PROGRAMOWANIE

Programowanie Win32 dla programistów języka assemblera x86

Witam w świecie programowania Win32.To prawda! Nie jest niemożliwe napisanie programu dla Windows w assemblerze. Tu skupię się na stosowaniu podstawowych funkcji systemu Windows. Powinieneś wiedzieć, lub nauczyć się z innych źródeł jak pisać kod dla procesorów Intela x86 (i kompatybilnych). Używam składni Intela dla instrukcji maszynowych więc informacja tu zawarta powinna być użyteczna przy znanych asemblerach ,takich jak MASM (Microsoftu) lub TASM (Borlanda/ Inprise).Zakładam, że używałeś programów Windows , ale nie zakładam ,że napisałeś program dla Windows .Zbiór podstawowych funkcji systemowych jest nazwany Win32 API (interfejs programowy aplikacji). Win32 API kiedyś był nazywany interfejsem SDK ponieważ był pierwotnym interfejs, wspieranym przez Microsoft's Windows SDK Software Development Kit). Było to odróżnienie od interfejsu MFC (Microsoft Foundation Classes) przeznaczonego dla C++. Teraz Microsoft wspiera więcej niż jeden SDK. Win32 API jest wspierany przez Platformę SDK.

Przykładowy kod by testowany na Win98 SE (drugie wydanie) z TASM 4.0 a linker i biblioteki z VC++ 4.0. Nie próbuj uczynić tych programów kompatybilnych z NT 3.xx.

- Programowanie
- Ściąganie kodu źródłowego ze stronami WWW
- Ściąganie kodu źródłowego bez stron WWW
- Łącza sieciowe i inne zasoby
- Platforma Win32
- Notka dla programistów MASM

Programowanie

- Bardzo początkujące
  - Wprowadzenie do aplikacji konsolowych i plików
  - Podstawowy program GUI Win32 - WINBASIC.ASM
  - Tłumaczenie tekstów Win32
- Podstawowe zasady
  - Okna wywoływane i prawo własności
  - Okna potomne
  - Trochę więcej o komunikatach i wprowadzenie do myszki o

    Wprowadzenie do grafiki
  - Odświeżanie z WM_PAINT
  - Wprowadzenie do kontrolek
- Pętle wiadomości
  - Pętle konwencjonalne i komunikatów wątków o

    Zachłanne pętle komunikatów
- Tematy GUI
  - Standardowe kontrolki
    - ③ PRZYCISK
  - Wprowadzenie do menu
  - Wprowadzenie do dialogów
- Tematy Nie - GUI
- Inne tematy
  - Style Windows
  - Teoria zakleszczenia
  - Więcej informacji programistycznych
  - Reszta Windows

Wprowadzenie do aplikacji konsolowych i plików

Najpierw przyjrzymy się Aplikacjom Konsolowym (Console App, w skrócie) które kopiują pliki. Program, nazwany "cp", pokazuje jak poddać analizie argumenty lini poleceń, i jak odczytać i zapisać pliki sekwencyjne. Console App są dostarczone z trzema otwartymi "standardami" plików, pozwalające nam tworzyć pogramy DOS'owskie i C, obejmując przekierowania I/O z wewnątrz DOS.

Wstęp

Aplikacje Win32 pracują w 32 bitowych segmentach używając modelu pamięci FLAT. Zatem twój pogram jest automatycznie w trybie chronionym. Adresy generowane i używane przez twój program są znane jako adresy liniowe. Cztery rejestry segmentowe (CS, DS, ES, i SS) są ustawiane więc nie ma znaczenia jakiego segmentu używasz adresując daną komórkę (adres liniowy). Jedynym wymagającym przesłonięcia segmentem jest segment FS, który jest używany do przechowania łańcucha obsługi wyjątków i informacji odnośnie wielowątkowości.

MASM i TASM używają domyślnie 16-bitowego trybu 8086, które czynią nowsze 32 bitowe instrukcje niedostępnymi. Więc przy tych asemblerach musimy mówić wyraźnie, przynamniej przy instrukcjach 386. Przy ostatnich wersjach, assemblery te pozwalają "uprościć" dyrektywy segmentowe, które są dostępne wraz z dyrektywą MODEL.

```
.model    flat
```

Ważną opcją assemblera jest rozróżnianie małych i dużych liter w nazwach zewnętrznych. Wszystkie nazwy Win32API rozróżniają duże i małe litery. W TASM, przełącznikiem między dużymi i małymi literami jest albo /MX albo /ML. Jeśli planujesz zastosować dyrektywę PUBLICDLL ,będziesz musiał użyć /ML, która czyni wszystkie zdefiniowane programowo symbole zdolnymi do rozróżniania wielkości liter

Uruchamianie programu

Wszystkie programy (aplikacje konsolowe i standardowe aplikacje GUI) mają "adres startowy" gdzie rozpoczyna się wykonywanie. Microsoft i Borland, jednak, używają różnych sposobów do określenia tego adresu.

Microsoft: Adres startowy jest symbolem PUBLIC. Kiedy wywołujemy linker MS linker, określamy /ENTRY: przełącznik ustawiający adres startowy. Jeśli określimy `/ENTRY:start`, będziemy musieli zdefiniować symbol PUBLIC nazwany `_start` (zauważ znak podkreślenia).

Borland: Adres startowy jest symbolem określonym w dyrektywie END . Nie może być PUBLIC.

```
  public  _start  ; status publiczny jest ignorowany przez linker Borlanda

 .code            ; uproszczona dyrektywa segmentu
  _start:

        ; reszta programu

  end    _start    ; adres ignorowany przez linker MS
```

Odzyskiwanie argumentów lini poleceń , i Win32 API

Najpierw spojrzymy na funkcje API ,a później pokażemy kod analizy składniowej. GetCommandLine jest funkcją API która zwraca wskaźnik (w EAX) do ciągu zakończonego zerem zawierającego linię poleceń. Cała linia poleceń jest dostępna , jeśli obejmuje rozszerzoną nazwę pliku EXE. Nie ma argumentów , wiec wszystko co musisz zrobić to napisać:

```
        .data
       cmd_line_ptr      dd  0

        extrn      GetCommandLine:near

        .code
        call     GetCommandLine
        mov      [cmd_line_ptr],eax

        call     parse_cmd_line     ; nasz własny parser
```

Czyli to co zobaczysz. To jest to co wszystkie teksty i wszystkie książki programistyczne pokazują. Chociaż funkcje zwracają wskaźnik do ciągu, są dwa rodzaje ciągów: te zawierające 8 bitowe znaki ANSII I te zawierające 16 bitowe znaki Unicode. A w konsekwencji, są rzeczywiście dwie rzeczywiste nazwy powiązane z tą funkcją: GetCommandLineA (A dla ANSI) i GetCommandLineW (W dla szerokiego Unicode). Programista C lub C++ odnosi się do plików "dołączanych" które predefiniują (jak makro) GetCommandLine jako jedną z dwóch rzeczywistych nazw w zależności od tego czy specjalna zmienna (makro) była zdefiniowana czy nie. Możemy użyć podobnej sztuczki w MASM i TASM. Win95 i Win98 wewnętrznie używają ANSI, a ponieważ większość wersji Unicode API jest zablokowanych, będziemy używać wersji ANSI. Naturalnym trybem NT jest Unicode, więc używając wersji Unicode pod NT unikniemy wewnętrznego tłumaczenia z ANSI na Unicode. Win32 API jest implementowane w DLL'ach (biblioteka dołączana dynamicznie). Biblioteki te pozostają oddzielone od twojego pliku EXE. W odróżnieniu od DOS , API nie są dostępne z INT. Podobnie jak zwykłe statyczne biblioteki ,zwykły mechanizm wywołanie - powrót stosu jest używany do uzyskania dostępu do funkcji w DLL'lach.. Jest drobny koszt tego ,że musisz użyć pośredniości .Specjalne biblioteki import są potrzebne do stworzenia łącz pośrednich. Microsoft i Borland, ponownie, używają różnych sposobów wykonania tego.

Microsoft: Jest oddzielny plik .LIB dla każdego DLL, wiec musisz wylistować każdy DLL do którego chcesz uzyskać dostęp. LIB dla KERNEL32.DLL jest nazywany KERNEL32.LIB, podobnie dla wszystkich innych DLL'i. Musisz tylko użyć jednego z dwóch łącz do każdego pliku .LIB dostarczając: łącze do procedury pośredniczącej JMP ,lub pośrednie łącze do danej. Nazwa łącza procedury pośredniczącej JMP jest "odnowioną" wersją nazwy API : poprzedzona znakiem podkreślenia i dołączony znak "@" + numerem argumentu bajtowego dziesiętnie. Zatem łącze GetCommandLineA jest nazwane `_GetCommandLineA@0`. Pośrednie łącze danych jest zawsze takie samo: zamienia znak podkreślenia z __imp__. Zatem pośrednie łącze danych jest nazywane `__imp__GetCommandLineA@0`. Te dwa są używane różnie:

```
        extrn   _GetCommandLineA@0:near
        call    _GetCommandLineA@0                  ; bezpośrednie wywołanie

        extrn   __imp__GetCommandLineA@0:dword      ; musi być DWORD  !!!
        call    __imp__GetCommandLineA@0            ; pośrednie wywołanie
```

Borland: W TASM 4.0, większość Win32 API jest zebranych w pojedynczym pliku IMPORT32.LIB . Nazwa łącza jest dokładnie taka sama jak nazwa API. Jest tylko jedna nazwa łącza, nazwa odcinka JMP ,a ty uzyskujesz dostęp z bezpośredniego wywołania.

```
        extrn      GetCommandLineA:near
        call     GetCommandLineA                    ; bezpoœrednie wywołanie
```

Jak wspomniałem, w obu linkerach, bezpośrednie wywołanie nie skacze do DLL!. Procedura pośrednicząca JMP jest pośrednim JMP który czyni końcowy skok do DLL Niezależnie od wyglądu , Microsoftowskie wywołanie pośrednie eliminuje JMP i, w konsekwencji jest szybszym wywołaniem. Moim przyjętym rozwiązaniem obsługi zarówno bibliotek Microsoft i Borland jest zastosowanie udokumentowanych nazw API, uchwyt pary nazw A/W w WIN32HST.INC, i użycie pliku dołączanego zawierającego teksty makr do zmiany nazwy wszystkich nazw łącz dla bibliotek Microsoft. Wyrafinowane makra mogą być użyte do minimalizowania liczby deklarowanych nazw, ale takie makra mogą być niekompatybilne pomiędzy dwoma asemblerami. Bądź świadom tego ,że WIN32HST.INC jest niekompletny. Brakuje wiele ze stałych i struktur Win32.

```
        ; Nie włączaj vclib.inc, jeśli używasz linkera Borland
        include vclib.inc     ; Nazwa łącza Microsoft  (Visual C++)

        include win32hst.inc  ; stałe, struktury, i podwójne nazwy
```

Wewnątrz VCLIB.INC są dwa wejścia:

```
    GetCommandLineA  equ    <_GetCommandLineA@0>
    GetCommandLineW equ  <_GetCommandLineW@0>
```

Wewnątrz WIN32HST.INC jest warunek i dwa wejścia:

```
    if UNICODE
    ;  ...
    GetCommandLine  equ  GetCommandLineW ;
    ...
    else
    ;  ...
    GetCommandLine  equ  GetCommandLineA ;
    ...
    endif
```

Otwieranie pliku

Zakładając ,że mamy wyekstrahowane dwie nazwy pliku z lini poleceń, możemy kontynuować aktualne przesyłanie plików. Do otwierania pliku używamy CreateFile, który ma siedem argumentów. Przyjrzyjmy się funkcji udokumentowanej w VC++

```
HANDLE CreateFile(
  LPCTSTR lpFileName,                           //wskaźnik do nazwy pliku
  DWORD dwDesiredAccess,                        //tryb dostępu(odczyt-zapis)
  DWORD dwShareMode,                            // tryb dzielenia
  LPSECURITY_ATTRIBUTES    lpSecurityAttributes,//wskaŸnik    do    deskryptora
bezpeiczeństwa
  DWORD dwCreationDistribution,                 // jak stworzć
  DWORD dwFlagsAndAttributes,                   //attrybuty pliku
  HANDLE hTemplateFile                        //uchwyt do pliku z atrybutami do
kopiowania
 );
```

Każdy z tych siedmiu argumentów musi być odłożony na stos, w odwrotnym porządku, zanim funkcja zostanie wywołana. Wszystkie z powyższych argumentów są 32 bitowe. Prefiksy LP i lp oznaczają, że argument jest wskaźnikiem. Musimy określić minimum cztery argumenty dla każdego I/O. Nieokreślone argumenty muszą być zerem.(operator "large" jest potrzebny dla TASM 4.0.) . API dokumentuje wiele stałych przez nazwę.

Poniżej zobaczysz je wszystkie pisane z dużej litery, jak to zwykle w C. Kilka z nich jest zdefiniowanych w WIN32HST.INC.

```
.data
  source_filename_ptr dd  0
  dest_filename_ptr     dd  0

  source_file_handle dd  0
  dest_file_handle     dd  0

      extrn     CreateFile:near

      .code
      push    large  0   ; plik szablonowy
      push    large FILE_ATTRIBUTE_NORMAL
      push    large OPEN_EXISTING
      push    large  0   ; atrybuty bezpieczeństwa
      push    large  0   ; tryb dzielony
      push    large GENERIC_READ
      push    [source_filename_ptr]
      call    CreateFile
      cmp     eax,INVALID_HANDLE_VALUE
      je      bad_source
      mov     [source_file_handle],eax

      push    large  0   ; plik szablonowy
      push    large FILE_ATTRIBUTE_NORMAL
      push    large CREATE_ALWAYS
      push    large  0   ; atrybuty bezpieczeństwa
      push    large  0   ; tryb dzielony
      push    large GENERIC_WRITE
      push    [dest_filename_ptr]
      call    CreateFile
      cmp     eax,INVALID_HANDLE_VALUE
      je      bad_dest
      mov     [dest_file_handle],eax
```

Jeśli jesteś zaznajomiony ze sposobem w jaki C jest zazwyczaj implementowany, odnotujesz ,że argumenty nie są usuwane ze stosu . Funkcje Win32 API robią to za ciebie. Jest to konwencja wywołania stdcall. Jedynym wyjątkiem są funkcje , które mają zmienną liczbę argumentów. Zwykła konwencja wywołania cdecl, generowana przez kompilatory C, jest używana przez te wyjątkowe funkcje, w przypadku których argumenty są odkładane po powrocie. W samym Win32 API, jest tylko jedna funkcja , która używa konwencjonalnego wywołania cdecl - wsprintf (ma ona dwie wersje wsprintfA i wsprintfW). Funkcje Win32 również zachowują rejestry EBX, ESI, EDI, i EBP ,więc możesz oczekiwać , że te rejestry mają taką samą wartość przed i po wywołaniu. Flaga kierunku , DF, musi być wyzerowana, aby ciąg ops w działaniu API w trybie wzrostowym. Flaga ta pozostaje wyzerowana kiedy funkcja zwraca wartość.

Odczytywanie i zapisywanie plików

Po otwarciu pliku możemy teraz wykonać kopiowanie. Funkcje ReadFile i WriteFile uzyskują dostęp do pliku strumienia bajtów, jak w DOS, oraz fread() i fwrite() w C. Każde przesłanie danych może być dowolnej długości a koniec pliku jest sygnalizowany poprzez zwrócenie przekazanych bajtów zerowych

```
BUFFER_SIZE     equ     32768

    .data
bytes_read     dd  0
bytes_written dd  0

    .data?
temp_buffer      db BUFFER_SIZE dup(?)
```

```
        extrn      ReadFile:near,WriteFile:near

        .code
    copy_loop:
        push    large  0        ;wskaźnik do struktury OVERLAPPED
        push    offset bytes_read
        push    large BUFFER_SIZE     ;maksimum bajtów do przekazania
        push    offset temp_buffer
        push    [source_file_handle]
        call    ReadFile
        cmp     [bytes_read],0

        je      end_copy

        push    large  0        ;wskaŸnik do struktury OVERLAPPED
        push    offset bytes_written
        push    [bytes_read]  ;zapis wszystkich bajtów, które były odczytane
        push    offset temp_buffer
        push    [dest_file_handle]
        call    WriteFile
        jmp     copy_loop
    end_copy:
```

Zamykanie plików i kończenie programu

Chociaż pliki są zazwyczaj zamykane na wyjściu, posprzątamy po sobie używając CloseHandle. Skończymy program używając ExitProcess.

```
        extrn      CloseHandle:near,ExitProcess:near

        .code
        push    [source_file_handle]
        call    CloseHandle

        push    [dest_file_handle]
        call    CloseHandle

        push    large  0   ; kod wyjœcia
        call    ExitProcess
```

Analiza składniowa lini poleceń

Funkcja GetCommandLine zwraca wskaźnik do ciągu zakończonego zerem (null). Obejmuje on poszerzoną nazwę pliku EXE naszego uruchomionego programu.

```
        .data?
    cmd_line_2       db       1024  dup(?)     ;przestrzeń  dla  rozszerzonych
argumentów

        .code
    parse_cmd_line:
        mov     esi,[cmd_line_ptr]     ; Ÿródło
        mov     edi,offset cmd_line_2  ; przeznaczenie
        call    scan_blanks
        call    scan_arg    ; skip EXE name

        call    scan_blanks
        mov     [source_filename_ptr],edi
        call    scan_arg

        call    scan_blanks
```

```
        mov     [dest_filename_ptr],edi
        call    scan_arg

        ret
```

Najpierw wykonamy eliminację początkowych odstępów .

```
    tab    equ    9

        .code
    scan_blanks_1:
        inc     esi
    scan_blanks:
        mov     al,[esi]
        cmp     al,' '
        je      scan_blanks_1
        cmp     al,tab
        je      scan_blanks_1
        ret     ; ESI wskazuje pierwszy znak różny od spacji
```

Nazwy plików, w ostatnich wersjach Windows mogą mieć wbudowaną przestrzeń, która może być sygnalizowana przez cudzysłowy. Będziemy stosować cudzysłów. Funkcja CreateFile wymaga ciągu zakończonego zerem (null) , więc dodamy go do niej.

```
    scan_arg:
        mov     al,[esi]
        cmp     al,0
        je      exit_scan_arg
        cmp     al,'"'
        je      scan_quoted
    scan_unquoted:
        mov     [edi],al
        inc     esi
        inc     edi
        mov     al,[esi]
        cmp     al,0
        je      exit_scan_arg

        cmp     al,' '
        je      exit_scan_arg
        cmp     al,tab
        je      exit_scan_arg
        cmp     al,'"'
        je      exit_scan_arg
        jmp     scan_unquoted
    scan_quoted:
        inc     esi    ; skip quote
        mov     al,[esi]
        cmp     al,0
        je      exit_scan_arg
        cmp     al,'"'
        je      exit_quoted
    scan_quoted_1:
        mov     [edi],al
        inc     esi
        inc     edi
        mov     al,[esi]
        cmp     al,0
        je      exit_scan_arg
        cmp     al,'"'
        je      exit_quoted
        jmp     scan_quoted_1
    exit_quoted:
        inc     esi    ; przeskoczenie cudzysłowu
    exit_scan_arg:
```

```
        mov     byte ptr  [edi],0    ; zakończenie ciągu przeznaczenia
        inc     edi
        ret             ; esi wskazuje ostatni argument
```

Procedura obsługi błędów i pliki standardowe

Tu wyświetlimy wiadomość o niepowodzeniu otwarcia. Standardowe aplikacje GUI nie gwarantują odebrania każdego "standardowego pliku". Jednak, aplikacje konsolowe pobierają odpowiednik stdin z C, stdout, i stderr. (Odpowiednik DOS'owskiego pliku logicznego 0, 1,i 2). Zarówno "stdin" i "stdout" mogą być przekierowane do lini poleceń w DOS. Uchwyty są uzyskiwane poprzez GetStdHandle.

```
        .data
bad_source_msg     db "Nie można otworzyć pliku źródłowego",13,10
bad_source_msg_len equ  $  - bad_source_msg

bad_dest_msg     db "Nie można otworzyć pliku przeznaczenia",13,10
bad_dest_msg_len equ  $  - bad_dest_msg

        extrn     GetStdHandle:near

        .code
bad_source:
        mov     esi,offset bad_source_msg
        mov     ecx,bad_source_msg_len
        jmp     error_exit
bad_dest:
        mov     esi,offset bad_dest_msg
        mov     ecx,bad_dest_msg_len
error_exit:
        push    large  0  ; ptr to OVERLAPPED structure
        push    offset bytes_written
        push    ecx             ; byte count
        push    esi             ; byte buffer

        push    large STD_OUTPUT_HANDLE
        call    GetStdHandle

        push    eax
        call    WriteFile

        push    large  0   ; kod wyjœcia

        call    ExitProcess
```

Linkowanie/tworzenie pliku wykonywalnego

Większość assemblerów x86 (wliczając w to MASM) będzie tworzyło Intelowskie pliki OMF z domyślnym rozszerzeniem .OBJ. Większość linkerów generuje pliki wykonywalne DOS, które nie mogą tworzyć plików wykonywalnych Win32. Dwa linkery są zilustrowane poniżej. Microsoftowski linker Win32 jest (dwuznacznie) nazwany LINK. Borlandowski linker toTLINK32.

Microsoft: 32-bitowy linker ma taką samą nazwę jak linker 16 bitowy. LINK oczekuje odbioru pliku wynikowego w Unixowym formacie COFF z domyślnym rozszerzeniem .OBJ. Jeśli plik .OBJ nie jest plikiem , linker (wersja 3.0 to robi) automatycznie konwertuje plik OMF na COFF. Plik wynikowy nie jest zachowywany. Ostatnie wersje MASM mogą tworzyć pliki COFF Win32 bezpośrednio. NASM jest innym asemblerem, który może generować pliki COFF Win32. Jak wcześniej mówiłem, będzie tylko jeden plik .LIB dla każdego linkowanego DLL'a. Wszystkie używane funkcje API znajdują się w KERNEL32.DLL, więc tylko jeden plik .LIB musi być zlinkowany. Win32 widzi aplikacje konsolową jako specjalny podsystem , więc musimy określić (/subsystem:console). I jak wspomniano wcześniej ,musi być określony punkt wejścia. Nazwy opcji są niezależne od wielkości liter. Zakładamy ,że pewne zmienne środowiskowe zostały ustawione

```
link cp kernel32.lib  /entry:start  /subsystem:console
```

Borland: TLINK32 może obsługiwać tylko pliki OMF . Więc zlinkowanie plików COFF (na przykład, pliki .LIB DirectX, lub pliki OBJ stworzone przez VC++) nie jest możliwe. Pod Win95, jeśli TLINK32 nie zdoła pracować w DOS, uruchom go w kompatybilnym trybie MS-DOS. Opcje linkera pokazane poniżej są dla stworzenia pliku wykonywalnego (/Tpe), aplikacji konsolowej (/ap), i linkowania z rozróżnieniem wielkości liter (/c). Opcje linkera (w przeciwieństwie do opcji assemblera) rozróżniają wielkość liter . Jak już mówiłem , większość z Win32 API jest zebranych w pojedynczej bibliotece, IMPORT32.LIB. Zakładamy, że pewne zmienne środowiskowe są ustawione.

```
tlink32  /Tpe  /ap  /c    cp,,,import32.lib
```

Podstawowy program GUI Win32 -WINBASIC.ASM

Program ten tworzy podstawową aplikację okna. Możesz przesuwać okna wokół. Możesz zmieniać rozmiar okna i wszystkie użyteczne przyciski pojawią się w pełni funkcjonalne. Jedynymi brakującymi rzeczami są paski przewijania i pasek menu.

Uruchomienie programu

Dla programu jednowątkowego, wartości początkowe rejestrów są prawie bez znaczenia. Ale dla Twojej ciekawości: CS:EIP = start programu; SS:ESP = start stosu; DS = ES = SS; FS = TIB, blok informacji wątku; GS = 0, selektor zera. CS i DS odwzorowywują ten sam adres liniowy. Flaga kierunku, DF jest zerowana.

Zaczniemy naszą aplikację GUI w ten sam sposób jak aplikację konsolową

```
    .386
    model    flat

    ; Jeśli używasz TLINK32,nie włączaj vclib.inc
    include vclib.inc     ; nazwa łączona  .lib Microsoft VC++

    include win32hst.inc ; stałe, struktury i nazwy wejść public

    _start

    .code
_start:
```

Klasa okna

Każde okno jest opisane przez strukturę danych czasu wykonania znaną jako klasa okna. ("klasa" ta nie jest klasą C++) Każda klasa okna osiada funkcję wywołania zwrotnego, znaną jako procedura okna, która definiuje większość zachowań wszystkich okien stworzonych tą klasą. Zanim stworzymy okno, jego klasa okna, zdefiniowana przez strukturę WNDCLASSEX , musi być podana nazwa i zarejestrowana przez wywołanie RegisterClassEx. Windows rejestruje wcześniej kilka klas okien, ale są one zazwyczaj używane do tworzenia kontrolek.
Najpierw spójrzmy na strukturę WNDCLASSEX zdefiniowaną w WIN32HST.INC. Dla ułatwienia tłumaczenia innym asemblerom nadałem prefiksy każdemu polu kiedy wywołuję "sygnaturę" struktury. Sygnatury te są pokazane jako komentarz w pliku nagłówkowym .h dla C. Reszta nazw pól, po znaku podkreślenia, jest dokładnie taka sama jak w plikach nagłówkowych C.

```
WNDCLASSEX STRUC
wcx_cbSize              dd  ?
wcx_style               dd  ?
wcx_lpfnWndProc         dd  ?
wcx_cbClsExtra          dd  ?
wcx_cbWndExtra          dd  ?
wcx_hInstance           dd  ?
```

```
wcx_hIcon               dd  ?
wcx_hCursor             dd  ?
wcx_hbrBackground       dd  ?
wcx_lpszMenuName        dd  ?
wcx_lpszClassName       dd  ?
wcx_hIconSm             dd  ?
WNDCLASSEX ENDS
```

Wow! To dużo pól. Minimalnie trzy pola są konieczne do stworzenia okna: lpfnWndProc, hInstance, and lpszClassName. Zdefiniujemy również cztery inne pola: style, hIcon, hCursor, i hbrBackground. Pola nie używane muszą być wyzerowane.

Użyjemy nazw zdefiniowanych w dokumentacji API, ale nadamy prefiksy nazwom pól z "sygnatury" klasy, która jest pokazana w komentarzach w plikach nagłówkowych dla C

Pole lpfnWndProc zawiera adres procedury okna, która definiuje jak okno tej klasy będzie się zachowywać.

