Wydział Elektrotechniki i Informatyki
Politechnika Rzeszowska im. Ignacego Łukasiewicza

Katedra Informatyki i Automatyki

Systemy operacyjne

Laboratorium

Kierunek: **Informatyka (EF-DI)**
Rok: 2

Rzeszów 2009

Systemy operacyjne

Laboratorium

Ćwiczenie 1

Architektura systemów Windows NT(2000,XP)

Architektura systemów Windows 95(98,Me)

Systemy Windows NT, 2000, XP

System operacyjny złożony jest z dwóch składników:

- komponentów trybu użytkownika
- komponentów trybu kernela

W skład komponentów trybu kernela wchodzą sterowniki urządzeń trybu kernela, kernel, sterowniki zarządzające operacjami wejścia-wyjścia, pamięcią, konfiguracją systemu, Plug & Play. Sterowniki urządzeń trybu kernela można porównać do wirtualnych urządzeń systemów Windows 9x. Obsługują one urządzenia fizyczne i mają dostęp do wewnętrznych struktur systemowych do których nie mają dostępu sterowniki trybu użytkownika. Sterowniki trybu kernela udostępniają standardowy zestaw funkcji służących do komunikacji z urządzeniem.

W skład komponentów trybu użytkownika wchodzą aplikacje i sterowniki trybu użytkownika. Sterowniki te można porównać ze sterownikami urządzeń systemów Windows 9x. Dostęp do niektórych struktur systemowych mogą one uzyskać za pomocą funkcji sterowników trybu kernela.

Podstawą kernela systemów opartych na Windows NT jest warstwa abstrakcji sprzętu (HAL- Hardware Abstraction Layer). Jest to składnik kernela, który izoluje oprogramowanie systemowe od fizycznych urządzeń. Komponenty trybu kernela uzyskują dostęp do sprzętu za pośrednictwem HAL.

Podstawowa różnica w działaniu systemów opartych na Windows NT w porównaniu z systemami opartymi na Windows 95 polega na tym, że Windows NT nie pozwala komponentom trybu użytkownika na bezpośredni dostęp do urządzeń sprzętowych.

Środowisko Plug & Play

W środowisku Plug & Play istnieją trzy składniki: sterownik konfiguracji, moduły wyliczające i moduły przydzielające zasoby. Kiedy uruchamiany jest system, sterownik konfiguracji uruchamia moduły wyliczające, które tworzą listę zasobów sprzętowych i wypełniają ja informacjami z rejestru systemu. Następnie sterownik konfiguracji używa modułów przydzielających zasoby, w celu ustalenia prawidłowej konfiguracji systemu.

Każde urządzenie posiada unikalny identyfikator - ciąg znaków utworzony przez moduł wyliczający. Identyfikator składa się z dwóch lub trzech części, oddzielonych znakiem '\'. Pierwsza część jest nazwą modułu wyliczającego, który wykrył urządzenie, druga część jest właściwym identyfikatorem urządzenia, trzecia służy do rozróżnienia dwóch urządzeń tej samej klasy (np. portów szeregowych).

Moduły wyliczające szukają informacji o wirtualnym urządzeniu, które obsługuje urządzenie fizyczne w rejestrze w kluczu `HKLM\System\CurrentControlSet\Enum\[id urządzenia]`. Wartość klucza o nazwie `Driver` określa klasę i nazwę urządzenia wirtualnego.

System przechowuje informacje o wszystkich zainstalowanych urządzeniach wirtualnych w rejestrze w kluczu

`HKLM\System\CurrentControlSet\Services\Class\[klasa]\[nazwa]`.

Wartość klucza o nazwie `StaticVxd` określa nazwę pliku zawierającego program obsługi urządzenia wirtualnego, który zostaje wczytany przez moduł wyliczający. Jeżeli nie ma wartości o nazwie `StaticVxd`, wartość `DevLoader` mówi, jaki program dynamicznie wczytuje urządzenie wirtualne. Wartość `Driver` zawiera nazwę pliku, który zostanie wczytany, gdy urządzenie

wirtualne będzie potrzebne do pracy systemu (np. karta sieciowa nie jest używana cały czas). W niektórych przypadkach nazwa dynamicznego urządzenia wirtualnego przechowywana jest pod inną wartością, zależy to od programu wczytującego.
Informacje o urządzeniach niezgodnych ze standardem PnP przechowywane są w kluczu `HKLM\System\CurrentControlSet\Enum\Root\[id urządzenia]`.

Nazwy statycznych urządzeń wirtualnych wczytywanych przy starcie systemu przechowywane są w kluczu `HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services`.

Przykład

Moduł wyliczający sprawdzający magistrali PCI po przydzieleniu karcie identyfikatora `VEN_5333&DEV_8811\BUS_00&DEV_05&FUNC_00` znajduje w rejestrze wpisy:

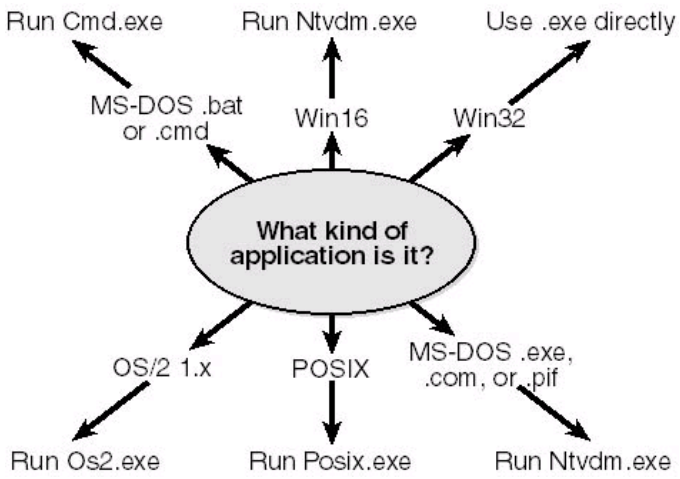
```
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Enum\PCI\VEN_5333&DEV_8811\BUS_00&DE
V_05&FUNC_00
CompatibleIDs="PCI\CC_030000,PCI\CC_0300"
DeviceDesc="S3 Trio32/64 PCI"
Driver="Display\0002"

HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\Class\Display\0002
  DevLoader="*vdd"
  InfPath="MICROS~2.INF"
  InfSection= "S3"
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\Class\Display\0002\DEFAULT
  CHIPID = 00 11 88 00 00 00
  minivdd="s3.vxd"
  drv="s3.drv"
```

Urządzenie wirtualne obsługujące kartę S3 Trio64 ładowane jest dynamicznie przez program `vdd.vxd`. Program ten szuka nazwy pliku, który ma wczytać w podkluczu `DEFAULT` w wartości o nazwie `minivdd`. Sterownikiem karty S3 jest biblioteka `s3.drv`.

Biblioteki DLL

W Microsoft Windows biblioteki DLL są modułami, które zawierają funkcje i dane. Biblioteka DLL jest wczytywana w czasie wykonywania aplikacji przez wywołujący ją proces. Kiedy kilka aplikacji używa tej samej biblioteki jednocześnie, każda dostaje kopię danych, ale współużytkują kod.

Interfejs programowania aplikacji Windows jest zbudowany jako zestaw bibliotek DLL. Aplikacje 16-bitowe korzystają z kreatora procesu Ntvdm.exe (*CreateProcess)*, aplikacje 32-bitowe używają funkcji z bibliotek gdi32.dll, user32.dll, kernel32.dll.



