

**POLITECHNIKA CZĘSTOCHOWSKA**  
**WYDZIAŁ INŻYNIERII MECHANICZNEJ I INFORMATYKI**



**Dokumentacja Projektu**  
**Prognoza pogody**

Piotr Zyszczak, Artur Śnioszek, Damian Łukasik

Nr albumu: 113066, 113055, 112993

Kierunek: Informatyka

Studia: stacjonarne

Poziom studiów: II

*Częstochowa, 2016*

## Spis treści

1	Cel i zakres projektu	3
2	Wykorzystane technologie	4
3	Implementacja	6
4	Opis użycia	28
5	Podsumowanie	30

## 1. Cel i zakres projektu

Zaimplementować program zawierający technologie takie jak:

- System okienkowy
- Grafika rastrowa oparta o GDI, Directx lub OpenGL
- Wielowątkowość
- Połączenie do bazy danych SQL
- Połączenie sieciowe i obsługa sieci na poziomie gniazd z przejściem układu I/O na system wiadomości windows(R) lub nowy watek z obsługą komunikacji sieciowej w technologii z obsługą gniazd bez przejścia z układu I/O na wiadomości systemu windows(R) (winsock.dll)

Zdecydowano się więc na serwis pogodowy, który będzie pobierał dane z internetu dzięki zapytaniom http. Połączenie z bazą będzie tylko demonstracją technologii.

## 2. Wykorzystane technologie

Windows API, lub krócej: WinAPI – interfejs programistyczny systemu Microsoft Windows – jest to zbiór funkcji, stałych i zmiennych umożliwiających działanie programu w systemie Microsoft Windows. Zbiór ten jest obszerny i zawiera funkcje do tworzenia okien programów, elementów interfejsu graficznego, obsługi zdarzeń oraz umożliwiające dostęp do innych aplikacji, funkcji sieciowych czy sprzętu w komputerze. Mianem WinAPI określamy standardowe funkcje przychodzące wraz z plikami bibliotek DLL (w 16-bitowych wersjach z rozszerzeniem .EXE) dostarczanych z systemem, np. kernel32.dll, user32.dll, gdi32.dll, wsock32.dll, znajdującymi się w katalogu /WINDOWS/system32. Liczba plików bibliotek wzrasta w nowszych wersjach systemu Microsoft Windows. Może to powodować pewne problemy z uruchomieniem aplikacji napisanej dla starszej wersji systemu. Ze względu na dużą popularność systemu Microsoft Windows, obecnie większość środowisk programistycznych posiada zaimplementowane odpowiednie pliki nagłówkowe umożliwiające korzystanie z WinAPI.

GDI (*ang. Graphics Device Interface*) - komponent Microsoft Windows pozwalający na odwzorowanie grafiki na urządzeniach zewnętrznych, np. drukarkach i monitorach.

ODBC (*ang. Open DataBase Connectivity*) - otwarte łącze baz danych) - interfejs pozwalający programom łączyć się z systemami zarządzającymi bazami danych. Jest to API niezależne od języka programowania, systemu operacyjnego i bazy danych. Standard ten został opracowany przez SQL Access Group we wrześniu 1992 roku. W skład ODBC wchodzi wywołania wbudowane w aplikacje oraz sterowniki ODBC. Pierwsza implementacja ODBC pojawiła się w systemie Microsoft Windows, lecz obecnie możliwe jest korzystanie z niego na platformach Unix, OS/2 oraz Macintosh. W systemach bazodanowych typu klient-serwer (np. Oracle lub PostgreSQL) sterowniki dają dostęp do silnika baz danych, natomiast w programach dla komputerów osobistych sterowniki sięgają bezpośrednio do danych. Mechanizm ODBC współpracuje zarówno z bazami obsługującymi język SQL, jak i nieobsługującymi go - w tym ostatnim przypadku język SQL jest przekładany na

## 2. Wykorzystane technologie

---

oryginalny język bazy danych.

Definicje wzięte z serwisu wikipedia.com.

## 3. Implementacja

W tym rozdziale opiszę po kolei funkcje i opowiem jak realizują założenia. Opis należy zacząć od głównej funkcji programu która wygląda następująco:

```
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int ↵
nCmdShow) {
    InitializeCriticalSection( & g_Section );
    InitializeCriticalSection( & g_Section1 );
    WNDCLASSEX wc; /* A properties struct of our window */
    MSG msg; /* A temporary location for all messages */

    /* zero out the struct and set the stuff we want to modify */
    memset(&wc,0,sizeof(wc));
    wc.cbSize      = sizeof(WNDCLASSEX);
    wc.lpfnWndProc  = WndProc; /* This is where we will send messages to */
    wc.hInstance    = hInstance;
    wc.hCursor      = LoadCursor(NULL, IDC_ARROW);

    /* White, COLOR_WINDOW is just a #define for a system color, try Ctrl+Clicking it ↵
    */
    wc.hbrBackground = (HBRUSH)(COLOR_WINDOW+1);
    wc.lpszClassName = "WindowClass";
    wc.hIcon         = LoadIcon(NULL, IDI_APPLICATION); /* Load a standard icon */
    wc.hIconSm        = LoadIcon(NULL, IDI_APPLICATION); /* use the name "A" to use the ↵
    project icon */

    /* zero out the struct and set the stuff we want to modify */
    memset(&wc1,0,sizeof(wc1));
    wc1.cbSize      = sizeof(WNDCLASSEX);
    wc1.lpfnWndProc  = WndProc1; /* This is where we will send messages↵
    to */
    wc1.hInstance    = hInstance;
    wc1.hCursor      = LoadCursor(NULL, IDC_ARROW);

    /* White, COLOR_WINDOW is just a #define for a system color, try ↵
    Ctrl+Clicking it */
    wc1.hbrBackground = (HBRUSH)(COLOR_WINDOW+1);
```

