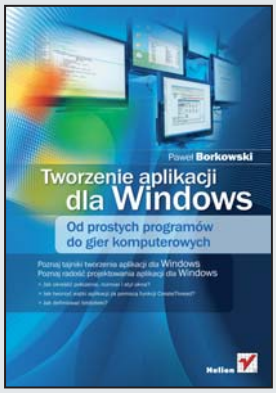
Tworzenie aplikacji dla Windows. Od prostych programów do gier komputerowych



Autor: Paweł Borkowski
ISBN: 978-83-246-1881-1
Format: 158x235, stron: 456

Poznaj tajniki tworzenia aplikacji dla Windows

- Jak określić położenie, rozmiar i styl okna?
- Jak tworzyć wątki aplikacji za pomocą funkcji CreateThread?
- Jak definiować biblioteki?

Dev-C++ to zintegrowane środowisko programistyczne, którego niewątpliwym atutem są tzw. DevPaki, czyli rozszerzenia programu, pozwalające korzystać z różnych bibliotek, szablonów i narzędzi. Środowisko Dev-C++ wspomaga także pracę nad nowym projektem Windows – gotowym kodem tworzącym okno z obsługą podstawowych komunikatów. Wszystko to sprawia, że mamy do czynienia z wygodnym i funkcjonalnym środowiskiem, zarówno dla początkujących, jak i zaawansowanych programistów.

Z książki „Tworzenie aplikacji dla Windows. Od prostych programów do gier komputerowych" może skorzystać każdy, kto chce nauczyć się programowania: zarówno studenci kierunków informatycznych, jak i osoby, które nie mają takiego przygotowania. Podręcznik kolejno odsłania poszczególne elementy wiedzy programistycznej – od najprostszych po najbardziej zaawansowane. Dowiesz się więc, jak wprowadzać niewielkie zmiany w kodzie, jak projektować aplikacje wielowątkowe i definiować biblioteki, jak budować duże, składające się z kilku plików projekty, aby na koniec samodzielnie stworzyć grę komputerową.

- Instalacja środowiska Dev-C++
- Tworzenie narzędzia pióro
- Obsługa map bitowych
- Obsługa komunikatów myszy i klawiatury
- Obiekty sterujące w oknie
- Menu i plik zasobów
- Projektowanie aplikacji wielowątkowych
- Biblioteki statyczne i dynamiczne
- Multimedia
- Programowanie gier

Naucz się programowania i twórz własne gry!

Helion SA
ul. Kościuszki 1c
44-100 Gliwice
tel. 032 230 98 63
e-mail: helion@helion.pl

Spis treści

Rozdział 8.

Projektowanie aplikacji wielowątkowych

8.1. Tworzenie wątków za pomocą funkcji CreateThread

Jesteśmy przyzwyczajeni do tego, że w systemie Windows możemy pracować na wielu oknach jednocześnie. Jak duże jest to udogodnienie, nie trzeba nikogo przekonywać. Tę cechę systemu nazywamy wielozadaniowością. Jej najpożyteczniejszym zastosowaniem jest możliwość ukrycia przed pracodawcą pod stosem otwartych okien jednego okienka z uruchomioną ulubioną grą. System Windows pozwala także na coś więcej — na wielowątkowość, czyli wykonywanie wielu zadań równocześnie w ramach działania jednego programu. Oczywiście, pojęcia pracy równoczesnej nie należy traktować zbyt dosłownie. System dzieli czas pracy procesora i przyznaje po kolei pewną jego część w celu obsługi kolejnego wątku. Ponieważ wspomniany proces wykonywany jest bardzo szybko, powstaje iluzja równoległego działania wątków.

Wątek tworzymy za pomocą funkcji `CreateThread` o następującej składni:

```
HANDLE CreateThread(LPSECURITY_ATTRIBUTES atrybuty_watku,
                    DWORD rozmiar_stosu,
                    LPTHREAD_START_ROUTINE adres_funkcji_watku,
                    LPVOID parametr,
                    DWORD flaga_watku,
                    LPDWORD identyfikator
                    );
```

Pierwszy argument `atrybuty_watku` miał zastosowanie w systemie Windows NT. Od tego czasu nie wykorzystuje się go i możemy w jego miejsce wpisać `NULL`. Drugi argument określa rozmiar stosu w bajtach. Jeżeli nie mamy pomysłu dotyczącego jego wielkości, wpiszmy zero, co oznacza zadeklarowanie stosu standardowego rozmiaru. W parametrze

adres_funkcji_watku powinien znajdować się, zresztą zgodnie z pomysłowo nadaną nazwą, adres funkcji, która zawiera kod obsługi wątku. Deklaracja funkcji musi mieć następującą postać:

```
DWORD NazwaFunkcji(LPVOID);
```

Parametr, który możemy przekazać do funkcji wątku, przekazujemy jako czwarty argument funkcji CreateThread. Argument flaga_watku określa chwilę uruchomienia wątku. Możemy użyć jednej z dwóch wartości:

- 0 — wątek uruchamiany jest w chwili utworzenia,
- CREATE_SUSPENDED — uruchomienie wątku jest wstrzymane aż do wywołania funkcji ResumeThread.

Ostatni parametr powinien zawierać wskaźnik do zmiennej, w której zostanie umieszczony identyfikator wątku. Jeżeli uruchomienie wątku powiedzie się, funkcja CreateThread zwraca uchwyt utworzonego wątku, w przeciwnym przypadku zwracana jest wartość NULL.

Wszystko, co właśnie przeczytaliście, jest niezwykle ciekawe, ale warto już przystąpić do budowania programu. Będzie to aplikacja z dziedziny helmintologii, czyli nauki o robakach (niestety, pasożytniczych). Wyobraźmy sobie obszar roboczy naszego okna. Kliknięcie lewego przycisku myszy powinno wywołać nowy wątek — robaka, którego początek niszczycielskiej działalności będzie się znajdował w miejscu, w jakim aktualnie znajduje się kursor myszy. Tworzymy nowy projekt Windows o nazwie *Program20.dev*. Spójrzmy na prosty program realizujący wspomniane zadanie (plik *Program20_main.cpp*), a następnie omówię kluczowe punkty programu.

```
#include <windows.h>

/* Stałe rozmiaru obszaru roboczego okna */
const int MaxX = 640;
const int MaxY = 480;

/* Zmienne globalne dla obsługi wątków */
int x, y;

/* Funkcja wątku */
DWORD Robak(LPVOID param);

LRESULT CALLBACK WindowProcedure (HWND, UINT, WPARAM, LPARAM);

char szClassName[ ] = "WindowsApp";

int WINAPI WinMain (HINSTANCE hThisInstance,
                    HINSTANCE hPrevInstance,
                    LPSTR lpszArgument,
                    int nFunsterStil)

{
    HWND hwnd;
    MSG messages;
    WNDCLASSEX wincl;
```

```
    wincl.hInstance = hThisInstance;
    wincl.lpszClassName = szClassName;
    wincl.lpfnWndProc = WindowProcedure;
    wincl.style = CS_DBLCLKS;
    wincl.cbSize = sizeof (WNDCLASSEX);
    wincl.hIcon = LoadIcon (NULL, IDI_APPLICATION);
    wincl.hIconSm = LoadIcon (NULL, IDI_APPLICATION);
    wincl.hCursor = LoadCursor (NULL, IDC_ARROW);
    wincl.lpszMenuName = NULL;
    wincl.cbClsExtra = 0;
    wincl.cbWndExtra = 0;
    wincl.hbrBackground = (HBRUSH) COLOR_BACKGROUND;

    if (!RegisterClassEx (&wincl))
        return 0;

    hwnd = CreateWindowEx (
           0,
           szClassName,
           "Program20 - aplikacja wielowątkowa",
           WS_OVERLAPPEDWINDOW,
           CW_USEDEFAULT,
           CW_USEDEFAULT,
           MaxX+8,
           MaxY+34,
           HWND_DESKTOP,
           NULL,
           hThisInstance,
           NULL
           );
    ShowWindow (hwnd, nFunsterStil);

    while (GetMessage (&messages, NULL, 0, 0))
    {
        TranslateMessage(&messages);
        DispatchMessage(&messages);
    }
    return messages.wParam;
}

LRESULT CALLBACK WindowProcedure (HWND hwnd, UINT message, WPARAM wParam,
↪LPARAM lParam)
{
    static HDC hdc;
    PAINTSTRUCT ps;
    DWORD pointer;

    switch (message)
    {
        case WM_PAINT:
            hdc = BeginPaint(hwnd, &ps);
            if(x>=5&&y>=5&&x<MaxX-5&&y<MaxY-5)
                Ellipse(hdc, x-5, y-5, x+5, y+5);
            EndPaint(hwnd, &ps);
            break;
        case WM_LBUTTONDOWN:
            x = LOWORD(lParam);
```

```
            y = HIWORD(lParam);
            CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)Robak, hwnd, 0,
            ↪&pointer);
            break;
        case WM_DESTROY:
            PostQuitMessage (0);
            break;
        default:
            return DefWindowProc (hwnd, message, wParam, lParam);
    }

    return 0;
}

DWORD Robak(LPVOID param)
{
    int wx = x, wy = y;

    //uruchomienie liczb losowych
    srand(GetTickCount());

    //pętla nieskończona
    while(TRUE)
    {
        //wylosuj ruch robaka
        wx += (rand()%5) - 2;
        wy += (rand()%5) - 2;
        //sprawdź, czy robak znajduje się w dozwolonym obszarze
        if(wx<5) wx = 5;
        if(wx>MaxX-5) wx = MaxX-5;
        if(wy<5) wy = 5;
        if(wy>MaxY-5) wy = MaxY-5;
        //załaduj nowe współrzędne do zmiennej globalnej
        x = wx;
        y = wy;
        //narysuj robaka w nowym położeniu
        InvalidateRect((HWND)param, NULL, FALSE);
        //zaczekaj 1/10 sekundy
        Sleep(100);
    }
    return 0;
}
```