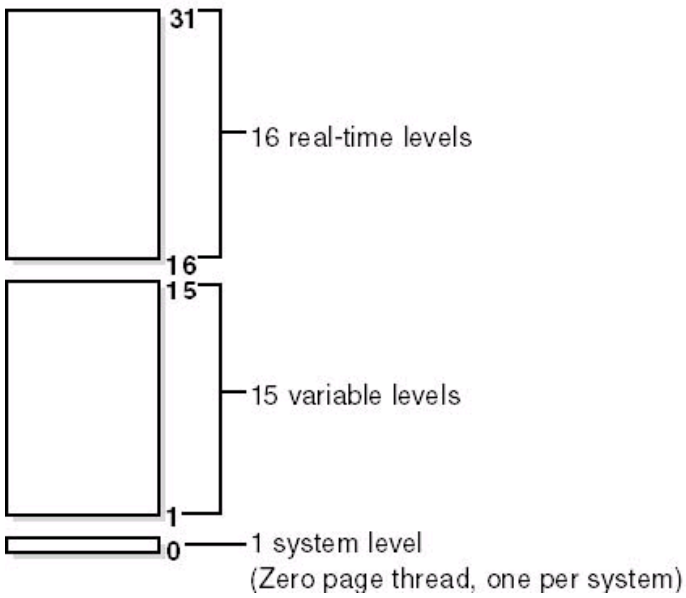
Twórcy oprogramowania mogą rozszerzać możliwości systemu przez tworzenie bibliotek DLL, które zawierają nowe funkcje ulepszające system i następnie przez udostępnienie tych bibliotek aplikacjom Windows. Biblioteki DLL najczęściej posiadają rozszerzenie .DLL, chociaż mogą mieć także rozszerzenie .EXE lub inne.

16- i 32-bitowe aplikacje Windows

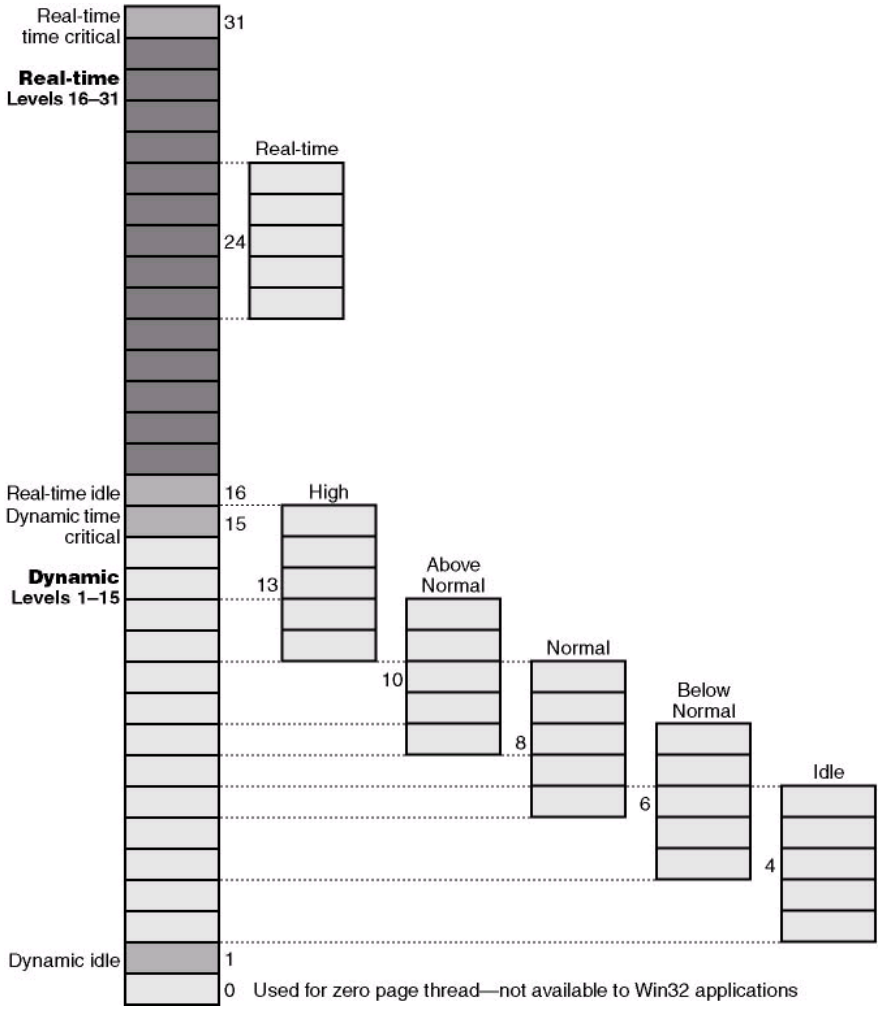
Windows pozwala uruchamiać 16-owe aplikacje napisane dla Windows w wersjach 3.x jak również aplikacje 32-bitowe, które używają funkcji Win32 lub Win32s. Dla aplikacji 16-bitowych Windows posiada tryb wielozadaniowości bez wywłaszczenia używany w Windows 3.x. Wszystkie aplikacje 16-bitowe wykonywane są jako jeden wątek. Inaczej jest w aplikacjach 32-bitowych: każda posiada jeden lub więcej wątków, każdy 32-bitowy wątek podlega mechanizmowi wielozadaniowości z wywłaszczeniem.

Procesy i wątki.

Aplikacja systemu Windows składa się z jednego lub kilku procesów. Procesem nazywamy wykonujący się program. Pojedynczy proces uruchamia jeden lub więcej wątków. Wątek jest podstawową jednostką, do której system operacyjny przydziela czas procesora. Każdy proces jest uruchamiany z pojedynczym wątkiem, nazywanym wątkiem głównym, ale może utworzyć dodatkowe wątki. Każdy wątek ma przydzielony priorytet wykonywania. Priorytet jest liczbą z zakresu od 0 (najniższy) do 31 (najwyższy).



Poziomy priorytetu wątków są wyznaczane z dwóch perspektyw: z Win32 API oraz z jądra Windows 2000. Win32 API organizuje procesy według priorytetu klasy do której są przeznaczone w tworzeniu (Czasu rzeczywistego , Wysoki, ponad Normalny, Normalny, poniżej Normalny, i bezczynny) i wtedy przez względne pierwszeństwo indywidualnych wątków w granicach tamtych procesów (Czasu - krytycznego, Najwyższy, ponad Normalny, Normalny, poniżej Normalny, Najniższy, i bezczynny).



Okna

Każda aplikacja graficzna systemu Windows tworzy przynajmniej jedno okno nazywane oknem głównym. Okno to służy do komunikacji między użytkownikiem a aplikacją. Większość aplikacji tworzy również inne okna, które wykorzystywane są przez okno główne.

Każde okno należy do pewnej klasy okien. Klasa okna określa wygląd okna i jego zachowanie. Głównych składnikiem klasy okna jest funkcja okienkowa, która otrzymuje i przetwarza wszystkie informacje dostarczane przez system do okna. System Windows komunikuje się z oknami za pomocą meldunków. Zmiana położenia kursora myszy, kliknięcie myszą, wciśnięcie klawisza powodują wysłanie przez system meldunku do aktywnego okna.

Okno może mieć nazwę. Nazwa jest identyfikatorem okna dla użytkownika. Okna główne i dialogowe wyświetlają nazwę na listwie tytułowej. W przypadku okien kontrolnych (przyciski, pola edycyjne, listy) widoczność nazwy okna zależy od jego klasy, np. przycisk (klasa *button*) wyświetla nazwę okna w zajmowanym obszarze.

Systemy Windows 95, 98, Me

Środowisko systemu operacyjnego Windows 9x składa się z części sprzętowej komputera i następujących składników programowych:

- sterownika maszyny wirtualnej (VMM - Virtual Machine Manager),
- urządzeń wirtualnych (Vxd),
- sterowników urządzeń,
- 16- i 32-bitowych bibliotek DLL Windows
- aplikacji opartych na systemie MS DOS
- 16- i 32-bitowych aplikacji Windows

Sterownik maszyny wirtualnej - VMM

VMM jest 32-bitowym systemem operacyjnym pracującym w trybie chronionym procesora. Jest to najbardziej wewnętrzny składnik systemu Windows. Jego pierwotnym zadaniem jest tworzenie, uruchamianie, nadzorowanie i przerywanie pracy maszyn wirtualnych. Do funkcji VMM należy zarządzanie pamięcią, procesami, przerwaniami i wyjątkami.

