Laboratorium Systemów Operacyjnych 1 *w środowisku* WINDOWS

Jak korzystać z funkcji plikowych SO ?

Na czym polega technologia *mapowania plików* ?

Jak z poziomu jednego programu uruchomić inny i przekazać do niego parametry?

W jaki sposób program może wykonywać klika czynności (wątki) jednocześnie?

Jak synchronizować działanie wątków i procesów?

Jak korzystać z czasomierzy do uruchamiania wątków w zadanym czasie ?

L I T E R A T U R A

[1] Jeffrey Richter: Programowanie Aplikacji dla Microsoft Windows. M, Warszawa 2002.

[2] Jeffrey Richter, Christophe Nasarre: Windows via C/C++. Microsoft Press, 2008.

TEMATYKA

I. Wstęp -plik **SO1_LAB0**: funkcja main(), typów danych w Windows, API, aplikacje systemu Windows, uchwyt instancji procesu, funkcja MessageBoxEx

1. Wybrane funkcje ogólne i pliki dyskowe - plik **SO1_LAB1**: pomiar czasu wykonania programu, czas i data, funkcje na pamięci RAM, funkcja GetLastError, funkcje obsługujące pliki dyskowe.
2. Mapowanie plików -plik **SO1_LAB2**: funkcje obsługujące pliki mapowane w RAM.
3. Procesy -plik **SO1_LAB3**: tworzenie procesu potomnego, metody kończenia procesu.
4. Przekazywanie informacji między procesami: -plik **SO1_LAB4**: dziedziczenie uchwytów Obiektu, nadawanie nazw Obiektom.
5. Wątki -plik **SO1_LAB5**:
6. Synchronizacja -plik **SO1_LAB6**:
7. Zaliczenie laboratorium: realizacja postawione zadania (czas 60 min – 70 min).

I. WSTĘP

Konsolowy program wyglądem podobny do aplikacji DOS-owych uruchamianych pod Windows.
Jest to 32-bitowa aplikacja nie posiadająca własnego interfejsu graficznego.
Pracuje w trybie znakowym. Komunikację z użytkownikiem zapewnia Okno Konsoli.
Punktem wejściowym jest funkcja **main**.

W zintegrowanym środowisku programistycznych, z reguły kompiluje się ***projekty*** a nie programy. Projekt *(project)* zawiera definicje czynności, jakie musi wykonać zintegrowane środowisko programistyczne, aby wygenerować wykonywalny kod.

❒ Parametry funkcji main

Uruchamiając program, czasami wygodnie jest podać parametry dla niego w wierszu poleceń.

MojProgram_exe 12 opis 23 56

Język C++ daje taką możliwość, poprzez użycie w funkcji **main** standardowych parametrów.

int **main**(int **argc**, char ***argv**[ ])

Pierwszy parametr to **liczba typu int**, drugim jest **tablica wskaźników** na łańcuchy.

argc (*argument count*) – po wykonaniu programu zawiera liczbę parametrów znajdujących się w wierszu poleceń, parametry oddzielają spacje;

argv[**0**] –wskaźnik łańcucha, zawierającego ścieżkę dostępu do uruchomionego programu;

argv[*pozostałe*] –**wskaźniki łańcuchów** zawierające kolejne parametry wołania (bez znaków rozdzielających poszczególne parametry jak *spacja*, *tabulator*).

Deklaracja: **char *argv[ ]**
jest równoważna: **char **argv;** .

Niech program **Test.exe** znajduje się na dysku **D** w katalogu SO_LAB.

Został uruchomiony z wiersza poleceń:

Test 1991 Maj 9

Wynik działania:

argc → 4
argv[0] → "D:\SO_LAB\Test.exe"
argv[1] → "2015"
argv[2] → "Maj"
argv[3] → "9"

```
#include <cstdlib>
#include <iostream>
#include <cmath>
using namespace std;
int main(int argc, char *argv[ ])          // Prog1
{
double x;
  if (argc < 2 ) {cout << "Brak parametrow w " << *argv ;
                  cout << endl;   cin.get();   return -1;}
for (int i=1; i < argc; i++) {
  x = atof(argv[i]);     // konwersja łańcucha na liczbę
  cout <<  "sqrt(" << x << ") = " << sqrt(x) << endl;
  }
cout << "Koniec Prog1";
// cin.get();
return 0;
}
```

Prog1 4 8.8 25

sqrt(4) = 2
sqrt(8.8) = 2.96648
sqrt(25) = 5

```
#include <cstdlib>
int atoi(const char *str);
double atof(const char *str)
```

atoi converts a string pointed to by **str** to **int**;
In this function, the first unrecognized character ends the conversion.
There are no provisions for overflow in atoi (results are undefined).
If the string cannot be converted to a umber of the corresponding type (int), **atoi** returns 0.