Pole hInstance zawiera uchwyt do modułu który "posiada" klasa okna. Win16 czyni różnicę pomiędzy instancją a uchwytem modułu, ale w Win32, są one jednym i tym samym. Te uchwyty instancji modułu identyfikują załadowany moduł (pliki EXE i DLL),w pewnym stopniu jak uchwyty pliku identyfikują otwarte pliki. Podobnie jak uchwyty plików, uchwyty moduł / instancja są unikalne tylko wewnątrz uruchomionej instancji programu lub procesu.

Pole lpszClassName zawiera adres ciągu zakończonego zerem (styl C) z nazwą klasy okna

Zamiast definiowania struktury z WNDCLASSEX, rozwiniemy ją i skomentujemy co jest w strukturze WNDCLASSEX.

```
    .data
    align   4
wcx dd      size WNDCLASSEX           ; cbSize
    dd      CS_VREDRAW or CS_HREDRAW  ; styl
    dd      WndProc                   ; lpfnWndProc
    dd      0,0                       ; cbClsExtra, cbWndExtra
    dd      0                         ; hInstance
    dd      0                         ; hIcon
    dd      0                         ; hCursor
    dd      COLOR_WINDOW+1            ; hbrBackground
    dd      0                         ; lpszMenuName
    dd      wndclsname                ; lpszClassName
    dd      0                         ; hIconSm

wndclsname db 'winmain',0
```

Pole cbSize ma całkowity rozmiar struktury WNDCLASSEX . Kilka struktur ma rozmiar struktury jako pierwsze pole. Czyni to łatwiejszym poszerzanie struktury. Późniejsze wydania Windows mogą używać tego pola do określania jaka wersja struktury będzie używana i odpowiednio zadziałać. Ważne jest ustawienie tego pola kiedy tak struktura jest używana do odbioru danych.

CS_VREDRAW i CS_HREDRAW w stylu pola wymuszają normalne zachowanie przerysowanego okna kiedy tylko zmieni się rozmiar. WndProc jest procedurą okna zdefiniowaną później w programie Pole hbrBackground określa domyślne lub podstawowe kolory obszaru ramki znanego jako obszar programu klienta. Wartość (COLOR_WINDOW + 1) czyni go takim samym jak inne okna obszaru programu klienta. Etykieta "wndclsname" definiuje lokację nazwy klasy okna zakończoną zerem

Teraz dla pola, które musi być wypełnione w czasie wykonania

```
    extrn     GetModuleHandle:near

    .code
    push    large  0            ; wskaźnik ciągu NULL
    call    GetModuleHandle     ; pobranie HINSTANCE/HMODULE pliku EXE
    mov     [wcx.wcx_hInstance],eax
```

Wywołanie GetModuleHandle ze wskaźnikiem NULL daje nam uchwyt instancji pliku EXE. Jest to moduł, który będzie "posiadał" klasę okna.

```
    extrn     LoadIcon:near,LoadCursor:near

    .code
    push    large IDI_WINLOGO
    push    large  0         ; hInstance,  0  = zasób ikon
```

```
    call    LoadIcon
    mov     [wcx.wcx_hIcon],eax

    push    large IDC_ARROW
    push    large  0        ; hInstance,  0  = zasób kursorów
    call    LoadCursor
    mov     [wcx.wcx_hCursor],eax
```

Pole hIcon daje nam ikonę "aplikacji" . A pole the hCursor daje nam obraz kursora, który jest używany kiedy kursor jest tworzony “ponad” oknem

```
    extrn     RegisterClassEx:near

    .code
    push    offset wcx
    call    RegisterClassEx
```

A zatem klasa okna jest zarejestrowana. Wszystkie wersje Windows używają tej struktury do stworzenia swojego własnego, wewnętrznego “obiektu” klasy okna, więc po zarejestrowaniu klasy możemy pozbyć się naszej "struktury klasy okna" bez żadnych ubocznych wpływów.

Tworzenie okna

Po tym jak nazwa klasy okna jest zarejestrowana, możemy stworzyć okno używając CreateWindowEx. Chociaż jest wiele pojedynczych argumentów my zasadniczo dostarczymy pięć (i unieważnimy inne): 1) nazwa klasy okna, 2) nagłówek okna, 3) lokacja okna, 4) rozmiar okna, i 5) jakaś standardowa opcja "styl"
Nazwa klasy okna jest kwalifikowana przez instancję modułu, którą posiada. Właściciel jest ustalony poprzez RegisterClassEx poprzez pole hInstance w strukturze danych WNDCLASSEX. W poniższym kodzie, użyjemy różnych podejść do przekazywania parametrów. Możemy traktować listę argumentów jako strukturę danych. Zamiast odkładania każdego elementu pojedynczo, tworzymy strukturę przechowującą wszystkie argumenty (we właściwym porządku), a potem odkładamy całą strukturę na stos. Możemy zmodyfikować każdą część zanim odłożymy strukturę, lub zmodyfikować strukturę na stosie. Poniższy kod wybiera ostatnie podejście

```
        .data
        align   4
cwargs    dd 0                  ; dwExStyle
        dd      wndclsname      ; lpszClass
        dd      wnd_title       ; lpszName
        dd      WS_VISIBLE or WS_OVERLAPPED or WS_SYSMENU or WS_THICKFRAME  \
             or WS_MINIMIZEBOX or WS_MAXIMIZEBOX    ; style
        dd   100                ; x
        dd   100                ; y
        dd   200                ; cx  (szerokość)
        dd   200                ; cy  (wysokość)
        dd   0                  ; hwndParent
        dd   0                  ; hMenu
        dd   0                  ; hInstance
        dd   0                  ; lpCreateParams

wnd_title db 'Application',0

        extrn     CreateWindowEx:near

        .code
        sub     esp,48            ; alokowanie listy argumentów
        mov     esi,offset cwargs  ; ustawienie bloku przesunięcia źródła
        mov       edi,esp                 ;  ustawienie  bloku  przesunięcia
przeznaczenia
        mov     ecx,12            ; liczba argumentów
        rep movsd
        mov     eax,[wcx.wcx_hInstance]
        mov       [esp+40],eax         ;  ustawienie  argumentu  hInstance  na
stosie
        call    CreateWindowEx
```

Pętla komunikatów

Po tym jak okno jest stworzone i wyświetlone możemy działać z komunikatem wysłanym do naszego programu. Nasz podstawowy program GUI wielokrotnie pobiera komunikaty z kolejki komunikatów GetMessage. Zrobimy to pętlą komunikatów. Pętla ta jest czasami nazywana pompą komunikatów. Jeśli komunikat pochodzi z jednej z kilku postaci SendMessage, GetMessage będzie wywoływała właściwą procedurę okna bezpośrednio. W przeciwnym razie wiadomość jest kopiowana do lokalnego bufora (argument lpMsgt), i zwróci sterowanie do wywołującej funkcji GetMessage. Wartość zero (FAŁSZ) jest zwracana jeśli GetMessage uzyskuje komunikat WM_QUIT .
W naszym programie, pętla komunikatów jest opuszczana jeśli jest odzyskiwany komunikat WM_QUIT. W przeciwnym razie wywołuje właściwą procedurę okna przy użyciu DispatchMessage.

```
    extrn GetMessage:near,DispatchMessage:near

    .data
msgbuf MSG     <>

    .code
msg_loop:
    push    large  0      ; uMsgFilterMax
    push    large  0      ; uMsgFilterMin
    push    large  0      ; hWnd  (filtr),  0  = wszystkie okna
    push    offset msgbuf  ; lpMsg
    call    GetMessage    ; zwraca FALS jeœli WM_QUIT
    or     eax,eax
    jz     end_loop

    push    offset msgbuf
    call    DispatchMessage

    jmp     msg_loop

end_loop:
```

Zakończenie programu

Opuszczamy program funkcją ExitProcess.

```
    extrn ExitProcess:near

    .code
    push    large  0    ;  (błąd) kod zakończenia
    call    ExitProcess
```

Procedura okna

Poza pewnymi inicjalizacjami i pewnym porządkowaniem, program GUI wykonuje wszystko wewnątrz różnych procedur okna. Te procedury okna mogą być nazwane przez siebie (np. kiedy okno jest tworzone po raz pierwszy), lub może być uzyskana poprzez DispatchMessage. Ta procedura okna odbiera cztery argumenty DWORD, w takim porządku: uchwyt okna (hWnd), ID komunikatu (nMessage), pierwszy parametr wiadomości (wParam), drugi parametr wiadomości (lParam). Argument wParam zawdzięcza swoją oryginalną nazwę Win16, gdzie był argumentem WORD . Kiedy okno jest zamykane, domyślnym działaniem, obsługiwanym przez DefWindowProc, jest jego usunięcie . Okno jest usuwane z ekranu a komunikat WM_DESTROY jest wysyłany do procedury okna. Jeśli chcemy zakończyć nasz program na zamknięciu określonego okna, wtedy to okno musi odpowiedzieć na komunikat powiązany z zamykanym oknem. Zalecanym komunikatem jest WM_DESTROY. Nasza procedura okna robi to dla jednego i tylko jednego okna, wywołując PostQuitMessage jako odpowiedź. Potem procedura okna jest zamykana, komunikat WM_QUIT będzie odebrany przez GetMessage w naszej pętli komunikatów. Wszystkie inne komunikaty przetwarzane są przez DefWindowProc.

Jedną z zalet przejścia z kodu 16 bitowego na 32 bitowy jest dostępny dodatkowy tryb adresowania . Wszystkie osiem podstawowych rejestrów 32 bitowych może być używanych w złożonych trybach adresowania.

WndProc wykorzystuje to poprzez zastosowanie ESP do uzyskania dostępu do swoich argumentów. Użyjemy rozłożonej postaci [ESP+4+4] do pokazania , która część offsetu jest do przeskoczenia przez EIP na stosie a która część jest offsetem drugiego argumentu (ID komunikatu).

```
    extrn DefWindowProc:near,PostQuitMessage:near

    .code
WndProc:
    cmp      dword ptr  [esp+4+4],WM_DESTROY jne
    DefWindowProc

on_destroy:
    push    large  0
    call     PostQuitMessage

    xor      eax,eax
    ret    16

    end    _start
```

Linkowanie

Linkowanie standardowej aplikacji jest podobne do linkowania aplikacji konsolowych. Przy linkerze Microsoft określamy różne podsystemy

```
  link winbasic kernel32.lib user32.lib  /entry:start  /subsystem:windows
```

Przy linkerze Borlanda ,określamy standardową aplikację Windows przez /aa zamiast /ap. Opcja /V mówi linkerowi, aby ustawił podsystem wersji 4.0. (Ostrzeżenie: Linker , który pochodzi z TASM 4.0 nie wspiera /V.)

```
    tlink32  /Tpe  /aa  /c  /V4.0 winbasic,,,import32.lib
```

Tłumaczenie tekstówWin32

Oficjalne teksty Win32 API są napisane dla programistów C i C++. Jeśli chcesz aby ktoś przetłumaczył to wszystko na ASM, czekałbyś długo, długo - tego jest bardzo duże. Więc dobrą rzeczą jest nauczyć się jak tłumaczyć składnię funkcji API z kodu C na kod ASM .

Standardowe funkcje Win32

Podprogramy te używane w większości w ASM będą podstawowymi funkcjami Win32 API. Większość z tych funkcji ma stałą liczbę argumentów, a funkcje te używają konwencji wywołania stdcall. Kilka, które ma zmienna liczbę argumentów (np. wsprintf) używa konwencji wywołania cdecl. Typowa funkcja API jest pokazana w poniższym formacie:

wartość zwracana-typ funkcji-nazwa (arg-typ-1 arg-nazwa-1, ...);

Więc na przykład definicja SetWindowText wygląda tak:

BOOL SetWindowText (HWND hWnd, LPCTSTR lpString);

We wszystkich przypadkach argumenty będą startowane od odłożenia argumentu najbardziej na prawo. Więc wywołując powyższą funkcję, odkładamy argumenty przed jej wywołaniem:

Push lpString
Push hWnd
Call SetWindowText

Jeśli wiesz co oznaczają typy HWND i LPCTSTR , będziesz mógł skonwertować tą funkcję do czegoś takiego jak to::

```
push    offset title1         ; adres nowego tytułu ciągu
push    dword ptr  [ebp+8]    ; hwnd z listą argumentów wndproc
call    SetWindowText         ; wynik BOOL sukces/porażka) w EAX
; argumenty już zdjęte przez SetWindowText
```

Powyżej jest zilustrowana konwencja wywołania stdcall używana przez większość funkcji Win32 API. SetWindowText zdejmuje argumenty, więc nie musisz tego robić. Jeśli używałeś konwencji wywołania cdecl , nie będziesz musiał przywracać ESP z dodatkową instrukcją , jak poniżej:

```
add     esp,8       ; zdjęcie dwóch argumentów DWORD
```

Ponieważ SetWindowText używa ciągu znaków, teksty w rzeczywistości odnoszą się do dwóch funkcji: SetWindowTextA, która używa ciągu znaków ANSI (8-bitów) , i SetWindowTextW, która używa ciągu znaków Unicode (16-bitów) . W programie C , plik dołączany windows.h ma makra, które automatycznie sterują i redefiniują te nazwy tekstowe do ich odpowiedników Unicode / ANSI. Zatem dla typowego programu będziemy używali tylko nazw tekstowych.
W MASM i TASM możemy użyć tekstów makr dla dostarczenia nazw domyślnych:

```
SetWindowText     equ     <SetWindowTextA>     ; użycie ANSI w Win95
```

Możesz , jeśli chcesz, użyć bardziej złożonych technik do wybrania jaką nazwę chcesz mieć jako domyślną

Interfejs COM (OLE2 i ActiveX)

Jest również interfejs oparty o obiekty COM (czasami znany jako OLE2 lub ActiveX) który używa innych konwencji wywołania. Jest dodatkowy, ukryty argument, który jest adresem rekordu interfejsu, a funkcja jest wywoływana pośrednio poprzez adres tablicy znanej jako vtable.
Interfejs COM jest zazwyczaj oprogramowany w C++. Składnia dla wywołania funkcji jest zazwyczaj taka:

wskaźnik_interfejsu -> funkcja (lista argumentów)

Argumenty są odkładane w odwrotnym porządku, podobnie jak funkcje SDK. Dodatkowy argument jest wskaźnikiem interfejsu, który jest odkładany ostatni. Potem używamy go do uzyskania dostępu do vtable i wywołania funkcji. Musisz znać dokładnie porządek w pamięci adresów funkcji. Zakładając, że wskaźnik_intefejsu jest w pamięci., instrukcje mogłyby wyglądać jak następuje:

```
;  ...
; wprowadzenie argumentu na stosie ; ...
mov     eax,[interface_pointer]
push    eax           ; odłożenie wskaźnika interfejsu
mov     ecx,[eax]     ;pierwszy dword interfejsu jest adresem vtable
call    [ecx+8]       ; wywołanie trzeciej funkcji
; argumenty już zdjęte
```

Ciąg danych w interfejsie COM powinien zawsze być w Unicode.

Rozmiar typów danych

Jeden typ, TCHAR, jest specjalny .Jego rozmiar zależy od tego czy program C jest domyślne ANSI (char) czy Unicode (short).
Pewne typy bez znaku są poprzedzone U. Wskaźniki są poprzedzone przez P lub LP. Uchwyty są poprzedzone przez H.
Niektóre rozmiary i typy:

BYTE: char (ze znakiem), BYTE (bez znaku)
WORD: short (ze znakiem), WORD (bez znaku), ATOM (bez znaku)
QWORD: double (zmienno przecinkowy)
DWORD: int (ze znakiem), DWORD (bez znaku), uchwyty, wskaźniki i miejmy nadzieję , że wszystko inne, co nie jest struct lub powiązane z poprzednimi typami

Konwencja zapisanych rejestrów

Źródłowe funkcje Win32 będą zabezpieczać wartości EBP, EBX, ESI, EDI, i DF. Na wejściu, oczekiwana jest wyzerowana flaga kierunku DF ,dla trybu wzrastającego w ops ciągu.

Wywołania zwrotne

Windows używa kilku funkcji z wywołaniem zwrotnym, które w większości są podobne do procedur okna. Funkcje te muszą następować po konwencji zapisanych rejestrów używanych przez funkcje API. Wszystkie z tych wywołań zwrotnych używają konwencji wywołania stdcall.
Ponieważ argumenty są odkładane w porządku odwrotnym, pierwszy wylistowany argument jest zawsze bezpośrednio po EIP na stosie. Jeśli używasz standardowego wstępu:

```
push    ebp
mov     ebp,esp
sub     esp,4     ; przykład alokacji jednego lokalnego DWORD
```

wtedy możesz uzyskać dostęp do pierwszego argumentu jako [ebp+8], drugiego argumentu jako [ebp+12], i tak dalej. Rozmiar każdego argumentu jest mnożony przez cztery bajty (DWORD). Większość argumentów to liczby całkowite, uchwyty lub wskaźniki: każdy z nich jest tylko jednym DWORD'em rozmiarowo. Standardowy wstęp musi mieć dobrany koniec:

```
mov     esp,ebp
pop     ebp
```

Argumenty mogą być zdjęte instrukcją powrotu

```
ret     16        ; przykład dla czterech argumentów DWORD
```

Okna wywoływane i posiadanie

Pierwsze okno jakie stworzyliśmy było to okno nakładkowe . Drugim typem okna jest okno wywoływane , lub w skrócie popup. Nie pojawia się żadna rzeczywista różnica pomiędzy oknami nakładkowymi a popup, z wyjątkiem inicjalizacji. Okno nakładkowe jest zawsze tworzone z obramowaniem i nagłówkiem (pasek tytułowy),ale okno popup może być tworzone bez tego. Możesz wymyślić w Windows gdzie i jak duże uczynić swoje okno nakładkowe przez użycie CW_USEDEFMAULT z argumentami x, y, cx, i cy CreateWindowEx, ale okno popup wymaga wyraźnej lokacji i rozmiaru argumentów. Czyniąc jedno okno zawsze pojawiające się przed innym oknem, możemy uczynić okno "w tle" właścicielem innego. Kiedy okno właściciela jest minimalizowane , okno posiadane będzie znikało. Przywrócenie okna zminimalizowanego spowoduje , ponowne pojawienie się okna podsianego. Tylko okna nakładkowe lub popup mogą być właścicielami lub posiadanymi. Okno popup często jest oknem dialogowym lub oknem komunikatu, więc jest sensownym uczynić okno główne właścicielem popup'ów Program winowner.asm pokazuje kluczowe cechy związku posiadacz/ posiadany . Program tworzy cztery okna popup , trzy z nich są posiadane przez okno główne. Okno czwarte jest pośrednio posiadane przez okno główne. Żadne z posiadanych okien nie będzie się pojawiało na pasku zadań. Ponieważ Popup1, Popup2, i Popup3 są posiadane na tym samym poziomie, jedno z nich może pojawić się "pomiędzy" pozostałymi dwoma. Ponieważ Popup1 posiada Popup4, inne popup'y (Popup2 i Popup3) nie mogą być umieszczone pomiędzy nimi. A żadne z okien popup nie może pojawić się "za" "głównym posiadaczem", oknem głównym. Jeśli jakieś okno jest częściowo przykryte, klikając na dowolną część okna programu przeniesiemy je na przód. Minimalizując okno właściciela (okno główne lub Popup1), popup'y będą znikały. Po zminimalizowaniu właściciela, przywracając go lub maksymalizując spowodujemy ,że jego popup'y zostaną przywrócone i pojawią się ponownie.

Klasy okien

Zarejestrujemy dwie klasy okien, jedną dla głównego okna rodzicielskiego i jedną dla okien popup Powstrzymując posiadane okna popup przed całkowitym zniknięciem (nie mamy jeszcze sposobu na ich odtworzenie), dodamy CS_NOCLOSE do stylu klasy popup .W przeciwnym razie, nie zrobimy nic specjalnego z oknami popup ,wiec uczynimy ich procedury okna domyślnymi, DefWindowProc.

```
.386
.model    flat

; Jeśli używamy TLINK32,nie dołączaj vclib.inc
include vclib.inc     ; nazwa linkowana  .lib Microsoft VC++

include win32hst.inc  ; stałe, struktury  ,i nazwy wejściowe

.data
align    4
```

```
wcx     dd      size WNDCLASSEX            ; cbSize
        dd      CS_VREDRAW or CS_HREDRAW   ; style
        dd      WndProc                    ; lpfnWndProc
        dd      0,0                        ; cbClsExtra, cbWndExtra
        dd      0                          ; hInstance
        dd      0                          ; hIcon
        dd      0                          ; hCursor
        dd      COLOR_WINDOW+1             ; hbrBackground
        dd      0                          ; lpszMenuName
        dd      wndclsname                 ; lpszClassName
        dd      0                          ; hIconSm

wndclsname db 'winmain',0

        align  4
popup_class     dd      size WNDCLASSEX            ; cbSize
                dd      CS_VREDRAW or CS_HREDRAW or CS_NOCLOSE  ; style
                dd      DefWindowProc              ; lpfnWndProc
                dd      0,0                        ; cbClsExtra, cbWndExtra
                dd      0                          ; hInstance
                dd      0                          ; hIcon
                dd      0                          ; hCursor
                dd      COLOR_WINDOW+1             ; hbrBackground
                dd      0                          ; lpszMenuName
                dd      popup_class_name           ; lpszClassName
                dd      0                          ; hIconSm

popup_class_name db 'popupclass',0

        .code

        public  _start
        extrn     GetModuleHandle:near
        extrn    LoadIcon:near,LoadCursor:near extrn
        RegisterClassEx:near

_start:
        push    large  0        ; ciąg NULL oznaczający wskaźnik
        call    GetModuleHandle  ; pobranie HINSTANCE/HMODULE pliku EXE
        mov     [wcx.wcx_hInstance],eax
        mov     [popup_class.wcx_hInstance],eax

        push    large IDI_WINLOGO
        push    large  0        ; hInstance,  0  = zbiór ikon
        call    LoadIcon
        mov     [wcx.wcx_hIcon],eax
        mov     [popup_class.wcx_hIcon],eax

        push    large IDC_ARROW
        push    large  0        ; hInstance,  0  = zbiór kursorów
        call    LoadCursor
        mov     [wcx.wcx_hCursor],eax
        mov     [popup_class.wcx_hCursor],eax

        push    offset wcx
        call    RegisterClassEx

        push    offset popup_class
        call    RegisterClassEx
```

Tworzenie głównego okna

Tu stworzymy okno główne. Po jego stworzeniu, zachowamy jego uchwyt i wywołamy podprogram tworzący okna popup .

```
        .data
        align   4

hMainWnd dd    0                    ; handle of main window

cwargs     dd    0                   ; dwExStyle
           dd      wndclsname        ; lpszClass
           dd      wnd_title         ; lpszName
           dd      WS_VISIBLE or WS_OVERLAPPED or WS_SYSMENU or WS_THICKFRAME  \
                 or WS_MINIMIZEBOX or WS_MAXIMIZEBOX    ; style
           dd    100                 ; x
           dd    100                 ; y
           dd    400                 ; cx  (szerokość)
           dd    200                 ; cy  (wysokość)
           dd    0                   ; hwndParent
           dd    0                   ; hMenu
           dd    0                   ; hInstance
           dd    0                   ; lpCreateParams

msgbuf MSG       <>

wnd_title  db  'Owner  of  popup  windows',0

        .code

        extrn      CreateWindowEx:near
        extrn         GetMessage:near,DispatchMessage:near
        extrn      ExitProcess:near

        sub      esp,48             ;alokacja listy argumentów
        mov     esi,offset cwargs  ; ustawienie przesuwanego bloku Ÿródłowego
        mov     edi,esp         ; ustawienie przesuwanego bloku przeznaczenia
        mov      ecx,12             ; liczba argumentów
        rep movsd
        mov      eax,[wcx.wcx_hInstance]
        mov      [esp+40],eax      ; ustawienie argumentu hInstance na stosie
        call     CreateWindowEx
        mov      [hMainWnd],eax

        call     create_popups      ; wywołanie naszej specjalnej procedury
```

Pętla komunikatów i zakończenie programu

Tu mamy minimalny kod pętli komunikatu i zakończenia programu.

```
msg_loop:
        push     large  0         ; uMsgFilterMax
        push     large  0         ; uMsgFilterMin
        push     large  0         ; hWnd  (filtr),  0  = wszystkie okna
        push     offset msgbuf    ; lpMsg
        call     GetMessage       ; zwracany FALS jeœli WM_QUIT
        or       eax,eax
        jz       end_loop

        push     offset msgbuf
        call     DispatchMessage

        jmp      msg_loop
```

```
end_loop:
        push    large  0  ;  (błąd) kod zakończenia
        call    ExitProcess
```

Procedura głównego okna

Tu mamy minimalną procedurę okna używaną przez okno główne.

```
        .code

        extrn      DefWindowProc:near,PostQuitMessage:near

WndProc:
        mov     eax,[esp+4+4]            ; ID komunikatu
        cmp     eax,WM_DESTROY           ; start destrukcji okna
        je      on_destroy
        jmp         DefWindowProc       ;  przekazanie  przetwarzania  innych
komunikatów
on_destroy:
        push    large  0
        call    PostQuitMessage

        xor     eax,eax
        ret     16
```