Pod pojęciem maszyny wirtualnej rozumiemy wykonujące się zadanie, które składa się z aplikacji, środowiska programowego (np. ROM BIOS i MS DOS), pamięci i rejestrów procesora.

VMM zapewnia wielozadaniowość z wywłaszczeniem. Uruchamia równolegle wiele aplikacji poprzez dzielenie czasu procesora pomiędzy wirtualne maszyny, na których zostały one uruchomione.

Urządzenia wirtualne - VxD

Urządzenia wirtualne są 32-bitowymi programami, które pozwalają uniezależnić VMM od sprzętu, poprzez zarządzanie sprzętowymi urządzeniami komputera i oprogramowaniem rozszerzającym system o dodatkowe funkcje. Urządzenia VxD obsługują wszystkie urządzenia sprzętowe typowego komputera, włączając programowalny sterownik przerwań, zegar, urządzenie DMA, sterownik dysku, porty szeregowe i równoległe, klawiaturę i kartę graficzną. Urządzenie wirtualne jest wymagane dla każdego urządzenia sprzętowego, które ma programowalne tryby pracy lub okresowo odbiera dane. Jeżeli stan urządzenia sprzętowego może zostać zaburzony w wyniku przełączenia między wirtualnymi maszynami lub aplikacjami, urządzenie musi posiadać własny sterownik VxD.

Niektóre urządzenia wirtualne obsługują oprogramowanie, a nie sprzęt. Ogólnie, urządzenie VxD może udostępnić dowolny rodzaj usług dla VMM lub innego wirtualnego urządzenia. Na przykład urządzeniem wirtualnym nie obsługującym żadnego urządzenia sprzętowego jest sterownik pamięci podręcznej dysku `smartdrv.exe`

Sterowniki urządzeń

Sterownik urządzenia systemu Windows jest biblioteką DLL, którą Windows używa w celu wymiany danych z urządzeniem sprzętowym, takim jak karta graficzna lub klawiatura. Zamiast bezpośrednio programować urządzenie, Windows wczytuje sterownik urządzenia i wywołuje funkcje sterownika, które wykonują odpowiednie operacje na urządzeniu. Każdy sterownik urządzenia posiada zestaw funkcji. Windows wywołuje te funkcje w celu wykonania zadania, takiego jak rysowanie okręgu lub tłumaczenie kodów klawiatury. Funkcje sterownika zawierają także specjalne polecenia potrzebne do wykonywanie operacji specyficznych dla urządzenia sprzętowego.

Windows wymaga sterowników urządzeń dla karty graficznej, klawiatury i portów komunikacyjnych. Inne sterowniki mogą być wymagane jeżeli użytkownik doda opcjonalne urządzenia do systemu.

Środowisko Plug & Play

W środowisku Plug & Play istnieją trzy składniki: sterownik konfiguracji, moduły wyliczające i moduły przydzielające zasoby. Kiedy uruchamiany jest system, sterownik konfiguracji uruchamia moduły wyliczające, które tworzą listę zasobów sprzętowych i wypełniają ja informacjami z rejestru systemu. Następnie sterownik konfiguracji używa modułów przydzielających zasoby, w celu ustalenia prawidłowej konfiguracji systemu.
Każde urządzenie posiada unikalny identyfikator - ciąg znaków utworzony przez moduł wyliczający. Identyfikator składa się z dwóch lub trzech części, oddzielonych znakiem '\'. Pierwsza część jest nazwą modułu wyliczającego, który wykrył urządzenie, druga część jest właściwym identyfikatorem urządzenia, trzecia służy do rozróżnienia dwóch urządzeń tej samej klasy (np. portów szeregowych).

Moduły wyliczające szukają informacji o wirtualnym urządzeniu, które obsługuje urządzenie fizyczne w rejestrze w kluczu `HKLM\Enum\[id urządzenia]`. Wartość klucza o nazwie `Driver` określa klasę i nazwę urządzenia wirtualnego.

System przechowuje informacje o wszystkich zainstalowanych urządzeniach wirtualnych w rejestrze w kluczu `HKLM\System\CurrentControlSet\Services\Class\[klasa]\[nazwa]`. Wartość klucza o nazwie `StaticVxd` określa nazwę pliku zawierającego program obsługi urządzenia wirtualnego, który zostaje wczytany przez moduł wyliczający. Jeżeli nie ma wartości o nazwie `StaticVxd`, wartość `DevLoader` mówi, jaki program dynamicznie wczytuje urządzenie wirtualne. Wartość `Driver` zawiera nazwę pliku, który zostanie wczytany, gdy urządzenie wirtualne będzie potrzebne do pracy systemu (np. karta sieciowa nie jest używana cały czas). W niektórych przypadkach nazwa dynamicznego urządzenia wirtualnego przechowywana jest pod inną wartością, zależy to od programu wczytującego.

Informacje o urządzeniach niezgodnych ze standardem PnP przechowywane są w kluczu `HKLM\Enum\Root\[id urządzenia]`.

Nazwy statycznych urządzeń wirtualnych wczytywanych przy starcie systemu przechowywane są w kluczu `HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\VxD`.

Przykład

Moduł wyliczający sprawdzający magistrali PCI po przydzieleniu karcie identyfikatora `VEN_5333&DEV_8811\BUS_00&DEV_05&FUNC_00` znajduje w rejestrze wpisy:

```
HKEY_LOCAL_MACHINE\Enum\PCI\VEN_5333&DEV_8811\BUS_00&DEV_05&FUNC_00
CompatibleIDs="PCI\CC_030000,PCI\CC_0300"
DeviceDesc="S3 Trio32/64 PCI"
Driver="Display\0002"

HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\Class\Display\0002
  DevLoader="*vdd"
  InfPath="MICROS~2.INF"
  InfSection= "S3"
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\Class\Display\0002\DEFAULT
  CHIPID = 00 11 88 00 00 00
  minivdd="s3.vxd"
  drv="s3.drv"
```

Urządzenie wirtualne obsługujące kartę S3 Trio64 ładowane jest dynamicznie przez program `vdd.vxd`. Program ten szuka nazwy pliku, który ma wczytać w podkluczu `DEFAULT` w wartości o nazwie `minivdd`. Sterownikiem karty S3 jest biblioteka `s3.drv`.

Biblioteki DLL

W Microsoft Windows biblioteki DLL są modułami, które zawierają funkcje i dane. Biblioteka DLL jest wczytywana w czasie wykonywania aplikacji przez wywołujący ją proces. Kiedy kilka aplikacji używa tej samej biblioteki jednocześnie, każda dostaje kopię danych, ale współużytkują kod.

Interfejs programowania aplikacji Windows jest zbudowany jako zestaw bibliotek DLL. Aplikacje 16-bitowe korzystają m. in. z bibliotek Gdi.exe, User.exe i Krnl386.exe, aplikacje 32-bitowe używają funkcji z bibliotek gdi32.dll, user32.dll, kernel32.dll.

Twórcy oprogramowania mogą rozszerzać możliwości systemu przez tworzenie bibliotek DLL, które zawierają nowe funkcje ulepszające system i następnie przez udostępnienie tych bibliotek aplikacjom Windows. Biblioteki DLL najczęściej posiadają rozszerzenie .DLL, chociaż mogą mieć także rozszerzenie .EXE lub inne.