Program **Test_BubbleSort** generuje dane dla parametrów z wiersza poleceń, następnie sortuje je.

```
#include<iomanip>
#include<iostream>
#include <cstdlib>
#include<cmath>
#include<ctime>
using namespace std;
void BubbleSort(double *, int);
void DispV(int, int, double *,char *);
double Generuj(float a, float b)
  { double w = (a + (b - a)*(double)rand()/RAND_MAX);  return floor(w*100+0.5)/100; }
const int MaxObieg = 5;
int main(int argc, char *argv[ ])                    // Test_BubbleSort
{
clock_t T1, T2;
double Time, oda, dob;
int  nData;
if (argc <= 1 ) { nData = 15000; oda = 2.2; dob = 88.8; }
  else { nData= atoi(argv[1]); oda = atof(argv[2]); dob = atof(argv[3]); }
double *A = new double[nData];                       // dynamiczna tablica
for (int k = 0; k < MaxObieg; k++) {
    for (int i = 0; i < nData; i++) A[i] = Generuj(oda, dob);      //generowanie danych
            if (k==0) { cout << "Fragment Danych ( z " << nData << "):"; DispV(0, 32, A , ""); }
    T1 = clock();
      BubbleSort(A, nData);                          // sortowanie
    T2 = clock();
    Time = (T2 - T1)/(double)CLOCKS_PER_SEC;
            cout << "Obieg Nr "<< k << "  czas = " << Time << endl;
  }
DispV(0, 32, A, "Fragment po Sortowaniu:" );
delete [ ] A;
cout << "Koniec Test_BubbleSort";  cin.get();
return 0;
}

void BubbleSort(double *X, int size)
{
double w;
for (int i = 1; i < size; i++)
  for (int j = size-1; j >= i; j--)
   if (X[j] < X[j - 1]) { w=X[j-1]; X[j-1]=X[j]; X[j]=w; };
}
void DispV(int a, int b, double V[ ], char *text )
{
  cout << text;
  for(int i=a; i < b; i++) {
     if (i%8 == 0) cout << endl;
     cout << setw(9) << V[i];
     }   cout << endl;
}
```

```
Fragment Danych ( z 15000):
    2.31  51.01  18.94  72.24  52.86  43.76  32.54  79.79
   73.46  66.86  17.28  76.58  63.73  46.67  28.53   3.5
   10.12  33.76  14.96  16.57  87.81   40.8  12.51   2.6
    2.97  34.92  48.24  51.66  54.31  54.78   16.6  59.62
Obieg Nr 0  czas = 2.484
Obieg Nr 1  czas = 2.5
Obieg Nr 2  czas = 2.469
Obieg Nr 3  czas = 2.485
Obieg Nr 4  czas = 2.484
Fragment po Sortowaniu:
     2.2  2.21  2.21  2.21  2.22  2.22  2.22  2.23
    2.23  2.23  2.25  2.26  2.26  2.27  2.27  2.29
    2.29   2.3   2.3   2.3   2.3  2.31  2.32  2.32
    2.33  2.33  2.33  2.34  2.34  2.35  2.35  2.35
Koniec Test BubbleSort
```


❒ Uwagi o konfiguracji środowiska dla aplikacji wielowątkowej

Tworząc wielowątkowy program w C++ należy korzystać z **wielowątkowej** wersji biblioteki czasu wykonywania C++, gdyż biblioteka jednowątkowa nie działa prawidłowo z aplikacjami wielowątkowymi.

Jeśli użyjesz funkcji **_beginthreadex** -*korzystając z jednowątkowej wersji tej biblioteki*- program łączący zgłosi błąd *„unresolved external symbol"*.

Deklaracje funkcji _beginthread i _beginthreadex znajdują się pliku nagłówkowego <Process.h> .
Nagłówki funkcji są definiowane tylko wtedy, gdy zostanie odnaleziony identyfikator **_MT**.

Tworząc nowy projekt wiele środowisk standardowo wybiera jednowątkową bibliotekę.