Program po uruchomieniu czeka na naciśnięcie lewego przycisku myszy.

```
case WM_LBUTTONDOWN:
    x = LOWORD(lParam);
    y = HIWORD(lParam);
    CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)Robak, hwnd, 0, &pointer);
    break;
```

Jeżeli procedura okna otrzyma komunikat o zajściu tego zdarzenia, tworzy wątek, którego obsługę powierza funkcji `Robak`. Współrzędne kursora myszy są ładowane do zmiennych globalnych `x` i `y` w celu przekazania ich do funkcji obsługi wątku. Aplikacja kontynuuje pracę, a równolegle z nią zaczyna działać nowy wątek. Spójrzmy na kod funkcji `Robak`.

```
DWORD Robak(LPVOID param)
{
    int wx = x, wy = y;

    //uruchomienie liczb losowych
    srand(GetTickCount());

    while(TRUE)
    {
        //wylosuj ruch robaka
        wx += (rand()%5) - 2;
        wy += (rand()%5) - 2;
        //sprawdź, czy robak znajduje się w dozwolonym obszarze
        if(wx<5) wx = 5;
        if(wx>MaxX-5) wx = MaxX-5;
        if(wy<5) wy = 5;
        if(wy>MaxY-5) wy = MaxY-5;
        //załaduj nowe współrzędne do zmiennej globalnej
        x = wx;
        y = wy;
        //narysuj robaka w nowym położeniu
        InvalidateRect((HWND)param, NULL, FALSE);
        //zaczekaj 1/10 sekundy
        Sleep(100);
    }

    return 0;
}
```

Początkowe współrzędne obiektu zostały przekazane przy użyciu zmiennych globalnych `x` i `y`.

```
int wx = x, wy = y;
```

Pozostała część kodu umieszczona jest w pętli nieskończonej, co oznacza, że wątek zostanie zniszczony dopiero w chwili zamknięcia aplikacji.

Bardzo ważną cechą robaka jest jego losowy ruch realizowany za pomocą uaktualniania współrzędnych.

```
wx += (rand()%5) - 2;
wy += (rand()%5) - 2;
```

Ponieważ w tym zadaniu korzystamy z funkcji `rand`, która zwraca liczby pseudolosowe, ważne jest, by wartość inicjalizująca tej funkcji różniła się dla każdego wątku. Liczbę początkową dla obliczeń funkcji `rand` przekazujemy za pomocą `srand`, której argumentem jest u nas funkcja `GetTickCount` podająca liczbę milisekund liczoną od chwili uruchomienia systemu Windows.

```
srand(GetTickCount());
```

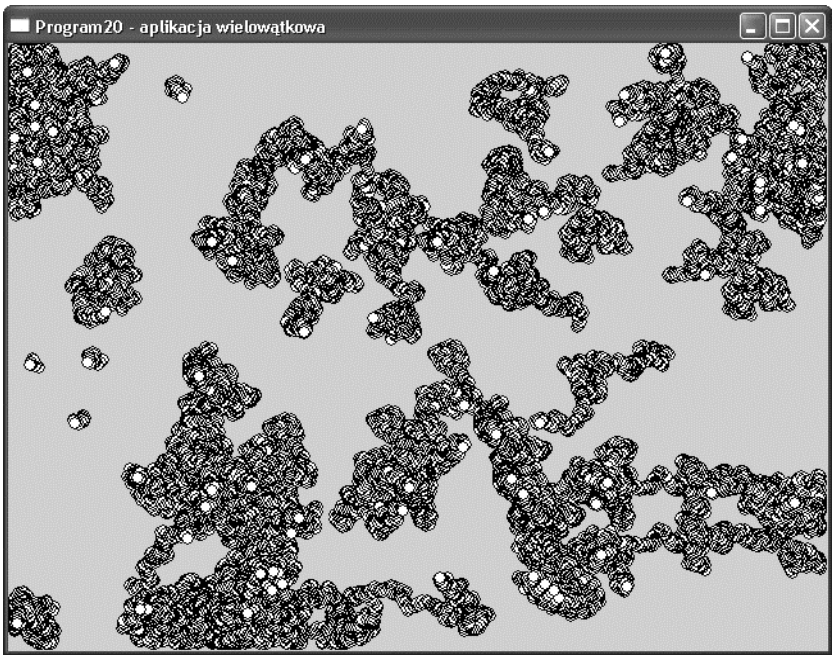
Po obliczeniu nowej współrzędnej położenia robaka i sprawdzeniu jej poprawności wymuszamy odmalowanie obszaru roboczego okna.

```
InvalidateRect((HWND)param, NULL, FALSE);
```

W ten sposób wątek komunikuje się z aplikacją główną i zmusza do zainteresowania się kodem obsługi komunikatu WM_PAINT.

```
case WM_PAINT:
    hdc = BeginPaint(hwnd, &ps);
    if(x>=5&&y>=5&&x<MaxX-5&&y<MaxY-5)
        Ellipse(hdc, x-5, y-5, x+5, y+5);
    EndPaint(hwnd, &ps);
    break;
```

Wizualną częścią działania wątku jest narysowana w nowym położeniu elipsa. Przykładowy efekt działania programu z uruchomionymi jednocześnie kilkudziesięcioma wątkami przedstawiam na rysunku 8.1.



Rysunek 8.1. *Okno działającej aplikacji Program20*

8.2. Czy praca z wątkami przyspiesza działanie aplikacji? Sekcja krytyczna i priorytety wątków

Odpowiedź na pytanie postawione w tytule podrozdziału nie jest prosta. Wszystko zależy od rodzaju czynności realizowanych przez wątek. Pamiętajmy, że każdy utworzony wątek korzysta z części czasu pracy procesora. Dlatego błędna jest myśl, że skomplikowane obliczenie rozłożone na wiele wątków wykona się szybciej niż to samo

obliczenie w jednowątkowej aplikacji. Natomiast istnieją sytuacje, w których procesor „czeka bezczynnie" lub jest tylko „nieznacznie zajęty", wtedy — oczywiście — stosowanie wątków jest zasadne. Operowanie wątkami ma przede wszystkim ułatwić pracę nam, czyli programistom.

Ilustracją do postawionego pytania będzie aplikacja, w której na dwóch częściach obszaru roboczego okna wyświetlimy zbiór Mandelbrota i jeden ze zbiorów Julii. Wykonanie każdego z tych zadań bardzo mocno obciąża procesor i kartę graficzną. Nasza aplikacja w pierwszej kolejności będzie rysowała oddzielnie zbiór Mandelbrota i zbiór Julii, a następnie obydwa zadania zostaną wykonane równoległe w dwóch wątkach. Policzymy czas wykonania zadań w wersji bez użycia wątków i z ich użyciem.