Rysunek 3.1: Główna funkcja.

### 3. Implementacja

---

```
    wc1.lpszClassName = "WindowClass1";
    wc1.hIcon          = LoadIcon(NULL, IDI_APPLICATION); /* Load a ↵
        standard icon */
    wc1.hIconSm        = LoadIcon(NULL, IDI_APPLICATION); /* use the name↵
        "A" to use the project icon */

    /* zero out the struct and set the stuff we want to modify */
    memset(&wc2,0,sizeof(wc2));
    wc2.cbSize         = sizeof(WNDCLASSEX);
    wc2.lpfnWndProc     = WndProc2; /* This is where we will send messages↵
        to */
    wc2.hInstance       = hInstance;
    wc2.hCursor         = LoadCursor(NULL, IDC_ARROW);
    /* White, COLOR_WINDOW is just a #define for a system color, try ↵
        Ctrl+Clicking it */
    wc2.hbrBackground   = (HBRUSH)(COLOR_WINDOW+1);
    wc2.lpszClassName   = "WindowClass2";
    wc2.hIcon           = LoadIcon(NULL, IDI_APPLICATION); /* Load a ↵
        standard icon */
    wc2.hIconSm         = LoadIcon(NULL, IDI_APPLICATION); /* use the name↵
        "A" to use the project icon */

    if(!RegisterClassEx(&wc)) {
        MessageBox(NULL, "Window Registration Failed!", "Error!", MB_ICONEXCLAMATION | ↵
            MB_OK);
        return 0;
    }

    if(!RegisterClassEx(&wc1)) {
        MessageBox(NULL, "Window Registration Failed!", "Error!", MB_ICONEXCLAMATION | ↵
            MB_OK);
        return 0;
    }

    if(!RegisterClassEx(&wc2)) {
        MessageBox(NULL, "Window Registration Failed!", "Error!", MB_ICONEXCLAMATION | ↵
            MB_OK);
        return 0;
    }
}
```

Rysunek 3.2: Główna funkcja.

### 3. Implementacja

---

```
hwnd = CreateWindowEx(WS_EX_CLIENTEDGE, "WindowClass", "Okno glowne", WS_VISIBLE | ↵
WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, /* x */
    CW_USEDEFAULT, /* y */
    420, /* width */
    350, /* height */
    NULL, NULL, hInstance, NULL);

if(hwnd == NULL) {
    MessageBox(NULL, "Window Creation Failed!", "Error!", MB_ICONEXCLAMATION | MB_OK);
    return 0;
}

hwnd2 = CreateWindowEx(WS_EX_CLIENTEDGE, "WindowClass1", "Prognoza", WS_VISIBLE | ↵
WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, /* x */
    CW_USEDEFAULT, /* y */
    420, /* width */
    350, /* height */
    NULL, NULL, hInstance, NULL);

if(hwnd2 == NULL) {
    MessageBox(NULL, "Window Creation Failed!", "Error!", MB_ICONEXCLAMATION | MB_OK);
    return 0;
}

hwnd3 = CreateWindowEx(WS_EX_CLIENTEDGE, "WindowClass2", "Autorzy", WS_VISIBLE | ↵
WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, /* x */
    CW_USEDEFAULT, /* y */
    420, /* width */
    350, /* height */
    NULL, NULL, hInstance, NULL);

if(hwnd3 == NULL) {
    MessageBox(NULL, "Window Creation Failed!", "Error!", MB_ICONEXCLAMATION | MB_OK);
    return 0;
}

GenerateButtons(hwnd, hInstance);
GenerateButtonsWeather(hwnd2, hInstance);
GenerateButtonsAuthors(hwnd3, hInstance);

ShowWindow(hwnd2, SW_HIDE);
ShowWindow(hwnd3, SW_HIDE);
```

Rysunek 3.3: Główna funkcja.



### 3. Implementacja

---

```
/*
    This is the heart of our program where all input is processed and
    sent to WndProc. Note that GetMessage blocks code flow until it receives ↔
    something, so
    this loop will not produce unreasonably high CPU usage
*/
while(GetMessage(&msg, NULL, 0, 0) > 0) { /* If no error is received... */
    TranslateMessage(&msg); /* Translate key codes to chars if present */
    DispatchMessage(&msg); /* Send it to WndProc */
}
DeleteCriticalSection(& g_Section);
DeleteCriticalSection(& g_Section1);
return msg.wParam;
}
```

Rysunek 3.4: Główna funkcja.

