدن خودکار، ایجاد گشت. پس از ایجاد شیء رخداد ، ترد ایجاد گردید. ترد پس از ایجاد ، بلافاصله اجرا نگشته و منتظر شیء رخداد می‌ماند تا مطابق کد زیر به حالت signalled درآید:

```
ThreadProc PROC USES ecx Param:DWORD
    invoke WaitForSingleObject,hEventStart,INFINITE
    mov  ecx,600000000
```

اولین خط رویه ترد، فراخوانی تابع WaitForSingleObject است. در اینجا برای حالت signalled شیء رخداد به مدت نامحدودی ، پیش از پایان تابع ، صبر خواهد شد. یعنی پس از ایجاد ترد ، آنرا اجبار به خواب خواهیم فرستاد!

هنگامیکه کاربر از منو گزینه run thread را انتخاب کند، شیء رخداد را به صورت زیر به حالت signalled در خواهیم آورد:

```
.if ax==IDM_START_THREAD
   invoke  SetEvent,hEventStart
```

در این حالت ترد شروع به کار خواهد کرد.

اگر کاربر از منوی برنامه گزینه را stop thread انتخاب کند ، مقدار متغیر عمومی EventStop را به TRUE تنظیم خواهیم کرد.

```
.if EventStop==FALSE
      add  eax,eax
      dec  ecx
.else
      invoke MessageBox,hwnd,ADDR StopString,ADDR AppName,MB_OK
      mov  EventStop,FALSE
      jmp ThreadProc
.endif
```

اینکار ترد را متوقف ساخته و برنامه مجدد به WaitForSingleObject پرش خواهد کرد. لازم به ذکر است که نیازی به فراخوانی تابع ResetEvent به صورت صریح نیست ، زیرا پارامتر bManualReset تابع CreateEvent را به FALSE تنظیم کرده ایم.

درخواستی از خواننده:

اگـر در مـتن فـوق خطـایی را مـشاهده فرمودیـد (فنـی یـا غیرفنـی) ، لطفـا آنـرا بـا نویـسنده بـه آدرس vahid_nasiri@yahoo.com مطرح بفرمایید.

مآخذ:

1- Iczelion's Win32 Assembly tutorials. Web: http://win32assembly.online.fr/tutorials.html
2- Win32Asm Tutorial by Thomas Bleeker. Web: http://www.madwizard.org
3- Masm32's help files.
4- An Introduction to Assembly Language I/II/III by Darwen.
Web: http://www.codeguru.com/Cpp/Cpp/cpp_mfc/tutorials/article.php/c9411
5- Win32 Assembly parts 1-6 by Chris Hobbs.
Web: http://www.gamedev.net/reference/list.asp?categoryid=20#101
6- Win32 Assembler Coding Tutorial. Web: http://www.deinmeister.de/w32asm1e.htm