Tworzymy nowy projekt *Program21.dev*. Ponieważ przewidujemy dużą ilość operacji graficznych, odpowiedni kod umieścimy w pliku *Program21.h*. Wygenerowany przez Dev-C++ kod (plik *Program21_main.cpp*) modyfikujemy tylko w czterech miejscach.

1. Dodajemy plik nagłówkowy.

```
#include "Program21.h"
```

2. Zmieniamy parametry funkcji `CreateWindowEx`.

```
hwnd = CreateWindowEx (
       0,
       szClassName,
       "Program21",
       WS_OVERLAPPEDWINDOW,
       CW_USEDEFAULT,
       CW_USEDEFAULT,
       648,
       340,
       HWND_DESKTOP,
       NULL,
       hThisInstance,
       NULL
       );
```

3. Dodajemy kod obsługi komunikatu `WM_PAINT`.

```
case WM_PAINT:
    hdc = BeginPaint(hwnd, &ps);
    MojaProcedura(hdc);
    EndPaint(hwnd, &ps);
    break;
```

4. Dodajemy kod obsługi komunikatu `WM_LBUTTONDOWN`. Dzięki temu możliwe będzie łatwe odświeżanie obszaru roboczego okna i tym samym wielokrotne powtórzenie doświadczenia.

```
case WM_LBUTTONDOWN:
    InvalidateRect(hwnd, NULL, TRUE);
    break;
```

Plik *Program21_main.cpp* w całości wygląda następująco (jak zwykle, pominięte zostały generowane przez Dev-C++ komentarze):

```
#include <windows.h>
#include "Program21.h"

LRESULT CALLBACK WindowProcedure (HWND, UINT, WPARAM, LPARAM);

char szClassName[ ] = "WindowsApp";

int WINAPI WinMain (HINSTANCE hThisInstance,
                    HINSTANCE hPrevInstance,
                    LPSTR lpszArgument,
                    int nFunsterStil)
{
    HWND hwnd;
    MSG messages;
    WNDCLASSEX wincl;

    wincl.hInstance = hThisInstance;
    wincl.lpszClassName = szClassName;
    wincl.lpfnWndProc = WindowProcedure;
    wincl.style = CS_DBLCLKS;
    wincl.cbSize = sizeof (WNDCLASSEX);
    wincl.hIcon = LoadIcon (NULL, IDI_APPLICATION);
    wincl.hIconSm = LoadIcon (NULL, IDI_APPLICATION);
    wincl.hCursor = LoadCursor (NULL, IDC_ARROW);
    wincl.lpszMenuName = NULL;
    wincl.cbClsExtra = 0;
    wincl.cbWndExtra = 0;
    wincl.hbrBackground = (HBRUSH) COLOR_BACKGROUND;

    if (!RegisterClassEx (&wincl))
        return 0;

    hwnd = CreateWindowEx (
           0,
           szClassName,
           "Program21",
           WS_OVERLAPPEDWINDOW,
           CW_USEDEFAULT,
           CW_USEDEFAULT,
           648,
           340,
           HWND_DESKTOP,
           NULL,
           hThisInstance,
           NULL
           );

    ShowWindow (hwnd, nFunsterStil);

    while (GetMessage (&messages, NULL, 0, 0))
    {
        TranslateMessage(&messages);
        DispatchMessage(&messages);
    }

    return messages.wParam;
}
```

```
LRESULT CALLBACK WindowProcedure (HWND hwnd, UINT message, WPARAM wParam,
↪LPARAM lParam)
{
    HDC hdc;
    PAINTSTRUCT ps;

    switch (message)
    {
        case WM_PAINT:
            hdc = BeginPaint(hwnd, &ps);
            MojaProcedura(hdc);
            EndPaint(hwnd, &ps);
            break;
        case WM_LBUTTONDOWN:
            InvalidateRect(hwnd, NULL, TRUE);
            break;
        case WM_DESTROY:
            PostQuitMessage (0);
            break;
        default:
            return DefWindowProc (hwnd, message, wParam, lParam);
    }

    return 0;
}
```

Kluczową część programu, wraz z wywoływaną w kodzie obsługi komunikatu `WM_PAINT` procedurą `MojaProcedura`, umieścimy w pliku *Program21.h*. Zawartość procedury będą stanowić:

1. Obliczenie czasu potrzebnego do narysowania zbioru Mandelbrota i Julii za pomocą funkcji wywoływanych w sposób sekwencyjny.
   - **a)** Pobranie liczby milisekund, które upłynęły od chwili uruchomienia systemu Windows i umieszczenie tej wielkości w zmiennej `m1`.
   - **b)** Wywołanie funkcji rysującej zbiór Mandelbrota.
   - **c)** Wywołanie funkcji rysującej zbiór Julii.
   - **d)** Pobranie liczby milisekund, które upłynęły od chwili uruchomienia systemu Windows i umieszczenie tej wielkości w zmiennej `m2`.
   - **e)** Obliczenie i wyświetlenie różnicy `m2-m1`. Wynik oznacza czas wykonania zadań opisanych w punktach b i c.
2. Obliczenie czasu potrzebnego do narysowania zbioru Mandelbrota i Julii za pomocą funkcji wywołanych w równolegle pracujących wątkach.
   - **a)** Pobranie liczby milisekund, które upłynęły od chwili uruchomienia systemu Windows, i umieszczenie tej wielkości w zmiennej `m1`.
   - **b)** Utworzenie wątku wywołującego funkcję rysującą zbiór Mandelbrota.
   - **c)** Utworzenie wątku wywołującego funkcję rysującą zbiór Julii.
   - **d)** Oczekiwanie na zakończenie pracy obu wątków.

e) Pobranie liczby milisekund, które upłynęły od chwili uruchomienia systemu Windows, i umieszczenie tej wielkości w zmiennej `m2`.

f) Obliczenie i wyświetlenie różnicy `m2-m1`. Wynik oznacza czas wykonania zadań opisanych w punktach b i c.

Popatrzmy na pierwszą wersję procedury `MojaProcedura`.

```
void MojaProcedura(HDC hdc)
{
    char tab[32];
    DWORD pointer;
    DWORD m, m1, m2;

//pobierz ilość milisekund od chwili uruchomienia systemu Windows
//(czas startu zadania)
    m1 = GetTickCount();
//wykonaj zadanie numer 1
    Zbior_Mandelbrota(hdc);
//wykonaj zadanie numer 2
    Zbior_Julii(hdc);
//pobierz ilość milisekund od chwili uruchomienia systemu Windows
//(czas zakończenia zadania)
    m2 = GetTickCount();
//wyświetl okres czasu potrzebny do wykonania zadania
    m = m2 - m1;
    TextOut(hdc, 10, 250, "Czas sekwencyjnego wykonania zadań:", 35);
    itoa(m, tab, 10);
    TextOut(hdc, 300, 250, tab, strlen(tab));

//wyczyść fragment okna
    Rectangle(hdc, 0, 0, 640, 240);
//wyzeruj zmienną globalną dla wątków
    zmienna_stop = 0;
//pobierz ilość milisekund od chwili uruchomienia systemu Windows
//(czas startu zadania)
    m1 = GetTickCount();
//wywołaj wątek wykonujący zadanie numer 1
    CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)Z_M, hdc, 0, &pointer);
//wywołaj wątek wykonujący zadanie numer 2
    CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)Z_J, hdc, 0, &pointer);
//czekaj na zakończenie pracy wątków
    do{}while(zmienna_stop<2);
//pobierz ilość milisekund od chwili uruchomienia systemu Windows
//(czas zakończenia zadania)
    m2 = GetTickCount();
//wyświetl okres czasu potrzebny do wykonania zadania
    m = m2 - m1;
    TextOut(hdc, 10, 270, "Czas równoległego wykonania zadań:", 34);
    itoa(m, tab, 10);
    TextOut(hdc, 300, 270, tab, strlen(tab));
}
```