W funkcji głównej znajdziemy przede wszystkim informacje o liczbie okien, kolorze tła, rozmiarze, nagłówkach. Są tam też odniesienia do funkcji gdzie znajdują się reakcje na zdarzenia.

### 3. Implementacja

---

```
LRESULT CALLBACK WndProc(HWND hwnd, UINT Message, WPARAM wParam, LPARAM lParam) {
    switch(Message) {
        /* Upon destruction, tell the main thread to stop */
        case WM_CLOSE: {
            switch(MessageBox(NULL, "Chcesz zamknac?", "Zamykanie?", MB_ICONQUESTION | ↵
                MB_YESNO)) {
                case IDYES: {
                    PostQuitMessage(0);
                    break;
                }
                case IDNO: {
                    MessageBox(NULL, "Nie!", "Ups", MB_ICONINFORMATION | MB_OK);
                    break;
                }
            }
            break;
        }

        case WM_COMMAND: {
            switch(wParam) {
                case B_Option1: {

                    ShowWindow(hwnd2, SW_SHOW);

                    break;
                }
                case B_Option2: {

                    ShowWindow(hwnd3, SW_SHOW);

                    break;
                }
                case B_Option3: {
                    switch(MessageBox(NULL, "Chcesz zamknac?", "Zamykanie?", ↵
                        MB_ICONQUESTION | MB_YESNO)) {
                        case IDYES: {
                            PostQuitMessage(0);
                            break;
                        }
                        case IDNO: {
                            MessageBox(NULL, "Nie!", "Odmowa", MB_ICONINFORMATION | MB_OK);
                            break;
                        }
                    }
                }
            }
        }
    }
}
```

Rysunek 3.5: Główne menu.

### 3. Implementacja

---

```
        break;
    }
}
default: {
    break;
}
}
break;
}
case WM_PAINT: {
    OnPaint(hwnd);
    break;
}

/* All other messages (a lot of them) are processed using default procedures */
default:
    return DefWindowProc(hwnd, Message, wParam, lParam);
}
return 0;
}
```

Rysunek 3.6: Główne menu.

Jak można zauważyć powyżej zdążenia są przechwytywane i rozpatrywane za pomocą funkcji switch. Funkcja ta rozpoznaje wcześniej zdefiniowaną nazwę przycisku i reaguje w sposób zdefiniowany przez programistę dla konkretnego przycisku. Można też dostrzec iż mechanizm okienkowy został zaimplementowany przy pomocy funkcji chowających i pokazujących okna gdyż nie było potrzeby ich tworzenia przy każdym naciśnięciu przycisku. Z tejże funkcji uruchamiana jest też funkcja rysująca przykładową grafikę.

### 3. Implementacja

---

```
LRESULT OnPaint(HWND hwnd){
    PAINTSTRUCT ps;
    HDC hdc;
    int l=0;
    //static int x,y;

    hdc = BeginPaint(hwnd, &ps);
    RECT rect;
    GetClientRect(hwnd, &rect);

    HBRUSH brush = CreateSolidBrush( RGB(0,0,255) );
    //FillRect(hdc, &rect, brush);
    SelectObject(hdc, brush);

    //Rainy claud
    Ellipse(hdc, 200,150,250,200 );
    Ellipse(hdc, 230,150,280,200 );
    Ellipse(hdc, 260,150,310,200 );
    Ellipse(hdc, 290,150,340,200 );
    l+=10;//+1
    Ellipse(hdc, 200+l,150+l,250+l,200 +l);
    Ellipse(hdc, 230+l,150+l,280+l,200 +l);
    Ellipse(hdc, 260+l,150+l,310+l,200+l );
    Ellipse(hdc, 290+l,150+l,340+l,200+l );
    MoveToEx(hdc, rect.left + 210, rect.top + 220, NULL);
    LineTo(hdc, rect.left + 210, rect.top + 250);
    MoveToEx(hdc, rect.left + 230, rect.top + 220, NULL);
    LineTo(hdc, rect.left + 230, rect.top + 250);
    MoveToEx(hdc, rect.left + 250, rect.top + 220, NULL);
    LineTo(hdc, rect.left + 250, rect.top + 250);
    MoveToEx(hdc, rect.left + 270, rect.top + 220, NULL);
    LineTo(hdc, rect.left + 270, rect.top + 250);
    MoveToEx(hdc, rect.left + 290, rect.top + 220, NULL);
    LineTo(hdc, rect.left + 290, rect.top + 250);
    MoveToEx(hdc, rect.left + 310, rect.top + 220, NULL);
    LineTo(hdc, rect.left + 310, rect.top + 250);
    MoveToEx(hdc, rect.left + 330, rect.top + 220, NULL);
    LineTo(hdc, rect.left + 330, rect.top + 250);

    EndPaint(hwnd, &ps);
    DeleteObject(brush);
}
```

Rysunek 3.7: Funkcja graficzna.

