Programowanie niskopoziomowe

Ćwiczenia laboratoryjne w środowisku Visual Studio 2019

Labolatoria 9

„Tryb graficzny”

Contents

1 Na laboratorium

1. Zasada działania aplikacji graficznej w środowisku WIN32,
2. Struktury oraz funkcje niezbędne do tworzenia nowego okna za pomocą API WIN32,
3. Obługa zdarzeń związanych z oknem,
4. Tworzenie elementów wewnątrz istniejącego okna(pod-okna),
5. Komunikacja między oknami.

2 Zadania

1. Napisz program wyświetlający główne puste okno aplikacji o nazwie <twoje nazwisko>(bez pętli obsługi komunikatów),

2. Wypisz na kanwie okna imię i nazwisko osoby siędzącej po twojej prawej stronie(nadal bez pętli obsługi komunikatów),

3. Dodaj pętlę obsługi komunikatów i obsłuż komunikat o utworzeniu okna. Po wykryciu komunikatu dodaj przycisk(pod-okno) do okna głównego(4.12).

4. Rozpoznaj wiadomość informującą o kliknięciu na przycisk i wyświetl bezpośrednio na kanwie głównej aplikacji informacje o kliknięciu(4.12,4.13).

5. Dodaj pole edycyjne do okna głównego(4.16).

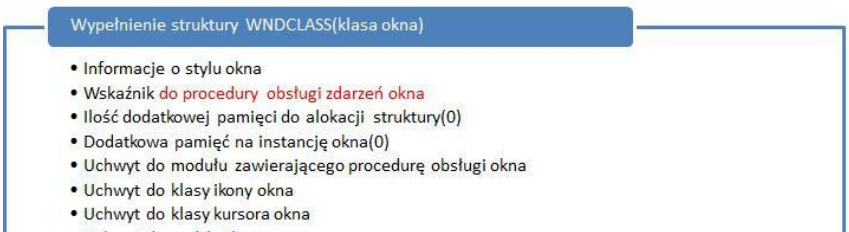
6. Dodaj drugi przycisk, i rozpoznaj na nim zdarzenie kliknięcia i skopiuj napis z pola tekstowego na kanwę głównego okna(4.12 i 4.16).

7. Zadanie dodatkowe od prowadzącego.

---

3 Wprowadzenie

Aplikacja graficzna zawsze posiada okno. Każdy graficzny element okna jest rozpatrywany również jako okno. Komponenty systemu Windows współdziałają za pomocą komunikatów o zdarzeniach. Aplikacja analizuje i obsługuje komunikat w tzw. procedurze okna. System Windows otrzymuje wskaźnik na procedurę okna aplikacji w strukturze *WNDCLASS*. Programista musi przygotować tą strukturę i zarejestrować ją za pomocą funkcji *RegisterClass*. Na bazie struktury *WNDCLASS* system Windows tworzy okno, gdy wywołana zostanie funkcja *CreateWindowsEx*, a następnie wyświetla okno po wywołaniu funkcji *ShowWindow*. System Windows kieruje ko-munikaty o zdarzeniach do procedury okna, na podstawie których procedura może podjąć od-powiednie działania. Komunikaty nie są wysyłane bezpośrednio do procedury okna. Aby okno otrzymywało komunikaty, w aplikacji musi działać tzw. pętla obsługi komunikatów. Wewnątrz pętli aplikacja wywołuje funkcję *GetMessage*, która pobiera bieżący komunikat z kolejki komuni-katów Windows. W pętli komunikatów aplikacja wywołuje funkcję *DispatchMessage* i przez nią przekazuje komunikat do systemu Windows, który kieruje komunikat do procedury okna. Przed wywołaniem funkcji *DispatchMessage* zwykle ma miejsce wywołanie funkcji *TranslateMessage*, która przekształca część komunikatów od klawiatury. Wyjście z pętli obsługi komunikatów musi nastąpić, gdy funkcja *GetMessage* zwróci wartość zerową. Podsumowując, aplikacja graficzna dla Windows ma następującą strukturę:



Rysunek 1: Przebieg programu

Dodatkowo należy napisać procedurę okna w postaci podprogramu z tradycyjną nazwą Wnd-Proc. W tej procedurze (funkcji) jest zaimplementowana specyfika aplikacji.