16- i 32-bitowe aplikacje Windows

Windows pozwala uruchamiać 16-owe aplikacje napisane dla Windows w wersjach 3.x jak również aplikacje 32-bitowe, które używają funkcji Win32 lub Win32s. Dla aplikacji 16-bitowych Windows posiada tryb wielozadaniowości bez wywłaszczenia używany w Windows 3.x. Wszystkie aplikacje 16-bitowe wykonywane są jako jeden wątek. Inaczej jest w aplikacjach 32-bitowych: każda posiada jeden lub więcej wątków, każdy 32-bitowy wątek podlega mechanizmowi wielozadaniowości z wywłaszczeniem.

Procesy i wątki.

Aplikacja systemu Windows składa się z jednego lub kilku procesów. Procesem nazywamy wykonujący się program. Pojedynczy proces uruchamia jeden lub więcej wątków. Wątek jest podstawową jednostką, do której system operacyjny przydziela czas procesora. Każdy proces jest uruchamiany z pojedynczym wątkiem, nazywanym wątkiem głównym, ale może utworzyć dodatkowe wątki. Każdy wątek ma przydzielony priorytet wykonywania. Priorytet jest liczbą z zakresu od 0 (najniższy) do 31 (najwyższy).

Okna

Każda aplikacja graficzna systemu Windows tworzy przynajmniej jedno okno nazywane oknem głównym. Okno to służy do komunikacji między użytkownikiem a aplikacją. Większość aplikacji tworzy również inne okna, które wykorzystywane są przez okno główne.

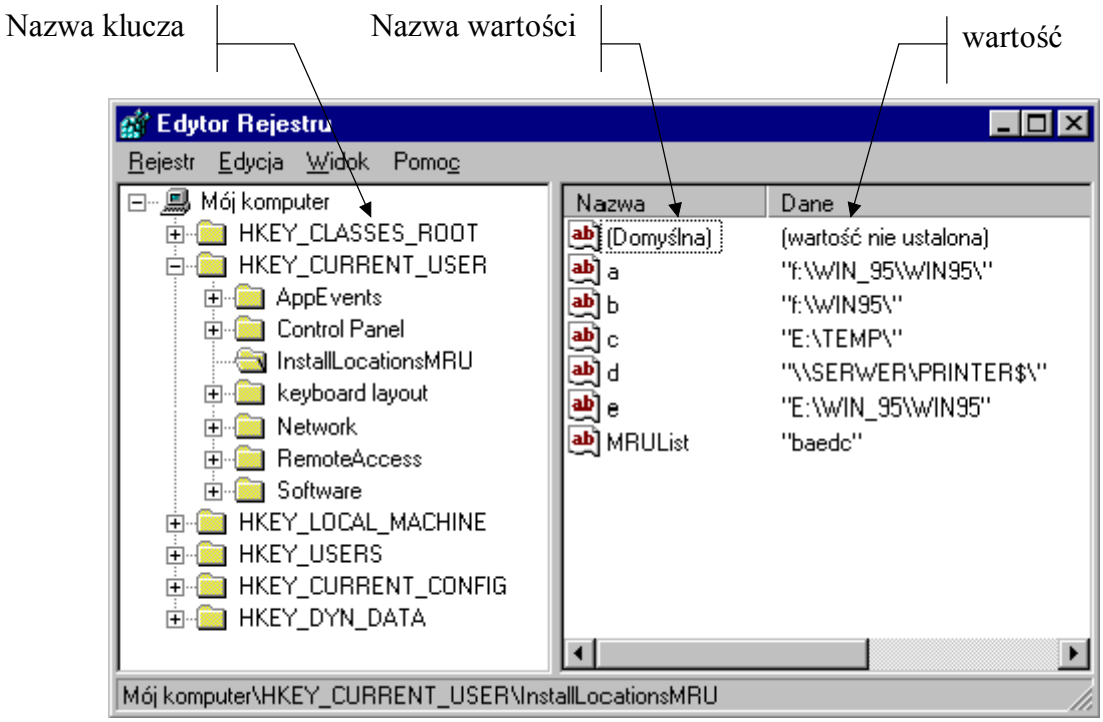
Każde okno należy do pewnej klasy okien. Klasa okna określa wygląd okna i jego zachowanie. Głównych składnikiem klasy okna jest funkcja okienkowa, która otrzymuje i przetwarza wszystkie informacje dostarczane przez system do okna. System Windows komunikuje się z oknami za pomocą meldunków. Zmiana położenia kursora myszy, kliknięcie myszą, wciśnięcie klawisza powodują wysłanie przez system meldunku do aktywnego okna.

Okno może mieć nazwę. Nazwa jest identyfikatorem okna dla użytkownika. Okna główne i dialogowe wyświetlają nazwę na listwie tytułowej. W przypadku okien kontrolnych (przyciski, pola edycyjne, listy) widoczność nazwy okna zależy od jego klasy, np. przycisk (klasa *button*) wyświetla nazwę okna w zajmowanym obszarze.

Programy narzędziowe używane w ćwiczeniu.

Edytor rejestru - regedit.exe

Edytor rejestru jest aplikacją pozwalającą modyfikować rejestr systemu Windows.

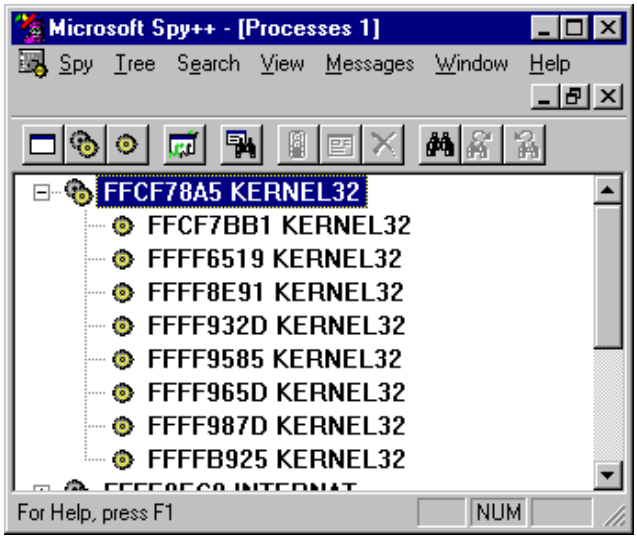


Ilustracja 1 Edytor rejestru

Spy++

Spy++ jest programem użytkowym, który przedstawia w graficzny sposób procesy, wątki, okna i meldunki.

Ilustracja 2 Spy++



Ilustracja 3 Proces Kernel32 i jego wątki

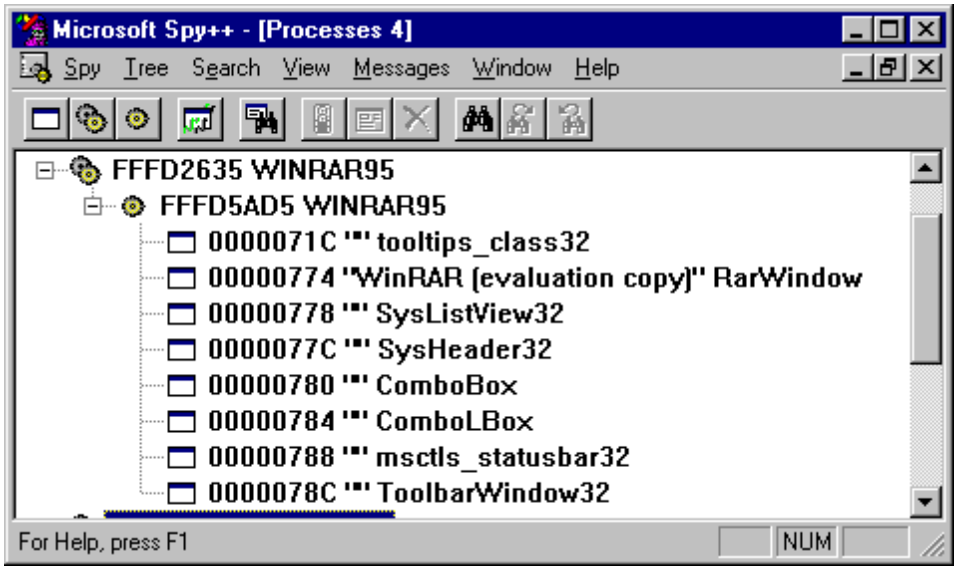
Aby otrzymać listę aktywnych procesów należy kliknąć przycisk  lub z menu **Spy** wybrać polecenie **Processes**.