### 3. Implementacja

---

Funkcja ta przy pomocy komend graficznych rysuje chmurę za pomocą prostych kształtów (linii i kółek) co realizuje jeden z punktów.

```
void GenerateButtons(HWND parent, HINSTANCE hInstance){
    CreateWindow(TEXT("STATIC"), TEXT("Witaj w programie Prognoza Pogody."),
        WS_CHILD | WS_VISIBLE,
        10, 10, 350, 25,
        parent, (HMENU)(502),
        hInstance, NULL);

    CreateWindowEx(WS_EX_CLIENTEDGE, "Button", "Wyszukaj Pogode", WS_VISIBLE | WS_CHILD | ↵
        BS_PUSHBUTTON,
        50, /* x */
        50, /* y */
        130, /* width */
        30, /* height */
        parent, (HMENU)B_Option1, hInstance, NULL);

    CreateWindowEx(WS_EX_CLIENTEDGE, "Button", "Autorzy", WS_VISIBLE | WS_CHILD | ↵
        BS_PUSHBUTTON,
        50, /* x */
        90, /* y */
        70, /* width */
        30, /* height */
        parent, (HMENU)B_Option2, hInstance, NULL);

    CreateWindowEx(WS_EX_CLIENTEDGE, "Button", "Koniec", WS_VISIBLE | WS_CHILD | BS_PUSHBUTTON ↵
        ,
        50, /* x */
        130, /* y */
        70, /* width */
        30, /* height */
        parent, (HMENU)B_Option3, hInstance, NULL);
}
```

Rysunek 3.8: Funkcja tworząca kontrolki do menu głównego.

Przyciski powstały w osobnej funkcji by ułatwić znalezienie ich.

### 3. Implementacja

---

```
LRESULT CALLBACK WndProc2(HWND hwnd, UINT Message, WPARAM wParam, LPARAM lParam) {
    switch(Message) {

        /* Upon destruction, tell the main thread to stop */
        case WM_CLOSE: {
            switch(MessageBox(NULL, "Chcesz zamknac?", "Zamykanie?", MB_ICONQUESTION | ↵
                MB_YESNO)){
                case IDYES:{
                    ShowWindow(hwnd, SW_HIDE);
                    break;
                }
                case IDNO:{
                    MessageBox(NULL, "Nie!", "Ups", MB_ICONINFORMATION | MB_OK);
                    break;
                }
            }
            break;
        }

        case WM_COMMAND: {
            switch(wParam) {
                case Closing: {
                    ShowWindow(hwnd, SW_HIDE);
                    //MessageBox(NULL, "Nie!", "Odmowa", MB_ICONINFORMATION | MB_OK);
                    break;
                }
            }
            break;
        }

        /* All other messages (a lot of them) are processed using default procedures */
        default:
            return DefWindowProc(hwnd, Message, wParam, lParam);
    }
    return 0;
}
```

Rysunek 3.9: Spis autorów.

Zdarzenia dla okna wypisującego autorów programu.

### 3. Implementacja

---

```
void GenerateButtonsAuthors(HWND parent, HINSTANCE hInstance){

    //static HWND hwnd_ed_u;
    CreateWindow(TEXT("STATIC"), TEXT("Mamy nastepujacy sklad:"),
        WS_CHILD | WS_VISIBLE ,
        50, 10, 200, 25,
        parent, (HMENU)(502),
        hInstance, NULL);

    CreateWindow(TEXT("STATIC"), TEXT("inz. Piotr Zyszczyk"),
        WS_CHILD | WS_VISIBLE ,
        50, 50, 200, 25,
        parent, (HMENU)(502),
        hInstance, NULL);

    CreateWindow(TEXT("STATIC"), TEXT("inz Artur Snioszek"),
        WS_CHILD | WS_VISIBLE ,
        50, 90, 200, 25,
        parent, (HMENU)(502),
        hInstance, NULL);

    CreateWindow(TEXT("STATIC"), TEXT("inz Damian Lukasik"),
        WS_CHILD | WS_VISIBLE ,
        50, 130, 200, 25,
        parent, (HMENU)(502),
        hInstance, NULL);

    CreateWindowEx(WS_EX_CLIENTEDGE, "Button", "Zamknij", WS_VISIBLE | WS_CHILD | ↵
        BS_PUSHBUTTON ,
        50, /* x */
        170, /* y */
        130, /* width */
        30, /* height */
        parent, (HMENU)Closing, hInstance, NULL);
}
```

Rysunek 3.10: Kontrolki do spisu autorów.

Funkcja robi to samo co tworząca kontrolki dla menu.