Procedura realizuje kolejno przedstawione w punktach zadania. Do obliczenia czasu działania części programu używamy znanej już funkcji `GetTickCount`. Taki sposób liczenia przedziałów czasu jest całkiem bezpieczny, gdyż zwracana przez funkcję wielkość

jest 32-bitowa, co oznacza, że możemy spodziewać się zrestartowania licznika dopiero po 49 dniach nieprzerwanej pracy systemu Windows. Funkcje tworzonych w procedurze wątków mają następującą postać:

```
DWORD Z_M(LPVOID param)
{
    Zbior_Mandelbrota((HDC)param);
    //zwiększ wskaźnik zakończenia pracy
    zmienna_stop++;
    return 0;
}

DWORD Z_J(LPVOID param)
{
    Zbior_Julii((HDC)param);
    //zwiększ wskaźnik zakończenia pracy
    zmienna_stop++;
    return 0;
}
```

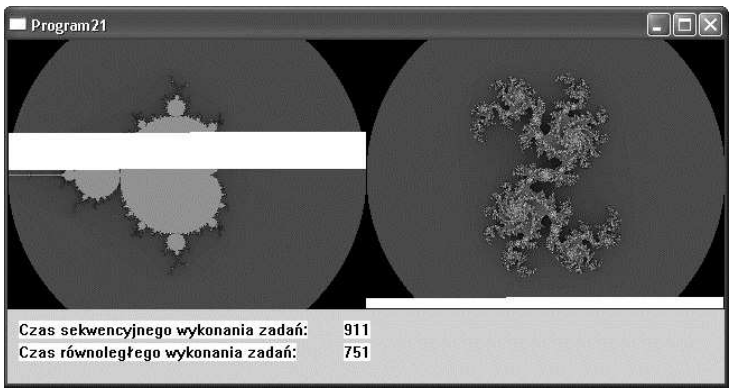
Kody funkcji `Zbior_Mandelbrota` i `Zbior_Julii` na razie w naszych rozważaniach nie są istotne. Należy zwrócić uwagę na sposób zakończenia funkcji wątków, polegający na inkrementacji zmiennej globalnej.

```
zmienna_stop++;
```

Dzięki temu możliwe staje się sprawdzenie zakończenia działania obu wątków za pomocą pętli:

```
do{}while(zmienna_stop<2);
```

Możemy przyjąć, że mamy pierwszą podstawową wersję aplikacji. Nie jest to jeszcze wersja optymalna. Popatrzmy na rysunek 8.2, na którym przedstawiam rezultat działania programu.



Rysunek 8.2. *Okno aplikacji Program21 w działaniu (wersja pierwsza, niepoprawna)*

Na początek należy zauważyć, że narysowanie zbiorów Mandelbrota i Julii za pomocą wątków wykonało się szybciej niż zadanie analogiczne, zrealizowane za pomocą sekwencyjnego wywołania funkcji. Dzieje się tak dlatego, gdyż okresy obliczeń wykorzystujące procesor są przerywane długimi okresami obciążania karty graficznej, podczas

których wyświetlany jest piksel o danej współrzędnej w odpowiednim kolorze. Istnieją więc krótkie przedziały czasowe małego obciążenia procesora, które może wykorzystać dla własnych obliczeń drugi wątek. Jednak niebezpieczeństwo pracy z wątkami polega na tym, że system może zarządzić wstrzymanie pracy wątku i przekazanie sterowania drugiemu w najmniej spodziewanym momencie. Jeżeli obydwa wątki korzystają w tym czasie ze wspólnych zasobów systemowych, a tak jest w przypadku operacji graficznych korzystających z obiektów GDI, dane mogą ulec zatraceniu, czego wynikiem są niezamalowane obszary, pokazane na rysunku 8.2.

Na szczęście, potrafimy przeciwdziałać przerwaniu pracy wątku w chwili dla nas niepożądanej (co za ulga!). Do tego celu służy *sekcja krytyczna*, a dokładniej *obiekty sekcji krytycznej*. W celu przybliżenia sensu istnienia tego rewelacyjnego rozwiązania programistycznego posłużę się przykładem. Wyobraźmy sobie bibliotekę, w której znajduje się jeden egzemplarz starego manuskryptu. Wyobraźmy sobie także, że setka bibliofilów chce w danym momencie ów egzemplarz przeczytać. Może to robić na raz tylko jeden fanatyk starego piśmiennictwa, natomiast reszta, obgryzając ze zdenerwowania palce, może tylko czekać, aż szczęśliwy chwilowy posiadacz obiektu zakończy pracę. Podobną rolę w pracy wątków odgrywa obiekt sekcji krytycznej: w jego posiadanie może w danej chwili wejść tylko jeden wątek, natomiast pozostałe wątki zawieszają pracę do czasu przejęcia obiektu sekcji krytycznej. Oczywiście, czekają tylko te wątki, w których zaznaczono konieczność wejścia w posiadanie obiektu sekcji krytycznej w celu wykonania fragmentu kodu.

Praca z obiektami sekcji krytycznej jest niezwykle prosta i składa się z kilku punktów. Oto one.

1. Deklaracja obiektu sekcji krytycznej. Nazwa obiektu jest dowolna, możemy zadeklarować dowolną liczbę obiektów.

```
CRITICAL_SECTION critical_section;
```

2. Inicjalizacja obiektu sekcji krytycznej.

```
InitializeCriticalSection(&critical_section);
```

3. Następne dwa podpunkty umieszczamy w funkcji wątków, są to:

   a) przejęcie obiektu sekcji krytycznej:

```
EnterCriticalSection(&critical_section);
```

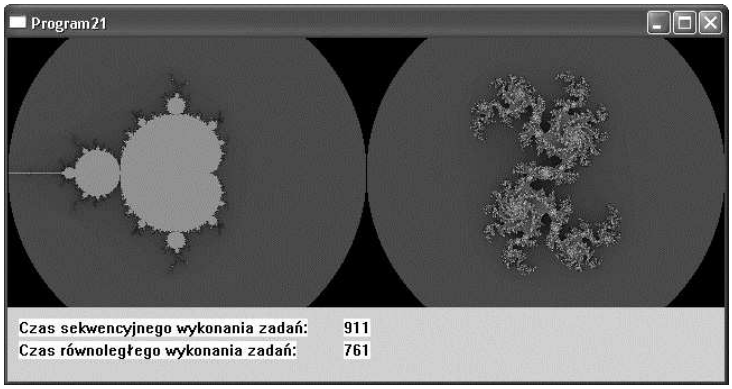
   b) zwolnienie obiektu sekcji krytycznej:

```
LeaveCriticalSection(&critical_section);
```

4. Zakończenie pracy z obiektem sekcji krytycznej powinno być związane z jego usunięciem.

```
DeleteCriticalSection(&critical_section);
```

Sposób użycia obiektów sekcji krytycznej zobaczymy na przykładzie kodu poprawionej procedury `MojaProcedura` i jednej z funkcji wątków `Z_M`. Rezultat działania tak poprawionego programu przedstawiam na rysunku 8.3.



Rysunek 8.3. *Okno aplikacji Program21 działającej z wykorzystaniem obiektu sekcji krytycznej*

```
/* Obiekt sekcji krytycznej */
CRITICAL_SECTION critical_section;

void MojaProcedura(HDC hdc)
{
    char tab[32];
    DWORD pointer, m, m1, m2;

//pobierz ilość milisekund od chwili uruchomienia systemu Windows
//(czas startu zadania)
    m1 = GetTickCount();
//wykonaj zadanie numer 1
    Zbior_Mandelbrota(hdc);
//wykonaj zadanie numer 2
    Zbior_Julii(hdc);
//pobierz ilość milisekund od chwili uruchomienia systemu Windows
//(czas zakończenia zadania)
    m2 = GetTickCount();
//wyświetl okres czasu potrzebny do wykonania zadania
    m = m2 - m1;
    TextOut(hdc, 10, 250, "Czas sekwencyjnego wykonania zadań:", 35);
    itoa(m, tab, 10);
    TextOut(hdc, 300, 250, tab, strlen(tab));

//wyczyść fragment okna
    Rectangle(hdc, 0, 0, 640, 240);
//wyzeruj zmienną globalną dla wątków
    zmienna_stop = 0;
//zainicjalizuj obiekt sekcji krytycznej
    InitializeCriticalSection(&critical_section);
//pobierz ilość milisekund od chwili uruchomienia systemu Windows
//(czas startu zadania)
    m1 = GetTickCount();
//wywołaj wątek wykonujący zadanie numer 1
    CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)Z_M, hdc, 0, &pointer);
//wywołaj wątek wykonujący zadanie numer 2
    CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)Z_J, hdc, 0, &pointer);
//czekaj na zakończenie pracy wątków
    do{}while(zmienna_stop<2);
//pobierz ilość milisekund od chwili uruchomienia systemu Windows
//(czas zakończenia zadania)
    m2 = GetTickCount();
```

```
//usuń obiekt sekcji krytycznej
    DeleteCriticalSection(&critical_section);
//wyświetl okres czasu potrzebny do wykonania zadania
    m = m2 - m1;
    TextOut(hdc, 10, 270, "Czas równoległego wykonania zadań:", 34);
    itoa(m, tab, 10);
    TextOut(hdc, 300, 270, tab, strlen(tab));
}

DWORD Z_M(LPVOID param)
{

    double a_lewy = -2.0, a_prawy = 2.0;
    double b_gora = 1.5, b_dol = -1.5;
    double krokX = (a_prawy - a_lewy)/320.0;
    double krokY = (b_gora - b_dol)/240.0;
    int kolor;

    for(int y=0; y<240; y++)
        for(int x=0; x<320; x++)
        {
            kolor = Kolor(a_lewy+double(x)*krokX, b_gora-double(y)*krokY);
            //wejdź w sekcję krytyczną
            EnterCriticalSection(&critical_section);
            //ustaw piksel
            SetPixel((HDC)param, x, y, kolor);
            //opuść sekcję krytyczną
            LeaveCriticalSection(&critical_section);
        }
    //zwiększ wskaźnik zakończenia pracy
    zmienna_stop++;
    return 0;
}
```