W przypadku aplikacji wielowątkowych trzeba jawnie zmienić ustawienie na wielowątkową bibliotekę czasu wykonywania C++.

☞ **W środowisku** Dev C++ **nie trzeba nic ustawiać.**
Środowisko Dev C++ jest automatycznie skonfigurowane dla aplikacji wielowątkowych.

❒ Interpretacja typów danych w Windows

HINSTANCE	void*
LPSTR	char*
LPINT	int*
LPCSTR	const char*
UINT	unsigned int

Nazwy typów związanych z uchwytami (wskaźnikami) rozpoczynają się od litery **H**.

Nazwy zawierające ciąg **STR** odnoszą się do tablicy znaków zakończonych bajtem zerowym.

Litera **P** oznacza wskaźnik (pointer).

Litera **L** zmienną typu long.

Litera **W** oznacza platformę Unicode.

Litera **T** oznacza niezależność od platformy (ANSI lub Unicode), np. **LPCTSTR** to adres początku stałej tablicy znaków na aktualnie obowiązującej platformie.

❒ API

API - *Application **P**rogramming **I**nterface***: zestaw funkcji umożliwiających komunikowania się z systemem operacyjnym.

Plik nagłówkowy **<windows.h>** zawiera ich deklaracje.

Interfejs **API** dostarcza funkcji umożliwiających:

- ➤ Dostęp do Obiektów Jądra,
  (wywołanie funkcji **Create**NazwaObiektu)
- ➤ Wspólne korzystanie z obiektów,
  (dziedziczenie uchwytu, nadanie nazwy obiektowi; DuplicateHandle)
- ➤ Zarządzanie procesami: 4-ry klasy priorytetów procesu)
- ➤ Komunikacja międzyprocesorowa,
  (wspólne używanie Obiektów Jądra, przekazywanie komunikatu)
- ➤ Zarządzanie pamięcią,
  (pamięć wirtualna, pliki odwzorowane w pamięci, sterty, lokalna pamięć wątku)
- ➤ Tworzenia aplikacji okienkowych.

Punktem wejściowym dla aplikacji okienkowej jest funkcja **WinMain.**

int **WINAPI** **WinMain**(HINSTANCE ***hInstance***, HINSTANCE ***hPrevInstance***,
LPSTR ***lpCmdLine***, int ***iCmdShow***)

***hInstance**: uchwyt wystąpienia programu.*

W systemie wielozadaniowym można uruchomić jednocześnie wiele kopii tego samego programu (obrazy programu w pamięci operacyjnej) nazywane egzemplarzami (instances).

hPrevInstance: pozostałością po 16-bitowych wersjach Windows, ma zawsze wartość **NULL**.

lpCmdLine: zawiera adres łańcucha znaków, w którym przechowywane są argumenty wywołania danego programu.

***iCmdShow**:* informuje program o sposobie, w jaki powinno ukazać się główne okno aplikacji.

SW_HIDE	Hides the window and activates another window.
SW_MINIMIZE	Minimizes the specified window .
SW_RESTORE	Activates and displays a window.
SW_SHOW	Activates a window and displays it in its current size and position.
SW_SHOWMAXIMIZED	Activates a window and displays it as a maximized window.

Identyfikator **WINAPI** oznacza, że funkcja **WinMain** w specjalny sposób komunikuje się z systemem operacyjnym.

SO może być napisany w dowolnym języku programowania i powinien współpracować z programami napisanymi w różnych językach, a nie tylko w C++.

Wszystkie funkcje interfejsu API są typu **WINAPI**.

Należy deklarować typy wszystkich pisanych funkcji wywoływanych *bezpośrednio* przez system operacyjny, jako **WINAPI** lub, w innym kontekście jako **CALLBACK** (funkcji wywoływanej w odpowiednim kontekście przez system operacyjny).

❒ Aplikacje systemu Windows

System Windows obsługuje dwa rodzaje aplikacji opartej na:

1. konsolowym interfejsie użytkownika (*Console User Interface* - **CUI**).
2. graficznym interfejsie użytkownika (*Graphical User Interface* - **GUI**)

Aplikacje **CUI** komunikują się w trybie tekstowym.
Zazwyczaj nie tworzą okien ani komunikatów procesowych.
Nie wymagają graficznego interfejsu użytkownika.
Zgłaszają się na ekranie wewnątrz okna jako okno tekstowe.
Interpreter poleceń - CMD.EXE.