### 3. Implementacja

---

```
LRESULT CALLBACK WndProc1(HWND hwnd, UINT Message, WPARAM wParam, LPARAM lParam) {

    int Data_Of_Thread_1 = 1;
    int Data_Of_Thread_2 = 1;
    HANDLE Array_Of_Thread_Handles[3];
    switch(Message) {

        /* Upon destruction, tell the main thread to stop */
        case WM_CLOSE: {
            switch(MessageBox(NULL, "Chcesz zamknac?", "Zamykanie?", MB_ICONQUESTION | ↵
                MB_YESNO)){
                case IDYES:{
                    ShowWindow(hwnd, SW_HIDE);
                    break;
                }
                case IDNO:{
                    MessageBox(NULL, "Nie!", "Ups", MB_ICONINFORMATION | MB_OK);
                    break;
                }
            }
            break;
        }

        case WM_COMMAND: {
            switch(wParam) {
                case Closing2: {
                    ShowWindow(hwnd, SW_HIDE);
                    break;
                }
                case Chconn: {
                    //InitializeCriticalSection( & g_Section );
                    HANDLE Handle_Of_Thread_1 = CreateThread( NULL, 0, FunkcjaConnectowa ↵
                        , &Data_Of_Thread_1, 0, NULL);
                    //Array_Of_Thread_Handles[0] = Handle_Of_Thread_1;
                    //WaitForSingleObject( Handle_Of_Thread_1, 500);
                    DWORD rs = WaitForSingleObject( Handle_Of_Thread_1, 10000);

                    if(rs == WAIT_OBJECT_0)
                    {
                        MessageBox(NULL, "Watek zakonczyl sie", "Komunikat", ↵
                            MB_ICONINFORMATION | MB_OK);
                    }
                }
            }
        }
    }
}
```

Rysunek 3.11: Zdarzenia okna prezentującego pogodę.



```
else if(rs == WAIT_TIMEOUT)
{
    MessageBox(NULL,"Przekroczono czas","Komunikat",↵
        MB_ICONINFORMATION|MB_OK);
} else if(rs == WAIT_FAILED)
{
    MessageBox(NULL,"Funkcja nie powiodła sie","Komunikat",↵
        MB_ICONINFORMATION|MB_OK);
}
else if(rs == WAIT_ABANDONED)
{
    MessageBox(NULL,"Bład","Komunikat",MB_ICONINFORMATION|MB_OK);
}
//MessageBox(NULL,buffer,"Watek pogodowy",MB_ICONINFORMATION|MB_OK)↵
;

//DeleteCriticalSection(& g_Section);
CloseHandle(Handle_Of_Thread_1);

if (StatusWatek1==1) {
    MessageBox(NULL,"Watek nie uruchomiony","Komunikat",↵
        MB_ICONINFORMATION|MB_OK);
}
else if (StatusWatek1==0) {
    MessageBox(NULL,"Zakonczono pobieranie","Komunikat",↵
        MB_ICONINFORMATION|MB_OK);
}
else if (StatusWatek1==1) {
    MessageBox(NULL,"Nadal pobieram dane","Komunikat",↵
        MB_ICONINFORMATION|MB_OK);
}
else if (StatusWatek1==2) {
    MessageBox(NULL,"Wysylam zapytanie","Komunikat",↵
        MB_ICONINFORMATION|MB_OK);
}
else if (StatusWatek1==3) {
    MessageBox(NULL,"Bład w funkcji connect","Komunikat",↵
        MB_ICONINFORMATION|MB_OK);
}
```

Rysunek 3.12: Zdarzenia okna prezentującego pogodę.

### 3. Implementacja

---

```
        else if (StatusWatek1==4) {
            MessageBox(NULL, "Bład inicjacji wsastartup", "Komunikat", ↵
                MB_ICONINFORMATION | MB_OK);
        }
        else if (StatusWatek1==7) {
            MessageBox(NULL, "Nie ma Internetu", "Komunikat", ↵
                MB_ICONINFORMATION | MB_OK);
        }
        MessageBox(NULL, buffer_w1, "Komunikat", MB_ICONINFORMATION | MB_OK);
        memset(buffer_w1, 0, sizeof buffer_w1);
        break;
    }
    case DBtest: {
        //InitializeCriticalSection( & g_Section1 );
        HANDLE Handle_Of_Thread_2 = CreateThread( NULL, 0, FunkcjaBazodanowa ↵
            , &Data_Of_Thread_2, 0, NULL);
        if (WaitForSingleObject( Handle_Of_Thread_2, 100000) != WAIT_TIMEOUT) ↵
        {
            if (StatusWatek2==0){
                MessageBox(NULL, "Bład w watku!", "Bład!", MB_ICONINFORMATION | ↵
                    MB_OK);
            }
            if (StatusWatek2==1){
                MessageBox(NULL, dest_buf_w2, "Wszystko ok!", MB_ICONINFORMATION | ↵
                    MB_OK);
            }
            if (StatusWatek2==2){
                MessageBox(NULL, message_w2, "Bład wdostepie do bazy!", ↵
                    MB_ICONINFORMATION | MB_OK);
            }
        }
        else{
            MessageBox(NULL, "Proces przekroczył czas!", "Wszystko ok", ↵
                MB_ICONINFORMATION | MB_OK);
        }
        memset(dest_buf_w2, 0, sizeof dest_buf_w2);
        memset(message_w2, 0, sizeof message_w2);
        CloseHandle(Handle_Of_Thread_2);
        break;
    }
}
break;
}

/* All other messages (a lot of them) are processed using default procedures */
default:
    return DefWindowProc(hwnd, Message, wParam, lParam);
}
return 0;
}
```