4 Pomoc

4.1 Struktura WNDCLASS

Struktura WNDCLASS zawiera informacje na podstawie których tworzona jest klasa okien gra-ficznych wykorzystywanych w aplikacji. Więcej informacji tutaj.

```
typedef struct tagWNDCLASSA {
  UINT      style;
  WNDPROC   lpfnWndProc;
  int       cbClsExtra;
  int       cbWndExtra;
  HINSTANCE hInstance;
  HICON     hIcon;
  HCURSOR   hCursor;
  HBRUSH    hbrBackground;
  LPCSTR    lpszMenuName;
  LPCSTR    lpszClassName;
} WNDCLASSA, *PWNDCLASSA, *NPWNDCLASSA, *LPWNDCLASSA;
```

Parametry:

1. style - style które zostaną wykorzystane w klasie. Dostępne parametry tutaj.

2. lpfnWndProc - wkaźnik do procedury okna. Prototyp możemy znaleźć tutaj. W celu ob-sługi zdarzeń musimy stworzyć własną implementację(4.7).

3. cbClsExtra - ilość dodatkowych bajtów niezbędna do alokacji stuktury klasy okna. Do-myślnie system inicjalizuje z wartoscią 0.

4. cbWndExtra - ilość dodatkowo zarezerwowanych bajtów za instancją okna. Domyślnie 0.

5. hInstance - uchwyt do instancji która zawiera procedurę obługi okna dla tej klasy(pobrany za pomocą 4.3).
6. hIcon - Uchwyt do klasy ikony(pobrany za pomocą 4.4).
7. hCursor - Uchwyt do klasy kursora(pobrany za pomocą 4.5).
8. hbrBackground - uchwyt do pędzla tła(pobrany za pomocą 4.6).
9. lpszMenuName - wskaźnik do pliku zasobów z menu. W przypadku przekazania 0 okno nie będzie posiadało menu.
10. lpszClassName - wskaźnik do nazwy klasy okna(null terminated string, max 255 znaków).

Prototyp:

```
WNDCLASS struct
        clsStyle            DWORD    0
        clsLpfnWndProc      DWORD    0
        clsCbClsExtra       DWORD    0
        clsHInstance        DWORD    0
        clsHIcon            DWORD    0
        clsHCursor          DWORD    0
        clsHbrBackground    DWORD    0
        clsLpszMenuName     DWORD    0
        clsLpszClassName    DWORD    0
WNDCLASS ends
```

Fragment danych

```
CSstyle EQU CS_HREDRAW + CS_VREDRAW + CS_GLOBALCLASS
wndc WNDCLASS S
```

Fragment kodu

```
mov [ wndc . clsStyle ] , CSstyle ; wypelnienie pierwszego elementu struktury
```

4.2 Rejestracja klasy okna

Funkcja RegisterClassA z bilbioteki User32.lib pozwala na zarejestrowanie klasy okien, które będą tworzone w ramach tej aplikacji. Więcej informacji tutaj.

```
ATOM RegisterClassA(
  const WNDCLASSA *lpWndClass
);
```

Parametry:

1. *lpWndClass - wskaźnik do struktury WNDCLASS(4.1).

Funkcja zwaraca uchwyt do utworzonej klasy. W przypadku błędu do rejestru EAX zapisana jest wartość 0, by uzyskać dodatkowe informacje o błędzie możemy wykorzystać procedurę Ge-tLastError(poprzednie laboratoria).

Prototyp:

```
RegisterClassA      PROTO  : DWORD
```

Fragment danych

```
Wndc                WNDCLASS    <?>
```

Fragment kodu

```
INVOKE RegisterClassA OFFSET wndc
```

4.3 Pobranie uchwytu do modułu

Funkcja GetModuleHandleA z biblioteki Kernel32.lib pobiera uchwyt do modułu aplikacji.

```
HMODULE GetModuleHandleA(
  LPCSTR lpModuleName
);
```

Parametry:

1. lpModuleName - parametr na podstawie którego pobierany jest uchwyt. Możliwe jest podanie pliku .dll lub .exe. W przypadku przekazania 0(null) pobierany jest uchwyt do aplikacji z której procedura została wywołana.

Uchwyt zapisywany jest do rejestru EAX. W przypadku błędu do rejestru EAX zapisana jest wartość 0, by uzyskać dodatkowe informacje o błędzie możemy wykorzystać procedurę GetLa-stError(poprzednie laboratoria).

Prototyp:

```
GetModuleHandleA    PROTO  : DWORD
```

Fragment danych

```
hinst        DWORD    0
```

Fragment kodu

```
invoke    GetModuleHandleA ,  0
mov  hinst ,  EAX
```

4.4 Pobranie uchwytu ikony

Funkcja LoadIconA z bilbioteki User32.lib pozwala na pobranie uchwytu do ikony. Więcej in-formacji tutaj.