Listę wątków, które utworzył dany proces można otrzymać zaznaczając proces i wybierając polecenie **Expand One Level** z menu **Tree**.

Informacje o priorytecie wątku otrzymujemy zaznaczając go i wybierając polecenie **Properties** z menu **View**.

W celu otrzymania listy okien otworzonych przez wątek należy zaznaczyć wątek i z menu **Tree** wybrać polecenie **Expand One Level**. Spy++ podaje następujące informacje o otwartych oknach: uchwyt okna, tytuł okna (jeżeli okno posiada tytuł) i klasę okna.

Aby śledzić meldunki wysyłane przez system do okna należy zaznaczyć okno, z menu **Spy** wybrać polecenie **Messages** lub wcisnąć przycisk  i w wyświetlonym oknie dialogowym wybrać Ok.



Ilustracja 4 Okna programu WinRAR95

Programy DOSa

Programy te mają na celu pokazanie pewnych właściwości urządzeń wirtualnych.

`Floppy.exe` Program odczytuje za pośrednictwem funkcji BIOSu dyskietkę ze stacji `a:` Uruchomiony z parametrem `nobios` (`floppy nobios`) włącza silnik stacji dyskietek bezpośrednio programując kontroler. Kontroler programowany jest wtedy w niestandardowy sposób, niezrozumiały dla urządzeń wirtualnych systemu Windows.

`Reset.exe` Program uruchomiony w systemie DOS resetuje komputer. Programowany jest sterownik klawiatury, który generuje sygnał sprzętowego resetu.

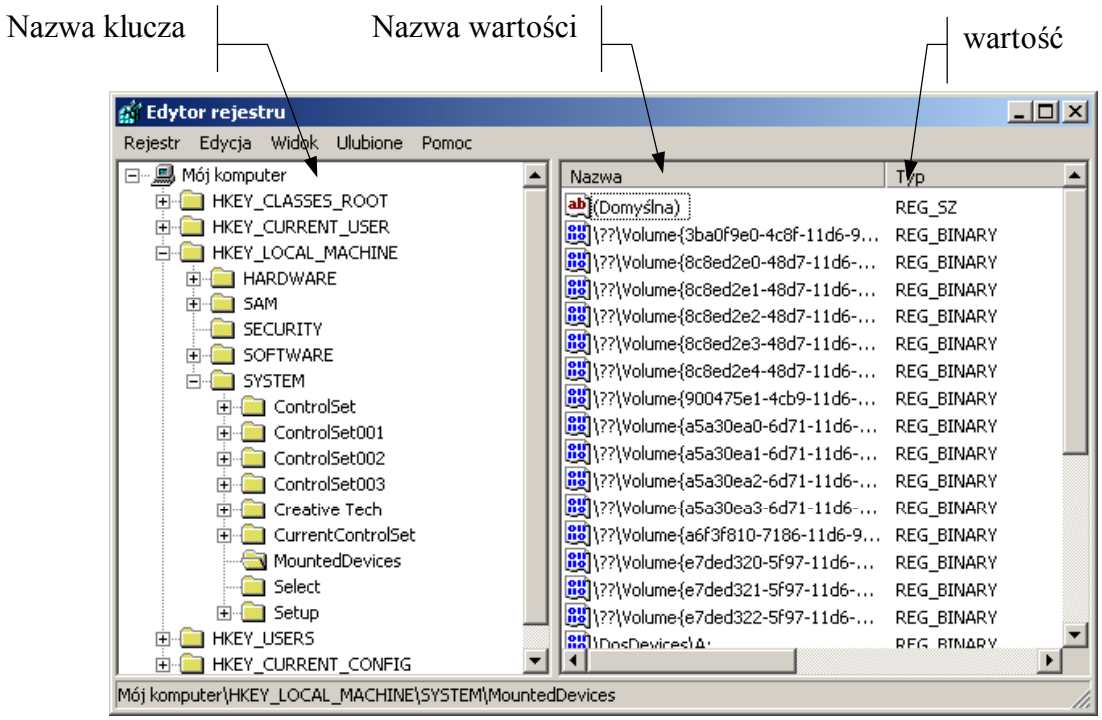
`Cmos.exe` Program wyświetla informację o ilości pamięci komputera zapisaną przez BIOS w pamięci CMOS (zwykle komputer ma więcej pamięci, niż BIOS może zapisać w pamięci CMOS). Z parametrem `zero` (`cmos zero`) program zeruje bajty przechowujące tę informację.

Programy Windows 16-bitowe

Programy `winfile.exe`, `winmine.exe`, `charmap.exe` używane są do pokazania pewnych zachowań systemu podczas pracy z 16-bitowymi aplikacjami Windows.

Programy narzędziowe używane w ćwiczeniu.

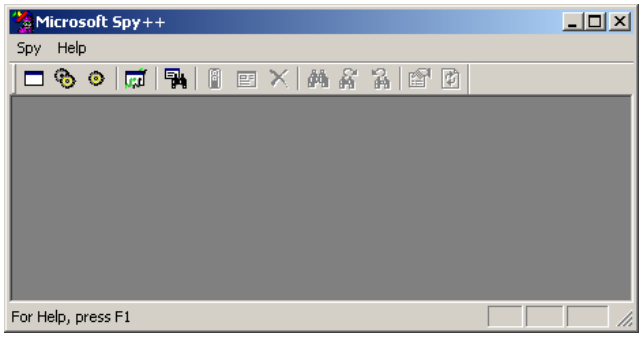
Edytor rejestru - regedit.exe
Edytor rejestru jest aplikacją pozwalającą modyfikować rejestr systemu Windows.



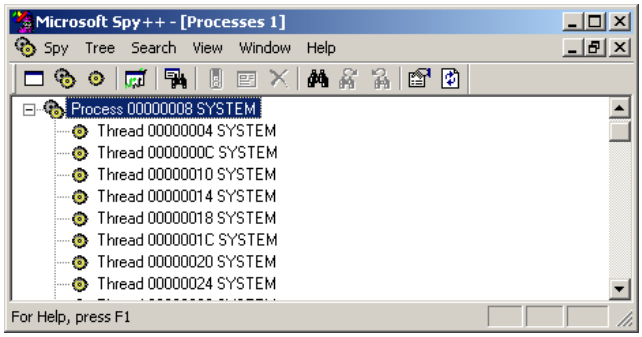
Ilustracja 5 Edytor rejestru

Spy++

Spy++ jest programem użytkowym, który przedstawia w graficzny sposób procesy, wątki, okna i meldunki.



Ilustracja 6 Spy++



Ilustracja 7 Proces System i jego wątki

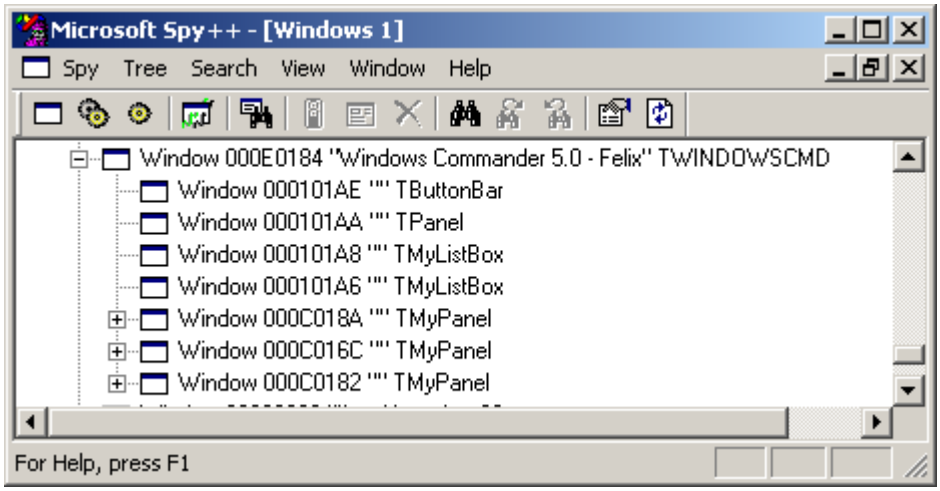
Aby otrzymać listę aktywnych procesów należy kliknąć przycisk  lub z menu **Spy** wybrać polecenie **Processes**.