Tworzenie okna popup

Popup'y są tworzone poprzez wywołanie CreateWindowEx z WS_POPUP zwartego w argumencie styl Argument hwndParent jest użyty w tym przypadku jako argument właścicielski. Kiedy typ okna (ustawiony w argumencie styl) jest ustawiony na WS_OVERLAPPED lub WS_POPUP, argument ten jest traktowany jako właścicielski. Zanotuj, że WS_POPUP wymienia WS_OVERLAPPED. Dodamy również WS_CAPTION ponieważ popup'y nie mają automatycznie nagłówka (paska tytułowego). Poniższy kod tworzy duża listę argumentów CreateWindowEx poprzez zbudowanie jej bezpośrednio w obszarze adresowalnym, a potem kopiuje ją na stos.

```
        .data
        align   4
cwp_args       dd   0                   ; dwExStyle
               dd      popup_class_name ; lpszClass
popup_caption dd    0                   ; lpszName
               dd      WS_VISIBLE or WS_POPUP or WS_SYSMENU or WS_THICKFRAME  \
                    or  WS_MINIMIZEBOX  or  WS_MAXIMIZEBOX  or  WS_CAPTION    ;
style
popup_x        dd   0                   ; x
popup_y        dd   0                   ; y
               dd   200                 ; cx  (szerokość)
               dd   145                 ; cy  (wysokość)
hOwner         dd   0                   ; hwndParent
               dd   0                   ; hMenu  (control ID)
hInstance      dd   0                   ; hInstance
               dd   0                   ; lpCreateParams

popup_title1 db      'Popup1',0
popup_title2 db      'Popup2',0
popup_title3 db      'Popup3',0
popup_title4 db      'Popup4',0

        .code

create_popups:
        mov     eax,[wcx.wcx_hInstance]
        mov     [hInstance],eax    ; ustawienie argumentu hInstance
```

```
        mov     eax,[hMainWnd]
        mov     [hOwner],eax       ;ustawienie posiadania dla okna głównego
        mov     [popup_x],150
        mov     [popup_y],150
        mov     [popup_caption],offset popup_title1
        call    create_one_popup

        mov      [hOwner],eax       ;  ustawienie  posiadania  dla  poprzedniego
okna
        mov     [popup_x],200
        mov     [popup_y],200
        mov     [popup_caption],offset popup_title2
        call    create_one_popup

        mov     eax,[hMainWnd]
        mov       [hOwner],eax         ;ustawienie  posiadania  okna  głównego
mov     [popup_x],450
        mov     [popup_y],150
        mov     [popup_caption],offset popup_title3
        call    create_one_popup

        mov     eax,[hMainWnd]
        mov     [hOwner],eax       ;ustawienie posiadania dla okna głównego
        mov     [popup_x],750
        mov     [popup_y],150
        mov     [popup_caption],offset popup_title4
        call    create_one_popup

        ret

create_one_popup:
        sub     esp,48             ; alokacja listy argumentów
        mov     esi,offset cwp_args  ; ustawienie przesuwanego bloku źródła
        mov     edi,esp       ; ustawienie przesuwanego bloku przeznaczenia
        mov     ecx,12             ; liczba argumentów
        rep movsd
        call    CreateWindowEx
        ret

        end     _start
```

Okna potomne

Teraz spojrzymy na trzeci i ostatni typ okien okna potomne. Kiedy tworzysz okno potomne, mówisz Windows które okno będzie jego oknem nadrzędnym. Windows kontynuuje wyświetlania okna potomnego "wbudowanego" w jego okno nadrzędne . Kiedykolwiek okno nadrzędne jest minimalizowane lub czynione niewidocznym, okno potomne staje się niewidoczne. Kiedy okno nadrzędne jest potem przywracane lub maksymalizowane, okno potomne pojawia się ponownie. Okno potomne może być również rodzicem dla innego okna potomnego. Program winchild.asm pokazuje kluczowe cechy związku rodzic / potomek. Program ten tworzy cztery okna potomne, trzy z nich są bezpośrednio potomkami okna głównego. Czwarte okno jest potomkiem okna potomnego. Ponieważ Child1, Child2, i Child3 są na takim samym poziomie, każdy z nich może pojawić się "pomiędzy" pozostałymi dwoma. Ponieważ Child1 jest rodzicem dla Child4, inne potomki (Child2 i Child3) nie mogą być umieszczone pomiędzy nimi . Minimalizujemy okno nadrzędne (okno główne lub Child1), a jego okna potomne znikają Po zminimalizowaniu rodzica , jego przywrócenie lub zmaksymalizowanie spowoduje przywrócenie i pojawienie się ponownie jego okien potomnych. Jak zobaczysz, program ten jest bardzo podobny do programu winowner.asm .Kiedy uruchomisz ten pogram i zabawisz z oknami potomnymi, zobaczysz inne zachowania, które są odrobinę inne od pozostałych dwóch typów okien

Klasy okna

Zarejestrujemy dwie klasy okna, jedną dla głównego okna nadrzędnego i jedną dla okien potomnych. Dla zachowania okna potomnego przed całkowitym zniknięciem (nie mamy jeszcze sposobu na ich odtworzenie), dodamy CS_NOCLOSE do stylu klasy potomka. W przeciwnym razie nie zrobimy niczego szczególnego z oknami potomnymi, więc uczynimy domyślnymi procedury okna, DefWindowProc.
Wysokość i szerokość okna głównego jest ustawiona do wyświetlenia wszystkich okien potomnych.

```
        .386
        .model   flat

        ; Jeśli używasz TLINK32 nie włączaj vclib.inc
        include vclib.inc     ;linkowana nazwa  .lib Microsoft VC++

        include win32hst.inc  ;stałe, struktury, i nazwy wejść

        .data
        align   4
wcx     dd      size WNDCLASSEX            ; cbSize
        dd      CS_VREDRAW or CS_HREDRAW   ; style
        dd      WndProc                    ; lpfnWndProc
        dd      0,0                        ; cbClsExtra, cbWndExtra
        dd      0                          ; hInstance
        dd      0                          ; hIcon
        dd      0                          ; hCursor
        dd      COLOR_WINDOW+1             ; hbrBackground
        dd      0                          ; lpszMenuName
        dd      wndclsname                 ; lpszClassName
        dd      0                          ; hIconSm

wndclsname db 'winmain',0

        align  4
child_class     dd      size WNDCLASSEX            ; cbSize
                dd      CS_VREDRAW or CS_HREDRAW or CS_NOCLOSE  ; style
                dd      DefWindowProc              ; lpfnWndProc
                dd      0,0                        ; cbClsExtra, cbWndExtra
                dd      0                          ; hInstance
                dd      0                          ; hIcon
                dd      0                          ; hCursor
                dd      COLOR_WINDOW+1             ; hbrBackground
                dd      0                          ; lpszMenuName
                dd      child_class_name           ; lpszClassName
                dd      0                          ; hIconSm

child_class_name db 'childclass',0

        .code

        public  _start
        extrn     GetModuleHandle:near
        extrn    LoadIcon:near,LoadCursor:near extrn
        RegisterClassEx:near

_start:
        push    large  0        ; oznaczenie wskaźnika ciągu NULL
        call    GetModuleHandle  ;pobranie HINSTANCE/HMODULE pliku EXE
        mov     [wcx.wcx_hInstance],eax
        mov     [child_class.wcx_hInstance],eax

        push    large IDI_WINLOGO
        push    large  0        ; hInstance,  0  = zbiór ikon
```

```
        call    LoadIcon
        mov     [wcx.wcx_hIcon],eax
        mov     [child_class.wcx_hIcon],eax

        push    large IDC_ARROW
        push    large  0        ; hInstance,  0  = zbiór kursorów
        call    LoadCursor
        mov     [wcx.wcx_hCursor],eax
        mov     [child_class.wcx_hCursor],eax

        push    offset wcx
        call    RegisterClassEx

        push    offset child_class
        call    RegisterClassEx
```

Tworzenie okna głównego

Tu tworzymy okno główne. Po jego stworzeniu, zapiszemy jego uchwyt i wywołamy podprogram tworzący okna popup .

```
        .data
        align   4

hMainWnd dd   0                   ; uchwyt głównego okna

cwargs    dd    0                 ; dwExStyle
          dd     wndclsname       ; lpszClass
          dd     wnd_title        ; lpszName
          dd      WS_VISIBLE or WS_OVERLAPPED or WS_SYSMENU or WS_THICKFRAME  \
                or WS_MINIMIZEBOX or WS_MAXIMIZEBOX    ; style
          dd    100               ; x
          dd    100               ; y
          dd    765               ; cx  (szerokość)
          dd    240               ; cy  (wysokość)
          dd    0                 ; hwndParent
          dd    0                 ; hMenu
          dd    0                 ; hInstance
          dd    0                 ; lpCreateParams

msgbuf MSG      <>

wnd_title  db  'Rodzic  okna  potomnego',0

        .code

        extrn     CreateWindowEx:near
        extrn        GetMessage:near,DispatchMessage:near
        extrn     ExitProcess:near

        sub     esp,48             ;alokacja listy argumentów
        mov     esi,offset cwargs  ; ustawienie przesuwanego bloku Ÿródła
        mov     edi,esp        ; ustawienie przesuwanego bloku przeznaczenia
        mov     ecx,12             ; liczba argumentów
        rep movsd
        mov     eax,[wcx.wcx_hInstance]
        mov     [esp+40],eax     ; ustawienie argumentu hInstance na stosie
        call    CreateWindowEx
        mov     [hMainWnd],eax
        call    create_children    ; wywołanie naszej specjalnej procedury
```

Pętla komunikatu i zakończenie programu

Tu mamy minimalny kod pętli komunikatu i zakończenia programu.

```
msg_loop:
        push    large  0        ; uMsgFilterMax
        push    large  0        ; uMsgFilterMin
        push    large  0        ; hWnd  (filter),  0  = all windows
        push    offset msgbuf   ; lpMsg
        call    GetMessage      ; zwracane FALS jeœli WM_QUIT
        or      eax,eax
        jz      end_loop

        push    offset msgbuf
        call    DispatchMessage

        jmp     msg_loop

end_loop:
        push    large  0  ;  (błąd) kod zakończenia
        call    ExitProcess
```

Procedura głównego okna

Tu mamy minimalną procedurę okna, używaną przez okno główne.

```
        .code

        extrn      DefWindowProc:near,PostQuitMessage:near

WndProc:
        mov     eax,[esp+4+4]           ; ID komunikatu
        cmp     eax,WM_DESTROY          ; start destrukcji okna
        je      on_destroy
        jmp     DefWindowProc           ; przetwarzanie innych komunikatów

on_destroy:
        push    large  0
        call    PostQuitMessage

        xor     eax,eax
        ret     16
```

Tworzenie okien potomnych

Okna potomne są tworzone przez wywołanie CreateWindowEx z WS_CHILD zawartej w argumencie styl. Kiedy CreateWindowEx jest używana do tworzenia okna potomnego, współrzędne x-y dostarczają współrzędnych obszaru klienta , gdzie (0,0) jest to górny lewy róg obszaru ramki, obszaru programu klienta. Okna nakładkowe i popup'y są tworzone przez współrzędne ekranu, gdzie (0,0) to lewy górny róg pulpitu. Argument hwndParent jest oczywiście oknem nadrzędnym. Kiedy typ okna (ustawiony w argumencie styl) jest ustawiony na WS_CHILD, argument ten jest traktowany jako nadrzędny. Zauważmy ,że WS_CHILD podmienia WS_POPUP. Włączamy WS_CAPTION ponieważ okna potomne nie mają automatycznie nagłówka (paska tytułowego). WS_CLIPSIBLINGS jest wymagane jeśli przesuwne okna potomne działają racjonalnie, kiedy są nakładkowe. Usuńmy tą opcję stylu i zobaczmy jak szalone może by zachowanie okien potomnych, kiedy stają się nakładkowe! Poniższy kod tworzy dużą listę argumentów CreateWindowEx przez zbudowanie jej w bezpośrednim obszarze adresowalnym, a potem skopiowanie jej na stos.

```
        .data
        align   4
cwp_args      dd   0                  ; dwExStyle
              dd      child_class_name ; lpszClass
child_caption dd    0                  ; lpszName
```

```
                dd      WS_VISIBLE or WS_CHILD or WS_SYSMENU or WS_THICKFRAME  \
                     or WS_MINIMIZEBOX or WS_MAXIMIZEBOX or WS_CAPTION  \
                     or WS_CLIPSIBLINGS  ; style
child_x         dd   0                   ; x
child_y         dd   0                   ; y
child_width       dd 0                   ; cx  (szerokość)
child_height     dd  0                   ; cy  (wysokoœć)
hParent         dd   0                   ; hwndParent
                dd   0                   ; hMenu  (ID kontrolki)
hInstance       dd   0                   ; hInstance
                dd   0                   ; lpCreateParams

child_title1 db      'Child1',0
child_title2 db      'Child2',0
child_title3 db      'Child3',0
child_title4 db      'Child4',0

        .code

create_children:
        mov     eax,[wcx.wcx_hInstance]
        mov     [hInstance],eax     ; ustawienie argumentu hInstance

        mov     eax,[hMainWnd]
        mov     [hParent],eax       ;ustawienie nadrzędnoœci okna głównego
        mov     [child_x],10
        mov     [child_y],10
        mov     [child_width],240
        mov     [child_height],180
        mov     [child_caption],offset child_title1
        call    create_one_child

        mov       [hParent],eax        ;ustawienie  nadrzędności  poprzedniego
okna
        mov     [child_x],10
        mov     [child_y],10
        mov     [child_width],200
        mov     [child_height],120
        mov     [child_caption],offset child_title2
        call    create_one_child

        mov     eax,[hMainWnd]
        mov     [hParent],eax       ; ustawienie nadrzędności okna głównego
        mov     [child_x],260
        mov     [child_y],10
        mov     [child_width],240
        mov     [child_height],180
        mov     [child_caption],offset child_title3
        call    create_one_child

        mov     eax,[hMainWnd]
        mov     [hParent],eax       ; ustawienie nadrzędności okna głównego
        mov     [child_x],510
        mov     [child_y],10
        mov     [child_width],240
        mov     [child_height],180
        mov     [child_caption],offset child_title4
        call    create_one_child

        ret
```

```
create_one_child:
        sub     esp,48              ; alokacja listy argumentów
        mov      esi,offset cwp_args  ;ustawienie przesuwanego bloku źródłowego
        mov      edi,esp       ; ustawienie przesuwanego bloku przeznaczenia
        mov     ecx,12              ;liczba argumentów
        rep movsd
        call    CreateWindowEx
        ret

        end     _start
```

Trochę więcej o komunikatach i wprowadzenie do myszki.

Komunikaty są podstawą w programowaniu GUI Windows .Prawie wszystkie programy GUI są wyzwalane przez komunikaty. Pisząc program, który wielokrotnie oczekuje na komunikaty, wynika z tego styl programowania sterowania zdarzeniami, gdzie każdy komunikat przedstawia jedno zdarzenie .
Komunikaty WM_QUIT i WM_DESTROY zostały wprowadzone aby aplikacja mogła zakończyć się z gracją. W tej sekcji wprowadzimy komunikaty myszki, które pokazują jak program może odpowiadać na zdarzenia myszki w prosty sposób. Komunikaty myszki są powiązane z obszarem ramki okna, obszaru programu klienta. Dla programistów MS-DOS , napisanie procedury okna jest czyś takim jak napisanie przerwania INT 21H,z liczbą komunikatów na stosie zamiast w AH. A zamiast skoku do starego adresu DOS dla standardowych zachowań, wywołamy (lub skoczymy do) DefWindowProc. Windows tworzy wiele komunikatów, ale typowa aplikacja dostarcza odpowiedzi tylko na mały procent z nich. Większość komunikatów może być zaklasyfikowana jako powiadomienie—komunikaty "to się wydarzyło" lub "to jest o tym co się wydarza. Kilka komunikatów może być interpretowanych jako polecenia.

System komunikatów

System komunikatów może być potraktowany jako zbiór nadawców (twórców) i odbiorców (konsumentów) komunikatów. Komunikaty są zazwyczaj prowadzone przez trzymanie ich w kolejce. W ten sposób komunikaty są wysyłane i odbierane przez przechowanie i pobieranie ich z kolejki . Pozwala to nadawcy zasygnalizować odbiorcy dogodny czas, a odbiorcy odpowiedzieć w dogodnym czasie.

Nasz podstawowy program GUI obsługuje przyjmowanie komunikatów w dwóch częściach: pierwsza część kontroluje kolejkę komunikatów a druga część decyduje jak odpowiedzieć na dany komunikat. Nasz podstawowy program GUI jest napisany głównie jako odbiorca lub konsument komunikatów. Nadawcami lub twórcami tych komunikatów są albo funkcje API Windows lub sam system Windows .

Komunikaty

Komunikat jest pakietem informacji, który zawiera numer identyfikacyjny. Numery nie są rezerwowane przez Windows więc mogą być używane do naszych własnych celów. Niektórym liczby nadał Windows i używamy tych nazw - zaczynają się one od WM. Komunikaty są dostarczane do okna przez wywołanie procedur okien powiązanych z oknem. Jak opisano wcześniej, procedury okna są przydzielane w dwóch krokach. Najpierw przez zarejestrowanie klasy okna zawierającego adres procedury okna; i w drugim przez stworzenie okna przy użyciu tej klasy. Większość, ale nie wszystkie komunikaty są dostarczane do okien. WM_QUIT jest przykładem komunikatu, który nigdy nie dochodzi do okna. Jest tak dlatego, że nie obsługujemy go w procedurze okna. Kod napisany do odpowiadania na komunikat często jest nazywany programem obsługi komunikatu dla tego komunikatu. Różne okna mogą różnie odpowiadać na taki sam identyfikator komunikatu. (sam "pakiet" komunikatu jest skierowany tylko d jednego okna) .

Alokacja identyfikatorów komunikatów

Numery identyfikatorów komunikatów lub, identyfikator komunikatu, są alokowane jak następuje::

- 0 do 03FFh

  Standardowy komunikat okna. To są WM_ komunikaty. Nieużywane liczby są zarezerwowane przez Windows. Standardowe kontrolki również mają numery komunikatów w tym zakresie

- 0400h (WM_USER) to 7FFFh

  Komunikaty określonej klasy. Dla każdej klasy okna można zdefiniować komunikat w tym zakresie dla działania określonej klasy. Numery są ponownie używane przez klasy okien dostarczane przez Windows, a my możemy ich użyć ponownie dla własnych klas okien. Nowe wspólne kontrolki mają liczby w tym zakresie . (Dla tego zapoznaj się z wartościami Win16 , standardowe kontrolki nie mają numeru komunikatu w tym zakresie)

- 8000h (WM_APP) do 0BFFFh

  Komunikaty określonych aplikacji. Dla komunikatów nie określających żadnej klasy okna. Być może ten zbiór komunikatów powinien być nazwany komunikatami określonych wątków.

- 0C000h do 0FFFFh

  Komunikaty systemu globalnego. Numery te są zarezerwowane przez funkcję RegisterWindowMessage. Funkcja ta generuje numer komunikatu w tym zakresie.

- 10000h to 0FFFFFFFFh

  System komunikatów wewnętrznych (zastrzeżonych).Numery te są zarezerwowane przez Windows. Ponieważ wymagają one więcej niż 16 bitów nie mogą być użyte do komunikowania się ze starymi 16 bitowymi programami Windows które działają na platformie Win32 .

Obszar programu klienta

Pierwszy komunikat myszki jaki zobaczymy jest ten dostarczany kiedy kursor (który reprezentuje myszkę) wskazuje miejsce na obszarze klienta. Obszar programu klienta jest wyświetlanym obszarem ograniczony obrzeżami okna. Jeśli istnieją, tytuł, menu i paski przewijania są wyłączone z obszaru programu klienta.

Przesunięcia myszki i przyciski

Dwu przyciskowa mysz może tworzyć następujące podstawowe komunikaty kiedy tylko kursor wskazuje lokację w obszarze programu klienta::

- WM_LBUTTONDOWN, kiedy jest naciśnięty lewy przycisk
- WM_LBUTTONUP, kiedy jest zwolniony lewy przycisk
- WM_RBUTTONDOWN kiedy jest naciśnięty prawy przycisk
- WM_RBUTTONUP, kiedy jest zwolniony prawy przycisk
- WM_MOUSEMOVE, kiedy myszka jest przesuwana

Przykładowy program

Nasz przykładowy program , winclick.asm, raportuje aktywność myszki wewnątrz naszego okna

Początek i rejestracja klasy

Tu stworzymy podstawową klasę okna.

```
.386
.model    flat

; Jeśli używasz TLINK32, nie włączaj vclib.inc
include vclib.inc     ;linkowana nazwa  .lib Microsoft VC++

include win32hst.inc  ; stałe, struktury, I nazwy wejścia
```

```
        .data
        align   4
wcx     dd      size WNDCLASSEX             ; cbSize
        dd      CS_VREDRAW or CS_HREDRAW    ; style
        dd      WndProc                     ; lpfnWndProc
        dd      0,0                         ; cbClsExtra, cbWndExtra
        dd      0                           ; hInstance
        dd      0                           ; hIcon
        dd      0                           ; hCursor
        dd      COLOR_WINDOW+1              ; hbrBackground
        dd      0                           ; lpszMenuName
        dd      wndclsname                  ; lpszClassName
        dd      0                           ; hIconSm

wndclsname db 'winmain',0

        .code

        public  _start
        extrn     GetModuleHandle:near
        extrn   LoadIcon:near,LoadCursor:near extrn
        RegisterClassEx:near

_start:
        push    large  0        ;wskaźnik ciągu NULL
        call    GetModuleHandle  ;pobranie HINSTANCE/HMODULE pliku EXE
        mov     [wcx.wcx_hInstance],eax

        push    large IDI_WINLOGO
        push    large  0        ; hInstance,  0  = stock icon
        call    LoadIcon
        mov     [wcx.wcx_hIcon],eax

        push    large IDC_ARROW
        push    large  0        ; hInstance,  0  = stock cursor
        call    LoadCursor
        mov     [wcx.wcx_hCursor],eax

        push    offset wcx
        call    RegisterClassEx
```

Tworzenie okna i pętla komunikatów

Tworzymy nasze okno i ustawiamy go do wyświetlenia tytułu startowego. Szerokość okna jest dość długa aby uniknąć przycięcia tekstu tytułu.

```
       .data
        align   4

cwargs     dd  0                    ; dwExStyle
         dd      wndclsname         ; lpszClass
         dd      wnd_title          ; lpszName
         dd      WS_VISIBLE or WS_OVERLAPPED or WS_SYSMENU or WS_THICKFRAME  \
              or WS_MINIMIZEBOX or WS_MAXIMIZEBOX    ; style

         dd   100                   ; x
         dd   100                   ; y
         dd   400                   ; cx  (szerokość)
         dd   200                   ; cy  (wysokość)
         dd   0                     ; hwndParent
         dd   0                     ; hMenu
```

```
         dd   0                   ; hInstance
         dd   0                   ; lpCreateParams

msgbuf MSG      <>

wnd_title db 'Aktywnoœć myszki',0

        .code

        extrn     CreateWindowEx:near
        extrn        GetMessage:near,DispatchMessage:near
        extrn     ExitProcess:near

        sub     esp,48             ;alokacja listy argumentów
        mov     esi,offset cwargs  ;ustawienie przesuwanego bloku Ýródła
        mov    edi,esp         ; ustawienie przesuwanego bloku przeznaczenia
        mov     ecx,12             ; liczba    argumentów
        rep movsd
        mov     eax,[wcx.wcx_hInstance]
        mov     [esp+40],eax      ;ustawienie argumentu hInstance    na stosie
        call    CreateWindowEx

msg_loop:
        push    large  0         ; uMsgFilterMax
        push    large  0         ; uMsgFilterMin
        push    large  0         ; hWnd  (filter),  0  = all windows
        push    offset msgbuf    ; lpMsg
        call    GetMessage       ; zwraca FALSE jeœli WM_QUIT
        or      eax,eax
        jz      end_loop

        push    offset msgbuf
        call    DispatchMessage

        jmp     msg_loop

end_loop:
        push    large  0  ;  (błąd) kod zakończenia
        call    ExitProcess
```

Wysyłanie komunikatów

Poniżej mamy jeden sposób wysyłania komunikatów. Nie jest zły ponieważ jest kilka komunikatów. Duży zbiór komunikatów byłby łatwiejszy w obsłudze przy pomocy tablicy adresów

```
    .code

        extrn      DefWindowProc:near,PostQuitMessage:near

WndProc:
        mov     eax,[esp+4+4]            ; ID komuikatu
        cmp     eax,WM_DESTROY           ; start destrukcji okna
        je      on_destroy
        cmp     eax,WM_LBUTTONDOWN       ;  naciœnięto  lewy  przycisk  myszy
        je      on_left_mouse_down
        cmp     eax,WM_LBUTTONUP         ;  zwolniono  lewy  przycisk  myszy
        je      on_left_mouse_up
        cmp     eax,WM_RBUTTONDOWN       ; naciśnięto prawy przycisk myszy
        je      on_right_mouse_down
        cmp     eax,WM_RBUTTONUP         ; zwolniono prawy przycisk myszy
        je      on_right_mouse_up
```

```
        cmp     eax,WM_MOUSEMOVE        ; przesunięta myszka
        je      on_mouse_move
        jmp     DefWindowProc           ; przetwarzanie innych komunikatów
on_destroy:
        push    large  0
        call    PostQuitMessage

        xor     eax,eax
        ret     16
```

Obsługa komunikatów myszki

Jesteśmy tu gdzie odpowiadamy na komunikat myszki. Tekst w pasku tytułowym jest zmieniany przez wywołanie SetWindowText. Tekst wyświetlany zależy od aktywności myszki kiedy kursor wskazuje coś wewnątrz obszaru programu klienta. Kod jest lekko nie zoptymalizowany aby pokazać bardziej wyraźnie skąd pochodzi parametr hwnd . (+4 przeskakuje EIP na stosie, a +0 jest to offset pierwszego argumentu.)

```
       .data
        align   4
text_ldown  db  'Lewy  przycisk  myszki  wciśnięty',0
text_lup        db 'Lewy przycisk myszy zwolniony',0
text_rdown  db  'Prawy  przycisk  myszy  wciśnięty',0
text_rup       db 'Prawy przycisk myszy zwolniony',0
text_mmove db 'Mysz przesunięta',0

        .code

        extrn     SetWindowText:near

on_left_mouse_down:
        mov     eax,[esp+4+0]    ;pobranie hwnd przed zmianą ESP
        push    offset text_ldown
        push    eax
        call    SetWindowText

        xor     eax,eax
        ret     16

on_left_mouse_up:
        mov     eax,[esp+4+0]    ; pobranie hwnd przed zmianą ESP
        push    offset text_lup
        push    eax
        call    SetWindowText

        xor     eax,eax
        ret     16

on_right_mouse_down:
        mov     eax,[esp+4+0]    ; pobranie hwnd przed zmianą ESP
        push    offset text_rdown
        push    eax
        call    SetWindowText

        xor     eax,eax

        ret     16

on_right_mouse_up:
        mov     eax,[esp+4+0]    ; pobranie hwnd przed zmianą ESP
        push    offset text_rup
        push    eax
```

```
        call    SetWindowText

        xor     eax,eax
        ret     16

on_mouse_move:
        mov     eax,[esp+4+0]    ; pobranie hwnd przed zmianą ESP
        push    offset text_mmove
        push    eax
        call    SetWindowText

        xor     eax,eax
        ret     16

        end     _start
```

Wprowadzenie do grafiki

Ostatecznie, cała grafika pod Windows jest wykonywana poprzez urządzenie kontekstowe , lub DC. DC jest informacją o środowisku rysowania. Obejmuje to odniesienie do powierzchni rysowania (nazywanej "urządzeniem"), pewnych powiązanych obiektów GDI (narzędzia rysowania), i kilku trybów rysowania. Aktualnie są trzy typy urządzeń: monitor, urządzenie kopiujące (drukarka lub plotter), i urządzenie wirtualne (bitmapa w pamięci).