### 3. Implementacja

---

W pewnym sensie nowością w tym oknie jest zastosowanie wątków do wywołania funkcji które będą ciałami tych wątków (funkcja `createThread`). Dodatkowo można wspomnieć o mechanizmie zmiennych oznaczających różne fazy wątku np. Brak internetu spowoduje awaryjne wyjście z funkcji w wątku wywołanym z kalwisa zdefiniowanego jako `Chconn`. W obu pewne dane trzeba było przekazać do głównego wątku. By te nie kolidowały ze sobą użyto mechanizmu sesji krytycznej.

### 3. Implementacja

---

```
int FunkcjaConnectowa() {

    char buffer[100000];
    buffer[0] = 0;

    char *mess;

    WSADATA wsaData;
    if (WSAStartup(MAKEWORD(2,2), &wsaData) != 0) {
        // MessageBox(NULL, "dsa", "WSA startup failed", MB_ICONINFORMATION|MB_OK);
        EnterCriticalSection( & g_Section );
        StatusWatek1=4;
        LeaveCriticalSection( & g_Section );
        system("pause");
        return 1;
    }

    SOCKET Socket=socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    struct hostent *host;

    host = gethostbyname("api.wunderground.com");
    if(host != NULL){
    }
    else{
        EnterCriticalSection( & g_Section );
        StatusWatek1=7;
        LeaveCriticalSection( & g_Section );
        system("pause");
        return 1;}
    //WSACleanup();

    SOCKADDR_IN SockAddr;
    SockAddr.sin_port=htons(80);
    SockAddr.sin_family=AF_INET;
    SockAddr.sin_addr.s_addr = *((unsigned long*)host->h_addr);
```

Rysunek 3.14: Funkcja pobierająca dane z internetu.

```
if(connect(Socket, (SOCKADDR*)&SockAddr, sizeof(SockAddr)) != 0){
    EnterCriticalSection( & g_Section );
    StatusWatek1=3;
    LeaveCriticalSection( & g_Section );
    system("pause");
    return 1;
}

DWORD dlugosc = GetWindowTextLength( hText );
LPSTR Bufor =( LPSTR ) GlobalAlloc( GPTR, dlugosc + 1 );
GetWindowText( hText, Bufor, dlugosc + 1 );

DWORD dlugosc2 = GetWindowTextLength( hText2 );
LPSTR Bufor2 =( LPSTR ) GlobalAlloc( GPTR, dlugosc2 + 1 );
GetWindowText( hText2, Bufor2, dlugosc2 + 1 );

char* char1=(char*)Bufor;
char* char2=(char*)Bufor2;
char* char3= "GET /api/5df3f8dcf842e4e7/geolookup/conditions/forecast/q/";
char* char4= "/";
char* char5= ".json HTTP/1.1\r\nHost: api.wunderground.com\r\n\r\n";
char dest_buf[100];
wsprintf (dest_buf, "%s%s", char3, char1);
wsprintf (dest_buf, "%s%s", dest_buf, char4);
wsprintf (dest_buf, "%s%s", dest_buf, char2);
wsprintf (dest_buf, "%s%s", dest_buf, char5);
```

Rysunek 3.15: Funkcja pobierająca dane z internetu.

```
mess = dest_buf;
if(send(Socket , mess , strlen(mess) , 0) < 0)
{
    EnterCriticalSection( & g_Section );
    StatusWatek1=2;
    LeaveCriticalSection( & g_Section );
}

int nDataLength;

while ((nDataLength = recv(Socket ,buffer ,2000,0)) > 0){

    // MessageBox(NULL, buffer , "Connecting",MB_ICONINFORMATION|MB_OK);
    EnterCriticalSection( & g_Section );
    wsprintf (buffer_w1, "%s%s", buffer_w1, buffer);
    StatusWatek1=1;
    LeaveCriticalSection( & g_Section );
}

//EnterCriticalSection( & g_Section );
// recv(Socket ,buffer ,100000,0);
//LeaveCriticalSection( & g_Section );

EnterCriticalSection( & g_Section );

// wsprintf(buffer_w1, "%s%s", buffer_w1, buffer);
StatusWatek1=0;
LeaveCriticalSection( & g_Section );
//wsprintf (dest_buf, "%s%s", dest_buf, char5);

closesocket(Socket);
WSACleanup();
}
```

Rysunek 3.16: Funkcja pobierająca dane z internetu.

Funkcja pobiera dane z serwisu za pomocą poleceń z biblioteki winsock i libws2 32.a. Dane są pobierane po wysłaniu zapytania do serwisu wundergroune w postaci http z danymi miasta dla którego chcemy dostać pogodę i klucza. By takowy klucz otrzymać trzeba się zarejestrować w serwisie. Sesję krytyczną zostały otoczone miejsca z których może korzystać tylko jeden wątek.