Listę wątków, które utworzył dany proces można otrzymać zaznaczając proces i wybierając polecenie **Expand One Level** z menu **Tree**.

Informacje o priorytecie wątku otrzymujemy zaznaczając go i wybierając polecenie **Properties** z menu **View**.

W celu otrzymania listy okien otworzonych przez wątek należy zaznaczyć wątek i z menu **Tree** wybrać polecenie **Expand One Level**. Spy++ podaje następujące informacje o otwartych oknach: uchwyt okna, tytuł okna (jeżeli okno posiada tytuł) i klasę okna.

Aby śledzić meldunki wysyłane przez system do okna należy zaznaczyć okno, z menu **Spy** wybrać polecenie **Messages** lub wcisnąć przycisk  i w wyświetlonym oknie dialogowym wybrać Ok.



Ilustracja 8 Okna programu WinRAR95

Programy DOSa

Programy te mają na celu pokazanie pewnych właściwości urządzeń wirtualnych.

`Floppy.exe` Program odczytuje za pośrednictwem funkcji BIOSu dyskietkę ze stacji `a:` Uruchomiony z parametrem `nobios` (`floppy nobios`) włącza silnik stacji dyskietek bezpośrednio programując kontroler. Kontroler programowany jest wtedy w niestandardowy sposób, niezrozumiały dla urządzeń wirtualnych systemu Windows.

`Reset.exe` Program uruchomiony w systemie DOS resetuje komputer. Programowany jest sterownik klawiatury, który generuje sygnał sprzętowego resetu.

`Cmos.exe` Program wyświetla informację o ilości pamięci komputera zapisaną przez BIOS w pamięci CMOS (zwykle komputer ma więcej pamięci, niż BIOS może zapisać w pamięci CMOS). Z parametrem `zero` (`cmos zero`) program zeruje bajty przechowujące tę informację.

Programy Windows 16-bitowe

Programy `winfile.exe`, `winmine.exe`, `charmap.exe` używane są do pokazania pewnych zachowań systemu podczas pracy z 16-bitowymi aplikacjami Windows.

Przebieg ćwiczenia w systemie Windows NT (2000, XP).

1. ***Urządzenia wirtualne.***

a) **Współdzielenie zasobów**

Uruchomić dwie konsole poleceń, 16-bitową aplikację Windows: Managera Plików Windows 3.x (`winfile.exe`) i Eksploratora Windows. Na pierwszej konsoli uruchomić program `floppy.exe`. W trakcie pracy programu wyświetlić zawartość dyskietki w Managerze Plików, Eksploratorze Windows i na drugiej konsoli (poleceniem `'dir a:'`). Zakończyć pracę programu `floppy.exe` i ponownie wyświetlić zawartość dyskietki.

W sprawozdaniu: Czy możliwe jest współdzielenie urządzeń fizycznych, gdy wszystkie aplikacje odwołujące się do nich korzystają z funkcji systemu?

b) **Współdzielenie zasobów z aplikacją bezpośrednio programującą sprzęt**

Na pierwszej konsoli uruchomić program `floppy.exe` z parametrem *nobios* (`floppy nobios`). W trakcie pracy programu wyświetlić zawartość dyskietki w Managerze Plików, Eksploratorze Windows i na drugiej konsoli (poleceniem `'dir a:'`). Zakończyć pracę programu `floppy.exe` i ponownie wyświetlić zawartość dyskietki. Zamknąć konsolę, na której pracował program `floppy.exe`. Wyświetlić zawartość dyskietki w Managerze Plików, Eksploratorze Windows i na konsoli poleceń. Powtórzyć punkt ***b***, ale program `floppy.exe` uruchomić w trybie pełnego ekranu (może pracować w tle).

W sprawozdaniu: Czy możliwe jest współdzielenie urządzeń fizycznych, gdy jedna z aplikacji programuje urządzenie fizyczne w sposób niezrozumiały dla urządzenia wirtualnego? Jak system traktuje aplikacje bezpośrednio programujące sprzęt?

c) **Współdzielenie zasobów z aplikacjami bezpośrednio programującymi sprzęt**

Uruchomić drugą konsolę poleceń. Na obydwu konsolach uruchomić program `floppy.exe` z parametrem *nobios* (`floppy nobios`). Powtórzyć punkt ***c***, ale drugi program `floppy.exe` uruchomić w trybie pełnego ekranu. Zakończyć pracę programu `floppy.exe` i pozamykać wszystkie programy używane w punkcie 1.

W sprawozdaniu: Wyjaśnić zachowanie systemu.

d) **Zmiana stanu komputera**

Uruchomić program `reset.exe` na konsoli oraz w trybie pełnego ekranu.

W sprawozdaniu: Wyjaśnić zachowanie systemu.

e) **Dostęp do pamięci CMOS**

Na konsoli uruchomić program `cmos.exe`. Uruchomić go powtórnie z opcją `zero` (`cmos zero`). Uruchomić drugą konsolę. Na obydwu konsolach uruchomić program `cmos.exe`.

W sprawozdaniu: Czy wirtualne urządzenie pamięci CMOS pracuje poprawnie? Czy modyfikacje pamięci CMOS przez proces DOSa mają wpływ na inne procesy?

2. *Funkcje urządzeń wirtualnych i głównych bibliotek DLL.*

a) Określić nazwy modułów wyliczających zainstalowanych w systemie. Podać nazwy modułów wyliczających, które wykryły kartę sieciową i kartę graficzną.
Podać nazwę urządzenia wirtualnego i sterownika obsługującego kartę sieciową i graficzną.
Jeżeli dane urządzenie wirtualne jest dynamiczne, podać nazwę programu, który je wczytuje.

b) Uruchomić aplikacje `charmap.exe` i `charmap16.exe`. Uruchomić aplikację *Process Viewer* z grupy programów Visual Studio. W oknie *processes* zaznaczyć proces `charmap` i wyświetlic informację o szczegółach pamięci. W liście rozwijanej *User address space for* sprawdzić, które biblioteki `user`, `kernel`, `gdi` używane są przez proces: 16-bitowe czy 32-bitowe. Powtórzyć ćwiczenie dla procesu `charmap16` (jest uruchomiony jako wątek procesu `ntvdm`).

W sprawozdaniu: Czy procesy 16- i 32-bitowe używaja różnych bibliotek systemowych?

3. *Procesy, wątki, okna.*

Uruchomić aplikacje `Notepad.exe` i `Charmap16.exe` z katalogu `c:\windows`. Z pakietu Visual Studio uruchomić aplikację `Spy++`. Korzystając z programu `Spy++` określić liczbę uruchomionych procesów i dla każdego z nich podać liczbę wątków. Dla każdego wątku podać jego priorytet.
Przy pomocy programu `Spy++` określić okna wchodzące w skład aplikacji `Notepad.exe` i `Charmap16.exe`. Podać klasę okna i jego tytuł (jeżeli posiada).
Używając programu `Spy++` podać jakie meldunki otrzymują okna aplikacji `Notepad.exe` i `Charmap16.exe` przy przesuwaniu myszy i przy kliknięciu myszą. W aplikacji `Notepad.exe` śledzić meldunki przesyłane do okna klasy `Edit`, w aplikacji `Charmap16.exe` do okna głównego. Określić, czy są różnice w przysyłanych meldunkach do aplikacji 16-bitowych (`Charmap16.exe`) i 32-bitowych (`Notepad.exe`).