Prototyp:

```
LoadIconA     PROTO  : DWORD ,: DWORD
```

Fragment danych

```
handleIcon                    DWORD    0
```

Fragment kodu

```
INVOKE LoadIconA , 0  IDI_APPLICATION
 mov handleIcon , EAX
```

4.5 Pobranie uchwytu kursora

Funkcja LoadCursorA z bilbioteki User32.lib pozwala na pobranie uchwytu do kursora. Więcej informacji tutaj.

Prototyp:

```
 LoadCursorA    PROTO  : DWORD ,: DWORD
```

Fragment danych

```
handleCursor                  DWORD    0
```

Fragment kodu

```
INVOKE  LoadCursorA ,  0 ,        ID C_ AR RO W
mov  handleCursor , EAX
```

4.6 Pobranie uchwytu do pędzli/czcionek/palet

Funkcja GetStockObject z bilbioteki Gdi32.lib pozwala na pobranie uchwytu do jednego z pędzli, czcionek lub palet. Więcej informacji tutaj.

Prototyp:

```
GetStockObject          PROTO  : DWORD
```

Fragment danych

```
handleBrush          DWORD    0
```

Fragment kodu

```
INVOKE  GetStockObject ,  W H I T E _ B R U S H
mov  handleBrush , EAX
```

4.7 Procedura obsługująca zdarzeń w głównym oknie aplikacji

W celu obsługi zdarzeń/wiadomości od systemu Windows, niezbędna jest implementacja odpo-wiedniej procedury po stronie aplikacji i przekazanie jej adresu w strukturze *WNDCLASS* (4.1). Więcej informacji tutaj.

```
LRESULT CALLBACK WindowProc(
  _In_ HWND   hwnd,
  _In_ UINT   uMsg,
  _In_ WPARAM wParam,
   In  LPARAM lParam
);
```

Parametry:

1. hwnd - uchwyt do okna aplikacji,
2. uMsg - wiadomość na podstawie której możemy podejmować odpowiednie kroki, listę niezbędnych wiadomości znajdziemy w podrozdziale 4.12,
3. wParam - dodatkowe informacje. Zawartość tego parametru jest zależna od parametru *uMsg*,
4. lParam - dodatkowe informacje. Zawartość tego parametru jest zależna od parametru *uMsg*,

Wewnątrz ciała procedury niezbędna jest obsługa nieobsłużonych komunikatów za pomocą pro-cedury *DefWNDProc*(4.11). Przykładowa implementacja z domyślną obsługą wiadomości:

Fragment kodu

```
WndProc PROC uses EBX ESI EDI windowHandle : DWORD , uMsg : DWORD ,
addParam1 : DWORD, addParam2 : DWORD

        INVOKE DefWindowProcA , windowHandle , uMsg , addParam1 , addParam2
         ret

WndProc ENDP
```

4.8 Tworzenie głównego okna aplikacji

Funkcja CreateWindowExA umożliwia utworzenie okna w aplikacji. Więcej informacji tutaj.

```
HWND CreateWindowExA(
  DWORD     dwExStyle,
  LPCSTR    lpClassName,
  LPCSTR    lpWindowName,
  DWORD     dwStyle,
```

```
  int         X,
  int         Y,
  int         nWidth,
  int         nHeight,
  HWND        hWndParent,
  HMENU       hMenu,
  HINSTANCE   hInstance,
  LPVOID      lpParam
);
```

Parametry:

1. dwExStyle - dodatkowe parametry styli dla okna naszej aplikacji, przekazujemy 0,
2. lpClassName - null-terminated string z nazwą do wcześniej utworzonej klasy za pomocą procedury *RegisterClassA*(4.2).
3. lpWindowName - null-terminated string z nazwą do tworzonego okna,
4. dwStyle - styl okna, więcej informacji tutaj,
5. x - początkowa horyzontalna pozycja okna,
6. y - początkowa wertykalna pozycja okna,
7. nWidth - szerokość okna,
8. nHeight - wysokość okna,
9. hWndParent - uchwyt do okna rodzica(jeśli tworzymy okno w oknie to musimy przekazać uchwyt do okna nadrzędnego), jeśli okno go nie posiada to możemy przekazać 0,
10. hMenu - uchwyt do menu, jeśli go nie potrzebujemy może być 0,
11. hInstance - uchwyt do modułu z którym ma być kojarzone okno. Pobrany za pomocą funkcji *GetModuleHandleA*(4.3).
12. lpParam - wskaźnik do dodatkowych parametrów, możemy przekazać 0.