### 3. Implementacja

---

```
int FunkcjaBazodanowa() {
    char dest_buf[500];

    EnterCriticalSection( & g_Section1 );
    //dest_buf[0] = 0;
    StatusWatek2=0;
    LeaveCriticalSection( & g_Section1 );

    SQLHANDLE sqlenvhandle;
    SQLHANDLE sqlconnectionhandle;
    SQLHANDLE sqlstatementhandle;
    SQLRETURN retcode;

    if (SQL_SUCCESS!=SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &sqlenvhandle))
        goto FINISHED;

    if (SQL_SUCCESS!=SQLSetEnvAttr(sqlenvhandle, SQL_ATTR_ODBC_VERSION, (SQLPOINTER)&
        SQL_OV_ODBC3, 0))
        goto FINISHED;

    if (SQL_SUCCESS!=SQLAllocHandle(SQL_HANDLE_DBC, sqlenvhandle, &sqlconnectionhandle))
        goto FINISHED;

    SQLCHAR retconstring[1024];
    switch (SQLDriverConnect (sqlconnectionhandle,
        NULL,
        (SQLCHAR*)"DSN=mysqlster;",
        SQL_NTS,
        retconstring,
        1024,
        NULL,
        SQL_DRIVER_COMPLETE)){
    case SQL_SUCCESS_WITH_INFO:
        show_error(SQL_HANDLE_DBC, sqlconnectionhandle);
        break;
    case SQL_INVALID_HANDLE:
    case SQL_ERROR:
        show_error(SQL_HANDLE_DBC, sqlconnectionhandle);
        goto FINISHED;
    default:
        break;
    }
}
```

Rysunek 3.17: Funkcja łącząca aplikację z bazą.

### 3. Implementacja

---

```
if(SQL_SUCCESS!=SQLAllocHandle(SQL_HANDLE_STMT, sqlconnectionhandle, &sqlstatementhandle))
    goto FINISHED;

if(SQL_SUCCESS!=SQLExecDirect(sqlstatementhandle, (SQLCHAR*)"select * from testtable", SQL_NTS)){
    show_error(SQL_HANDLE_STMT, sqlstatementhandle);
    goto FINISHED;
}
else{
    char name[64];
    char address[64];
    char id[64];
    while(SQLFetch(sqlstatementhandle)==SQL_SUCCESS){
        SQLGetData(sqlstatementhandle, 1, SQL_C_CHAR, id, 64, NULL);
        SQLGetData(sqlstatementhandle, 2, SQL_C_CHAR, name, 64, NULL);
        SQLGetData(sqlstatementhandle, 3, SQL_C_CHAR, address, 64, NULL);

        //EnterCriticalSection( & g_Section1 );
        wprintf (dest_buf, "%s%s", dest_buf, id);
        wprintf (dest_buf, "%s%s", dest_buf, name);
        wprintf (dest_buf, "%s%s", dest_buf, address);

        EnterCriticalSection( & g_Section1 );
        StatusWatek2=1;
        wprintf(dest_buf_w2, "%s%s", dest_buf_w2, dest_buf);
        LeaveCriticalSection( & g_Section1 );
    }
}
FINISHED:
SQLFreeHandle(SQL_HANDLE_STMT, sqlstatementhandle );
SQLDisconnect(sqlconnectionhandle);
SQLFreeHandle(SQL_HANDLE_DBC, sqlconnectionhandle);
SQLFreeHandle(SQL_HANDLE_ENV, sqlenvhandle);
}
```

Rysunek 3.18: Funkcja łącząca aplikację z bazą.

Funkcja łączy aplikację z bazą za pomocą sterownika odbc. By takowy zastosować potrzeba było bibliotek libodbc32.a, libodbcpp32.a. Trzeba dodatkowo w systemie dodać ustawienie w panelu sterowania dla odbc.



### 3. Implementacja

---

```
void show_error(unsigned int handletype, const SQLHANDLE handle){
    SQLCHAR sqlstate[1024];
    SQLCHAR message[1024];
    if(SQL_SUCCESS == SQLGetDiagRec(handletype, handle, 1, sqlstate, NULL, message, 1024, NULL)){
        EnterCriticalSection( & g_Section1 );
        StatusWatek2=2;
        wprintf (message_w2, "%s%s", message_w2, message);
        //cout<<"Message: "<<message<<"\nSQLSTATE: "<<sqlstate<<endl;
        LeaveCriticalSection( & g_Section1 );
    }
}
```

Rysunek 3.19: Dodatkowa funkcja z komunikatem błędu.

Ta funkcja wysyła do wątku głównego szczegółowy komunikat błędu. Może to być w postaci kodu 08001.