Wyświetlające urządzenie kontekstowe (monitor DC)

DC używane z monitorem jest również znane jako wyświetlające urządzenie kontekstowe. Pozwala ci ono rysować obrazy na monitorze. Każdy wyświetlacz DC przechowuje stan obszaru rysowania na monitorze. Każdy DC definiuje obszar przycinania obrazu--wszystko" rysowane" poza tym obszarem jest ignorowane , I skutecznie usuwane (lub "wycinane"). Powszechnie używanym DC przez programistów Windows jest ten, który pozwala rysować w obszarze klienta okna . Kiedy potrzeba uchwyt do tego DC jest normalnie odzyskiwany z programu obsługi komunikatu po jego wywołaniu , a zwolniony przed zakończeniem programu obsługi komunikatu. Jeśli komunikatem jest WM_PAINT, są używane dwie funkcje BeginPaint i EndPaint. W przeciwnym razie, dwie funkcje GetDC i ReleaseDC.

Wspólne i prywatne DC

Dwa podstawowe typy kontekstowych urządzeń wyświetlających to wspólne i prywatne. (Trzeci typ, klasa, jest przestarzały.) Kiedy odzyskujemy uchwyt DC , DC będzie prywatny jeśli klasa okna była zarejestrowana w stylu klasy CS_OWNDC , w przeciwnym razie będzie wspólny. Uchwyt DC jest odzyskiwany poprzez wywołanie GetDC. (Kiedy odpowiadamy na WM_PAINT, musimy wywołać BeginPaint.) Jeśli DC jest wspólny, odnosi się do nowego zainicjalizowanego DC z domyślnie ustawionymi narzędziami do rysowania i trybami. To DC powinno być zwolnione przez wywołanie ReleaseDC przed wyjściem z procedury okna. (Kiedy odpowiadamy na WM_PAINT, musimy wywołać EndPaint.) Kiedy jest tworzony prywatny DC kiedy jest tworzone okno, nie ma potrzeby wywoływać ReleaseDC. Chyba ,że wspólne DC, nie jest ponownie zainicjalizowane po odzyskaniu W wyniku tego, unikniesz konieczności resetowania wszystkich swoich narzędzi malarskich i trybów przy każdym komunikacie żądającym grafiki .Jednakże jeśli wyświetlasz wiele okien naprzemiennie , biorąc pod uwagę każde okno , prywatne DC będzie zużywało dużo pamięci.

Przykładowy program

Przykładowy program windraw1.asm, rysuje okrąg, kiedy klikniesz w obszarze programu klienta “głównego: okna.

Tworzenie, pętla i wysłanie

```
        .386
        .model   flat

        ; Jeśli używasz TLINK32,nie dołączaj vclib.inc
        include vclib.inc     ;linkowana nazwa  .lib Microsoft VC++

        include win32hst.inc  ;stałe, struktury i nazwy wejścia
        .data
        align   4
wcx     dd      size WNDCLASSEX            ; cbSize
        dd      CS_VREDRAW or CS_HREDRAW   ; style
        dd      WndProc                    ; lpfnWndProc
        dd      0,0                        ; cbClsExtra, cbWndExtra
        dd      0                          ; hInstance
        dd      0                          ; hIcon
        dd      0                          ; hCursor
        dd      COLOR_WINDOW+1             ; hbrBackground
        dd      0                          ; lpszMenuName
        dd      wndclsname                 ; lpszClassName
        dd      0                          ; hIconSm

wndclsname db 'windraw',0

        .code

        public  _start
        extrn     GetModuleHandle:near
        extrn   LoadIcon:near,LoadCursor:near extrn
        RegisterClassEx:near

_start:
        push    large  0        ; wskaźnik ciągu NULL
        call    GetModuleHandle  ;pobranie HINSTANCE/HMODULE pliku EXE
        mov     [wcx.wcx_hInstance],eax

        push    large IDI_WINLOGO
        push    large  0        ; hInstance,  0  = stock icon
        call    LoadIcon
        mov     [wcx.wcx_hIcon],eax

        push    large IDC_ARROW
        push    large  0        ; hInstance,  0  = stock cursor
        call    LoadCursor
        mov     [wcx.wcx_hCursor],eax

        push    offset wcx
        call    RegisterClassEx

        .data
        align   4
cwargs   dd   0                  ; dwExStyle
         dd     wndclsname       ; lpszClass
         dd     wnd_title        ; lpszName
         dd     WS_VISIBLE or WS_OVERLAPPED or WS_SYSMENU or WS_THICKFRAME  \
                or WS_MINIMIZEBOX or WS_MAXIMIZEBOX    ; style
         dd   100                ; x
         dd   100                ; y
         dd   200                ; cx  (szerokość)
         dd   200                ; cy  (wysokość)
         dd   0                  ; hwndParent
         dd   0                  ; hMenu
```

```
         dd   0                   ; hInstance
         dd   0                   ; lpCreateParams

msgbuf MSG      <>

wnd_title db 'Rysowanie okręgu',0

        .code

        extrn     CreateWindowEx:near
        extrn        GetMessage:near,DispatchMessage:near
        extrn     ExitProcess:near

        sub     esp,48             ;alokacja listy argumentów
        mov     esi,offset cwargs  ;ustawienie przesuwanego bloku Ÿródła
        mov     edi,esp       ; ustawienie przesuwanego bloku przeznaczenia
        mov     ecx,12         ; liczba argumentów
        rep movsd
        mov     eax,[wcx.wcx_hInstance]
        mov     [esp+40],eax       ;ustawienie argumentu hInstance na stosie
        call    CreateWindowEx

msg_loop:
        push    large  0         ; uMsgFilterMax
        push    large  0         ; uMsgFilterMin
        push    large  0         ; hWnd  (filtr),  0  = wszystkie okna
        push    offset msgbuf    ; lpMsg
        call    GetMessage       ; zwraca FALSE jeœli WM_QUIT
        or      eax,eax
        jz      end_loop

        push    offset msgbuf
        call    DispatchMessage

        jmp     msg_loop

end_loop:
        push    large  0  ;  (błąd) kod zakończenia
        call    ExitProcess
;
;  Procedura okna...gdzie jest przetwarzany komunikat dla jednej klasy okna
;
;
; Parametrami są hWnd, komunikat, wParam, lParam. ;
hWnd jest oknem odbierającym ten komunikat. ;    komunikat
jest ID komunikatu.
;       wParam i lParam zależą od ID komunikatu.
;
; Musisz zachować EBX, ESI, i EDI.
;
        extrn      DefWindowProc:near,PostQuitMessage:near

        .code
WndProc:
        mov     eax,[esp+4+4]    ; ID komunikatu
        cmp     eax,WM_DESTROY   ;start destrukcji okna
        je      on_destroy
        cmp     eax,WM_LBUTTONDOWN       ;kliknięto lewy przycisk myszy
        je      on_left_mouse_button_down
        jmp     DefWindowProc    ;przetwarzanie innych komunikatów
;
```

```
; Proces WM_DESTROY. Wysyłany po tym jak okno usunięto z ekranu , ale ; zanim
zacznie się destrukcja.
;
; Zwracane zero jeśli przetworzono.
;
; Musisz zachować EBX, ESI, i EDI.
;
on_destroy:
        push    large  0
        call    PostQuitMessage

        xor     eax,eax
        ret     16
```

Rysowanie

Tu jest serce całego przykładu - grafika pod kliknięciem przycisku myszy. Pokazuje pobieranie (GetDC) I zwolnienie (ReleaseDC) wspólnego DC. Jeśli struktura WNDCLASSEX "wc" miała CS_OWNDC w polu styl, GetDC pobierałby uchwyt prywatnego DC, a ReleaseDC nie robiłby nic - innym słowy, wywołanie ReleaseDC może być wyeliminowane. Rysująca okrąg, funkcja Ellipse jest wywoływana ze współrzędnymi zamkniętego prostokąta .

```
; Process WM_LBUTTONDOWN.     Lewy przycisk myszy naciśnięto.
;
; wParam is mouse flags.
; lParam to y:x   (współrzędne klienta).
;
; Zwrócono zero jeśli przetworzono.
;
; Musisz zachować EBX, ESI, i EDI.
;
        .data
        align   4
wndDC     dd  0          ; uchwyt DC handle  (hDC) okna obszaru klienta

        .code

        extrn   GetDC:near,ReleaseDC:near
        extrn Ellipse:near

on_left_mouse_button_down:
        push    dword ptr  [esp+4+0]     ; hWnd
        call    GetDC
        mov     [wndDC],eax              ; hDC dla okna

        push    large  150               ; y, niższy prawy
        push    large  150               ; x
        push    large  50                ; y, górny lewy
        push    large  50                ; x
        push    [wndDC]                  ; hDC
        call    Ellipse

        push    [wndDC]
        push    dword ptr  [esp+4+0]     ; hWnd
        call    ReleaseDC

        xor     eax,eax
        ret     16

        end     _start
```

Odœwieżanie z WM_PAINT

Jedną ze słabości windraw1 jest to ,że rysowany obraz może stać się złamany. Jeśli przykryjesz część okręgu innym oknem, a potem przesuniesz to okno gdzie indziej, zgubisz część okręgu. Windows oczekuje ,że ponownie narysujesz tą część, i sygnalizuje ci to przez komunikat WM_PAINT .

WM_PAINT, BeginPaint, i EndPaint

Komunikat WM_PAINT jest wysyłany kiedy obszar klienta okna wymaga ponownego przemalowania. Jednym z takich przypadków jest kiedy część obszaru klienta została "odkryta" przez zamknięcie ,przesunięcie lub zmianę rozmiaru okna nakładkowego. Ponieważ wyświetlacz DC nie przechowuje obrazu , który był tam oczekiwany, odkryta część musi być ponownie przerysowana przez aplikację. Kiedy obsługujemy WM_PAINT, zawsze wywołujemy BeginPaint i EndPaint, nawet jeśli nie robimy grafiki. Funkcje te obsługują specjalne potrzeby komunikatu WM_PAINT. Jeśli nie obsłużysz tego komunikatu, DefWindowProc będzie wywoływał te funkcje dla ciebie. BeginPaint zwraca uchwyt DC , a EndPaint zwalnia go. Więc nie jest konieczne wywoływanie GetDC i ReleaseDC.

Program przykładowy

Program przykładowy windraw2.asm, rysuje okrąg w obszarze klienta "głównego" okna i utrzymuje go. Większość kodu jest taka sama jak dla windraw1.asm, więc pokażemy tylko różnice.

Wysłanie komunikatu

Odpowiadamy na WM_PAINT, zamiast WM_LBUTTONDOWN.

```
; Procedura okna...gdzie jest przetwarzany komunikat dla jednej klasy okna
;
;
; Parametry to hWnd, komunikat, wParam, lParam.
;      hWnd jest oknem odbierającym komunikat.
;      komunikat jest ID komunikatu ID.
;      wParam i lParam zależą od ID komunikatu.
;
; Musisz zachować EBX, ESI, i EDI.
;
        extrn      DefWindowProc:near,PostQuitMessage:near

        .code
WndProc:
        mov     eax,[esp+4+4]    ; ID komunikatu
        cmp     eax,WM_DESTROY   ; start destrukcji okna
        je      on_destroy
        cmp     eax,WM_PAINT     ; polecenie malowania
        je      on_paint
        jmp     DefWindowProc    ; przetwarzanie innych komunikatów
```

Malowanie

Stary program obsługi przycisku myszki staje się teraz nowym programem obsługi WM_PAINT . Z powodu komunikatu, musimy użyć BeginPaint i EndPaint. Definiujemy PAINTSTRUCT która jest konieczna tym dwóm funkcjom
Teraz , kiedy okno jest odkryte, przywrócimy go. Przywrócimy go również po zmianie rozmiaru okna

```
; Process WM_PAINT.     Niektóre obszary okna wymagające przemalowania.
;
; wParam is  ???.
```

```
; lParam is  ???.
;
; Zwraca zero jeśli przetworzone.
;
; Musisz zachować EBX, ESI, i EDI.
;
        .data
        align   4
wndDC      dd  0         ; Uchwyt DC  (hDC) okna obszaru klienta

ps      PAINTSTRUCT     <>      ; to zazwyczaj ląduje na stosie

        .code

        extrn   BeginPaint:near,EndPaint:near
        extrn Ellipse:near

on_paint:
        mov     eax,[esp+4+0]           ; hWnd
        push    offset ps               ; lpPaint
        push    eax
        call    BeginPaint
        mov     [wndDC],eax             ; hDC for window

        push    large  150              ; y, prawy dolny
        push    large  150              ; x
        push    large  50               ; y, lewy górny
        push    large  50               ; x
        push    [wndDC]                 ; hDC
        call    Ellipse

        mov     eax,[esp+4+0]           ; hWnd
        push    offset ps               ; lpPaint
        push    eax
        call    EndPaint

        xor     eax,eax
        ret     16
```

Wprowadzenie do kontrolek

Co to jest kontrolka? Dobre pytanie. Przed (VB), była pomyślana jako okno potomne stworzone z jednej z sześciu predefiniowanych klas okna. Tworząc okno z wbudowaną kontrolką dostarczamy "panelu kontrolnego", z kontrolkami działającymi jako kanały wejścia i wyjścia dla aplikacji. Obecnie, są kontrolki niestandardowe - stworzone przez programistów nie-MS. Microsoft dodał uniwersalne kontrolki do Win95. A dodając do tego , VB (Visual Basic) nadał inne znaczenie kontrolkom - jest to moduł oprogramowania który dostarcza specjalnej funkcji „sterującej" (np., zegar). Będziemy ignorować znaczenie kontrolek dla VB

Rejestrowanie klas kontrolek

Wywołanie przez Windows kontrolki jest implementowane jako okno, więc jest klasa okna powiązana z każdym typem kontrolki. Mamy sześć predefiniowanych klas kontrolek przez Windows . Te klasy okien są rejestrowane zanim aplikacja się uruchomi. Są to: BUTTON, COMBOBOX, EDIT, LISTBOX, SCROLLBAR, i STATIC. Te wspólne kontrolki są rejestrowane przez wywoływani InitCommonControls. Jeśli używasz kontrolek niestandardowych, czy ty sam czy ktoś inny, musisz zarejestrować swoją klasę okna przez wywołanie RegisterWindowEx.

Tworzenie okien potomnych i kontrolek

Okna potomne są tworzone przez wywołanie CreateWindowEx WS_CHILD zawartej w argumencie styl. Są zazwyczaj tworzone jako okna widoczne w trybie WS_VISIBLE. Wyższe 16-bitów argumentu styl jest używane do kodowania ogólnych zadań opcji WS_ styl. Niższe 16 bitów argumentu styl jest używanych do kodowania stylu opcji określonej klasy okna. Współrzędne x-y muszą być współrzędnymi klienta, gdzie (0, 0) jest to górny , lewy róg obszaru klienta w oknie nadrzędnym. A, oczywiście, argument hParent będzie oknem nadrzędnym. Argument hMenu nie jest używany do tworzenia menu, zamiast tego jest używany do przydzielenia ID kontrolki do okna potomnego. Ponieważ ID kontrolki dostarcza konwencjonalnego odniesienia do okna potomnego, jest dobrą praktyką nadawanie unikalnych numerów oknom potomnym okna nadrzędnego. Kiedy okno jest tworzone, komunikat WM_CREATE jest wysyłany do tej procedury okna po jego stworzeniu.. Program obsługi dla tego komunikatu jest dogodnie umieszczony dla tworzenia okien potomnych dla nowo stworzonych okien. Kontrolki są zazwyczaj tworzonej jako widzialne okna potomne (WS_CHILD z WS_VISIBLE).

Żądania i zapytania kontrolek, SendMessage

Kiedy chcemy zmodyfikować lub zapytać kontrolkę, wysyłamy komunikat do niej poprzez wywołanie SendMessage. Każda kontrolka definiuje swój własny zbiór komunikatów do tego celu. Większość czasu , SendMessage działa jak wywołany podprogram. W tym czasie nie musimy się martwić komplikacjami wielowątkowości.

Odpowiedzi kontrolek

Większość predefiniowanych kontrolek (i wspólnych kontrolek) ma zdolność do wysyłania określonego zbioru zawiadomień do swojego okna rodzicielskiego w postaci komunikatu WM_COMMAND . Kontrolki będą wysyłały WM_COMMAND ze swojego uchwytu okna, swojego ID kontrolki i swojego kodu powiadomienia. Kontrolki "suwaka" są wyjątkami. Wysyłają one komunikat WM_VSCROLL lub WM_HSCROLL w zależności od tego czy suwak ma orientację poziomą czy pionową. Komunikaty te były pierwotnie zaprojektowane tylko dla kontrolki paska przewijania (klasa SCROLLBAR). Kontrolki niestandardowe mają opcję powiadamiania okna rodzicielskiego komunikatem WM_NOTIFY. Niektóre z nich również generują komunikat WM_NOTIFY. wParam jest ID kontrolki (takim samym jak poniższy nmh_idFrom), a lParam jest adresem NMHEADER (nagłówek komunikatu powiadomienia). Informacja określonego powiadomienia następuje bezpośrednio po NMHEADER.

```
NMHEADER STRUC
nmh_hwndFrom DD  ?    ; uchwyt kontrolki która wysyła WM_NOTIFY
nmh_idFrom     DD  ?     ; ID kontrolki, która wysyła WM_NOTIFY
nmh_code      DD  ?    ; NM lub kod zdefiniowany przez użytkownika
NMHEADER ENDS
```

Predefiniowany kod NM ma wartości ujemne i obejmuje: NM_OUTOFMEMORY, NM_CLICK, NM_DBLCLICK, NM_RETURN, NM_RCLICK, NM_RDBLCLK, NM_SETFOCUS, i NM_KILLFOCUS.

Prosty edytor notatnikowy--WINPAD.ASM

Poniższy kod implementuje edytor tekstowy, nie przetwarzający pliku - notatnik. Wszystko co musimy zrobić to stworzyć kontrolkę EDIT . Wszystko inne, tzn. Przewijanie i menu kontekstowe jest obsługiwane przez kontrolkę EDIT.

Rejestracja

Najpierw rejestrujemy klasę okna dla okna głównego.

```
        .386
        .model    flat

        ;Jeśli używasz TLINK32,nie włączaj    vclib.inc
        include vclib.inc     ;linkowane nazwy  .lib Microsoft VC++
```

```
        include win32hst.inc  ;stałe, struktury, i nazwy wejść

        .data
        align   4
wcx     dd      size WNDCLASSEX           ; cbSize
        dd      CS_VREDRAW or CS_HREDRAW  ; style
        dd      WndProc                   ; lpfnWndProc
        dd      0,0                       ; cbClsExtra, cbWndExtra
        dd      0                         ; hInstance
        dd      0                         ; hIcon
        dd      0                         ; hCursor
        dd      COLOR_WINDOW+1            ; hbrBackground
        dd      0                         ; lpszMenuName
        dd      wndclsname                ; lpszClassName
        dd      0                         ; hIconSm

wndclsname db 'winpad',0

        .code

        public  _start
        extrn     GetModuleHandle:near
        extrn   LoadIcon:near,LoadCursor:near extrn
        RegisterClassEx:near

_start:
        push    large  0        ;wskaźnik ciągu NULL
        call    GetModuleHandle  ;pobranie HINSTANCE/HMODULE pliku EXE
        mov     [wcx.wcx_hInstance],eax

        push    large IDI_WINLOGO
        push    large  0        ; hInstance,  0  = stock icon
        call    LoadIcon
        mov     [wcx.wcx_hIcon],eax

        push    large IDC_ARROW
        push    large  0        ; hInstance,  0  = stock cursor
        call    LoadCursor
        mov     [wcx.wcx_hCursor],eax

        push    offset wcx
        call    RegisterClassEx
```

Tworzenie i pętla komunikatów

W przeciwieństwie do poprzednich programów, eliminujemy opcję rozmiaru (żadnych ramek, minimalizacji lub maksymalizacji), i dodajemy TranslateMessage do naszej pętli komunikatów. Kontrolka EDIT nie działa jeśli nie wywołamy TranslateMessage.

```
        .data
        align   4
cwargs    dd  0                 ; dwExStyle
        dd      wndclsname      ; lpszClass
        dd      wnd_title       ; lpszName
        dd      WS_VISIBLE or WS_OVERLAPPED or WS_SYSMENU  ; style
        dd   50                 ; x
        dd   50                 ; y
        dd   400                ; cx  (szerokość)
        dd   400                ; cy  (wysokość)
        dd   0                  ; hwndParent
        dd   0                  ; hMenu
```

```
        dd   0                  ; hInstance
        dd   0                  ; lpCreateParams

msgbuf MSG      <>

wnd_title db 'Scratch Pad Editor',0

        .code

        extrn     CreateWindowEx:near
        extrn   GetMessage:near,TranslateMessage:near,DispatchMessage:near extrn
        ExitProcess:near

        sub     esp,48            ;alokacja listy argumentów
        mov     esi,offset cwargs  ;ustawienie przesuwanego bloku Ÿródłowego
        mov     edi,esp      ; ustawienie przesuwanego bloku    przeznaczenia
        mov     ecx,12            ; liczba argumentów
        rep movsd
        mov     eax,[wcx.wcx_hInstance]
        mov     [esp+40],eax      ;ustawienie argumentu hInstance na stosie
        call    CreateWindowEx

msg_loop:
        push    large  0        ; uMsgFilterMax
        push    large  0        ; uMsgFilterMin
        push    large  0        ; hWnd  (filter),  0  = all windows
        push    offset msgbuf   ; lpMsg
        call    GetMessage      ; zwraca FALSE jeœli WM_QUIT
        or      eax,eax
        jz      end_loop

        push    offset msgbuf
        call    TranslateMessage

        push    offset msgbuf
        call    DispatchMessage

        jmp     msg_loop

end_loop:
        push    large  0        ;  (błąd) kod powrotu
        call    ExitProcess
```

Wysyłanie komunikatu

Przetwarzamy WM_CREATE, komunikat, który jest wysyłany do naszego okna kiedy tworzymy go po raz pierwszy.

```
        extrn     DefWindowProc:near,PostQuitMessage:near

        .code
WndProc:
        mov     eax,[esp+4+4]   ; ID komunikatu
        cmp     eax,WM_CREATE   ; tworzenie okna
        je      on_create
        cmp     eax,WM_DESTROY  ;start destrukcji okna
        je      on_destroy
        jmp     DefWindowProc   ;przetwarzanie innych komunikatów

on_destroy:
        push    large  0
```

```
        call    PostQuitMessage

        xor     eax,eax
        ret     16
```

Tworzenie kontrolki

Zilustrujemy jak stworzyć kontrolkę kiedy okno nadrzędne zostało stworzone. GetClientRect pobiera współrzędne obszaru klienta naszego głównego okna . W ten sposób nie musimy wcześniej obliczać rozmiaru kontrolki EDIT. Ponieważ kilka z argumentów CreateWindowEx nie jest stałymi wcześniej obliczonymi możemy woleć odłożyć argumenty w tym czasie

```
        .data
        align   4
wndeditargs     dd      0               ; dwExStyle
                dd      editclsname     ; lpszClass
                dd      0               ; lpszName
                dd       WS_CHILD+WS_HSCROLL  or  WS_VSCROLL  or  WS_VISIBLE  or
\
                        ES_AUTOHSCROLL or ES_AUTOVSCROLL or ES_MULTILINE
                dd      0               ; x
                dd      0               ; y
                dd      0               ; cx  (szerokoœć)
                dd      0               ; cy  (wysokoœć)
                dd      0               ; hwndParent
                dd      0               ; hMenu
                dd      0               ; hInstance
                dd      0               ; lpCreateParams

client_rect     RECT    <>

editclsname db 'edit',0

        .code

        extrn     GetClientRect:near

on_create:
        mov     eax,[esp+4+0]    ;przechwyt hwnd zanim zmieni się ESP

        push    offset client_rect
        push    eax             ; hwnd
        call    GetClientRect

        mov     eax,[esp+4+0]    ; przechwyt hwnd zanim zmieni się ESP

        push    esi             ; musimy zachować ESI i EDI
        push    edi

        sub     esp,48          ; alokacja argumentów
        mov        esi,offset  wndeditargs   ;ustawienie  przesuwanego  bloku
źródła
        mov     edi,esp
        mov     ecx,12
        rep movsd
        mov     [esp+32],eax    ;uczynienie okna głównego rodzicem "edit"
        mov     eax,[wcx.wcx_hInstance]
        mov     [esp+40],eax    ;ustawienie argumentu hInstance na stosie
        mov     eax,client_rect.rc_left
        mov     [esp+16],eax    ;ustawienie argumentu x
        mov     eax,client_rect.rc_top
```

```
        mov     [esp+20],eax    ; ustawienie    argumentu y
        mov     eax,client_rect.rc_right
        inc     eax
        sub     eax,client_rect.rc_left
        mov     [esp+24],eax    ; ustawienie argumentu cx(szerokość)
        mov     eax,client_rect.rc_bottom
        inc     eax
        sub     eax,client_rect.rc_top
        mov     [esp+28],eax    ;ustawienie argumentu cy  (szerokość)
        call    CreateWindowEx

        pop     edi             ; przywrócenie ESI i EDI
        pop     esi

        xor     eax,eax         ; powrót OK
        ret     16

        end     _start
```

Pętle konwencjonalne i wątków komunikatów

Konwencjonalna pętla komunikatu

Pierwszą pętlą komunikatu jaką pokażemy jest konwencjonalna pętla komunikatu. Może być sparametryzowana aby wyglądała jak poniżej:

```
    .data
msgbuf MSG   <>

    .code
    call    setup       ;  *** ustawienia zdefiniowane przez użytkownika  ***
msg_loop:
    push    large  0       ; uMsgFilterMax
    push    large  0       ; uMsgFilterMin
    push    large  0       ; hWnd  (filtr),  0  = wszystkie okna
    push    offset msgbuf  ; lpMsg
    call    GetMessage     ; zwraca FALSE jeœli WM_QUIT
    or      eax,eax
    jz      end_msg_loop

    call      dispatch_message  ;***wysyłanie  zdefiniowane  przez  użytkownika

    jmp     msg_loop

end_msg_loop:
    call    cleanup      ;  ***czyszczenie zdefiniowane przez użytkownika***
```

Jeśli nie ma komunikatów oczekujących w kolejce komunikatów, GetMessage zwraca sterowanie do Windows. To zwalnia CPU, pozwalając mu na używanie innych programów. Alternatywnie, możemy powiedzieć , że GetMessage daje nam poślizg czasowy programu, maksymalną ilość czasu daną programowi przed wywłaszczeniem. Wszystko to pozwala całemu systemowi być bardziej wrażliwym niż w innym przypadku

Kod komunikatu wysyłania dla podstawowej aplikacji GUI:

```
dispatch_message:
    push    offset msgbuf
    call    TranslateMessage
    push    offset msgbuf
    call    DispatchMessage
    ret
```

Wywołując TranslateMessage sprawdzamy komunikaty klawiatury i pewne kombinacje komunikatów, dodajemy komunikat WM_CHAR do kolejki komunikatów. (jaka niespodzianka -- WM_CHAR nie jest komunikatem klawiatury!). Pokazaliśmy kod wysyłania jako podprogram, ale ta krótka sekwencja jest zazwyczaj wpleciona tak, że eliminuje koszt wywołania - powrotu

Pętla komunikatów wątków

Każde okno uruchamia swoją procedurę okna z jakimś wątkiem. Jednakże, wątek może działać bez tworzenia jakiegoś okna. Możesz jeszcze przekazać komunikaty do wątku przez PostThreadMessage. Obsługując odbieranie komunikatów, używamy konwencjonalnej pętli i zmieniamy kod wysyłania do bezpośredniego obsługiwania komunikatów.

```
dispatch_message:
    mov     eax,msgbuf.msg_message
    cmp     eax,WM_APP+0
    je      on_app_message0
    cmp     eax,WM_APP+1
    je      on_app_message1
    ret
```

Zachłanne pętle komunikatów

Niektórzy ludzi (w znacznej mierze programiści gier) chcą wykonać procesy drugoplanowe kiedy nie przetwarzają żadnych komunikatów. Dla nich, pokażę "zachłanną" pętlę komunikatów. Zamiast podawania odcinka czasu , możemy zamienić GetMessage z PeekMessage.

```
    .data
msgbuf MSG     <>

    .code
    call    setup
msg_loop:
    push    large PM_REMOVE
    push    large  0       ; uMsgFilterMax
    push    large  0       ; uMsgFilterMin
    push    large  0       ; hWnd  (filtr),  0  = wszystkie okna
    push    offset msgbuf  ; lpMsg
    call    PeekMessage    ; zwraca nie zero  (prawda) jeśli jest komunikat
    or      eax,eax
    jnz     got_msg

    call    do_background_process
    jmp     msg_loop

got_msg:
    cmp     msgbuf.msg_message,WM_QUIT
    je      end_msg_loop

    call    dispatch_message
    jmp     msg_loop

end_msg_loop:
    call    cleanup
```

Powyższy kod jest ekstremalnie zachłanny. Będzie używał czasu jeśli nie ma żadnych procesów drugoplanowych do wykonania. Będzie również używał czasu kiedy nie będzie wyświetlane żaden okno .Z powodu wywłaszczenia, nie zabezpieczymy innych programów przed uruchomieniem. Ale ponieważ używamy całego odcinka czasu, spowolnimy je.