W przypadku błędu funkcja zwróci do rejestru EAX wartość różną od 0, do sprawdzenia błędu możemy wykorzystać procedurę *GetLastErr* z poprzednich laboratoriów. Przykład wykorzysta-nia:

Prototyp:

```
CreateWindowExA PROTO : DWORD ,: DWORD ,: DWORD ,: DWORD ,: DWORD ,: DWORD
,: DWORD ,: DWORD ,: DWORD ,: DWORD ,: DWORD
```

Fragment danych

```
WNDstyle EQU WS_CLIPCHILDREN + WS_OVERLAPPEDWINDOW + WS_HSCROLL +
WS_VSCROLL
Hinst          DWORD      0              ; uchwyt
cname          BYTE       " MainClass " , 0
 hwnd          DWORD      0
 tytul         BYTE       " Pyra " ,0
```

Fragment kodu

```
INVOKE CreateWindowExA , 0 , OFFSET cname , OFFSET tytul
, WNDstyle , 50 , 50 , 600 , 400 , 0 , 0 , hinst , 0
mov hwnd , EAX
```

4.9 Wyświetlenie głównego okna aplikacji

Funkcja *ShowWindow* z biblioteki User32.lib pozwala na aktywację okna, oraz zmianę statusu jego wyświetlania(np minimalizacja, maxymalizacja). Więcej informacji tutaj.

```
BOOL ShowWindow(
  HWND hWnd,
  int  nCmdShow
);
```

Parametry:

1. hWnd - uchwyt do okna aplikacji uzyskany z funkcji *CreateWindowExA*(4.8),
2. nCmdShow - parametr oznaczający w jaki sposób ma zostać wyświetlone okno. Np 1 dla aktywowania i wyświetlenia okna w domyślnej pozycji i rozmiarze.

Prototyp:

```
ShowWindow      PROTO  : DWORD ,: DWORD
```

Fragment danych

```
hwnd            DWORD           0
```

Fragment kodu

```
INVOKE  ShowWindow ,  hwnd ,  SW_SHOWNORMAL
```

4.10 Pobranie kanwy głównego okna

Procedura *GetDC* pozwala na pobranie kanwy głównego okna co umożliwia bezpośrednie przesyłanie wiadomości. Segment danych:

Fragment danych

```
naglow          BYTE            " Naglowek " ,0
rozmN           DWORD           0
hwnd            DWORD           0
hdc             DWORD           0
```

Fragment kodu

```
INVOKE  GetDC , hwnd
mov  hdc , EAX

INVOKE  lstrlenA , OFFSET       naglow
mov  rozmN , EAX
INVOKE TextOutA , hdc ,100 ,100 , OFFSET naglow , rozmN
INVOKE UpdateWindow , hwnd
```

Gdzie *hwnd* to uchwyt do okna głównego z funkcji *CreateWindowExA*(4.8).

4.11 Domyślna procedura obsługi zdarzeń dla okna

Domyślna procedura obsługująca wiadomości przekazane do okna to *DefWNDProc*. Przykład wykorzystania możemy znaleźć w podrozdziale 4.7 w segmencie kodu. Więcej informacji tutaj.

4.12 Typy wiadomości dla WindowProc

Podczas laboratoriów niezbędne będzie nam rozróżnienie 5 rodzajów wiadomości pod parame-trem *uMSG* w procedurze obsługującej zdarzenia naszego okna. Typy te zdefiniowane są pod stałymi w pliku grafika.inc:

1. WM CREATE - wiadomość o utworzeniu okna głównego aplikacji. W tym przypadku powinniśmy obsłużyć dodawanie pod-elementów okna głównego.

2. WM DESTROY - wiadomość o zamknięciu okna głównego aplikacji. W tym momencie powinniśmy zniszczyć obiekty zbędne za pomocą procedury *DeleteObject*, oraz wysłać wiadomość o zamknięciu aplikacji za pomocą procedury *PostQuitMessage*.

3. WM COMMAND - wiadomość informująca o kliknięciu jednego z pod-elementów okna, w tym przypadku lParam jest uchwytem do klikniętego obiektu

4. WM LBUTTONDOWN - wiadomość informująca o wciśnięciu lewego przycisku myszy w ramach głównego okna aplikacji(nie pod-elementu).