Przebieg ćwiczenia w systemie Windows 95 (98,Me).

1. ***Maszyny wirtualne i urządzenia wirtualne.***

a) **Maszyny wirtualne**

Uruchomić Monitor systemu (`sysmon.exe`). Dodać element „komputery wirtualne" z kategorii „kernel". Uruchamiając lub zamykając kolejne aplikacje obserwować liczbę maszyn wirtualnych. Uruchomić Managera Plików Windows 3.x (`winfile.exe`), Eksploratora Windows i trzy konsole MS-DOS. Uruchomić Notatnik (`notepad.exe`) i grę Saper (`winmine.exe`). Zamknąć jedną konsolę MS-DOS. Włożyć dyskietkę do stacji dysków.

W sprawozdaniu: Jakie reguły stosuje VMM przy tworzeniu maszyn wirtualnych?

b) **Współdzielenie zasobów**

Na pierwszej konsoli MS-DOS uruchomić program `floppy.exe`. W trakcie pracy programu wyświetlić zawartość dyskietki w Managerze Plików, Eksploratorze Windows i na drugiej konsoli MS-DOS. Zakończyć pracę programu `floppy.exe` i ponownie wyświetlić zawartość dyskietki.

W sprawozdaniu: Czy możliwe jest współdzielenie urządzeń fizycznych, gdy wszystkie aplikacje odwołujące się do nich korzystają z funkcji systemu?

c) **Współdzielenie zasobów z aplikacją bezpośrednio programującą sprzęt**

Na jednej konsoli MS DOS uruchomić program `floppy` z parametrem *nobios*. W trakcie pracy programu wyświetlić zawartość dyskietki w Managerze Plików, Eksploratorze Windows i na drugiej konsoli MS-DOS. Zakończyć pracę programu `floppy` i ponownie wyświetlić zawartość dyskietki. Zamknąć sesję DOSa, w której pracował program `floppy`. Wyświetlić zawartość dyskietki w Managerze Plików, Eksploratorze Windows i na konsoli MS-DOS. Powtórzyć punkt ***c***, ale program `floppy` uruchomić w trybie pełnego ekranu (może pracować w tle).

W sprawozdaniu: Czy możliwe jest współdzielenie urządzeń fizycznych, gdy jedna z aplikacji programuje urządzenie fizyczne w sposób niezrozumiały dla urządzenia wirtualnego?

d) **Współdzielenie zasobów między aplikacjami bezpośrednio programującymi sprzęt**

Uruchomić drugą konsolę MS-DOS. Na obydwu konsolach MS-DOS uruchomić program `floppy.exe` z parametrem *nobios*. Powtórzyć punkt ***d***, ale drugi program `floppy.exe` uruchomić w trybie pełnego ekranu. Zakończyć pracę programu `floppy.exe` i pozamykać wszystkie programy używane w punkcie 1.

W sprawozdaniu: Czy możliwe jest współdzielenie urządzeń fizycznych między kilkoma aplikacjami programującymi urządzenie fizyczne w sposób niezrozumiały dla urządzenia wirtualnego?

e) **Zmiana stanu komputera**

Uruchomić program `reset.exe` w okienku MS-DOS oraz w trybie pełnego ekranu.

W sprawozdaniu: Wyjaśnić zachowanie systemu.

f) **Dostęp do pamięci CMOS**

W okienku MS-DOS uruchomić program `cmos.exe`. Uruchomić go powtórnie z opcją `zero`. Uruchomić drugią sesję DOSa. Na obydwu konsolach uruchomić program `cmos.exe`.

W sprawozdaniu: Czy wirtualne urządzenie pamięci CMOS pracuje poprawnie?

2. ***Funkcje urządzeń wirtualnych i głównych bibliotek DLL.***

a) Określić nazwy modułów wyliczających zainstalowanych w systemie. Podać nazwy modułów wyliczających, które wykryły kartę sieciową i kartę graficzną.
Podać nazwę urządzenia wirtualnego i sterownika obsługującego kartę sieciową i graficzną.
Jeżeli dane urządzenie wirtualne jest dynamiczne, podać nazwę programu, który je wczytuje.

b) Odszukać w katalogu `c:\windows` aplikacje `Winfile.exe` i `Notepad.exe`. Kliknąć prawym przyciskiem myszy i z menu podręcznego wybrać Szybki podgląd. Określić jakie biblioteki DLL używane są przez te aplikacje (w sekcji Import Table).

W sprawozdaniu: Czy procesy 16- i 32-bitowe używaja różnych bibliotek systemowych?

3. ***Procesy, wątki, okna.***

Uruchomić aplikacje `Notepad.exe` i `Charmap.exe` z katalogu `c:\windows`. Z pakietu Visual Studio uruchomić aplikację `Spy++`. Korzystając z programu `Spy++` określić liczbę uruchomionych procesów i dla każdego z nich podać liczbę wątków. Dla każdego wątku podać jego priorytet.
Przy pomocy programu `Spy++` określić okna wchodzące w skład aplikacji `Notepad.exe` i `Charmap.exe`. Podać klasę okna i jego tytuł (jeżeli posiada).
Używając programu `Spy++` podać jakie meldunki otrzymują okna aplikacji `Notepad.exe` i `Charmap.exe` przy przesuwaniu myszy i przy kliknięciu myszą. W aplikacji `Notepad.exe` śledzić meldunki przesyłane do okna klasy `Edit`, w aplikacji `Charmap.exe` do okna głównego. Określić, czy są różnice w przysyłanych meldunkach do aplikacji 16-bitowych (`Charmap.exe`) i 32-bitowych (`Notepad.exe`).

Pytania na wejściówkę

1. Wyjaśnić zadania sterownika maszyny wirtualnej, urządzeń wirtualnych, sterowników urządzeń.
2. Opisać krótko proces przydzielania zasobów w systemach PnP.
3. Podać różnice między statycznym a dynamicznym urządzeniem.
4. Jakie są korzyści z używania bibliotek dynamicznych.
5. Jak wygląda wielozadaniowość aplikacji 16-bitowych i 32-bitowych w Windows.
6. Do czego służy klasa okna i funkcja okienkowa.

Literatura

1. Microsoft Windows Resource Kit: Windows Reference: Windows Architecture,
2. Inside Windows 2000, Third Edition

Systemy operacyjne

Laboratorium

Ćwiczenie 2

Zarządzanie pamięcią w Windows

Pamięć procesu

Każdy proces w Windows otrzymuje własną przestrzeń adresową o rozmiarze 4 GB. Pamięć o adresach 0 - 2 GB jest dostępna dla procesu, pamięć powyżej 2 GB jest zarezerwowana dla systemu. Adres wirtualny zamieniany jest przez system na adres fizyczny przy użyciu mechanizmu stronicowania. Przestrzeń adresowa procesu podzielona jest na strony o wielkości 4 KB. Każda strona może być w jednym z trzech stanów:

wolna	strona nie jest dostępna,
zarezerwowana	strona zarezerwowana nie jest dostępna i nie jest jej przydzielona fizyczna pamięć, proces może rezerwować strony w celu późniejszego użycia
przydzielona	stronie została przydzielona pamięć fizyczna (w RAM lub na dysku), taka strona może mieć określone prawa, np. tylko odczyt, wykonywanie programu, zapis

Organizacja stron

Każda strona posiada deskryptor strony, który określa jej położenie w pamięci fizycznej oraz prawa dostępu. Deskryptory stron są grupowane przez system w struktury nazywane tablicami stron. Tworząc wiele tablic stron system tworzy osobne przestrzenie adresowe. Adresy tablic stron przechowywane są w strukturze nazywanej katalogiem stron. Każda maszyna wirtualna posiada odrębny katalog stron.