Mniej zachłanna wersja, poniżej, używa flag dla sygnalizowania aktywności, stanu zachłanności. Flaga może być przełączana przez program obsługi komunikatu

```
ACTIVE_FLAG equ  1

    .data
flags    dd   ACTIVE_FLAG
msgbuf MSG    <>

    .code
    call     setup
msg_loop:
    test     flags,ACTIVE_FLAG  ;zy jest aktywna?
    jz       get_next_msg   ; nie  ,nie bądŸ achłanny

    push     large PM_NOREMOVE  ;  *** *POZOSTAWIENIE KOMUNIKATU W KOLEJCE*****
    push     large  0       ; uMsgFilterMax
    push     large  0       ; uMsgFilterMin
    push     large  0       ; hWnd  (filtr),  0  = wszystkie okna
    push     offset msgbuf  ; lpMsg
    call        PeekMessage  ;zwraca  wartość  niezerową  (prawda)  jeśli  jest
komunikat
    or       eax,eax
    jnz      get_next_msg

    call     do_background_process
    jmp      msg_loop

get_next_msg:
    push     large  0       ; uMsgFilterMax
    push     large  0       ; uMsgFilterMin
    push     large  0       ; hWnd  (filter),  0  = all windows
    push     offset msgbuf  ; lpMsg
    call     GetMessage     ; zwraca FALSE jeœli komuikat to WM_QUIT
    or       eax,eax
    jz       end_msg_loop

    call     dispatch_message
    jmp      msg_loop

end_msg_loop:
    call     cleanup
```

Kontrolka BUTTON

Kontrolka Button jest używana do wyzwalania lub przełączania. Przełączanie może być 2-stopniowe lub 3-stopniowe

Typ i styl Button

Typy Button :

BS_PUSHBUTTON
BS_DEFPUSHBUTTON

BS_CHECKBOX
BS_AUTOCHECKBOX
BS_STATE
BS_AUTOSTATE

BS_RADIOBUTTON
BS_AUTORADIOBUTTON

BS_GROUPBOX

BS_USERBUTTON
BS_OWNERDRAW

Style Button :

BS_TEXT
BS_LEFTTEXT, BS_RIGHTBUTTON

BS_ICON
BS_BITMAP

BS_LEFT
BS_CENTER
BS_RIGHT
BS_TOP
BS_VCENTER
BS_BOTTOM

BS_PUSHLIKE
BS_MULTILINE
BS_FLAT

BS_NOTIFY

Komunikaty Button

Jest kilka API dla buttonów i kilka ogólnych API takich jak SetFocus, EnableWindow, i SetWindowText.

Komunikaty Button:

BM_GETCHECK
BM_SETCHECK ; wParam = nowe sprawdzanie stanu
BM_GETSTATE
BM_SETSTATE ; wParam = nowy stan odłożenia
BM_SETSTYLE ; wParam = nowy styl, lParam = przerysowanie
BM_CLICK ; symulacja kliknięcia
BM_GETIMAGE
BM_SETIMAGE ; lParam = nowy program obsługi obrazu

Stany Button :

BST_UNCHECKED
BST_CHECKED
BST_INDETERMINATE

; Stany Button maskowane dla BM_GETSTATE

; pobranie sprawdzenia stanu = 3h
BST_PUSHED ;pobranie stanu odłożenia / zaznaczenia
BST_FOCUS ; pobranie stanu ogniska

Powiadomienia Button

Buttony powiadamiają swoich rodziców przez wysłanie komunikatu WM_COMMAND do nich z kodem powiadomienia.

Kody powiadomienia Button:

BN_CLICKED
BN_PAINT
BN_HILITE, BN_PUSHED
BN_UNHILITE, BN_UNPUSHED
BN_DISABLE

BN_DOUBLECLICKED, BN_DBLCLK
BN_SETFOCUS
BN_KILLFOCUS

Pewne specjalizowane funkcje API button

Funkcje te były stworzone dla stosowania w okienkach dialogowych. Jednakże będą działały dla każdego okna, które ma kontrolkę button. Kiedy działamy z okienkami dialogowymi, argument Wnd jest okienkiem dialogowym.

DWORD IsDlgButtonChecked(HWND Wnd, int ButtonID)
Tak samo wysyłamy BM_GETSTATE. Ostatecznie używane kiedy button musi być skolejkowany przez program obsługi innej kontrolki.

void CheckDlgButton(HWND Wnd, int ButtonID, DWORD NewState)
Tak samo wysyłamy BM_SETSTATE. Ostatecznie używane kiedy button musi być zmieniony przez inny program obsługi kontrolki.

void CheckRadioButton(HWND Wnd, int FirstBtnID, int LastBtnID, int SelectBtnID)
To jest bardzo użyteczna funkcja. Czyni ,że kilka buttonów działających razem jako "wybrana jedna" kontrolka. Działające buttony mają ID kontrolki w zakresie od FirstBtnID do LastBtnID. Button z SelectBtnID jest jedynym pobierającym "zaznaczenie", i inne stające "nie zaznaczonymi”".

Kontrolka COMBOBOX

Pola kombi są polami listy, które są połączeniem albo z kontrolką statyczną albo kontrolką edycyjną. Pole listy może być opcjonalnie rozwijana jak menu.

Style pola Combo

Style pola combo:

CBS_SIMPLE
CBS_DROPDOWN
CBS_DROPDOWNLIST
CBS_OWNERDRAWFIXED
CBS_OWNERDRAWVARIABLE
CBS_AUTOHSCROLL
CBS_OEMCONVERT
CBS_SORT
CBS_HASSTRINGS
CBS_NOINTEGRALHEIGHT
CBS_DISABLENOSCROLL
CBS_UPPERCASE
CBS_LOWERCASE

Komunikaty pola combo

Wartości zwracane pola combo:

CB_OKAY
CB_ERR
CB_ERRSPACE

Komunikaty pola combo:

CB_GETEDITSEL
CB_LIMITTEXT
CB_SETEDITSEL
CB_ADDSTRING
CB_DELETESTRING
CB_DIR
CB_GETCOUNT
CB_GETCURSEL
CB_GETLBTEXT
CB_GETLBTEXTLEN
CB_INSERTSTRING
CB_RESETCONTENT
CB_FINDSTRING
CB_SELECTSTRING
CB_SETCURSEL
CB_SHOWDROPDOWN
CB_GETITEMDATA
CB_SETITEMDATA
CB_GETDROPPEDCONTROLRECT
CB_SETITEMHEIGHT
CB_GETITEMHEIGHT
CB_SETEXTENDEDUI
CB_GETEXTENDEDUI
CB_GETDROPPEDSTATE
CB_FINDSTRINGEXACT
CB_SETLOCALE
CB_GETLOCALE

CB_GETTOPINDEX
CB_SETTOPINDEX
CB_GETHORIZONTALEXTENT
CB_SETHORIZONTALEXTENT
CB_GETDROPPEDWIDTH
CB_SETDROPPEDWIDTH
CB_INITSTORAGE

Powiadomienia pola combo

Pole combo powiadamia swojego rodzica poprzez wysłanie komunikatu WM_COMMAND do niego z kodem powiadomienia.

Kod powiadomienia pola combo:

CBN_ERRSPACE

CBN_SELCHANGE
CBN_DBLCLK
CBN_SETFOCUS
CBN_KILLFOCUS
CBN_EDITCHANGE
CBN_EDITUPDATE
CBN_DROPDOWN
CBN_CLOSEUP
CBN_SELENDOK
CBN_SELENDCANCEL

Kontrolka EDIT

Kontrolki edycji są używane do wprowadzania danych tekstowych. Mogą również funkcjonować jako mini edytory tekstu. Wersja Win95 tej kontrolki ma ograniczenia ponieważ jest implementowana w kodzie 16

bitowym. Możesz życzyć sobie użyć kontrolki Rich Edit do przejścia tych ograniczeń lub uzyskać więcej cech edycyjnych

Styl kontrolki Edit

Style kontrolki Edit :

ES_LEFT
ES_CENTER
ES_RIGHT
ES_MULTILINE
ES_UPPERCASE
ES_LOWERCASE
ES_PASSWORD
ES_AUTOVSCROLL
ES_AUTOHSCROLL
ES_NOHIDESEL
ES_OEMCONVERT
ES_READONLY
ES_WANTRETURN
ES_NUMBER

Komunikaty kontrolki Edit

Komunikaty kontrolki Edit:
;
; Parametry okna Edit
;EM_SETMARGIN
EC_LEFTMARGIN
EC_RIGHTMARGIN
EC_USEFONTINFO
;
; Komunikaty okna Edit
; Niektóre z nich sąą współdzielone z oknem Rich Edit
;
EM_GETSEL
EM_SETSEL
EM_GETRECT
EM_SETRECT
EM_SETRECTNP
EM_SCROLL
EM_LINESCROLL
EM_SCROLLCARET
EM_GETMODIFY
EM_SETMODIFY
EM_GETLINECOUNT
EM_LINEINDEX
EM_SETHANDLE
EM_GETHANDLE
EM_GETTHUMB
EM_LINELENGTH
EM_REPLACESEL
EM_GETLINE
EM_LIMITTEXT
EM_CANUNDO
EM_UNDO
EM_FMTLINES
EM_LINEFROMCHAR
EM_SETTABSTOPS

EM_SETPASSWORDCHAR
EM_EMPTYUNDOBUFFER
EM_GETFIRSTVISIBLELINE
EM_SETREADONLY
EM_SETWORDBREAKPROC
EM_GETWORDBREAKPROC
EM_GETPASSWORDCHAR
EM_SETMARGINS
EM_GETMARGINS
EM_SETLIMITTEXT equ EM_LIMITTEXT ; zmiana nazwy win40
EM_GETLIMITTEXT
EM_POSFROMCHAR
EM_CHARFROMPOS

Powiadomienia kontrolki Edit

Kontrolki Edit powiadamiają swoich rodziców przez wysłanie komunikatu WM_COMMAND do nich z kodem powiadomienia.

Kod powiadomienia kontrolki Edit :

EN_SETFOCUS
EN_KILLFOCUS
EN_CHANGE
EN_UPDATE
EN_ERRSPACE
EN_MAXTEXT
EN_HSCROLL
EN_VSCROLL

Kontrolka ListBox

Kontrolki pola listy dostarczają łatwego sposobu uzyskania wybranych list. Style

pola listy

Style pola listy:

LBS_NOTIFY
LBS_SORT
LBS_NOREDRAW
LBS_MULTIPLESEL
LBS_OWNERDRAWFIXED
LBS_OWNERDRAWVARIABLE
LBS_HASSTRINGS
LBS_USETABSTOPS
LBS_NOINTEGRALHEIGHT
LBS_MULTICOLUMN
LBS_WANTKEYBOARDINPUT
LBS_EXTENDEDSEL
LBS_DISABLENOSCROLL
LBS_NODATA
LBS_NOSEL

LBS_STANDARD equ LBS_NOTIFY or LBS_SORT or WS_VSCROLL or WS_BORDER

Komunikaty pola listy

Komunikaty pola listy:
;
; kod dialogu pola listy
;
LB_CTLCODE
;
; Zwracana wartość pola listy
;
LB_OKAY
LB_ERR
LB_ERRSPACE

LB_INSERTSTRING
LB_DELETESTRING
LB_SELITEMRANGEEX
LB_RESETCONTENT
LB_SETSEL
LB_SETCURSEL
LB_GETSEL
LB_GETCURSEL
LB_GETTEXT
LB_GETTEXTLEN
LB_GETCOUNT
LB_SELECTSTRING
LB_DIR
LB_GETTOPINDEX
LB_FINDSTRING
LB_GETSELCOUNT
LB_GETSELITEMS
LB_SETTABSTOPS
LB_GETHORIZONTALEXTENT
LB_SETHORIZONTALEXTENT
LB_SETCOLUMNWIDTH
LB_ADDFILE
LB_SETTOPINDEX
LB_GETITEMRECT
LB_GETITEMDATA
LB_SETITEMDATA
LB_SELITEMRANGE
LB_SETANCHORINDEX
LB_GETANCHORINDEX
LB_SETCARETINDEX
LB_GETCARETINDEX
LB_SETITEMHEIGHT
LB_GETITEMHEIGHT
LB_FINDSTRINGEXACT
LB_SETLOCALE
LB_GETLOCALE
LB_SETCOUNT
LB_INITSTORAGE
LB_ITEMFROMPOINT

Powiadomienia pola listy

Pola listy powiadamiają swoich rodziców poprzez wysłanie komunikatu WM_COMMAND do nich z kodem powiadomienia.

Kody powiadomienia pola listy:

LBN_ERRSPACE

LBN_SELCHANGE
LBN_DBLCLK
LBN_SELCANCEL
LBN_SETFOCUS
LBN_KILLFOCUS

Kontrolka SCROLLBAR

Paski przesuwania są podstawowymi kontrolkami suwaków. Same z siebie niczego nie przesuwają. Synchronizacja paska przewijania i obrazu przewijanego musi być oprogramowana. Klasa SCROLLBAR jest ograniczona do wyświetlania prostokątnych suwaków.

Typy style pasków przewijania

Typy pasków przewijania:

```
SBS_HORZ    equ 0000h
SBS_VERT    equ 0001h
```

Style pasku przewijania:

```
;
; Stałe pasku przewijania
;
SB_HORZ equ 0
SB_VERT equ 1
SB_CTL equ 2
SB_BOTH equ 3
```

```
SBS_TOPALIGN  equ 0002h
SBS_LEFTALIGN  equ 0002h
SBS_BOTTOMALIGN  equ 0004h
SBS_RIGHTALIGN  equ 0004h
SBS_SIZEBOXTOPLEFTALIGN equ 0002h
SBS_SIZEBOXBOTTOMRIGHTALIGN equ 0004h
SBS_SIZEBOX    equ 0008h
SBS_SIZEGRIP  equ 0010h
```

Komunikaty paska przewijania

Komunikaty pasku przewijania:

```
;
; Info flag  pasku przewijania
;
SIF_RANGE
SIF_PAGE
SIF_POS
SIF_DISABLENOSCROLL
SIF_TRACKPOS
SIF_ALL  equ  SIF_RANGE  or  SIF_PAGE  or  SIF_POS  or  SIF_TRACKPOS
;
; Komunikaty pasku przewijania
;
SBM_SETPOS
SBM_GETPOS
SBM_SETRANGE
SBM_SETRANGEREDRAW
SBM_GETRANGE
```

SBM_ENABLE_ARROWS
SBM_SETSCROLLINFO
SBM_GETSCROLLINFO

Powiadomienia paska przewijania

Pionowy pasek przewijania powiadamia swoich rodziców poprzez wysłanie komunikatu WM_VSCROLL z kodem powiadomienia. Poziomy pasek przewijania wysyła komunikat WM_HSCROLL. (W przeciwieństwie do innych standardowych kontrolek, te nie wysyłają komunikatu WM_COMMAND.)

Kod powiadamiania paska przewijania:
;
; Polecenia paska przewijania
;
SB_LINEUP
SB_LINELEFT
SB_LINEDOWN
SB_LINERIGHT
SB_PAGEUP
SB_PAGELEFT
SB_PAGEDOWN
SB_PAGERIGHT
SB_THUMBPOSITION
SB_THUMBTRACK
SB_TOP
SB_LEFT
SB_BOTTOM
SB_RIGHT
SB_ENDSCROLL

Kontrolka STATIC

Kontrolki statyczne są używane do umieszczania niemodyfikowalnych tekstów w oknie Style

statyczne

Style statyczne:

SS_LEFT
SS_CENTER
SS_RIGHT
SS_ICON
SS_BLACKRECT
SS_GRAYRECT
SS_WHITERECT
SS_BLACKFRAME
SS_GRAYFRAME
SS_WHITEFRAME
SS_USERITEM
SS_SIMPLE
SS_LEFTNOWORDWRAP
SS_BITMAP
SS_OWNERDRAW
SS_ENHMETAFILE
SS_ETCHEDHORZ
SS_ETCHEDVERT
SS_ETCHEDFRAME

SS_TYPEMASK
SS_NOPREFIX ; nie tłumaczy znaku "&"
SS_NOTIFY
SS_CENTERIMAGE
SS_RIGHTJUST
SS_REALSIZEIMAGE
SS_SUNKEN

Komunikaty statyczne

Komunikaty statyczne:

STM_SETICON
STM_GETICON
STM_SETIMAGE
STM_GETIMAGE

Powiadomienia statyczne

Kontrolki statyczne powiadamiają swoich rodziców poprzez wysłanie komunikatu WM_COMMAND do nich z kodem powiadomienia.

Kody powiadomienia statyczne:

STN_CLICKED
STN_DBLCLK
STN_ENABLE
STN_DISABLE

Wprowadzenie do menu

Win32 API dostarcza kilku funkcji do tworzenia dynamicznych menu , ale preferowanym sposobem jest stworzenie szablon menu i przechowanie ich jako zasobów w pliku EXE . Jest to zazwyczaj wykonywane przy pomocy edytora menu i kompilatorem zasobów , ale tu pokażę jak stworzyć szablon menu bez tych specjalnych narzędzi. Program to winmenu.asm.

Wstępny program

Najpierw zwykły start programu - zanim wywołamy CreateWindowEx.

```
        .386
        .model    flat

        ;Jeśli używasz TLINK32,nie włączaj vlib.inc
        include vclib.inc     ;łącze nazwy Microsoft VC++  .lib

        include win32hst.inc     ;stałe, struktury, i nazwy wejść

        .data
        align   4
wcx     dd      size WNDCLASSEX            ; cbSize
        dd      CS_VREDRAW or CS_HREDRAW   ; style
        dd      WndProc                    ; lpfnWndProc
        dd      0,0                        ; cbClsExtra, cbWndExtra
        dd      0                          ; hInstance
        dd      0                          ; hIcon
        dd      0                          ; hCursor
```

```
        dd      COLOR_WINDOW+1              ; hbrBackground
        dd      0                           ; lpszMenuName
        dd      wndclsname                  ; lpszClassName
        dd      0                           ; hIconSm

wndclsname db 'winmain',0

        .code

        public  _start
        extrn     GetModuleHandle:near
        extrn    LoadIcon:near,LoadCursor:near extrn
        RegisterClassEx:near

_start:
        push    large  0         ;wskaźnik do ciągu NULL
        call    GetModuleHandle  ;pobranie HINSTANCE/HMODULE pliku EXE
        mov     [wcx.wcx_hInstance],eax

        push    large IDI_WINLOGO
        push    large  0         ; hInstance,  0  = stock icon
        call    LoadIcon
        mov     [wcx.wcx_hIcon],eax

        push    large IDC_ARROW
        push    large  0         ; hInstance,  0  = stock cursor
        call    LoadCursor
        mov     [wcx.wcx_hCursor],eax

        push    offset wcx
        call    RegisterClassEx

        .data
        align   4
cwargs   dd   0                   ; dwExStyle
         dd      wndclsname       ; lpszClass
         dd      wnd_title        ; lpszName
         dd      WS_VISIBLE or WS_OVERLAPPED or WS_SYSMENU or WS_THICKFRAME  \
              or WS_MINIMIZEBOX or WS_MAXIMIZEBOX    ; style
         dd   100                 ; x
         dd   100                 ; y
         dd   400                 ; cx  (szerokość)
         dd   200                 ; cy  (wysokość)
         dd   0                   ; hwndParent
         dd   0                   ; hMenu
         dd   0                   ; hInstance
         dd   0                   ; lpCreateParams

msgbuf MSG       <>

wnd_title db 'Window with menu',0

        .code

        extrn     CreateWindowEx:near
        extrn        GetMessage:near,DispatchMessage:near
        extrn     ExitProcess:near

        sub     esp,48             ;alokowanie listy argumentów
        mov     esi,offset cwargs  ;ustawienie przesuwanego bloku Ÿródłowego
        mov     edi,esp         ;ustawienie przesuwanego bloku przeznaczenia
```

```
        mov     ecx,12              ;liczba argumentów
        rep movsd
        mov     eax,[wcx.wcx_hInstance]
        mov     [esp+40],eax        ;ustawienie argumentu hInstance na stosie
```

Dodawanie menu do okna

Nasz program będzie definiował szablon menu w pamięci. Tu tworzymy „obiekt" menu przez wywołanie LoadMenuIndirect, która zwraca uchwyt menu. Menu przyłączone do okna stworzymy przez ustawienie argumentu CreateWindowEx.
Możesz również stworzyć okno bez menu i później dodać go SetMenu. Ta funkcja API może również być użyta do zmiany menu. Menu mogą również być usuwane tą funkcją API przez użycie uchwytu menu NULL (zero)

Później menu może być zarządzane i manipulowane poprzez jego uchwyty.

```
        .code

        extrn      LoadMenuIndirect:near

        push    offset appMenuTemplate
        call    LoadMenuIndirect
        mov     [esp+36],eax    ;ustawienie argumentu hMenu na stosie
```

Pozostała część wstępna

Tu, kończymy wywoływanie CreateWindowEx, i dodajemy pętlę komunikatów.

```
        .code
        call    CreateWindowEx

msg_loop:
        push    large  0        ; uMsgFilterMax
        push    large  0        ; uMsgFilterMin
        push    large  0        ; hWnd  (filtr),  0  = wszystkie okna
        push    offset msgbuf   ; lpMsg
        call    GetMessage      ; zwraca FALSE jeœli    WM_QUIT
        or      eax,eax
        jz      end_loop

        push    offset msgbuf
        call    DispatchMessage

        jmp     msg_loop

end_loop:
        push    large  0  ;  (błąd) kod zakończenia
        call    ExitProcess
```