Ponieważ funkcje graficzne zostały w aplikacji zamknięte w sekcje krytyczne, nie ma na wykresie obszarów błędnych. Wyświetlane przez program liczby mogą się różnić od zaprezentowanych na ilustracji, co jest wynikiem różnej mocy obliczeniowej komputera, a nawet ilością procesów korzystających w danej chwili z procesora. Ostatnie udoskonalenie programu będzie związane ze zmianą priorytetu wykonywanych wątków.

Uruchomione w aplikacji wątki muszą współdzielić czas pracy procesora nie tylko między siebie, ale też z wywołującą je aplikacją główną i innymi uruchomionymi w systemie wątkami. To wydłuża czas działania wątków. Zmiana tej sytuacji jest możliwa za pomocą zmiany priorytetu uruchomionych wątków. Do tego zadania służy funkcja `SetThreadPriority` o następującej deklaracji:

```
BOOL SetThreadPriority(HANDLE uchwyt_watku, int priorytet);
```

Parametr `uchwyt_watku` powinien zawierać uchwyt wątku, którego priorytet chcemy zmienić, a argumentem `priorytet` określamy wartość priorytetu wątku. Zestawienie możliwych do wyboru wartości priorytetu przedstawiam w tabeli 8.1.

W naszej aplikacji użyjemy najwyższego priorytetu `THREAD_PRIORITY_HIGHEST`, czego wynikiem będzie prawie dwukrotne przyspieszenie czasu rysowania wykresów zbioru Mandelbrota i Julii. Oto gotowy plik *Program21.h.*

Tabela 8.1. *Priorytety uruchamianych wątków*

Stała priorytetu	Wartość priorytetu
THREAD_PRIORITY_HIGHEST	+2
THREAD_PRIORITY_ABOVE_NORMAL	+1
THREAD_PRIORITY_NORMAL	0
THREAD_PRIORITY_BELOW_NORMAL	-1
THREAD_PRIORITY_LOWEST	-2

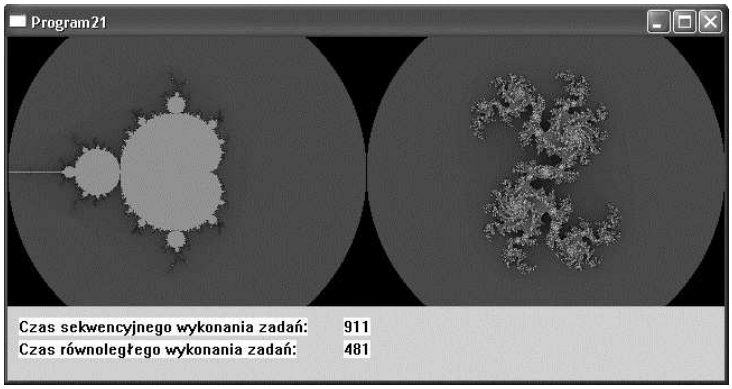
```
#include <math.h>

//zmienna globalna do sprawdzenia stanu wątków
int zmienna_stop;

/* Deklaracja procedur */
void MojaProcedura(HDC);
void Zbior_Mandelbrota(HDC);
void Zbior_Julii(HDC);

/* Funkcje wątków */
DWORD Z_M(LPVOID);
DWORD Z_J(LPVOID);

/* Obiekt sekcji krytycznej */
CRITICAL_SECTION critical_section;

void MojaProcedura(HDC hdc)
{
    char tab[32];
    DWORD pointer, m, m1, m2;
    HANDLE h;

//pobierz ilość milisekund od chwili uruchomienia systemu Windows
//(czas startu zadania)
    m1 = GetTickCount();
//wykonaj zadanie numer 1
    Zbior_Mandelbrota(hdc);
//wykonaj zadanie numer 2
    Zbior_Julii(hdc);
//pobierz ilość milisekund od chwili uruchomienia systemu Windows
//(czas zakończenia zadania)
    m2 = GetTickCount();
//wyświetl okres czasu potrzebny do wykonania zadania
    m = m2 - m1;
    TextOut(hdc, 10, 250, "Czas sekwencyjnego wykonania zadań:", 35);
    itoa(m, tab, 10);
    TextOut(hdc, 300, 250, tab, strlen(tab));

//wyczyść fragment okna
    Rectangle(hdc, 0, 0, 640, 240);
//wyzeruj zmienną globalną dla wątków
    zmienna_stop = 0;
//zainicjalizuj obiekt sekcji krytycznej
    InitializeCriticalSection(&critical_section);
```

```
//pobierz ilość milisekund od chwili uruchomienia systemu Windows
//(czas startu zadania)
    m1 = GetTickCount();
//wywołaj wątek wykonujący zadanie numer 1
    h = CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)Z_M, hdc, 0, &pointer);
    SetThreadPriority(h, THREAD_PRIORITY_HIGHEST);
//wywołaj wątek wykonujący zadanie numer 2
    h = CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)Z_J, hdc, 0, &pointer);
    SetThreadPriority(h, THREAD_PRIORITY_HIGHEST);
//czekaj na zakończenie pracy wątków
    do{}while(zmienna_stop<2);
//pobierz ilość milisekund od chwili uruchomienia systemu Windows
//(czas zakończenia zadania)
    m2 = GetTickCount();
//usuń obiekt sekcji krytycznej
    DeleteCriticalSection(&critical_section);
//wyświetl okres czasu potrzebny do wykonania zadania
    m = m2 - m1;
    TextOut(hdc, 10, 270, "Czas równoległego wykonania zadań:", 34);
    itoa(m, tab, 10);
    TextOut(hdc, 300, 270, tab, strlen(tab));
}

int Kolor(double a0, double b0)
{
    double an, a = a0, b = b0;
    int k = 0;
    while((sqrt(a*a+b*b)<2)&&(k<256))
    {
        an = a*a - b*b + a0;
        b  = 2.0*a*b + b0;
        a  = an;
        k++;
    }
    return 0xF8*k;
}

void Zbior_Mandelbrota(HDC hdc)
{
    double a_lewy = -2.0, a_prawy = 2.0;
    double b_gora = 1.5, b_dol = -1.5;
    double krokX = (a_prawy - a_lewy)/320.0;
    double krokY = (b_gora - b_dol)/240.0;

    for(int y=0; y<240; y++)
        for(int x=0; x<320; x++)
            SetPixel(hdc, x, y, Kolor(a_lewy+double(x)*krokX,
                                      b_gora-double(y)*krokY));
}

int jKolor(double a0, double b0)
{
    double an, a = a0, b = b0;
    int k = 0;
    while((sqrt(a*a+b*b)<2)&&(k<256))
    {
        an = a*a - b*b + 0.373;
        b  = 2.0*a*b + 0.133;
```

```
        a  = an;
        k++;
    }
    return 0xF8*k;
}

void Zbior_Julii(HDC hdc)
{
    double a_lewy = -2.0, a_prawy = 2.0;
    double b_gora = 1.5, b_dol = -1.5;
    double krokX = (a_prawy - a_lewy)/320.0;
    double krokY = (b_gora - b_dol)/240.0;

    for(int y=0; y<240; y++)
        for(int x=0; x<320; x++)
            SetPixel(hdc, 320+x, y, jKolor(a_lewy+double(x)*krokX,
                                           b_gora-double(y)*krokY));
}

DWORD Z_M(LPVOID param)
{


    double a_lewy = -2.0, a_prawy = 2.0;
    double b_gora = 1.5, b_dol = -1.5;
    double krokX = (a_prawy - a_lewy)/320.0;
    double krokY = (b_gora - b_dol)/240.0;
    int kolor;

    for(int y=0; y<240; y++)
        for(int x=0; x<320; x++)
        {
            kolor = Kolor(a_lewy+double(x)*krokX, b_gora-double(y)*krokY);
            //wejdź w sekcję krytyczną
            EnterCriticalSection(&critical_section);
            //zapal piksel
            SetPixel((HDC)param, x, y, kolor);
            //opuść sekcję krytyczną
            LeaveCriticalSection(&critical_section);
        }

    //zwiększ wskaźnik zakończenia pracy
    zmienna_stop++;
    return 0;
}

DWORD Z_J(LPVOID param)
{
    double a_lewy = -2.0, a_prawy = 2.0;
    double b_gora = 1.5, b_dol = -1.5;
    double krokX = (a_prawy - a_lewy)/320.0;
    double krokY = (b_gora - b_dol)/240.0;
    int kolor;

    for(int y=0; y<240; y++)
        for(int x=0; x<320; x++)
```

```
            {
                kolor = jKolor(a_lewy+double(x)*krokX, b_gora-double(y)*krokY);
                //wejdź w sekcję krytyczną
                EnterCriticalSection(&critical_section);
                //ustaw piksel
                SetPixel((HDC)param, 320+x, y, kolor);
                //opuść sekcję krytyczną
                LeaveCriticalSection(&critical_section);
            }

        //zwiększ wskaźnik zakończenia pracy
        zmienna_stop++;
        return 0;
    }
```