Aplikacje GUI komunikują się przy użyciu okien, menu, interakcyjnych okien dialogowych.
`Notatnik`, **`Kalkulator`** czy **`WordPad`** oparte są na **GUI**.

☞ Można utworzyć aplikację **CUI**, która będzie wyświetlać okna dialogowe.

☞ Można utworzyć **GUI**, która będzie wyświetlać napisy w oknie konsoli.

Aby osadzić stosowny podsystem w kodzie wynikowym Visual C++, zintegrowane środowisko ustawia klucz konsolidatora (linkera) na:

/SUBSYSTEM:WINDOWS dla aplikacji **GUI**,
/SUBSYSTEM:CONSOLE dla aplikacji **CUI**,

Po uruchomieniu aplikacji przez użytkownika konsolidator systemu operacyjnego (loader) zagląda do nagłówka obrazu kodu i odczytuje rodzaj podsystemu.

Dla aplikacji **CUI** konsolidator otwiera okna konsoli tekstowej.
Dla aplikacji **GUI** konsolidator nie otwiera okna konsoli tylko ładuje tę aplikację.

Po rozpoczęciu działania aplikacji System Operacyjny nie interesuje się, jakiego typu interfejs użytkownika ma ta aplikacja.

Aplikacja Windows musi zawierać funkcję stanowiącą „***punkt wejścia***".
Jest to funkcja wywoływana w momencie rozpoczęcia działania aplikacji.

Dla aplikacji wykorzystującej kod ANSI:

int WINAPI **WinMain**(HINSTANCE hInstance, HINSTANCE, LPSTR lpCmdLine, int iCmdShow);

int __cdecl **main**(int argc, char *argv[ ] , char *envp[ ]) ;

Dla aplikacji wykorzystującej Unicod: **wWinMain(...), wmain(...)**

☛ W rzeczywistości system operacyjny nie wywołuje funkcji użytkownika, wskazanej jako punkt wejścia, lecz specjalną **funkcję startową czasu wykonywania C/C++**.

Funkcja ta inicjalizuje bibliotekę czasu wykonywania C++, oraz przygotowuje zadeklarowane globalne i statyczne obiekty C++ przed wykonaniem kodu.

Wyboru funkcji startowej czasu wykonywania C/C++ dokonuje konsolidator (linker).

➡ Dla klucza /SUBSYSTEM:WINDOWS, konsolidator poszukuje w kodzie funkcję **WinMain**.
Jeśli nie znajdzie zwraca błąd "*unresolved external symbol*", w przeciwnym razie wybiera funkcję **WinMainCRTStartup**.

➡ Dla klucza /SUBSYSTEM:CONSOLE, konsolidator poszukuje w kodzie funkcję **main**.
Jeśli nie znajdzie zwraca błąd "*unresolved external symbol*", w przeciwnym razie wybiera funkcję **mainCRTStartup**.

Można usunąć klucz /SUBSYSTEM z **projektu**.

Wtedy konsolidator automatycznie ustali rodzaj podsystemu używany przez aplikację, na podstawie odnalezionej w kodzie funkcji ***punktu wejścia***.

Jeżeli aplikacja **Win32** (Visual C++) będzie miała punkt wejścia funkcję **main** to wystąpi zgłoszenie błędu, gdyż ***projekt*** aplikacji Win32 ma automatycznie ustawiony klucz /SUBSYSTEM:WINDOWS, a kod nie zawiera funkcji **WinMain**.

➔ Aplikacja **Win32** w Dev C++ akceptuje jako punkt wejścia funkcje **main**.

Funkcja startowa wykonuje:

- pobiera wskaźnik do linii polecenia nowego procesu;
- pobiera wskaźnik do zmiennych środowiskowych nowego procesu;
- inicjalizuje zmienne globalne czasu wykonywania C/C++, dostępne w kodzie użytkownika (plik <stdlib.h>)

  np. : unsigned int **__argc**, char ** **__argv**, **__environ** , char * **__pgmpir ;**
- inicjalizuje stertę używaną przez funkcje czasu wykonywania C++ alokujące pamięć;
- wywołuje konstruktory wszystkich globalnych i statycznych obiektów klas C++.

Po inicjalizacji funkcja startowa C++ wywołuje funkcję stanowiącą punkt wejścia do aplikacji; postać wywołania w przypadku funkcji **main**:

```
int nMainRetVal = main(__argc, __argv, _environ);
```

❒ Uchwyt instancji procesu

Każdy plik wykonywalny lub DLL po załadowaniu do przestrzeni adresowej procesu ma przypisywany unikatowy **uchwyt** instancji.