Definicja szablonu Menu

Pierwsze dwa szablony wejść są częścią nagłówka szablonu. Te dwa wejścia są numerem wersji szablonu menu, i offsetem do listy pozycji. Offset ten pozwala przeskoczyć dodatkową informację nagłówka. Wersja 0 jest pierwotnie dla NT I starszych szablonów Win16 .Będziemy używali nowej wersji 1 szablonu. Dodamy ID pomocy dla F1 do nagłówka. Zauważ ,że wejście offsetu w nagłówku jest ustawione do przeskoczenia tej dodatkowej informacji Nagłówek musi być wyrównany do granicy DWORD . Każda pozycja menu , która następuje po nagłówku musi być również wyrównany do granicy DWORD . Jest pięć pól (wejść) i sześć pól, które dostarczają tylko pozycji menu podręcznego. Są one, po kolei: typ pozycji, pozycja stanu, ID pozycji, informacja o strukturze menu , pozycja tekstu, i ID pomocy F1.
Pozycja typ pola zawiera poprawne kombinacje MFT_ stałe. Wskazuje rodzaj informacji menu do pokazania. Nasz przykład używa tylko typów MFT_STRING i MFT_SEPARATOR .Typ MFT_SEPARATOR tworzy

pasek pomiędzy pozycjami menu Help... i About...
Pole pozycji stanu zawiera poprawne kombinacje MFS_ stałe. Użyjemy tylko typu MFS_ENABLED .Wskazuje stan wyświetlania lub wyboru.
Pozycja ID jest numerem ID wysyłanym poprzez komunikat WM_COMMAND do procedury okna , które ma menu. Chociaż to pole jest 32-bitowe, tylko 16-bitów jest wysyłanych poprzez WM_COMMAND.
Standardowym jest nazywanie ich stałymi zaczynającymi się od IDM_.
Kolejne pole jest to info o strukturze menu. Jest to pole bajtowe wyłożone do granicy WORD . Intelowski format liczb rosnącego porządku bitów pozwala nam ustawić go i wyłożyć dyrektywą DW . API nie ma standardowych nazw dla poprawnych danych tego pola.. Definiujemy MFR_ stałe do tego celu. Są tylko dwa: flaga sygnalizująca MFR_POPUP ,że pozycja następująca po niej jest częścią podmenu podręcznego. Ostatnią pozycją w menu głównym lub podmenu ma flagę MFR_END w tym polu. Te dwie flagi mogą być łączone
Pozycja tekstu jest tekstem, który będzie wyświetlany przez pozycję MFT_STRING .Pomimo faktu, że używamy wersji "A" LoadMenuIndirect, ciągi w szablonie muszą być szerokości formatu Unicode. To dlatego DW jest używane zamiast DB. Jak w C, ciągi są zakończone znakiem zera (NUL) . Pole to zawsze istnieje z powodu potrzeby dojścia do granicy DWORD . '&' na pozycji ciągu powoduje , że znak następujący po nim jest pokazany jako podkreślony. Znak podkreślenia jest selektorem menu klawiatury . Więc jeśli naciśniesz i zwolnisz klawisz Alt , pierwsza pozycja w pasku menu będzie podświetlona. Potem naciskasz klawisz 'F', a pojawi się podmenu File. Jeśli podmenu pojawi się, naciśnięcie innego klawisza może spowodować wybór pozycji w podmenu. Chcąc wyświetlić '&' jako znak, musisz podwójnie wpisać '&', '&'.
W końcu IF pomocy F1. To pole musi istnieć dla wejścia MFR_POPUP ; nie musi istnieć dla innych. Musi być wyrównane do granicy DWORD .

```
;
; Menu item  (command) IDs
;
IDM_EXIT equ  101
IDM_HELP equ  901
IDM_ABOUT equ  902
;
; Menu template
;
MFR_END equ  80h
MFR_POPUP        equ  01h

        .data
        align   4       ;musisz wyrównać do granicy DWORD
appMenuTemplate dw  1   ; wersja szablonu menu
                        dw  4    ;offset od końca tego słowa do pozycji menu
                        dd  0    ;pasek menu ID pomocy

                        dd MFT_STRING,MFS_ENABLED,0
                        dw MFR_POPUP                    ; pierwsza kolumna
                        dw  '&','F','i','l','e',0,0;podstawa wyrównania do  4
                        dd  0                    ;ID pomocy menu podręcznego

                        dd MFT_STRING,MFS_ENABLED,IDM_EXIT
                        dw MFR_END                      ; dół kolumny
                        dw 'E','&','x','i','t',0,0;podstawa wyrównania do  4

                        dd MFT_STRING,MFS_ENABLED,0
                        dw MFR_POPUP or MFR_END         ;druga kolumna,
ostatnia
                       dw  '&','H','e','l','p',0,0;podstawa wyrównania do  4
                       dd  0                    ;ID pomocy menu Podręcznego

                        dd MFT_STRING,MFS_ENABLED,IDM_HELP
                        dw  0
                        dw  '&','H','e','l','p','.','.','.',0

                        dd MFT_SEPARATOR,0,0
                        dw  0
                        dw  0    podstawa wyrównania do  4
```

```
                        dd MFT_STRING,MFS_ENABLED,IDM_ABOUT
                        dw MFR_END                          ;dół kolumny
                        dw '&','A','b','o','u','t','.','.','.',0,0
```

Wysyłanie menu

Kiedy pozycja menu jest wybrana i "aktywowana", Windows wysyła komunikat WM_COMMAND do okna zawierającego menu. Pozycja menu ID funkcji jako polecenie ID przekazywany jest do okna procedury w niższych szesnastu bitach wParam.

```
        .code
        extrn      DefWindowProc:near,PostQuitMessage:near

WndProc:
        mov     eax,[esp+4+4]    ;ID komunikatu
        cmp     eax,WM_COMMAND   ;z menu, skrót lub kontrolka
        je      on_command
        cmp     eax,WM_DESTROY   ;okno będzie usunięte
        je      on_destroy
        jmp     DefWindowProc    ;przetwarzanie innych komunikatów

on_destroy:
        push    large  0
        call    PostQuitMessage

        xor     eax,eax
        ret     16

on_command:
        mov     eax,[esp+4+8]    ; wParam
        and     eax,large  0FFFFh        ;ID polecenia
        cmp     eax,IDM_EXIT
        je      on_exit_command
        cmp     eax,IDM_HELP
        je      on_help_command
        cmp     eax,IDM_ABOUT
        je      on_about_command
        xor     eax,eax          ;ignorujemy inne polecenia
        ret     16
```

Akcje menu

Dla zakończenia programu ,wysyłamy komunikat WM_CLOSE do okna „głównego” . Spowoduje to wywołanie DestroyWindow, domyślnie, i wysyłamy komunikat WM_DESTROY do naszej procedury okna.

Użyjemy MessageBox do wyświetlenia komunikatu kiedy jest wybrana pozycja menu nie - wyjście

```
        .data
helpCaption equ wnd_title
helpText db 'Help not implemented.',0

aboutCaption db 'About Win32 Assembly',0
aboutText db 'Place your copyright here.',0

        .code
        extrn      SendMessage:near
        extrn      MessageBox:near

on_exit_command:
        ; trigger the main window close function
        mov     eax,[esp+4+0]    ;uzyskanie hWnd przed zmianą stosu
```

```
        push    large  0        ; lParam
        push    large  0        ; wParam
        push    large WM_CLOSE  ; msgid
        push    eax             ; hwnd
        call    SendMessage

        xor     eax,eax
        ret     16

on_help_command:
        mov     eax,[esp+4+0]   ;uzyskanie hWnd przed zmianą stosu
        push    large MB_OK
        push    offset helpCaption
        push    offset helpText
        push    eax                     ;okno właściciela
        call    MessageBox

        xor     eax,eax
        ret     16

on_about_command:
        mov     eax,[esp+4+0]   ; uzyskanie hWnd przed zmianą stosu
        push    large MB_OK
        push    offset aboutCaption
        push    offset aboutText
        push    eax                     ;okno właściciela
        call    MessageBox

        xor     eax,eax
        ret     16

        end     _start
```

Wprowadzenie do okienek dialogowych

Dialogi, znane oficjalnie jako okienka dialogowe, są oknami popup które są tworzone z szablonów dialogów. .Użycie szablonów upraszcza tworzenie okien zawierających kontrolki. Predefiniowane procedury okna dla dialogów również dają ci standardowy zbiór funkcji nawigacyjnych klawiatury.

Dialogi modalne i niemodalne

Okienka dialogowe mogą być albo modalne lub niemodalne. Dialogi modalne wyłączają swoje okno właścicielskie i, w konsekwencji , swoje właścicielskie okna potomne. Wykonują również własną pętlę komunikatów, wywłaszczają wątki dopóki nie są zakończone. Chociaż dialog właścicielski nie odbiera generowanych komunikatów myszki lub klawiatury, może zaakceptować i przetworzony komunikat wysłać do niego w inny sposób.
Dialogi niemodalne działają jak inne okna popup .

Tworzenie okienka dialogowego

Modalne okienko dialogowe jest tworzone albo przez DialogBoxParam albo DialogBoxIndirectParam.Okienko dialogowe niemodalne jest tworzone albo przez CreateDialogParam lub CreateDialogIndirectParam. (Jeśli pisałeś Win32 w C or C++, niespodzianka!--windows.h używa makr do konwersji innych funkcji tworzących dialogi w tej postaci.)
Funkcje te ładują szablon dialogu, "bezpośrednie" wersje ładują z sekcji zasobów plików EXE lub DLL, wersje pośrednie z pamięci.

Procedury dialogu

Każdy dialog jest tworzony z klasy okna dialogu (nazywanego #32770).Procedura okna tej klasy wywołuje aplikację dostarczającą procedurę dialogu do obsługi tych komunikatów. Kiedy dialog jest stworzony, procedura okna tworzy dialog i wszystkie kontrolki, a potem wywołuje procedurę dialogu komunikatem WM_INITDIALOG . Oprócz dalszych inicjalizacji dialogu, procedura dialogu powinna odpowiadać temu komunikatowi przez zwracanie wartości niezerowej (TRUE) jeśli domyślne ustawienia skupiają się na pierwszej żądanej kontrolce

Dla większości komunikatów; procedura dialogu będzie zwracała wartość niezerową (TRUE) jeśli dany komunikat został przetworzony. Jeśli komunikat wymaga wartości zwracanej powinna być użyta SetWindowLong do ustawienia pola DWL_MSGRESULT w strukturze (wewnętrznej) okna dialogowego Jest kilka wyjątków, które zostały stworzone przez mądry MS . Zamiast stanu "przetwarzania?" poniższe komunikaty oczekują procedury dialogu dla zwracanej wartości:

WM_CTLCOLORxxxxx
WM_COMPAREITEM
WM_VKEYTOITEM
WM_CHARTOITEM
WM_QUERYDRAGICON

Jeśli okienko dialogowe jest modalne, EndDialog będzie kończył dialog.
Jeśli okienko dialogowe jest niemodalne, DestroyWindow będzie kończyło dialog.

Komunikacja z okienkiem dialogowym

Procedura okienka dialogowego występuje tam gdzie okienko dialogowe komunikuje się ze swoimi kontrolkami. Ale jak zrobić abyśmy uzyskali daną do i z okienka dialogowego?

Jednym rozwiązaniem jest użycie pamięci globalnej, to znaczy, zezwolenie na uzyskanie dostępu do okienka dialogowego ze stałej lokacji (np. w segmencie danych). Wielu ludzi nie lubi tego rozwiązania. Jest złe jeśli chcesz mieć więcej niż jedną instancję określonego dialogu

Powszechne dialogi używają różnych rozwiązań -- strukturę informacji. Adres struktury informacji, która może być alokowana gdziekolwiek w pamięci (globalnej, na stosie , lub stercie), jest przekazywany do okienka dialogowego kiedy jest tworzone. Może obejmować wartości startowe, których okienko dialogowe może używać do inicjalizacji swoich kontrolek. Okienko dialogowe może wprowadzić pola, w strukturze informacji, przez ukolejkowanie swoich kontrolek lub jako odpowiedź na powiadomienie kontrolek. Po tym jak okienko dialogowe jest zakończone, aplikacja może użyć przechwytywanej informacji. Funkcja tworząca okienko dialogowe ma argument parametr który może być użyty w tym celu. Argument ten staje się lParam komunikatu WM_INITDIALOG.

Nawigacja klawiatury

Dialog modalny automatycznie będzie obsługiwał nawigację klawiatury.
Aby wykonać obsługę dialogu niemodalnego nawigacją klawiatury, wywołujemy IsDialogMessage w pętli komunikatu. Jeśli przetwarza komunikat klawiatury, funkcja zwraca wartość niezerową (TRUE), a reszta pętli komunikatu powinna być przeskoczona.

Tworzenie szablonu dialogu z zasobu kompilowanego

Edytory dialogu jest to preferowany sposób tworzenia okienek dialogowych. Upraszczają one sortowanie według wymiarów i umieszczanie kontrolek. Mogą być one częścią edytorów zasobów .Edytory zasobów, takie jak edytory dialogów mogą tworzyć pliki tekstowe, które mogą być kompilowane przez kompilator zasobów. Efekt kompilacji zasobów może być potem dołączony do pliku EXE albo DLL jako zasób.
Kompilacja zasobów stworzy szablony dialogów do wliczenia jako zasoby. Funkcje okienek dialogowych mogą potem uzyskać dostęp do szablonów dialogów poprzez nazwę lub numer zasobu.

Tworzenie szablonów dialogów w pamięci bez kompilatora zasobów

Możliwe jest również zbudowanie szablonu dialogu w pamięci i użycie pośredniej funkcji okienka dialogowego do stworzenia dialogu.

Szablon dialogu jest “wewnętrznym” formatem, a ponieważ wersja Win32 była najpierw zaimplementowana

w NT, wszystkie ciągi których używamy muszą w Unicode. Jest tak pomimo faktu ,że są dwa punkty wejścia (A i W) dla funkcji pośredniej.

Poniżej mamy poszerzony szablon dialogu Win95. Sprawdź dokumentację SDK dla starych szablonów dialogów NT (Win95 może obsługiwać stare szablony dialogów)

Szablon dialogów Win95 jest dzielony na trzy części : nagłówek, opcjonalna pozycja czcionki i sekwencja pozycji dialogów.

Nagłówek zawiera sekcję o stałych rozmiarach po której następują trzy pola zmiennych o stałych rozmiarach

```
DLGTEMPLATEEX STRUC          ; nagłówek
dlttex_wDlgVer       DW  1         ; nakładki starego stylu dltt
dlttex_Signature     DW  0FFFFh    ; nakładki starego stylu dltt
dlttex_HelpID        DD  ?
dlttex_ExStyle       DD  ?
dlttex_style         DD  ?
dlttex_cDlgItems     DW  ?     ; liczba pozycji w dialogu
dlttex_x             DW  ?     ;lokacja lewego górnego rogu
dlttex_y             DW  ?
dlttex_cx            DW  ?     ; rozmiar
dlttex_cy            DW  ?
DLGTEMPLATEEX ENDS
```

Trzy pola zmiennych o stałym rozmiarze to menu, klasa i tytuł. Pierwsze słowo każdego pola ,ma specjalne znaczenie. Jeśli nie jest jedną ze specjalnych wartości, jest pierwszym znakiem ciągu Unicode zakończonego zerem.

Menu: 0000h = żadnego menu, 0FFFFh = kolejne słowo to id menu
Class: 0000h = klasa standardowego dialogu , 0FFFFh = kolejne słowo jest klasą okna
Title: 0000h = żadnego tytułu

{Pozycja czcionka istnieje tylko jeśli dlttex_styl zawiera DS_SETFONT.
pozycja czcionka ma sekcję o stałym rozmiarze po której następuje nazwa czcionki w zakończonym zerem ciągu Unicode.

```
FONTINFOEX STRUC
dfont_PointSize      DW  ?
dfont_Weight         DW  ?   ; FW_ stałe
dfont_Italic         DW  ?   ; TRUE lub FALSE
FONTINFOEX ENDS
```

Sekcja końcowa jest to sekwencja (lista) pozycji dialogu. Każda pozycja dialogu ma sekcję o stałym rozmiarze po której następują pola trzech zmiennych

```
DLGITEMTEMPLATEEX STRUC
dlitex_HelpID        DD  ?
dlitex_ExStyle       DD  ?
dlitex_Style         DD  ?
dlitex_x             DW  ?     ;lokacja lewego górnego rogu
dlitex_y             DW  ?
dlitex_cx            DW  ?     ; rozmiar
dlitex_cy            DW  ?
dlitex_id            DD  ?     ;ID kontrolki
DLGITEMTEMPLATEEX ENDS
```

Te trzy zmienne są to klasa, tytuł i data stworzenia. Pierwsze słowo każdego pola ma specjalne znaczenie Jeśli nie jest to jeden ze specjalnych wartości, pierwszy znak to ciąg Unicode zakończony zerem.

Class: 0000h = klasa okna standardowego, 0FFFFh = kolejne słowo jest klasą okna
Title: 0000h = żadnego tytułu
Creation data: pierwsze słowo jest bajtem liczącym rzeczywistą datę stworzenia która następuje bezpośrednio

Style Wndows

Style klas
CS_VREDRAW: przerysowanie obszaru klienta kiedy zmienia się rozmiar pionowy
CS_HREDRAW: przerysowanie obszaru klienta kiedy zmienia się rozmiar poziomy
CS_KEYCVTWINDOW: nieudokumentowane???
CS_NOKEYCVT: nieudokumentowane ???
CS_DBLCLKS: włączanie wykrywania podwójnego kliknięcia
CS_CLASSDC: tworzenie trwałego DC, dzielonego przez wszystkie okna
CS_OWNDC: tworzenie stałego DC dla każdego okna
CS_PARENTDC: wypożyczenie (dziedziczenie) okna rodzicielskiego DC
CS_NOCLOSE: Zamknięcie opcji usuwanej z menu systemu, zamknięcie wyłączonego przycisku
CS_SAVEBITS: zachowanie przykrytej części ekranu dla przemalowania po usunięciu okna
CS_BYTEALIGNWINDOW
CS_BYTEALIGNCLIENT
CS_GLOBALCLASS

Style okna

WS_OVERLAPPED
WS_POPUP
WS_CHILD

WS_DISABLED: nie wykryto żadnego wejścia klawiatury lub kliknięcia myszki, unieważniając wszystkie włączenia

WS_VISIBLE: czyni widzialnym używane dane współrzędne i rozmiar
WS_MINIMIZE: czyni widzialnym w postaci zminimalizowanej
WS_MAXIMIZE: czyni widzialnym w postaci zmaksymalizowanej

WS_CLIPSIBLING: nie przerysowuje obszaru pokrytego przez obszary siostrzane
WS_CLIPCHILDREN: nie przerysowuje obszaru pokrytego przez obszary potomne

WS_VSCROLL: pokazuje i włącza show pionowy pasek przesuwania
WS_HSCROLL: pokazuje i włącza poziomy pasek przewijania
WS_THICKFRAME: pokazuje i włącza grube obramowanie

WS_BORDER: pokazuje cienkie obramowanie
WS_DLGFRAME: pokazuje obramowanie okna dialogowego
WS_CAPTION: pokazuje i włącza nagłówek (pasek tytułowy) i cienkie obramowanie
WS_SYSMENU: pokazuje i włącza system menu ikon
WS_MINIMIZEBOX: pokazuje i włącza przycisk minimalizacji
WS_MAXIMIZEBOX: pokazuje i włącza przycisk maksymalizacji

Jeśli nie ma nagłówków

WS_TABSTOP: oznacza pozycję tabulatora
WS_GROUP: oznacza pole grupy

Style rozszerzone

WS_EX_WINDOWEDGE: Obramowanie okna 3-d
WS_EX_CLIENTEDGE: Obramowanie klienckie 3-d
WS_EX_STATICEDGE: Obramowanie 3-d (potomek?)

WS_EX_DLGMODALFRAME: podwójne obramowanie, może być używany z WS_CAPTION
WS_EX_TRANSPARENT: czyni przeźroczyste okno
WS_EX_CONTEXTHELP: pokazuje i włącza przyciski pomocy kontekstowej

WS_EX_LEFTSCROLLBAR: wkłada pionowy przycisk przewijania po lewej zamiast jak zazwyczaj po prawej
WS_EX_RIGHT: wyrównanie do prawej (niewyrównany do lewej) zamiast zazwyczaj lewego wyrównania , działa tylko z WS_EX_RTLREADING l
WS_EX_RTLREADING: odczytywanie od prawej do lewej zamiast zazwyczaj od lewej do prawej

WS_EX_TOPMOST: oznacza najwyższe okno , styl ten tworzy da zbiory okien, najwyższego i nie najwyższego

WS_EX_MDICHILD: czyni styl okna potomnego MDI
WS_EX_TOOLWINDOW: czyni styl narzędzia okna
WS_EX_APPWINDOW: czyni widocznym okno zminimalizowane na pasku zadań

WS_EX_CONTROLPARENT: włącza pozycję tabulatora potomstwa
WS_EX_NOPARENTNOTIFY: okno potomne nie wysyła WM_PARENTNOTIFY przy tworzeniu i destrukcji
WS_EX_ACCEPTFILES: przeciąganie i upuszczanie docelowe do pliku

Platformy Win32

Są podstawowe cztery platformy Win32 : NT, Win9x, WinCE, Win32s(przestarzałe)

NT

Ta platforma jest zbudowana na bazie kodu 32 bitowego. Obsługuje zarówno zbiór znaków ANSI i Unicode .

NT 3.1

Była to pierwsza platforma Win32 . Działała na Intel 386, MIPS, i Digital Alpha . Podobnie jak Unix, programy prekompilowane dla procesorów Intela , nie mogą działać na procesorach nie - Intelowskich, chyba ,że jest kompatybilny taki jak AMD lub Cyrix. Na pierwszy rzut oka ,ta wersja wyglądała dużo lepiej jak 16-bitowy Windows 3.1. Kilka wersji NT 3 zostało wprowadzono przed nadejściem NT 4 .

NT 4

Wersja ta przyniosła nowy Windows 95 wyglądający jak NT.

Windows 2000

Znany również jako Win2k. Był nazwany NT 5. W przeciwieństwie do NT 4, może czytać nowy system plików VFAT32 używanym przez Windows 95 OSR2 i Windows 98. Właściwie napisane sterowniki w postaci plików WDM , mogą być używane przez Win2k, Win98 SE, i Win ME.

Windows XP

To dodatkowa "domowa" wersja Win2k. Oczekuje się powolnej wymiany wszystkich wersji f Win9x, dla ludzi posiadających maszyny z minimalną ilości RAM 128M .

Win9x

Ta platforma jest hybrydą kodu 32- i 16-bitowego. Obsługuje najczęściej ANSI. Tylko bardzo niewiele podstawowych funkcji API może obsługiwać Unicode.

Windows 95

Znany również jako Win95, , był wersją, która zmieniła spojrzenie na Windows. Przyniósł również system plików VFAT ,który nie może być odczytany przez NT w czasie jego wprowadzenia. Ostatnia wersja , Windows 95 OSR2 (OEM System Release 2), wprowadza dużo bardziej wydajny system plików VFAT32 , ale te późniejsze wersje nie były dostępne jako produkt detaliczny . Musiałeś być producentem , lub kupić maszynę z Windows 95 OSR2 zainstalowanym przez producenta.
Kiedy pytasz o pierwotny numer wersji Win95, dowiesz się , że jest wersja 4.00. Win95 OSR2 pokaże ci wersję 4.00 B.

Windows 98

Znany również jako Win98, wersja ta uczyniła dostępnym poprawiony Win95 OSR2 w wersji detalicznej. Kiedy pytasz o numer wersji Win98 dowiesz się, ze jest to wersja 4.10. Win98 SE (wydanie drugie) to wersja 4.10.2222.

Windows ME (Millenium Edition)

Jest to oczekiwany ostatni system z serii Win9x.

WinCE

Podzbiór platformy zbudowany na bazie kodu 32-bitowego .Obsługuje tylko Unicode. Jest

to Win32 dla bardzo małych systemów, takich jak komputery przenośne.

Win32s (przestarzały)

Windows 3.1, 16-bitowa wersja Windows, może używać pewnych zdolności procesorów 32 bitowych Intela działających w Trybie Ulepszonym 386 . Pozwala to Windows dostarczyć dużą przestrzeń danych, i uruchamiać wiele sesji DOS równocześnie z Windows.

Możesz również ulepszyć ten system poprzez ściągnięcie z Microsoft, zbioru plików , które implementują set podzbiór Win32s z Win32. Implementacja Win32s cierpi z powodu tego ograniczenia (np. żadnej wielowątkowości) i kilku dzielących problem plików . To czyni pewne produkty Win32 bardzo złe. Win95 uczynił przestarzałym Win32s .

Reszta Win32

Jak widzisz na poniższej liście tematów nie objętych w tym tutorialu, programowanie Win32 jest tematem rozległym .

- Wspólne kontrolki, ponad dwanaście!!
- Okno komunikatu
- Wspólne dialogi, dziesięć!
- GDI, bitmapy, ikony, pędzle, pióra itd.
- Urządzenia kontekstowe dla drukarki / plottera i bitmap
- Okna nadklas i podklas
- Atomy
- Metapliki
- Zasoby
- Zasoby ciągów i internacjonalizacja
- Wersja Kontrolki
- Zarządzanie pamięcią
- Łączność
- Odwzorowywanie plików
- Dziedziczenie plików (i inne programy obsługi)
- Drukowanie
- Dźwięk
- DLL
- Hooki i injekcja DLL
- Powłoka i start pozostałych aplikacji
- Oczekiwanie na zakończenie aplikacji
- Restartowanie Windows
- Zmienne środowiskowe
- Profile: Rejestry i pliki .INI

- Wątki i procesy
- Strukturalna obsługa wyjątków
- Synchronizacja
- Myszka i kursor
- Klawiatura
- Schowek
- MDI
- DDE
- MFC !
- Winsock
- TAPI
- MAPI
- COM/OLE2/ActiveX
- DirectX
- Sterowniki: VxD and WDM
- System usług (NT)

Więcej informacji programistycznych

Uruchomienie programu

Kiedy program Win32 startuje, rejestry segmentowe są ustawiane w następujący sposób::
CS = selektor kodu
DS=ES=SS=selektor danych
FS= selektor danych do TIB (blok informacji wątków) dla pierwszego wątku GS =
0, selektor zera

CS odwzorowuje taki sam adres liniowy jak DS, ES, i SS. FS odwzorowuje do TIB który jest osadzony w bloku danych znanym jako TDB (baza danych wątków). TIB zawiera łańcuch obsługi wyjątków określonych wątków i wskaźnik do TLS (lokalna pamięć wątków) . Lokalna pamięć wątków nie jest tym samym co pamięć lokalna C.ESP jest ustawiony na wysoki adres , nie koniecznie przy znaku 2G. DF (flaga opadająca /kierunku) jest wyzerowana, więc ciąg ops będzie powodował zwiększenie (tryb rosnący) .

Jeśli kiedykolwiek napisałeś kod przełączający z trybu rzeczywistego do trybu chronionego wiesz jak CS ma różne wartości "segmentu" i jeszcze może adresować taki sam obszar pamięci z takim samym offsetem Dla programistów x86 ASM którzy nie są zaznajomieni z trybem chronionym "segment" nie jest dłużej wartością (paragrafu) , jest selektorem. Selektor odnosi się do deskryptora, gdzie rzeczywista informacja o "segmencie" (który może być dowolnego rozmiaru) jest przechowywana.

Inne wartości uruchomieniowe

Poprzednia instancja uchwytu:
Jeśli napisałeś program Windows a C lub C++, możesz być zaznajomiony z tymi parametrami. W Windowsie 16 bitowym było to najczęściej używał tego do upewnienia się że była tylko jedna instancja (uruchomiona kopia) aplikacji. Istnieją inne techniki dla zapewnienia tylko jednej instancji aplikacji. Kiedy systemy miały dużo mniej pamięci niż dzisiaj, pozwalało to kilku instancjom 16 bitowych aplikacji dzielić wspólnych danych (takich jak

grafika). W Win32, jest to zawsze 0 (NULL) ponieważ uchwyty instancji nie mogą być używane do uzyskaniu dostępu do danych z innych procesów (instancja).