5. WM RBUTTONDOWN - wiadomość informująca o wciśnięciu prawego przycisku myszy w ramach głównego okna aplikacji(nie pod-elementu).

4.13 Utworzenie przycisku

W celu utworzenia pod-elementów w ramach głównego okna możemy ponownie wykorzystać funkcję *CreateWindowExA*. Wewnątrz procedury obsługującej zdarzenia okna głównego(4.7) i po wykryciu zdarzenia *WM CREATE* możemy utworzyć przycisk.

Fragment danych

```
BSstyle     EQU   BS_PUSHBUTTON + WS_VISIBLE + WS_CHILD + WS_TABSTOP
tstart      BYTE                                        " Start " , 0
tbutt       BYTE                                        " BUTTON " , 0
hbutt       DWORD                                       0
```

Fragment kodu

```
INVOKE CreateWindowExA , 0 , OFFSET tbutt , OFFSET tstart ,
BSstyle , 10 , 50 , 100 , 40 , windowHandle , 0 , hinst , 0
```

Gdzie windowHandle to parametr procedury *WNDProc,* a hinst to uchwyt do głównego modułu aplikacji.

4.14 Utworzenie prostokątu

Fragment danych

```
Kolor       EQU   000000 FFh ; czerwony ; kolory : G B R
Hbrush      DWORD             0
rt          RECT              <100,50,20,20>
holdbrush   DWORD             0
```

Fragment kodu

```
INVOKE   CreateSolidBrush , kolor
mov  hbrush , EAX
INVOKE  SelectObject , hdc , hbrush
mov  holdbrush ,      EAX
INVOKE   FillRect , hdc , OFFSET  rt , ho ld br us h
```

4.15 Utworzenie comboboxa

W celu wysłania informacji do utworzonego pod-okna możemy wykorzystać funkcję *SendMes-sageA* na temat której więcej informacji znajdziemy tutaj.

Fragment danych

```
CBstyle EQU WS_VISIBLE + WS_CHILD + WS_TABSTOP + WS_BORDER
+ CBS_DROPDOWNLIST + CBS_HASSTRINGS

hcb             DWORD       0
tcb             BYTE        " COMBOBOX " , 0
tnazwacb        BYTE        " " , 0
ttekst          BYTE        " tekst " , 0
ttekst1         BYTE        " tekst1 " , 0
ttekst2         BYTE        " tekst2 " , 0
```

Fragment kodu

```
INVOKE CreateWindowExA , 0 , OFFSET tcb , OFFSET tnazwacb , CBstyle , 120 , 50 , 100 , 140 ,
windowHandle , 0 , hinst , 0
 mov hcb , EAX
 INVOKE SendMessageA , hcb , CB_ADDSTRING ,0 , OFFSET ttekst1
 INVOKE SendMessageA , hcb , CB_ADDSTRING ,0 , OFFSET ttekst2
INVOKE SendMessageA , hcb , CB_SELECTSTRING , -1 , OFFSET ttekst
```

4.16 Utworzenie pola edycyjnego

W celu wysłania informacji do utworzonego pod-okna możemy wykorzystać funkcję SendMessageA na temat której więcej informacji znajdziemy tutaj.

Fragment danych

```
 EDTstyle EQU WS_VISIBLE + WS_CHILD + WS_TABSTOP + WS_BORDER
hedt            DWORD       0
tedt            BYTE        " EDIT " , 0
tnazwaedt       BYTE        " POTATO " , 0
ttekst          BYTE        " tekst " , 0
```

Fragment kodu

```
INVOKE CreateWindowExA , 0 , OFFSET tedt , OFFSET tnazwaedt , EDTstyle , 10 , 100 , 100 , 40
, windowHandle , 0 , hinst , 0
mov hedt , EAX
INVOKE SendMessageA , hedt , WM_SETTEXT ,0 , OFFSET ttekst
INVOKE SetFocus , hedt
```

Gdzie windowHandle to pierwszy parametr procedry WNDProc(uhwyt do okna głównego), a hinst to uchwyt do modułu aplikacji. Pobranie tekstu z pola edycyjnego do bufora:

INVOKE SendMessageA , hedt , WM_GETTEXT ,128 , OFFSET bufor

5 Podziękowania

Za cenne uwagi dotyczące laboratoriów:

II rok inf stacjonartne 2016-2017

1. Kamil B.
2. Marta K.