Rezultat działania programu przedstawiam na rysunku 8.4.



Rysunek 8.4. *Okno aplikacji Program21 działającej z wykorzystaniem obiektu sekcji krytycznej i ustawionym najwyższym priorytetem pracy wątków*

8.3. Wstrzymywanie pracy i usuwanie wątków

Zapewne przynajmniej raz spotkaliście się z irytującym edytorem tekstowym, który — doskonale wiedząc, co chcemy napisać — z zadziwiającym uporem poprawiał postać wpisywanego tekstu. Oto nadszedł czas zemsty i w niniejszym podrozdziale zajmiemy się konstruowaniem podobnego, tylko jeszcze bardziej złośliwego edytora. Niezwykle pomocna przy tym będzie umiejętność tworzenia wątków, które niczym duch będą śledziły wpisywany przez użytkownika tekst, dokonując jego modyfikacji.

Znamy już trzy czynności związane z wątkami.

1. Utworzenie wątku. Do tego zadania służy funkcja `CreateThread`.
2. Obsługa obiektów sekcji krytycznej. Wykorzystywane do tego celu funkcje to: `InitializeCriticalSection`, `EnterCriticalSection`, `LeaveCriticalSection` i `DeleteCriticalSection`.
3. Zmiana priorytetu wątku. Do tego zadania służy funkcja `SetThreadPriority`.

Jeżeli chcemy chwilowo zawiesić działanie wątku, posłużymy się funkcją SuspendThread o następującej składni:

```
DWORD SuspendThread(HANDLE uchwyt_watku);
```

Jedynym parametrem funkcji jest uchwyt wstrzymywanego wątku. Wznowienie pracy wątku jest możliwe za pomocą funkcji ResumeThread:

```
DWORD ResumeThread(HANDLE uchwyt_watku);
```

Gdy działalność wątku zupełnie nam zbrzydła, możemy go usunąć za pomocą funkcji TerminateThread:

```
BOOL TerminateThread(HANDLE uchwyt_watku, DWORD kod_wyjscia);
```

Argument uchwyt_watku to — oczywiście — uchwyt usuwanego wątku. Posługując się parametrem kod_wyjscia, możemy wysłać kod zakończenia pracy wątku. Ten parametr najczęściej nie będzie nas interesował i w jego miejsce będziemy wpisywać liczbę zero.

Przystępujemy do budowania aplikacji uwzględniającej nowo poznane funkcje: SuspendThread i ResumeThread. Otwieramy nowy projekt o nazwie *Program22.dev*. Obszar roboczy okna podzielimy na dwie części: w części pierwszej umieścimy okno potomne klasy edycji, w części drugiej — przyciski typu BS_AUTORADIOBUTTON. Okno potomne tworzymy za pomocą funkcji CreateWindow. Okno klasy edycji wykreujemy przy użyciu następującego kodu:

```
static HWND hwndEdytora = CreateWindowEx(
       0,
       "EDIT",
       NULL,
       WS_VISIBLE | WS_CHILD | WS_BORDER | ES_MULTILINE | WS_VSCROLL | WS_HSCROLL,
       0, 0, 400, 200,
       hwnd, NULL, hInst, NULL);
```

Nazwę klasy okna potomnego podajemy jako drugi parametr funkcji CreateWindow. Natomiast styl okna zdefiniowany za pomocą stałych:

```
WS_VISIBLE | WS_CHILD | WS_BORDER | ES_MULTILINE | WS_VSCROLL | WS_HSCROLL
```

utworzy edytor wielowierszowy o poziomym i pionowym pasku przewijania. Podstawowe własności okna klasy edycji to możliwość edycji tekstu, jego pobierania i wstawiania tekstu nowego. Załóżmy, że chcemy pobrać wpisany w oknie tekst. Tę niezwykłą chęć możemy zaspokoić na dwa sposoby: wykorzystując funkcję GetWindowText lub wysyłając komunikat WM_GETTEXT. Kod programu, w którym zastosowaliśmy funkcję GetWindowText, będzie wyglądać następująco:

```
const int MaxT = 0xFFFF;
char tekst[MaxT];
GetWindowText(hwndEdytora, tekst, MaxT);
```

Analogiczny efekt uzyskamy, wysyłając komunikat WM_GETTEXT:

```
const int MaxT = 0xFFFF;
char tekst[MaxT];
SendMessage(hwndEdytora, WM_GETTEXT, MaxT, (LPARAM)tekst);
```

Parametrem MaxT oznaczamy maksymalną liczbę pobieranych znaków.

Wysłanie tekstu do okna edycji również możemy zrealizować na dwa sposoby. Pierwszym z nich jest użycie funkcji SetWindowText:

```
SetWindowText(hwndEdytora, tekst);
```

Drugim sposobem jest wysłanie komunikatu WM_SETTEXT:

```
SendMessage(hwndEdytora, WM_SETTEXT, 0, (LPARAM)(LPCTSTR)tekst);
```

Dość kłopotliwą własnością klasy edycji jest resetowanie pozycji karetki po każdorazowym wysłaniu tekstu do okna edycji. Do ustawienia pozycji karetki służy funkcja SetCaretPos, jednak — zgodnie z opisem *Pomocy Microsoft Windows* — funkcja działa tylko dla karetek utworzonych przez okno, czyli nie działa dla obiektów klasy edycji. Posłużymy się wybiegiem, a dokładniej wysłaniem komunikatu EM_SETSEL.

```
SendMessage(hwndEdytora, EM_SETSEL, poczatek, koniec);
```

Komunikat EM_SETSEL jest wysyłany w celu zaznaczenia bloku tekstu, począwszy od litery o indeksie poczatek i końcu oznaczonym indeksem koniec. Po wykonaniu tego zadania karetka jest ustawiana w punkcie oznaczonym indeksem koniec. Użycie komunikatu w następujący sposób:

```
SendMessage(hwndEdytora, EM_SETSEL, x, x);
```

jest łatwym sposobem przemieszczania pozycji karetki do pozycji x w oknie edycji.