Uchwyt instancji / modułu EXE:

```
    push    large  0              ; wskaźnik ciągu NULL
    call    GetModuleHandle       ; EAX  = hInstance EXE
```

Argumenty lini poleceń:

```
    call    GetCommandLine        ; EAX  = wskaźnik pełnej lini poleceń
```

[Zmienne środowiskowe:]

Standardowe I/O, nCmdShow:

```
STARTUPINFO struc
si_cb               dd  ?    ; rozmiar STARTUPINFO
                    dd  ?    ; lpReserved
si_lpDesktop        dd  ?    ;  (ciąg) nazwy pulpitu
;  *** Początek informacji pulpitu okna
si_lpTitle          dd  ?    ;  (ciąg) tytuł/nagłówek
si_dwX              dd  ?    ; górny lewy róg
si_dwY              dd  ?    ; górny lewy róg
si_dwXSize          dd  ?    ; szerokość
si_dwYSize          dd  ?    ; wysokość
si_dwXCountChars    dd  ?    ; szerokość pulpitu
si_dwYCountChars    dd  ?    ; wysokość pulpitu
si_dwFillAttribute dd  ?     ; atrybuty pulpitu okna
;  *** Koniec informacji pulpitu okna
si_dwFlags          dd  ?    ; opcja flag, używany przez CreateProcess
si_wShowWindow      dw  ?    ; używany w pierwszym wywołaniu ShowWindow ;
                    dw  ?    cbReserved2
                    dd  ?    ; lpReserved2
si_hStdInput        dd  ?    ; standardowa obsługa wejścia
si_hStdOutput       dd  ?    ; standardowa obsługa wyjścia
si_hStdError        dd  ?    ; standardowa obsługa błędu
STARTUPINFO ends

    .data
start_info     STARTUPINFO     <size  STARTUPINFO>     ;ustawienie  wartości  w
pierwszym polu

    .code
    push     offset start_info
    call      GetStartupInfo      ;funkcja pusta
```

Wersja podsystemu

Z wersją 4 kompilatora, IDE Microsoftu automatycznie ustawia wersję podsystemu 4.0 lub późniejszą. Linker nie będzie tego ustawiał, chyba ,że użyjesz opcji /subsystem:. Dla uzyskania dosyć wysokiej wersji użyj /SUBSYSTEM:CONSOLE lub /SUBSYSTEM:WINDOWS. Z powodu w jaki nasze przykłady zostały napisane , wersja 6 linkera wymaga opcji /SUBSYSTEM. Linker Borlanda TASM 4.0 nie dostarcza takiej możliwości. Późniejsze linkery mają opcję /Vn.n dla ustawienia tej ważnej wartości.

Wersja podsystemu wpływa na zachowanie aplikacji Win32.

- Jeśli poprzednia to było 4.0, standardowa aplikacja Win32 nie będzie startowała przy Win9x z pod DOS chyba ,że użyjesz polecenia START . Aplikacje konsolowe mogą zawsze być wywołane poleceniami DOS

- Jeśli poprzednio było to4.0, kolor tła standardowej kontrolki będzie białe (stary styl). Jest to denerwujące kiedy budujesz okienka dialogowe w nowym stylu.

Tworzenie ważmych bibliotek z arbitralnych DLL

Przy linkerze Borlanda, używamy IMPDEF do stworzenia pliku .DEF z DLL. Potem używamy IMPLIB do stworzenia biblioteki importu z pliku .DEF .
Przy linkerze Microsoft , używamy DUMPBIN /EXPORTS do uzyskania listy punktów wejścia i ich "porządków". Przekierowuje dane wyjściowe do pliku .DEF. Zmieniamy każdą linię nazwy punktu wejścia:

```
entryname
```

Eliminujemy wszystkie inne linie. Potem dodajemy na początek pliku:

```
LIBRARY    dllbasename
EXPORTS
```

Potem używamy LIB /DEF do wygenerowania pliku .LIB z wyedytowanego pliku .DEF. LIB będzie poprzedzał znakiem podkreślenia , "_", każdą nazwę wejścia. Ze względu na brak argumentu informacji, LIB nie będzie dodawał żadnych innych znaków do linkowanej nazwy.

Teoria zakleszczenia

Zakleszczenie , lub śmiertelny uścisk, jest stanem gdzie procesy oczekują nieustannie na pozyskanie zasobów , które nie mogą być wyzwolone. Na potrzeby teorii zakleszczenia wątek Win32 może być traktowany jako proces a zasób jest definiowany jako dowolnie przydzielany element. Dla rozumowania na temat zakleszczenia z powodu wiadomości okna, jest bardziej owocne traktowanie każdego okna jako procesu, a każdego wątku jako zasobu, który może powodować zakleszczenie
Są cztery konieczne i wystarczające warunki dla stworzenia zakleszczenia.

Konieczny oznacza, że wszystkie warunki muszą zaistnieć dla wystąpienia zakleszczenia.

Wystarczający oznacza ,że kiedy wszystkie warunki istnieją, wystąpi zakleszczenie.

Jak twierdzi Tanenbaum [odnoszę się do artykułu Coffmana, Elphicka i Shoshani'ego], są:

- Warunek wzajemnego wykluczania. Każdy zasób jest albo jest aktualnie przypisane do dokładnie jednego procesu lub jest dostępny
- Warunek trzymaj i czekaj. Procesy aktualnie przechowujące zasoby udostępnione wcześniej, mogą wymagać nowych zasobów.
- Warunek żadnego wywłaszczania. Zasoby poprzednio udostępnione nie mogą być na siłę zabrane z procesu. Musza być one wyraźnie zwolnione przez proces je przechowujący.
- Warunek cyklicznego oczekiwania. Musi być cykliczny łańcuch dwóch lub więcej procesów, każdy z nich oczekuje na zasób trzymany przez kolejny element łańcucha.

Zbiór Brincha Hansena może być bardziej odkrywczy:

- Wzajemne wykluczenie: Zasób może być tylko nabyty przez jeden proces w czasie.
- Kolejkowanie żadnego wywłaszczenia: Zasób może być tylko wyzwolony przez proces , który go nabył.
- Alokacja częściowa: Proces może nabywać swoje zasoby stopniowo.
- Oczekiwanie cykliczne: Poprzednie warunki zezwalają zbieżnym w czasie procesom nabywać część swoich zasobów i wprowadzać stan w którym oczekują nieokreślony czas na nabycie jeden od drugiego zasobów.

Anomalie paska tytułu (nagłówka) i przycisku paska zadań

Standardowy pasek tytułowy Win95 (wymagany albo domyślnie albo z WS_CAPTION) nie jest tak niestandardowy jak stary pasek tytułowy Win3.1 (NT 3.x lub 16-bitowy). Jeśli chcesz aby pasek tytułu zachowywał się lepiej, prawdopodobnie będziesz musiał tworzyć swój własny. Kiedy blokujesz przycisk paska tytułowego on niekoniecznie znika. Przyciski minimalizacji i maksymalizacji znikają jako para .Jeśli blokujesz

jeden a uaktywniasz inny, oba będą pokazane a przycisk zablokowany będzie szary . Przycisk zamknięcia (X) może być zablokowany (szary) z CS_NOCLOSE, ale nie pojawi się do usunięcia. Odblokowanie przycisku minimalizacji stworzy przycisk paska zadań, but odblokowanie przycisku maksymalizacji już nie. Przycisk zamknięcia nie może być odblokowany chyba ,że system menu jest również odblokowany.
Jest kilka dziwacznych zachowań kiedy żadne menu systemowe i żaden rozmiar przycisku nie są określone. Kiedy jest używany skrót do startu z oknem zmaksymalizowanym, pobierasz przycisk paska zadań. Kiedy skrót jest używany do startu z oknem zminimalizowanym , pobierasz "przycisk" , który nie siedzi w pasku zadań

Udokumentowane funkcje Win32 które nie istnieją

Jest kilka oficjalnych funkcji API, które są konwertowane przez makra C do odpowiednich funkcji z dodatkowymi argumentami. Dodatkowe argumenty to 0 lub NULL.

- CreateWindow, użycie CreateWindowEx
- DialogBox, użycie DialogBoxParam
- DialogBoxIndirect, użycie DialogBoxIndirectParam

Przestarzałe funkcje Win16

Jest kilka funkcji API Win16 , które albo nie istnieją dłużej w Win32, lub istnieją wyłącznie dla kompatybilności z programowaniem Win16 w C.

- Funkcje które nie istnieją
  - GetModuleUsage, byłą głównie używana do oczekiwania na koniec innych aplikacji. Do tego celu używamy CreateProcess, OpenProcess, and GetExitCode or aitForSingleObject.
  - MoveTo, używamy MoveToEx

Przykład pola wyboru: opcje nagłówka

Ten przykład pokazuje jeden ze sposobów użycia pół wyboru w oknie niedialogowym. Pokazuje również co się wydarzy kiedy te nagłówki opcji są łączone. Pokazuje jak zmieniać styl okna po tym jak go stworzono. I w końcu, pokazuje jak wymusić przerysowanie ramkę okna.
Plik źródłowy to wincaptn.asm.

Najpierw musimy uważać na okno główne, na start aplikacji i jej koniec

```
        .386
        .model   flat

        ;Jeśli używasz TLINK32,nie włączaj vclib.inc
        include vclib.inc     ;linkowana nazwa  .lib Microsoft VC++

        include win32hst.inc   ;stałe, struktury i nazwy wejść
        public  _start

        .code
_start:

        .data
        align   4
wc      dd      CS_VREDRAW or CS_HREDRAW   ; styl
        dd      WndProc                    ; lpfnWndProc
        dd      0,0                        ; cbClsExtra, cbWndExtra
        dd      0                          ; hInstance
        dd      0                          ; hIcon
        dd      0                          ; hCursor
        dd      COLOR_WINDOW+1             ; hbrBackground
        dd      0                          ; lpszMenuName
        dd      wndclsname                 ; lpszClassName
```

```
wndclsname db 'winmain',0

        extrn     GetModuleHandle:near

        .code
        push    large  0        ;wskaźnik ciągu NULL
        call    GetModuleHandle  ;pobranie HINSTANCE/HMODULE pliku EXE
        mov     [wc.wc_hInstance],eax

        extrn     LoadIcon:near,LoadCursor:near

        .code
        push    large IDI_WINLOGO
        push    large  0        ; hInstance,  0  = stock icon
        call    LoadIcon
        mov     [wc.wc_hIcon],eax

        push    large IDC_ARROW
        push    large  0        ; hInstance,  0  = stock cursor
        call    LoadCursor
        mov     [wc.wc_hCursor],eax

        extrn     RegisterClass:near

        .code
        push    offset wc
        call    RegisterClass

        .data
        align   4
cwargs   dd   0                  ; dwExStyle
         dd   wndclsname         ; lpszClass
         dd   wnd_title          ; lpszName
         dd   WS_VISIBLE or WS_OVERLAPPED or WS_SYSMENU or WS_THICKFRAME  \
              or WS_MINIMIZEBOX or WS_MAXIMIZEBOX   ; styl
         dd   40                 ; x
         dd   40                 ; y
         dd   300                ; cx  (szerokość)
         dd   110                ; cy  (wysokość)
         dd   0                  ; hwndParent
         dd   0                  ; hMenu
         dd   0                  ; hInstance
         dd   0                  ; lpCreateParams

wnd_title db 'Caption  (title bar) styles',0

        extrn     CreateWindowEx:near

        .code
        sub     esp,48            ; alokacja listy argumentów
        mov     esi,offset cwargs  ;ustawienie przesuwanego bloku źródła
        mov     edi,esp         ;ustawienie przesuwanego bloku przeznaczenia
        mov     ecx,12            ;liczba argumentów
        rep movsd
        mov     eax,[wc.wc_hInstance]
        mov     [esp+40],eax      ;stawienie argumentu hInstance na stosie
        call    CreateWindowEx

        extrn     GetMessage:near,DispatchMessage:near
```

```
        .data
        align   4
msgbuf MSG      <>

        .code
msg_loop:
        push    large  0        ; uMsgFilterMax
        push    large  0        ; uMsgFilterMin
        push    large  0        ; hWnd  (filtr),  0  = wszystkie okna
        push    offset msgbuf   ; lpMsg
        call    GetMessage      ; zwraca FALSE jeœli WM_QUIT
        or      eax,eax
        jz      end_loop

        push    offset msgbuf
        call    DispatchMessage

        jmp     msg_loop

end_loop:

        extrn      ExitProcess:near

        .code
        push    large  0  ;  (błąd) kod zakończenia
        call    ExitProcess
```

Następnie definiujemy nasze okno testowe. Wyraźnie dodajemy w WS_CAPTION ponieważ będziemy używali kopii tego argumentu stylu jako podstawy dla zmiany stylu. Okazuje się ,że chociaż okna nakładkowe zawsze tworzą nagłówek (pasek tytułowy), pasek tytułowy może być usunięty po tym jak stworzono okno przez zmianę stylu okna.

Procedura okna dla tego okna to DefWindowProc ponieważ nie jest wymagana żaden program obsługi komunikatu.

```
;
; test wyświetlania okna
;
        .data
        align  4

testwnd dd  0

wc2     dd      CS_VREDRAW or CS_HREDRAW   ; styl
        dd      DefWindowProc              ; lpfnWndProc
        dd      0,0                        ; cbClsExtra, cbWndExtra
        dd      0                          ; hInstance
        dd      0                          ; hIcon
        dd      0                          ; hCursor
        dd      COLOR_WINDOW+1             ; hbrBackground
        dd      0                          ; lpszMenuName
        dd      wndclsname2                ; lpszClassName

test     dd   0                   ; dwExStyle
         dd      wndclsname2      ; lpszClass
         dd      caption2         ; lpszName
         dd      WS_VISIBLE or WS_OVERLAPPED or WS_CAPTION  ; styl
         dd   160                 ; x
         dd   160                 ; y
         dd   200                 ; cx  (szerokość)
         dd   200                 ; cy  (wysokość)
         dd   0                   ; hwndParent
         dd   0                   ; hMenu
         dd   0                   ; hInstance
```

```
         dd   0                   ; lpCreateParams

wndclsname2 db 'testdisplay',0
caption2 db 'Display test',0
```

Teraz definiujemy pola wyborów, które są zdefiniowane jako przyciski. Nie ma struktury WNDCLASS ponieważ klasa przycisku jest już zdefiniowana i zarejestrowana. Będziemy obsługiwali wszystkie ważne pola wyboru aktywne w oknie głównym, które jest rodzicem (i kontenerem) lub wszystkie te pola wyboru.

```
; to przechowuje bity stylu używane do zmiany
;      opcje stylu okna testowego
;
        align   4

test_style dd  0

;
; ID kontrolek dla przycisków
;
IDCTL_SYSMENU  equ   101
IDCTL_SIZEBOX  equ   102
IDCTL_MINBOX     equ   103
IDCTL_MAXBOX     equ   104
;
; Lista argumentów Temporary CreateWindowEx dla pola wyboru "przycisków”
okna
;
        align   4

chkbox     dd  0                   ; dwExStyle
          dd     btnclsname        ; lpszClass
          dd   0                   ; lpszName
          dd     WS_VISIBLE or WS_CHILD or BS_CHECKBOX  ; style
          dd   0                   ; x
          dd   0                   ; y
          dd   0                   ; cx  (width)
          dd   0                   ; cy  (height)
          dd   0                   ; hwndParent
          dd   0                   ; hMenu
          dd   0                   ; hInstance
          dd   0                   ; lpCreateParams

btnclsname    db 'button',0         ; letter case doesn't matter
;
; option/button table
;
        align  4

sysmenu dd WS_SYSMENU,sysmenu_str,10,10,250,15,IDCTL_SYSMENU
sizebox   dd   WS_THICKFRAME,sizebox_str,10,25,250,15,IDCTL_SIZEBOX
minbox       dd  WS_MINIMIZEBOX,minbox_str,10,40,250,15,IDCTL_MINBOX
maxbox       dd  WS_MAXIMIZEBOX,maxbox_str,10,55,250,15,IDCTL_MAXBOX
;
; button texts
;
sysmenu_str db 'WS_SYSMENU',0
sizebox_str db 'WS_THICKFRAME/WM_SIZEBOX',0
minbox_str    db 'WS_MINIMIZEBOX',0
maxbox_str    db 'WS_MAXIMIZEBOX',0
```

Jest jedna procedura okna zdefiniowana przez nas, i jedna dla okna głównego, z polami wyboru w nim. Zaczniemy od wysłania komunikatu i polecenia "zamknięcia aplikacji”.

```
        extrn      DefWindowProc:near
```

```
        .code
WndProc:
        mov     eax,[esp+4+4]           ;ID komunikatu
        cmp     eax,WM_COMMAND          ;kliknięcie kontrolki
        je      on_command
        cmp     eax,WM_CREATE           ;stworzenie okna
        je      on_create
        cmp     eax,WM_DESTROY          ;start destrukcji okna
        je      on_destroy
        jmp     DefWindowProc           ;przetwarzanie innych komunikatów


        extrn     PostQuitMessage:near

on_destroy:
        push    large  0
        call    PostQuitMessage

        xor     eax,eax
        ret     16
```

Tu tworzymy pola wyboru, jako odpowiedź na WM_CREATE.W tym czasie, okno główne zostanie stworzone I jest uchwyt okna dla niego. Uchwyt jest używany do przypisania okna jako rodzica do każdego pola wyboru, jakie jest tworzone jako okno potomne. Okno potomne musi być powiązane z rodzicem kiedy jest tworzone
Zarejestrujemy również klasę okna testowego

```
;
; okno główne
;
        .data
        align   4

mainwnd dd  0

        .code
on_create:
        mov     eax,[esp+4+0]   ; hwnd
        mov     [mainwnd],eax
        mov     eax,[wc].wc_hInstance
        mov     [wc2].wc_hInstance,eax
        push    offset wc2
        call    RegisterClass

        push    esi
        push    edi

        mov     esi,offset test
        mov     eax,[esi+12]            ;argument stylu
        mov     [test_style],eax        ;zachowanie oryginalnego stylu
        call    install_subwindow
        mov     [testwnd],eax


        mov     esi,offset chkbox
        mov     edi,offset sysmenu
        call    install_chkbox

        mov     esi,offset chkbox
        mov     edi,offset sizebox
        call    install_chkbox
```

```
        mov     esi,offset chkbox
        mov     edi,offset minbox
        call    install_chkbox

        mov     esi,offset chkbox
        mov     edi,offset maxbox
        call    install_chkbox

        pop     edi
        pop     esi

        xor     eax,eax    ;sygnał poprawnego STWORZENIA
        ret     16
;
; Tworzenie pola wyboru
;
;       ESI  = adres    argumentów CreateWindowEx
;       EDI  = tablica wejść pola wyboru
;
; Zwracanie:
;
;       EAX  = uchwyt BUTTON
;
install_chkbox:
        mov     eax,4[edi]
        mov     [chkbox+8],eax     ; lpszName, ustawienie nazwy okna  =tekst
okna
        mov     eax,8[edi]
        mov     [chkbox+16],eax    ; x
        mov     eax,12[edi]
        mov     [chkbox+20],eax    ; y
        mov     eax,16[edi]
        mov     [chkbox+24],eax    ; cx
        mov     eax,20[edi]
        mov     [chkbox+28],eax    ; cy
        mov     eax,24[edi]
        mov     [chkbox+36],eax    ; hMenu, ctl ID kiedy WS_CHILD
        call    install_subwindow
        ret
;
; Tworzenie podokien okna głównego
;
;       ESI  = adres argumentów CreateWindowEx
;
; Zwracanie:
;
;       EAX  = uchwyt podokna
;
install_subwindow:
        mov     eax,[mainwnd]
        mov     [esi+32],eax             ; hwndParent, zrobienie    "mainbox"
                                          ;właścicielem lub rodzicem
                                                ;nowego okna

        mov     eax,[wc].wc_hInstance    ;pobranie instancji
        mov     [esi+40],eax             ;ustawienie hInstance klasy okna
        sub     esp,48    ;alokacja argumentów
        mov     edi,esp
        mov     ecx,12
        rep movsd
        call    CreateWindowEx
```

```
        ret
```

Tu wykonuje się większość pracy w odpowiedzi na WM_COMMAND. Jeśli polecenie nie jest przyciskiem (pole wyboru) wychodzimy. W przeciwnym razie pobieramy opcję stylu. Ponieważ opcja jest zakodowana pojedynczym bitem, podwójna opcja jest maską bitu.

```
        extrn     SendMessage:near
        extrn     SetWindowLong:near
        extrn     RedrawWindow:near

on_command:
        mov     eax,[esp+4+8]    ; wParam
        mov     edx,[esp+4+12]   ; lParam
        cmp     eax,(BN_CLICKED shl  16)+IDCTL_SYSMENU
        je      select_sysmenu
        cmp     eax,(BN_CLICKED shl  16)+IDCTL_SIZEBOX
        je      select_sizebox
        cmp     eax,(BN_CLICKED shl  16)+IDCTL_MINBOX
        je      select_minbox
        cmp     eax,(BN_CLICKED shl  16)+IDCTL_MAXBOX
        je      select_maxbox
        jmp     exit_on_command
;
; Pobranie stylu
;
select_sysmenu:
        mov     ecx,[sysmenu]
        jmp     toggle
select_sizebox:
        mov     ecx,[sizebox]
        jmp     toggle
select_minbox:
        mov     ecx,[minbox]
        jmp     toggle
select_maxbox:
        mov     ecx,[maxbox]
        jmp     toggle
```

Teraz pobieramy I resetujemy stan pola wyboru. Tu zobaczymy że komunikaty są głównym środkiem przekazywania informacji do i z kontrolek.

```
toggle:
        push    ecx             ;zachowanie stylu

        push    edx             ;zachowanie uchwytu kontrolek

        push    large  0
        push    large  0
        push    large BM_GETCHECK
        push    edx             ;uchwyt kontrolki z lParam
        call    SendMessage

        pop     edx             ;odzyskanie uchwytu kontrolki

        xor     eax,1           ;przełączenie stanu wyboru

        push    large  0
        push    eax             ;nowy stan wyboru
        push    large BM_SETCHECK
        push    edx             ;uchwyt kontrolki
        call    SendMessage

        pop     ecx             ;odzyskanie stylu
```

Then we change the window style of our test window.

```
        xor     [test_style],ecx        ;[przełączenie bitu stylu
        push    [test_style]            ;nowy styl
        push    large GWL_STYLE
        push    [testwnd]               ;okno testowe
        call    SetWindowLong
```

Zmiana stylu okna nie powoduje przerysowania okna. Bez poniższego kodu, okno testowe nie zmieni się dopóki jest wybrane. A przerysowanie jest niezupełne.

Wiec dodamy tą funkcję API dla wymuszenia przerysowania tego okna testowego bez jego wybierania. Pasek tytułowy jest częścią "ramki".

```
        push    large  (RDW_FRAME+RDW_INVALIDATE+RDW_UPDATENOW)
        push    large  0        ;aktualizacja prostokąta
        push    large  0        ;aktualizacja regionu(uchylenie prostokąta)
        push    [testwnd]       ; test window
        call    RedrawWindow

exit_on_command:
        xor     eax,eax    ;został przetworzony sygnał WM_COMMAND
ret     16

        end     _start
```

Minimalny program Win32 GUI -- WINDOW01.ASM

Program ten wyświetla okno używając bardzo mało informacji. Ilustruje on minimalne wymagania dla tworzenia zwykłego okna. Ten szczególny program jest prawie najmniejszym skończonym programem GUI ze zwykłym oknem. Aby uzyskać go takim małym będzie wymagał użycia predefiniowanych klas okien (okienka komunikatów, dialogów lub kontrolek)

Możesz przesunąć to okno. Jeśli wybierzesz jakieś inne okno, możesz kliknąć na okno dla reaktywacji (ponownego wyboru) naszego programu. Możesz reaktywować go również klawiszami Alt-Tab. Używamy AltF4, kiedy jest wybrany, kończąc program.

Możesz również stworzyć skrót I użyć go do startu programu z oknem w postaci zminimalizowanej lub zmaksymalizowanej. Podwójne kliknięcie paska nagłówka / tytułu zmieni okno na "normalny" rozmiar.

Startowanie programu

Dla programów jedno wątkowych wartości początkowe rejestrów prawie nie są. Ale w przypadku jaki nas interesuje: CS:EIP = start programu; SS:ESP = start stosu; DS = ES = SS; FS = TIB, blok informacji wątku; GS = 0, selektor null . CS i DS odwzorowują taki sam adres liniowy. Flaga kierunku , DF, jest wyzerowana.
Przy linkowaniu linkerem Borlanda, adres startowy jest określony w dyrektywie `END`.
Przy linkowaniu linkerem Microsoft, adres startowy musi być `PUBLIC I określony przez przełącznik linkera /ENTRY:` . Linker automatycznie doda znak podkreślenia, "_".

```
    .386
    model   flat

    public  _start

    ; ...

    .code
_start:

    ; ...

    end     _start
```

Klasa okna

Każde okno jest opisane strukturą danych znaną jako klasa okna. (Ta klasa nie jest klasą C++ .) Każda klasa okna ma nazwę. Powiązana z klasą okna jest funkcja wywołania zwrotnego znana jako procedura okna. Funkcja wywołania zwrotnego definiuje zachowania dal wszystkich okien stworzonych ta klasą Zanim stworzysz okno, okno to musi być zarejestrowane. Windows pre - rejestruje kilka klas okien, ale są one zazwyczaj używane do tworzenia kontrolek.

Są dwa sposoby rejestrowania zwykłych klas okien: stary sposób przez wywołanie RegisterClass ze struktury WNDCLASS , lub nowy sposób przez wywołanie RegisterClassEx ze struktury WNDCLASSEX . Nowszy sposób pozwala określić nowe małe ikony Win95 . Stary sposób jest wymaganay jeśli chcesz być kompatybilny ze starszymi (przed 4.0) systemami NT

```
WNDCLASSEX struc          ; comments show the C/C++ name
wcxSize           dd  ?   ; cbSize, size of WNDCLASSEX
wcxStyle          dd  ?   ; style
wcxWndProc        dd  ?   ; lpfnWndProc
wcxClsExtra       dd  ?   ; cbClsExtra
wcxWndExtra       dd  ?   ; cbWndExtra
wcxInstance       dd  ?   ; hInstance
wcxIcon           dd  ?   ; hIcon
wcxCursor         dd  ?   ; hCursor
wcxBkgndBrush     dd  ?   ; hbrBackground
wcxMenuName       dd  ?   ; lpszMenuName
wcxClassName      dd  ?   ; lpszClassName
wcxSmallIcon      dd  ?   ; hIconSm
WNDCLASSEX ends
```

Wow! Jak dużo pól. Cztery pola są minimalnie wymagane do stworzenia okna: cbSize, lpfnWndProc, hInstance, and lpszClassName. Pola nieużywane muszą być wyzerowane.