### 3. Implementacja

---

```
#ifndef WIN32_LEAN_AND_MEAN
#define WIN32_LEAN_AND_MEAN
#endif

#include <string.h>
#include <windows.h>
#include <winsock2.h>
#include <ws2tcpip.h>
#include <iphlpapi.h>
#include <sqltypes.h>
#include <sql.h>
#include <sqlext.h>
#pragma comment(lib, "libws2_32.a")
#pragma comment(lib, "libodbc32.a")
#pragma comment(lib, "libodbc32.a")
#define BUFFERSIZE 1024
#define Label 99
#define B_Option1 100
#define B_Option2 101
#define B_Option3 102
#define TI_Edit 103 //Kraj
#define TI_Edit1 110 //Miasto
#define Closing 104
#define Closing2 104
#define Chconn 105
#define DBtest 106

WNDCLASSEX wc1;
WNDCLASSEX wc2;
HWND hwnd;
HWND hwnd2;
HWND hwnd3;
HANDLE Handle_Of_Thread_1 = 0;
HWND hText, hText2;
CRITICAL_SECTION g_Section;
int StatusWatek1=-1;
CRITICAL_SECTION g_Section1;
int StatusWatek2=-1;
char buffer1[1024];
char buffer_w1[100000]; // dane z watku1
char dest_buf_w2[500]; // dane z watku2
SQLCHAR message_w2[500]; // komunikat bledu w2
```

Rysunek 3.20: Biblioteki, zdefiniowane kontrolki i zmienne globalne.

Biblioteki, zdefiniowane kontrolki i zmienne globalne zastosowane w programie. Dodatkowo trzeba było dociągnąć biblioteki zewnętrzne podłączone komendą `pragma com-`

### 3. Implementacja

---

ment().

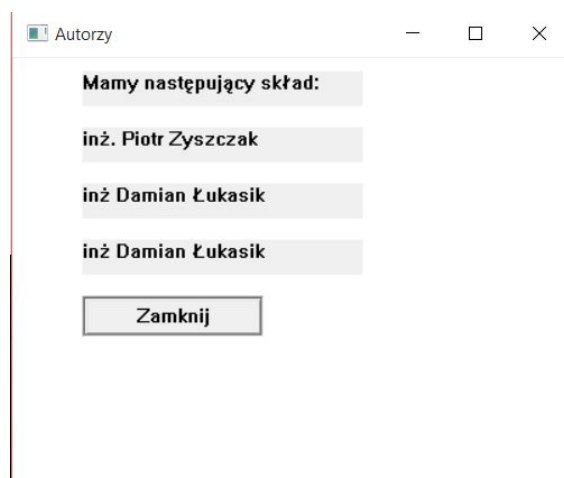
## 4. Opis użycia

Po uruchomieniu programu ukazuje nam się menu główne. Pierwsza opcja zabierze nas do ekranu gdzie możemy zdobyć informacje o pogodzie w dowolnym mieście na ziemi. Druga wyświetli listę autorów. Trzecia zakończy program.



Rysunek 4.1: Menu główne.

Ekran z autorami zawiera elementy typu label z danymi autorów programu (tytuł, imię i nazwisko). Dodatkowo jest przycisk zamykający ten ekran.



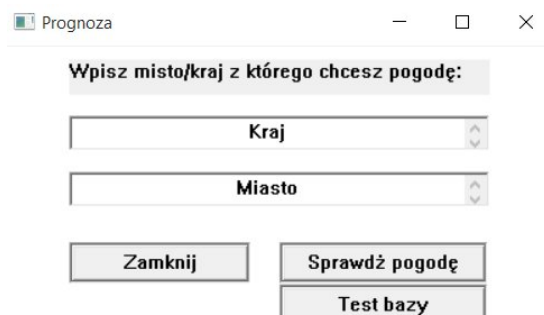
Rysunek 4.2: Menu główne.

Następne okno zawiera dwie kontrolki z edycją tekstu gdzie można zgodnie z opisem

#### 4. Opis użycia

---

wprowadzić dane potrzebne do zapytania http. Przycisk Sprawdź pogodę wyświetli nam raport z danymi pogodowymi i informacjami o połączeniu. Przycisk testuj bazę łączy z bazą MySQL i zwraca pobrane z niej rekordy.



The screenshot shows a window titled "Prognoza" with standard Windows window controls (minimize, maximize, close). Inside the window, there is a label "Wpisz miasto/kraj z którego chcesz pogodę:". Below this label are two text input fields. The first field is labeled "Kraj" and the second is labeled "Miasto". Below the input fields are three buttons: "Zamknij" (Close), "Sprawdź pogodę" (Check weather), and "Test bazy" (Test database). The "Sprawdź pogodę" and "Test bazy" buttons are stacked vertically.

Rysunek 4.3: Menu główne.

## 5. Podsumowanie

Zrealizowano wszystkie założenia projektu:

- System okienkowy zaimplementowano zgodnie z zaleceniami na zajęciach.
- Grafika rastrowa została stworzona w oparciu o GDI.
- Wielowątkowość zaimplementowano w postaci dwóch dodatkowych wątków na połączenie z bazą i pobieranie danych ze strony.
- Połączono z bazą danych MySQL przy pomocy sterownika ODBC.
- Zastosowano wątek z obsługą komunikacji sieciowej w technologii z obsługą gniazd bez przejścia z układu I/O na wiadomości systemu windows(R) (winsock.dll)