Przejdźmy do omawiania kluczowej części programu, czyli do obsługi wątków. W chwili uruchomienia aplikacji tworzone są dwa wątki, z których aktywny może być tylko jeden lub żaden.

```
//uruchom pierwszy wątek sprawdzania pisowni
watek1 = CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)PisowniaPierwszaDuza,
                      hwndEdytora, 0, &p);
//uruchom drugi wątek sprawdzania pisowni
watek2 = CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)PisowniaWszystkieDuze,
                      hwndEdytora, 0, &p);
//wstrzymaj działanie wątku drugiego
SuspendThread(watek2);
//ustaw wskaźnik uruchomionego wątku
flaga = 1;
```

Zmienna globalna flaga służy do zaznaczenia aktywności wątku. Przełączanie aktywności wątków, zgodnie z konfiguracją przycisków klasy BS_AUTORADIOBUTTON, zostało zaimplementowane w kodzie obsługi komunikatu WM_COMMAND.

```
case WM_COMMAND:
    switch(LOWORD(wParam))
    {
        case 1001:
            switch(flaga)
            {
                case 2:
                    //wstrzymaj działanie wątku drugiego
                    SuspendThread(watek2);
```

```
                case 3:
                    //początek nowego formatowania ustawiany jest
                    //na ostatni wprowadzony znak
                    GetWindowText(hwndEdytora, tekst, MaxT);
                    pocz_format = strlen(tekst);
                    //ponownie uruchom wątek pierwszy
                    ResumeThread(watek1);
                    break;
            }
            flaga = 1;
            break;
        case 1002:
            switch(flaga)
            {
                case 1:
                    //wstrzymaj działanie wątku pierwszego
                    SuspendThread(watek1);
                case 3:
                    //początek nowego formatowania ustawiany jest na ostatni wprowadzony znak
                    GetWindowText(hwndEdytora, tekst, MaxT);
                    pocz_format = strlen(tekst);
                    //ponownie uruchom wątek drugi
                    ResumeThread(watek2);
                    break;
            }
            flaga = 2;
            break;
        case 1003:
            switch(flaga)
            {
                case 1:
                    //wstrzymaj działanie wątku pierwszego
                    SuspendThread(watek1);
                    break;
                case 2:
                    //wstrzymaj działanie wątku drugiego
                    SuspendThread(watek2);
                    break;
            }
            flaga = 3;
            break;
        case 1010:
            SendMessage(hwnd, WM_DESTROY, 0, 0);
            break;
    }
    break;
```

Dzięki uwzględnieniu zmiennej `pocz_poz` modyfikacje tekstu nie wywołują zmian w tekście wpisanym przed wybraniem kolejnego sposobu formatowania. Stałe parametru `wParam` odpowiadają identyfikatorom okien potomnych klasy `BS_AUTORADIOBUTTON`. Aktywny wątek co sekundę sprawdza wpisany tekst, generując zmiany zgodne z wybranym sposobem formatowania. Dzięki zastosowanym wątkom kod pliku *Program22_main.cpp* nie jest skomplikowany, w całości wygląda następująco:

```
#include <windows.h>

const int MaxT = 0xFFFF;
char tekst[MaxT];

/* znacznik uruchomionego wątku*/
int flaga;
/* Uchwyty wątków*/
HANDLE watek1, watek2;
/* wskaźnik początku formatowania tekstu */
int pocz_format;

/* Deklaracja funkcji wątku */
DWORD PisowniaPierwszaDuza(LPVOID parametr);
DWORD PisowniaWszystkieDuze(LPVOID parametr);

LRESULT CALLBACK WindowProcedure (HWND, UINT, WPARAM, LPARAM);
char szClassName[ ] = "WindowsApp";

int WINAPI WinMain (HINSTANCE hThisInstance,
                    HINSTANCE hPrevInstance,
                    LPSTR lpszArgument,
                    int nFunsterStil)
{
    HWND hwnd;
    MSG messages;
    WNDCLASSEX wincl;

    wincl.hInstance = hThisInstance;
    wincl.lpszClassName = szClassName;
    wincl.lpfnWndProc = WindowProcedure;
    wincl.style = CS_DBLCLKS;
    wincl.cbSize = sizeof (WNDCLASSEX);
    wincl.hIcon = LoadIcon (NULL, IDI_APPLICATION);
    wincl.hIconSm = LoadIcon (NULL, IDI_APPLICATION);
    wincl.hCursor = LoadCursor (NULL, IDC_ARROW);
    wincl.lpszMenuName = NULL;
    wincl.cbClsExtra = 0;
    wincl.cbWndExtra = 0;
    wincl.hbrBackground = (HBRUSH) COLOR_BTNSHADOW;

    if (!RegisterClassEx (&wincl))
        return 0;

    hwnd = CreateWindowEx (
           0,
           szClassName,
           "Program22",
           WS_OVERLAPPEDWINDOW,
           CW_USEDEFAULT,
           CW_USEDEFAULT,
           410,
           375,
           HWND_DESKTOP,
           NULL,
           hThisInstance,
```

```
            NULL
            );

    ShowWindow (hwnd, nFunsterStil);

    while (GetMessage (&messages, NULL, 0, 0))
    {
        TranslateMessage(&messages);
        DispatchMessage(&messages);
    }

    return messages.wParam;
}

LRESULT CALLBACK WindowProcedure (HWND hwnd, UINT message, WPARAM wParam,
↪LPARAM lParam)
{
    DWORD p;

    switch (message)
    {
        case WM_CREATE:
            static HINSTANCE hInst = (HINSTANCE)GetWindowLong(hwnd, GWL_HINSTANCE);

            //utwórz okno edycji
            static HWND hwndEdytora = CreateWindowEx(
            0, "EDIT", NULL,
            WS_VISIBLE | WS_CHILD | WS_BORDER | ES_MULTILINE | WS_VSCROLL |
            ↪WS_HSCROLL,
            0, 0, 400, 200, hwnd, NULL, hInst, NULL);

            CreateWindowEx(0, "BUTTON", "Opcje formatowania tekstu",
            WS_VISIBLE | WS_CHILD | BS_GROUPBOX,
            5, 210, 220, 110, hwnd, NULL, hInst, NULL);

            static HWND hwndRadio1 = CreateWindowEx(
            0, "BUTTON", "Duże pierwsze litery",
            WS_VISIBLE | WS_CHILD | WS_GROUP | BS_AUTORADIOBUTTON,
            10, 240, 170, 20, hwnd, (HMENU)1001, hInst, NULL);

            static HWND hwndRadio2 = CreateWindowEx(
            0, "BUTTON", "Duże wszystkie litery",
            WS_VISIBLE | WS_CHILD | BS_AUTORADIOBUTTON,
            10, 260, 170, 20, hwnd, (HMENU)1002, hInst, NULL);

            static HWND hwndRadio3 = CreateWindowEx(
            0, "BUTTON", "Bez modyfikacji",
            WS_VISIBLE | WS_CHILD | BS_AUTORADIOBUTTON,
            10, 280, 170, 20, hwnd, (HMENU)1003, hInst, NULL);

            CreateWindowEx(0, "BUTTON", "Koniec",
            WS_VISIBLE | WS_CHILD, 260, 230, 110, 80,
            hwnd, (HMENU)1010, hInst, NULL);

            //uruchom pierwszy wątek sprawdzania pisowni
            watek1 = CreateThread(NULL, 0,
                 (LPTHREAD_START_ROUTINE)PisowniaPierwszaDuza, hwndEdytora, 0, &p);
```