Pole lpfnWndProc zawiera adres procedury okna, która definiuje jak okno tej klasy będzie się zachowywać. Pole hInstance zawiera uchwyt do modułu który "posiada" klasę okna Win16 rozróżnia pomiędzy instancją a uchwytem modułu ,ale w Win32, są one jednym i tym samym. Te uchwyty instancji modułów identyfikują załadowane moduły (pliki EXE i DLL), coś jak uchwyt pliku identyfikuje otwarte pliki. Podobnie jak uchwyt pliku, uchwyty moduł / instancja są unikalne tylko wewnątrz uruchomionej instancji programu lub procesu Pole lpszClassName zawiera adres ciągu zakończonego zerem (styl C) , który nazywa klasę okna. Chociaż Win32,generalnie pozwala na ciągi ze znalami albo ANSI albo Unicode, Win95,w większości przypadków obsługuje tylko ANSI. Więc zakodujemy nazwę klasy okna w .

```
extrn        GetModuleHandle:near,RegisterClassEx:near
    .data
wc  WNDCLASSEX <size WNDCLASSEX,0,WndProc,0,0, 0, 0,0,0, 0,wndclsname, 0> wndclsname
db 'window01',0

    .code
    push    large  0            ;wskaźnik ciągu NULL
    call    GetModuleHandle     ;uzyskanie HINSTANCE/HMODULE pliku EXE
    mov     [wc.wcxInstance],eax
    push    offset wc
    call    RegisterClassEx
```

Wszystkie wersje Windows używają tej struktury do tworzenia swoich własnych, wewnętrznych “obiektów” klas okien, po zarejestrowaniu klasy okna, porzucimy naszą "strukturę klasy okna" bez skutków ubocznych .

Tworzenie okna

Po zarejestrowaniu nazwy klasy okna, stworzymy okno używając CreateWindowEx.

Chociaż jest wiele argumentów dostarczymy głównie cztery (wyzerowując pozostałe): 1) nazwa klasy, 2) lokacja okna, 3) rozmiar okna, i 4) opcja startu z widzialnym oknem. Ostatnie trzy mogą być wyzerowane dostarczając chęci do wykonania odpowiednika wyświetlenia funkcji gdziekolwiek.

Uchwyt instancji modułu musi być "posiadaczem" danego okna klasy, wiec razem wyjątkowo identyfikują klasę okna. To posiadanie jest ustalone przez RegisterClassEx poprzez pole hInstance w strukturze danych WNDCLASSEX .

```
WS_VISIBLE equ  10000000h
```

```
    extrn      CreateWindowEx:near

    .code
    push    large  0            ; lpParam
    push    [wc.wcxInstance]    ; hInstance
    push    large  0            ; menu hmenu
    push    large  0            ; hwnd rodzica
    push    large  200          ; wysokość
    push    large  200          ; szerokość
    push    large  100          ; y
    push    large  100          ; x
    push    large WS_VISIBLE    ; Styl
    push    large  0            ; Tekst okna  (nagłówek)
    push    offset wndclsname   ; Nazwa klasy
    push    large  0            ; poszerzony styl
    call    CreateWindowEx
```

Pętle komunikatów

Po stworzeniu i wyświetleniu okna, możemy potem zadziałać z wysyłaniem komunikatów do naszego programu. Nasz minimalny program wielokrotnie pobiera komunikaty z kolejki komunikatów poprzez GetMessage. Robimy to pętlą komunikatów. Pętla ta jest czasami nazywana pompą komunikatów.

Jeśli jakiś komunikat pochodzi z jednej z różnych postaci SendMessage, GetMessage będzie wywoływał bezpośrednio właściwą procedurę okna. W przeciwnym razie komunikat jest kopiowany do lokalnego bufora (jednego z parametrów GetMessage), a sterowanie jest zwracane do programu wywołującego GetMessage. Wartość zero (FALSE) jest zwracana jeśli GetMessage odzyskuje komunikat WM_QUIT.

W naszym programie, pętla komunikatów jest opuszczana jeśli jest uzyskany komunikat WM_QUIT. W przeciwnym razie wywołujemy właściwą procedurę okna używając DispatchMessage.

```
MSG struc           ;komentarze pokazują nazwy C/C++
msgHwnd     dd  ?   ; hwnd
msgMessage dd  ?    ; komunikat
msgWparam    dd  ?  ; wParam
msgLparam    dd  ?  ; lParam
msgTime     dd  ?   ; czas, czas wysłania komunikatu
msgPtX      dd  ?   ; pt.x, pozycja kursora
msgPtY      dd  ?   ; pt.y, pozycja kursora
MSG ends

    extrn GetMessage:near,DispatchMessage:near

    .data
msgbuf MSG     <>

    .code
msg_loop:
    push    large  0     ; uMsgFilterMax
    push    large  0     ; uMsgFilterMin
    push    large  0     ; hWnd  (filter),  0  = all windows
    push    offset msgbuf  ; lpMsg
    call    GetMessage   ;zwraca FALSE jeœli WM_QUIT
    or     eax,eax
    jz     end_loop

    push    offset msgbuf
    call    DispatchMessage

    jmp     msg_loop

end_loop:
```

Zakończenie programu

Wychodzimy z programu przez ExitProcess.

```
    extrn ExitProcess:near

    .code
    push    large  0    ;  (błąd) kod zakończenia
    call    ExitProcess
```

Procedura okna

Z wyjątkiem pewnych inicjalizacji i czyszczenia program GUI jest oczekiwany wszędzie wewnątrz różnych procedur okna. Procedura okna może być wywoływana przez sam Windows (np. Kiedy jest najpierw tworzone), lub może być wywoływane przez DispatchMessage.

Kiedy okno jest zamykane (np, Alt-F4), domyślną akcją, obsługiwaną przez DefWindowProc, jest jego usunięcie. Okno jest usuwane z ekranu, a komunikat WM_DESTROY jest wysyłany do procedury okna. Jeśli chcemy zakończyć nasz program przez zamknięcie jakiegoś określonego okna, wtedy okno musi odpowiedzieć na powiązany komunikat z zamykanym oknem. Zalecanym komunikatem jest WM_DESTROY. Nasze procedura okna robi to dla jednego i tylko jednego okna, wywołując PostQuitMessage jako odpowiedź. Potem procedura okna jest zamykana, komunikat WM_QUIT będzie odebrany przez GetMessage w naszej pętli komunikatów.

Wszystkie inne komunikaty zostają przetworzone przez DefWindowProc.

Jedną z zalet przejścia z kodu 16-bitowego na kod 32-bitowy jest dostępność dodatkowych trybów . Wszystkie osiem podstawowych 32-bitowych rejestrów może być użytych w złożonych trybach adresowania.

WndProc wykorzystuje to przez zastosowanie ESP do uzyskania dostępu do swoich argumentów. Użyjemy rozłożonej postaci [ESP+4+4] do pokazania, że część offsetu przeskakuje EIP na stosie a pozostałą część jest offsetem drugiego argumentu (ID komunikatu).

```
    extrn DefWindowProc:near,PostQuitMessage:near


WM_DESTROY equ  2

    .code
WndProc:
    cmp      dword ptr  [esp+4+4],WM_DESTROY jne
    DefWindowProc

start_destroy:
    push    large  0
    call     PostQuitMessage

    xor      eax,eax
    ret    16
```

Ciągi, ANSI i Unicode

Dana znaku może być albo ANSI (8-bitów) albo Unicode (16-bitów). W konsekwencji, większość (nie wszystkie!) funkcje Win32 które obsługują ciągi, bezpośrednio lub pośrednio, mają dwie wersje - jedną dla ANSI i jedna dla Unicode. Funkcje są rozróżniane poprzez dodanie A lub W (rozszerzony) do nazwy funkcji.

Unicode jest bardziej wydajny na NT, ale Win95 ma ograniczone zdolności Unicode. Więc z wyjątkiem kilku przypadków programy Win95 używają znaków i ciągów ANSI.

Większość ciągów Windows występuje w standardzie z zerowym znakiem zakończenia (NUL)

Linkowanie

Jeśli jesteś programistą DOS ,możesz zastanawiać się ,jakie INT'y są używane do wywołania podprogramów API takich jak ExitProcess. Odpowiedź to żadne. Funkcje API będące wywoływanymi są umieszczone w plikach DLL (biblioteki łączone dynamicznie). (Jak program przełącza pomiędzy "aplikacją" a trybem "jądra" OS'a wymaga wiedzy na temat jak 386 I późniejsze chipy implementują tryb chroniony I stronicowanie pamięci.)

Linker nie dodaje kodu DLL do pliku EXE . Zamiast tego, tworzy odniesienie do punktu wejścia DLL w pliku EXE. Jest to robione poprzez dowiązanie biblioteki importu . Musisz przekazać linkerowi jakie biblioteki importów są konieczne. Niestety, nie ma standardowej postaci biblioteki importów. Poniżej jest przykład stosowania bibliotek Borlanda i Microsoftu.

Ważną cechą konieczną w twoim assemblerze jest zachowanie wielkości liter, ponieważ nazwy Win32 API rozróżniają pomiędzy małymi i dużymi literami. W wielu asemblerach opcja ta jest zazwyczaj wyłączona.

Borland używa jednej biblioteki , import32.lib, dla zbioru źródłowego funkcji Win32 .Nazwa łącza importowanego jest taka sama jak nazwa punktu wejścia w DLL .

```
CreateWindowEx      equ  <CreateWindowExA>
DefWindowProc      equ  <DefWindowProcA>
DispatchMessage     equ  <DispatchMessageA>
GetMessage         equ  <GetMessageA>
GetModuleHandle       equ   <GetModuleHandleA>
RegisterClassEx     equ  <RegisterClassExA>
```

Microsoft używa kilku bibliotek, jedna na DLL. Nazwy łącz dla funkcji Win32 są nazwami "dekorowanymi" . Zasada jest prosta: znak podkreślenia (_) dołączony znak at (@) i liczba bajtów argumentu dziesiętnie. Tak więc wersja ANSI GetMessage, który ma cztery argumenty DWORD ,jest linkowany jako `_GetMessageA@16` (_ + GetMessage + A + @ + 16 [= 4*4]).

Schemat ten jest głównie dla kompilatorów Microsoft . Nazwa linkowana nie może mieć takiego związku z nazwą punktu wejścia, ale schemat ten jest używany przez biblioteki Microsoft Win32.

```
CreateWindowEx      equ  <_CreateWindowExA@48>
DefWindowProc      equ  <_DefWindowProcA@16>
DispatchMessage     equ  <_DispatchMessageA@4>
ExitProcess        equ  <_ExitProcess@4>
GetMessage         equ  <_GetMessageA@16>
GetModuleHandle       equ   <_GetModuleHandleA@4>
PostQuitMessage       equ   <_PostQuitMessage@4>
RegisterClassEx     equ  <_RegisterClassExA@4>
```

Wprowadzenie do komunikatów

Komunikaty są sercem programowania Windows GUI . Prawie wszystkie programy GUI program są wyzwalane przez komunikaty. Pisanie programów, które wielokrotnie oczekują na komunikaty, prowadzi do stylu programowania sterowanego zdarzeniami.

Wysyłanie i przekazanie

Zazwyczaj pętle komunikatów są używane do wysyłania odebranych komunikatów .A komunikaty te są zazwyczaj wysyłane przez wywoływanie procedur okna. Jednakże, komunikaty mogą być albo wysłane do okna lub przekazane do wątku. Zatem komunikat przekazany nie musi wywoływać okna procedury..
Najoczywistszym przykładem przekazania komunikatu, który nie idzie do okna jest WM_QUIT, który zazwyczaj jest przekazywany z PostQuitMessage.

Pętle komunikatów zazwyczaj oczekują na komunikaty przez wywołanie GetMessage. Jeśli przyszedł jakiś "wysłany" komunikat, GetMessage bezpośrednio wywołuje właściwą procedurę okna (omijając widzialny kod obsługi komunikatów). Jeśli są jakieś "przekazane" komunikaty, wracają do programu wywołującego i pozwalają pętli komunikatów obsłużyć ten komunikat. Jeśli okno jest powiązane z komunikatem, właściwa procedura okna jest zazwyczaj wywoływana poprzez DispatchMessage.

Pewne komunikaty nie muszą być przetwarzane przez pętlę komunikatów. Te komunikaty są jedynymi wysyłanymi przez SendMessage w jednym wątku do okna w tym samym wątku. W tym przypadku, procedura okna jest wywoływana bezpośrednio

Windows tworzy wiele komunikatów, ale typowa aplikacja będzie dostarczała własnej odpowiedzi tylko na mały ich procent. Większość komunikatów może być sklasyfikowanych jako powiadomienie - -- komunikat "to się zdarzyło" lub "to jest o tym co się zdarzyło" . Kilka komunikatów np, WM_PAINT, może być interpretowane jako polecenia.

Identyfikatory komunikatów

Komunikat jest pakietem informacji , która zawiera numer identyfikacyjny. Chociaż możemy użyć numerów wierszy, poprzez konwencję podajemy ich wszystkie nazwy. To staje się "nazwami komunikatów".

Numery / nazwy komunikatów są alokowane jak następuje:

- 0 do WM_USER - 1
  Standardowe okno komunikatu. To są WM_ komunikaty.
- WM_USER do WM_APP - 1
  Komunikaty określonej klasy.
- WM_APP do 0BFFFh
  Komunikaty określonej aplikacji.
- 0C000h do 0FFFFh
  Komunikaty globalnego systemu. Funkcja RegisterWindowMessage tworzy komunikat w tym zakresie.
- 10000h do 0FFFFFFFFh
  Komunikaty wewnętrzne. Ponieważ wymagają one więcej niż 16 bitów, nie mogą być używane do komunikacji z programami 16-bitowego Windows .

Nazwa komunikatu WM_USER ma wartość 0400h a WM_APP ma wartość 8000h.

Przykładowy program

Nasz przykładowy program , winclick.asm, odpowiada na kliknięcie lewego i prawego przycisku myszki przez zmianę paska tytułowego. Poniżej pokazujemy jaka jest różnica z podstawowym programem GUI

Tytuły

Definiujemy dwa tytuły i ustawiamy nasze okno na wyświetlenie jednego z nich na starcie. Szerokość okna jest dosyć duża dla uniknięcia przycięcia tekstu tytułu.

```
DEFAULT_STYLE equ WS_VISIBLE  + WS_OVERLAPPED  + WS_CAPTION  + WS_SYSMENU  +  \
                  WS_THICKFRAME  + WS_MINIMIZEBOX  + WS_MAXIMIZEBOX
DEFAULT_EXSTYLE  equ  WS_EX_WINDOWEDGE   +  WS_EX_CLIENTEDGE
DEFAULT_X    equ  100
DEFAULT_Y    equ  100
DEFAULT_WIDTH equ  400
DEFAULT_HEIGHT equ  200

    .data

    align  4

cwargs CREATEARGS  <DEFAULT_EXSTYLE,wndclsname,title1,DEFAULT_STYLE,  \
          DEFAULT_X,DEFAULT_Y,  DEFAULT_WIDTH,DEFAULT_HEIGHT,   0,0,   0,   0>
title1 db 'Left-click to change title',0
title2 db 'Restore title by right-clicking',0
```

Wysyłanie

Poniżej mamy jeden ze sposobów wysyłania komunikatów. Nie jest zły ponieważ jest kilka komunikatów. Duży zbiór komunikatów byłby łatwiejszy do obsługi z tablicą adresów.

Każdy program obsługi komunikatów używa dokładnie takiego samego bloku listy argumentów (możemy wywołać go pakietem komunikatu) jako WndProc.

```
    extrn DefWindowProc:near

    .code
WndProc:
    mov     eax,[esp+4+4]       ;ID komunikatu
    cmp     eax,WM_LBUTTONDOWN  ;naciśnięty lewy przycisk myszki
    je      left_mouse_down
    cmp     eax,WM_RBUTTONDOWN  ; naciśnięty prawy przycisk myszki
    je      right_mouse_down
    cmp     eax,WM_DESTROY      ;start destrukcji okna
    je      start_destroy
    jmp     DefWindowProc       ;przetwarzanie innych komunikatów
```

Komunikaty myszki

Tu mamy jak odpowiadać na komunikaty myszki. Wybrany tytuł zależy od tego jaki przycisk jest naciśnięty.
Kod jest niezoptymalizowany aby lepiej pokazać wyraźniej skąd pochodzi parametr hwnd (+4 przeskakuje EIP na stosie, a +0 jest offsetem pierwszego argumentu .)

```
    extrn    SetWindowText:near

    .code
left_mouse_down:
    mov     eax,[esp+4+0]  ;pobranie hwnd przed zmianą ESP
    push      offset title2
    push      eax
    call      SetWindowText

    xor     eax,eax
    ret     16

right_mouse_down:
    mov     eax,[esp+4+0]  ;pobranie hwnd przed zmianą ESP
    push      offset title1
    push      eax
    call      SetWindowText

    xor     eax,eax
    ret     16
```

Wprowadzenie do myszki i kursora

Z wyjątkiem danych tekstowych , myszka jest preferowana jako postać wyjściowa. Przesunięcie myszki jest używane do sterowania przesunięciem kursora. Kursor daje użytkownikowi wizualny sygnał tego co dzieje się kiedy naciśnie przyciski myszki..

Komunikaty myszki

Sama myszka daje tylko dwa rodzaje informacji, przesunięcie i kliknięcie przycisków myszki. Są one tłumaczone na komunikaty Windows. Komunikaty odbierane przez okno zależą od tego gdzie jest umieszczony gorący punkt kursora.

Każde okno ma zarówno obszar klienta jak i obszar nie kliencki. Jeśli gorący punkt kursora jest umieszczony w obszarze nie klienckim , są przekazywane nie klienckie komunikaty myszki . Dla celów tego wprowadzenia

zignorujemy je.

Jeśli gorący punkt kursora jest umieszczony w obszarze klienta, komunikaty odbierane przez okno są następujace

Przesunięcie myszki:
WM_MOUSEMOVE

Przyciski myszki lewy i prawy:
WM_LBUTTONUP
WM_LBUTTONDOWN
WM_LBUTTONDBLCLK
WM_RBUTTONUP
WM_RBUTTONDOWN
WM_RBUTTONDBLCLK

Środkowy przycisk myszki (tylko jeśli jst 3-przyciskowa):
WM_MBUTTONUP
WM_MBUTTONDOWN
WM_MBUTTONDBLCLK

Współrzędne kursora (myszki)

Mówiąc ściśle, nie ma takiej rzeczy jak współrzędne myszki - tylko współrzędne kursora. Jednakże, łatwo jest popełnić pomyłkę, ponieważ komunikaty myszki obejmują informacje o współrzędnych.

Więcej o komunikatach i wątkach

Wątki i okna

Za każdym razem kiedy jest tworzone okno, jest "związane" z bieżącym wątkiem. Wątek ten jest jedynym wątkiem , który może wykonać procedurę okna dla tego określonego okna. W konsekwencji jest to jedyny wątek, który może przetwarzać komunikat dal tego okna. W tym sensie wątek “posiada” okno.
Jednakże inne okno (tej samej klasy) może by stworzone w innym wątku, pozwalając dwóm oknom przetwarzać komunikat równocześnie. Ponieważ więcej niż jeden wątek może być wykonywany na danej procedurze okna , poniżej omówimy funkcje będące “wywoływanymi przez wątki”

Wywłaszczanie czy nie wywłaszczanie?

Podczas gdy system wątków jest z wywłaszczeniem, system komunikatów już nie. Watek jest wywłaszczony jedynie przy pozwoleniu na uruchomienie innych wątków. Wątek nigdy nie jest z wywłaszczeniem zmieniając swój punkt wykonywania. A zatem wątek , generalnie, nie odpowiada bezpośrednio na nowy komunikat.

Jest tak ponieważ system komunikatów ma trzy podstawowe składniki: nadawcę , kolejki i odbiorcę.
Podprogram nadawczy odkładał będzie komunikat do przeznaczonej kolejki, a podprogram odbiorczy musi odebrać go zanim może go przetwarzać program obsługi komunikatów. Jeśli podprogram odbiorczy nie jest wywołany przez wątek, wątek nie może obsłużyć danych komunikatów z kolejki. Windows dostarcza dwóch funkcji dla odbioru komunikatów: GetMessage i PeekMessage.

Wątki i kolejki komunikatów

"Przekazane" komunikaty są dostarczane wątkom. To oznacza , ze każdy wątek , który oczekuje komunikatów będzie miał swoją własną kolejkę komunikatów. Ponieważ każde okno jest powiązane z dokładnie jednym oknem, PostMessage może określić jaki wątek przekazać (bez konieczności argumentu wątku).

"Wysłane" komunikaty są, pozornie, dostarczane do okien. Ponieważ wątki wykonują swoją pracę obsługując odebrane komunikaty, "wysłane" komunikaty muszą w rzeczywistości być dostarczone wątkom. Z wyjątkiem

pewnych komunikatów wątków, takie komunikaty muzą być umieszczone w kolejce komunikatów wątków -- które są powiązane z oknem przeznaczenia . Tak czy inaczej ta kolejka jest taka sam jak nieważna kolejka "PostMessage" . Jest ważna tylko dla systemu , który wie czy komunikat jest z funkcji "SendMessage" czy funkcji "PostMessage" .

Blokada wątków i SendMessage

Kiedy wysyłamy komunikat do okna, funkcja wysyłająca może musieć czekać na odpowiedź. Podczas oczekiwania , mówimy, że wątek który wywołał funkcję jest blokowany. Sytuacja ta zabezpiecza wątek przed obsługą innego komunikatu dla pozostałych okien jakie „posiada". A zatem okno zawartość okna nie ulegnie aktualizacji, a kiedy okno jest odkrywane nie będzie ponownie przerysowane.

Kiedy SendMessage wysyła komunikat do okna w innym wątku, oczekuje na odpowiedź ,która jest udzielana kiedy wątek odbiorczy wywoła ReplyMessage. Jasna odpowiedź nie jest zazwyczaj konieczna ponieważ kiedy procedura okna się kończy, ReplyMessage jest wywoływana automatycznie . Jasna ReplyMessage będzie konieczna aby uniknąć zakleszczenia jeśli jest łańcuch nieskończonych wywołań SendMessage , które wiążą się z więcej niż jednym wątkiem w sposób cykliczny
Na przykład:

Łańcuch SendMessage

(window A, thread 1) --> (B, thread 2) --> ... --> (C, thread n) --> (D, [back to] thread 1)

zakleszczy się kiedy C (w wątku n) wyśle komunikat do D, jeśli żadna z procedur okna nie wywoła ReplyMessage. Jeśli jakaś wywołana procedura okna (inna niż D) wywoła ReplyMessage zanim wywoła SendMessage, łańcuch zostanie przełamany.

Wysyłanie takich samych wątków

SendMessage, funkcja blokowania wątku, jest używana dosyć często do wysyłania komunikatu do okna w tym samym wątku. Przy normalnych zasadach, funkcja ta odłożyłaby komunikat do kolejki a potem wymusiła wątek oczekujący na odpowiedź. Ponieważ oczekuje ,nie może odpowiedzieć - zakleszczenie W tym specjalnym przypadku wysyła "taki sam wątek", Windows będzie bezpośrednio wywoływał właściwą procedurę okna jeśli była podprogramem. W efekcie jest to wątek wywłaszczony dla celu odpowiedzenia na nowy komunikat.

Ponieważ nie wymaga wycofania komunikatu z kolejki, zachowanie to pozwala aby komunikat był przetwarzany bez pętli komunikatu. Na przykład CreateWindowEx może wywołać twój specjalny kod WM_CREATE I narysować ramkę okna zanim twoja aplikacja wprowadzi pętlę komunikatów. Również każde okno (stworzone w tym samym wątku) może być bezpośrednio aktualizowane przez wysłanie do niego właściwego komunikatu.

Ale wiedz, że bezpośrednia odpowiedź wystąpi tylko kiedy komunikat jest wysłany (przez SendMessage) do okna w tym samym wątku.

Formy przekazywania komunikatów

SendMessage jest funkcją blokady wątków. Zawsze oczekuje na odpowiedź. SendMessageTimeout blokuje jak SendMessage, ale odblokowuje jeśli odbiorca zbyt długo zwleka z odpowiednią. Ty określasz przedział czasowy.

SendNotifyMessage jest funkcja nie blokującą. Nie oczekuje na odpowiedź, chyba ,że okno przeznaczenia jest w tym samym wątku.

SendMessageCallback jest inną funkcją nie blokującą Ona również nie oczekuje na odpowiedź, chyba ,że okno przeinaczenia jest w tym samym wątku. Określasz podprogram wywołania zwrotnego, który będzie wykonywany kiedy odpowie wątek odbiorczy.

PostMessage jest zupełnie nie blokująca. Nigdy nie oczekuje na odpowiedź. Musisz określić okno . Okno to określa jaki wątek będzie odpowiadała na komunikat.

PostThreadMessage jest nie blokująca I nie może określać okna. Musisz określić wątek.

PostQuitMessage jest nie blokująca i przekazuje komunikat WM_QUIT do bieżącego wątku

Programowanie w języku assemblera x86 dla Win32

Łącza sieciowe i inne zasoby

Zasoby sieciowe

Win32ASM Message Board
Sieciowa tablica dla omawiania programowania Win32 w ASM.

Iczelion's Win32 Assembly Language Page
Strona ta posiada doskonały tutorial o ASM Wiin32. Wykorzystuje cechy MASM. Możesz nawet ściągnąć MASM32, kompletny zestaw rozwojowy dla Win32, wliczając w to assembler i linker Microsoft na potrzeby tutorialu Iczeliona. Ma również duży zbiór linków o ASM w Win32 I pliki do ściągnięcia. Jeśli masz problemy z dostępem do tej wyśmienitej strony spróbuj mirrorów

http://spiff.tripnet.se/~iczelion/
http://win32assembly.online.fr/
http://users.daex.ufsc.br/~iczelion/
http://203.157.250.93/win32asm/

Platform SDK
Kolekcja użytecznych zasobów, które Microsoft pozwala ściągnąć za darmo. Podstawowym komponentem jest Build Environment. Komponent Toolkit jest również prawdopodobnie użyteczny .

MSDN Online Documentation
Nawiguj zużywając spisu treści. Przegląd i opisy źródłowych funkcji API udokumentowanych pod Platformę SDK.

Assembly Language Programming
Zawiera szczegółowy przewodnik programowania Intel x86. Obejmuje kurs programowania w 32 bitowym assemblerze, którego używa Randy HLA (Assembler Wysokiego Poziomu).

Intel Secrets
Nieudokumentowany Intel.

File Formats at wotsit.org
Kilka popularnych formatów plików, obejmujących pliki Windows i binaria, dostępne jako linki lub pliki do ściągnięcia.

GoRC resource compiler
Darmowe kompilatory zasobów dla przetwarzania plików .RC , standardowych sposobów tworzenia zasobów.

ALINK x86 linker
Darmowy linker dla tworzenia EXE I DLLA Win32 zarówno z plików OMF jak i MS-COFF . można również zbudować program DOS . Dobre dla użytkowników, którzy chcą połączyć biblioteki COFF Microsoftu.

WALK32 v1.00
Niekonwencjonalny system linkowania oparty o MASM 6.xx dla tworzenia EXE i DLL Win32 .