Instancja wykonywalnego pliku przekazywana jest przez pierwszy parametr **WinMain**(..).

Faktyczną wartością **1-go** parametru funkcji **WinMain** jest adres **bazowy** pamięci, pod jakim system załadował obraz pliku wykonywalnego do przestrzeni adresowej procesu.

Adres bazowy, pod który system ładuje obraz pliku wykonywalnego, jest ustalany przez konsolidator.

Dla Visual C++ konsolidator używa adresu 0x00400000.

Funkcja *GetModuleHandle* zwraca uchwyt bazowy pliku wykonywalnego lub DLL załadowanego do przestrzeni adresowej procesu:

HMODULE **GetModuleHandle**(LPCTSTR ***lpModuleName***);

lpModuleName -wskaźnik do napisu z nazwą pliku wykonywalnego lub DLL, załadowanego do przestrzeni adresowej procesu wywołującego funkcję.

Funkcja zwraca adres bazowy miejsca załadowania lub zwraca NULL.

☛ Wywołanie **GetModuleHandle**(*NULL*) zwraca adres bazowego pliku wykonywalnego w przestrzeni adresowej procesu wywołującego tę funkcję.

☛ Funkcja sprawdza wyłącznie przestrzeń adresową **wywołującego ją** procesu.

Uwaga: Umieszczając wywołanie w kodzie DLL, funkcja zwróci adres bazowy pliku wykonywalnego procesu, **a nie pliku DLL**

Funkcja MessageBoxEx

Jest to funkcja interfejsu API służąca do tworzenia aplikacji okienkowych.
Funkcje tego typu nie są tematem bieżącego laboratorium.

int **MessageBoxEx**(HWND ***hWnd***, LPCTSTR ***lpText***, **LPCTSTR *lpCaption***, UINT ***uType***, WORD ***wLanguageId***);

hWnd: uchwyt okna będącego właścicielem (rodzicem) tworzonego właśnie okna, wartość NULL oznacza brak takiego okna.

pText: adres tekstu wyświetlanego w oknie,

lpCaption: tytuł tworzonego okna,

uType: styl tworzonego okna. Jest on kombinacją flag (bitów) określających właściwości okienka. Flagi łączy się przy pomocy operatora sumy bitowej.
Flagom odpowiadają identyfikatory zdefiniowane w pliku <windows.h>.

wLanguageId : język, w którym wyświetlane będą napisy na przyciskach.

Funkcja **MessageBoxEx** zwraca liczbę całkowitą, identyfikującą naciśnięty przycisk.
Może przyjąć wartości:
IDABORT, **IDCANCEL**, **IDIGNORE**, **IDNO**, **IDOK**, **IDRETRY** lub **IDYES**;
co odpowiada przyciśnięciu klawisza:
ABORT, CANCEL, IGNORE, NO, OK, RETRY lub YES.

Przykładowa kombinacja flag: **MB_OKCANCEL | MB_ICONQUESTION | MB_HELP**
powoduje wyświetlenie w okienku dwóch przycisków z napisem: **OK** oraz **Cancel**.
dodatkowo wyświetlona zostanie ikona ze znakiem ? oraz klawisz z napisem Help, który będzie również reagował na naciśnięcie klawisza F1-uruchomienie systemu pomocy (jeżeli został dołączony do programu).

Przyciski definiują flagi: **MB_ABORTRETRYIGNORE** **MB_OKCANCEL**
MB_RETRYCANCEL **MB_YESNO** **MB_YESNOCANCEL**

Standardowe ikony np.:  definiują następujące flagi:

MB_ICONEXCLAMATION **MB_ICONWARNING** **MB_ICONINFORMATION**
MB_ICONASTERISK **MB_ICONQUESTION** **MB_ICONSTOP**
MB_ICONERROR **MB_ICONHAN**

```
#include <windows.h>                    // Prog2   uruchomiony konsolowo w Dev-C++
int WINAPI WinMain(HINSTANCE hInstance,   HINSTANCE hPrevInstance,
                   LPSTR lpCmdLine,   int uCmdShow)
{
  LPCTSTR tekst="Opis dzialania";
  MessageBoxEx(NULL, l tekst, "Tytul Nowego Okna", IDABORT, 0);
  return 0;
}
```