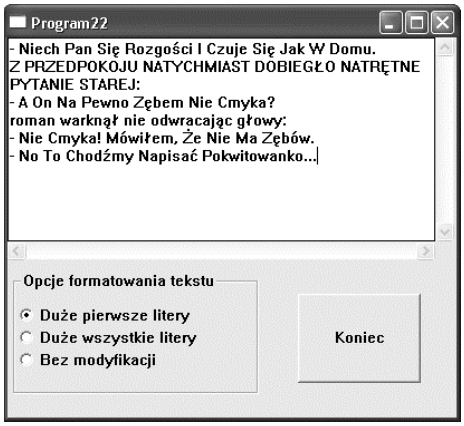
```
        //uruchom drugi wątek sprawdzania pisowni
        watek2 = CreateThread(NULL, 0,
            (LPTHREAD_START_ROUTINE)PisowniaWszystkieDuze, hwndEdytora, 0, &p);
        //wstrzymaj działanie wątku drugiego
        SuspendThread(watek2);
        //ustaw wskaźnik uruchomionego wątku
        flaga = 1;
        //ustaw domyślną opcję
        SendMessage(hwndRadio1, BM_SETCHECK, 1001, 0);
        //ustaw zakres zmian tekstu
        pocz_format = 0;
        break;
    case WM_COMMAND:
        switch(LOWORD(wParam))
        {
            case 1001:
                switch(flaga)
                {
                    case 2:
                        //wstrzymaj działanie wątku drugiego
                        SuspendThread(watek2);
                    case 3:
                        //początek nowego formatowania ustawiany jest na ostatni
                        //wprowadzony znak
                        GetWindowText(hwndEdytora, tekst, MaxT);
                        pocz_format = strlen(tekst);
                        //ponownie uruchom wątek pierwszy
                        ResumeThread(watek1);
                        break;
                }
                flaga = 1;
                break;
            case 1002:
                switch(flaga)
                {
                    case 1:
                        //wstrzymaj działanie wątku pierwszego
                        SuspendThread(watek1);
                    case 3:
                        //początek nowego formatowania ustawiany jest na ostatni
                        //wprowadzony znak
                        GetWindowText(hwndEdytora, tekst, MaxT);
                        pocz_format = strlen(tekst);
                        //ponownie uruchom wątek drugi
                        ResumeThread(watek2);
                        break;
                }
                flaga = 2;
                break;
            case 1003:
                switch(flaga)
                {
                    case 1:
                        //wstrzymaj działanie wątku pierwszego
                        SuspendThread(watek1);
                        break;
```

```
                        case 2:
                            //wstrzymaj działanie wątku drugiego
                            SuspendThread(watek2);
                            break;
                    }
                    flaga = 3;
                    break;
                case 1010:
                    SendMessage(hwnd, WM_DESTROY, 0, 0);
                    break;
            }
            break;
        case WM_DESTROY:
            PostQuitMessage (0);
            break;
        default:
            return DefWindowProc (hwnd, message, wParam, lParam);
    }

    return 0;
}

DWORD PisowniaPierwszaDuza(LPVOID parametr)
{
    int i, pozycja;

    //działaj aż do odwołania
    while(TRUE)
    {
        //pobierz tekst okna
        GetWindowText((HWND)parametr, tekst, MaxT);

        //popraw tekst
        i = pocz_format;
        while(tekst[i])
        {
            //postaw dużą literę, gdy jest to pierwszy znak
            if((i==pocz_format)&&(tekst[i]>=97&&tekst[i]<=122)) tekst[i] -= 32;
            //lub jest to znak po spacji, lub znaku nowej linii
            if(tekst[i]==32||tekst[i]=='\n')
            {
                if(tekst[i+1]>=97&&tekst[i+1]<=122) tekst[i+1] -= 32;
                //sprawdź polskie znaki
                if(tekst[i+1]=='ą') tekst[i+1] = 'Ą';
                if(tekst[i+1]=='ć') tekst[i+1] = 'Ć';
                if(tekst[i+1]=='ę') tekst[i+1] = 'Ę';
                if(tekst[i+1]=='ł') tekst[i+1] = 'Ł';
                if(tekst[i+1]=='ń') tekst[i+1] = 'Ń';
                if(tekst[i+1]=='ó') tekst[i+1] = 'Ó';
                if(tekst[i+1]=='ś') tekst[i+1] = 'Ś';
                if(tekst[i+1]=='ź') tekst[i+1] = 'Ź';
                if(tekst[i+1]=='ż') tekst[i+1] = 'Ż';
            }

            //zwiększ licznik
            i++;
```

```
        }

        //wyślij poprawiony tekst do okna edycji
        SetWindowText((HWND)parametr, tekst);
        //ustaw karetkę na koniec tekstu
        SendMessage((HWND)parametr, EM_SETSEL, i, i);

        //zaczekaj
        Sleep(1000);
    }

    return 0;
}

DWORD PisowniaWszystkieDuze(LPVOID parametr)
{
    int i, pozycja;

    //działaj aż do odwołania
    while(TRUE)
    {
        //pobierz tekst okna
        GetWindowText((HWND)parametr, tekst, MaxT);

        //popraw tekst
        i = pocz_format;
        while(tekst[i])
        {
            //postaw dużą literę w miejsce małej
            if(tekst[i]>=97&&tekst[i]<=122) tekst[i] -= 32;
            //sprawdź polskie znaki
            if(tekst[i]=='ą') tekst[i] = 'Ą';
            if(tekst[i]=='ć') tekst[i] = 'Ć';
            if(tekst[i]=='ę') tekst[i] = 'Ę';
            if(tekst[i]=='ł') tekst[i] = 'Ł';
            if(tekst[i]=='ń') tekst[i] = 'Ń';
            if(tekst[i]=='ó') tekst[i] = 'Ó';
            if(tekst[i]=='ś') tekst[i] = 'Ś';
            if(tekst[i]=='ź') tekst[i] = 'Ź';
            if(tekst[i]=='ż') tekst[i] = 'Ż';

            //zwiększ licznik
            i++;
        }

        //wyślij poprawiony tekst do okna edycji
        SetWindowText((HWND)parametr, tekst);
        //ustaw karetkę na koniec tekstu
        SendMessage((HWND)parametr, EM_SETSEL, i, i);

        //zaczekaj
        Sleep(1000);
    }

    return 0;
}
```

Okno główne działającej aplikacji prezentuję na rysunku 8.5. W celu zilustrowania działania programu wykorzystałem fragment wspaniałej powieści Arkadija i Borysa Strugackich *Poniedziałek zaczyna się w sobotę*[1].

Rysunek 8.5.
Okno główne aplikacji Program22



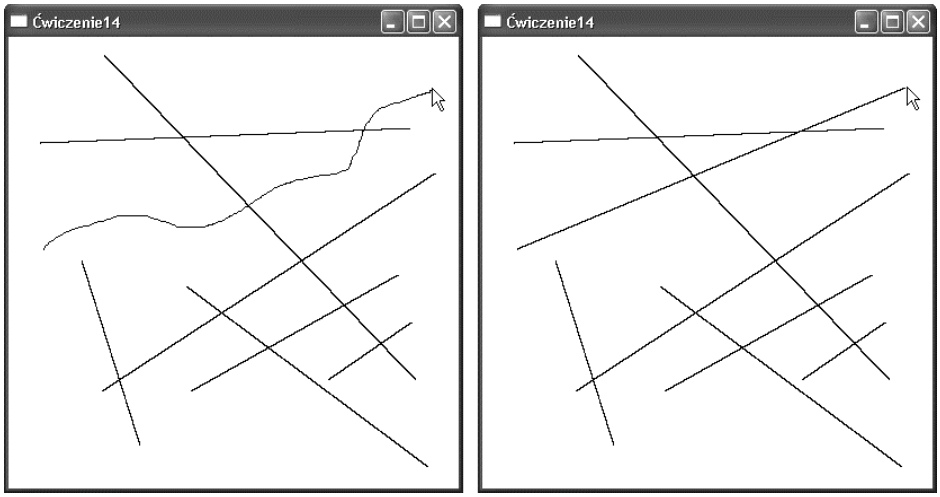
8.4. Ćwiczenia

Ćwiczenie 13. Zaprojektuj aplikację, w której w obszarze roboczym okna wyświetlany będzie czas systemowy. W lewej części obszaru okna czas będzie wyświetlany w postaci cyfrowej, prawą część okna zajmie tarcza zegara o klasycznej postaci (patrz rysunek 8.6). Do obsługi zegarów użyj dwóch wątków.

Rysunek 8.6.
Okno główne aplikacji Cwiczenie13

Ćwiczenie 14. Uruchomione w aplikacji *Program22* wątki służyły do automatycznego poprawiania pisowni w edytorze tekstowym. Zaprojektuj podobną aplikację działającą w edytorze graficznym, w której uruchomiony wątek będzie zamieniał narysowane linie krzywe w linie proste (sposób działania programu zilustrowano na rysunku 8.7).

[1] Arkadij Strugacki, Borys Strugacki, *Poniedziałek zaczyna się w sobotę*, tłum. Irena Piotrowska, Warszawa 1970, s. 17.



Rysunek 8.7. *Ilustracja działania programu Cwiczenie14*