Adres w wirtualnej przestrzenie adresowej procesu składa się z następujących części:

bity adresu	znaczenie
31-22	numer tablicy stron w katalogu stron
21-12	numer deskryptora strony w tablicy stron
11-0	pozycja bajtu na stronie

Ze względu na mechanizmy ochrony aplikacja nie może zmieniać położenia katalogu stron i tablicy stron. Dostęp do tych struktur mają urządzenia wirtualne.

Biblioteka Microsoft Foundation Class

Klasy biblioteki MFC tworzą szkielet aplikacji Windows. Klasy te opisują także wszystkie składniki systemu Windows, takie jak elementy kontrolne (przyciski, napisy, okna edycyjne), standardowe okna dialogowe (Otwórz, Zapisz jako).

Procesy i wątki

Aplikacja systemu Windows składa się z jednego lub więcej procesów. Proces może uruchomić jeden lub więcej wątków. Procesowi odpowiada klasa CWinApp, która jest klasą bazową dla aplikacji używającej MFC. Aplikacja może posiadać tylko jeden obiekt klasy CWinApp. Wątkowi odpowiada klasa CWinThread. Ponieważ klasa ta jest klasą bazową klasy CWinApp, podczas tworzenia nowego procesu automatycznie tworzony jest jego główny wątek.

Okna, plansze dialogowe, elementy kontrolne

CWnd jest podstawową klasą opisującą okno systemu Windows. Plansze dialogowe opisywane są przez klasę CDialog. Każdy element kontrolny posiada odpowiadającą mu klasę, np. dla przycisku jest to klasa CButton, dla okienka edycyjnego CEdit. Oknem głównym aplikacji może być dowolne okno, którego klasą bazową jest CWnd, np. plansza dialogowa.

Zasoby

Zasoby to dane które dołączane są do wykonywalnego pliku. Standardowe zasoby to plansze dialogowe, ikony, kursory, bitmapy, fonty.

Elementy kontrolne ActiveX

Elementy ActiveX ułatwiają tworzenie aplikacji. Podobnie jak klasa w języku C++ charakteryzuje się zmiennymi i funkcjami składowymi, elementy ActiveX udostępniają trzy rodzaje obiektów:

- metody - są poleceniami, które obiekt może wykonać. Przykładowo metoda Move przesuwa element kontrolny w określone miejsce planszy dialogowej,
- właściwości - są funkcjami określającymi stan obiektu, np. właściwość Visible określa czy element jest widoczny
- zdarzenia - to akcje rozpoznawane przez obiekt, takie jak kliknięcie myszą lub wciśnięcie klawisza.

Klasą MFC opisującą obiekt ActiveX jest COleDispatchDriver lub CWnd, jeżeli obiekt ActiveX jest umieszczony na planszy dialogowej. Class Wizard automatycznie tworzy klasę dla dodawanego elementu ActiveX.

Funkcje wykorzystane w ćwiczeniu

```
BOOL VirtualProtectEx(HANDLE hProcess, void *lpAddress, DWORD dwSize,
                      DWORD flNewProtect, PDWORD lpflOldProtect);
```

Funkcja zmienia prawa dostępu przydzielonych stron określonego procesu

parametr	opis
hProcess	identyfikuje proces, którego przestrzeń adresowa będzie modyfikowana
lpAddress	Adres pierwszej strony, która będzie mieć zmienione prawa dostępu. Adres musi być wielokrotnością rozmiaru strony
dwSize	rozmiar w bajtach regionu modyfikowanych stron
`flNewProtect`	nowe prawa dostępu do stron, jedna ze stałych: `PAGE_READONLY`, `PAGE_READWRITE`, `PAGE_WRITECOPY`, `PAGE_EXECUTE`, `PAGE_EXECUTE_READ`, `PAGE_EXECUTE_READWRITE`, `PAGE_EXECUTE_WRITECOPY`, `PAGE_GUARD`, `PAGE_NOACCESS`, `PAGE_NOCACHE`
`lpflOldProtect`	adres zmiennej typu unsigned long, w której funkcja umieści prawa dostępu strony sprzed modyfikacji.

```
HANDLE OpenProcess(DWORD dwDesiredAccess, BOOL bInheritHandle,
                   DWORD dwProcessId);
```

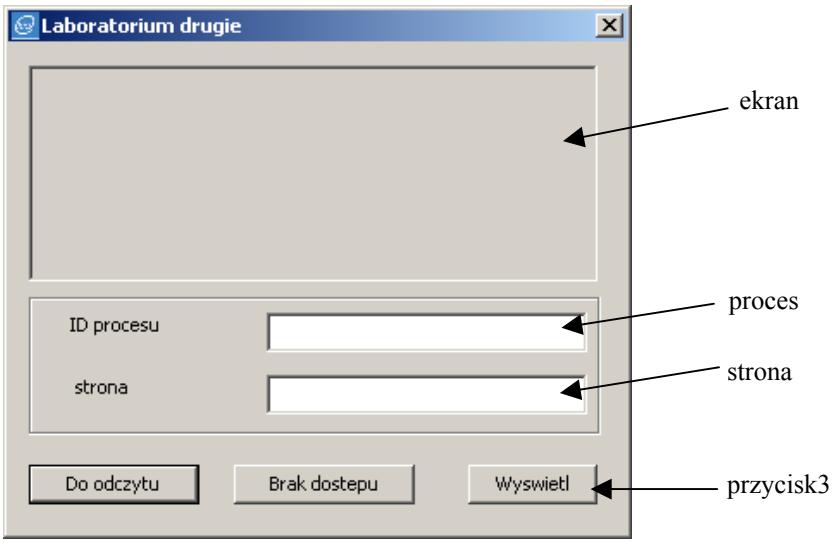
Funkcja zwraca uchwyt procesu. Uchwyt jest potrzebny m. in. funkcjom modyfikującym przestrzeń adresową procesu.

parametr	opis
`dwDesiredAccess`	Określa prawa dostępu do procesu. Stała `PROCESS_ALL_ACCESS` pozwala na wykonywanie dowolnych operacji na procesie
`bInheritHandle`	Określa, czy zwrócony uchwyt procesu będzie przekazywany procesom potomnym
`dwProcessId`	identyfikator procesu, można go uzyskać wykorzystując program Spy++

Przebieg ćwiczenia

Celem ćwiczenia jest napisanie programu, który przegląda i modyfikuje strony w przestrzeni adresowej innego procesu. Procesem, który będzie używany w ćwiczeniu jest aplikacja `tester-nt.exe`. Program ten należy uruchomić przed rozpoczęciem ćwiczenia (program ten znajduje się w katalogu ćwiczenia).
`Tester.exe` informuje o adresie bajtu, który jest modyfikowany i stronie, na której leży ten bajt. Należy przeglądnąć całą przestrzeń adresową procesu `tester.exe` oraz zmienić prawa strony na której leży modyfikowany bajt na "tylko do odczytu", a następnie na "brak dostępu". Okno główne programu, który wykona te funkcje wygląda jak na Ilustracji 1.

Po wciśnięciu przycisku "Do odczytu" program powinien uzyskać dostęp do przestrzeni adresowej procesu o identyfikatorze z pola "ID procesu" i nadać stronie wpisanej w polu "strona" prawa "tylko do odczytu". Podobnie, po wciśnięciu przycisku "Brak dostępu" prawa wskazanej strony powinny zostać zmienione na "brak dostępu".
Po wciśnięciu przycisku ”Wyślietl” okno tekstowe powinno pokazać zawartość wirtualnej przestrzeni adresowej procesu o podanym identyfikatorze.



Ilustracja 1 Okno główne programu

Identyfikatory elementów w oknie głównym

Zostały utworzone następujące identyfikatory i zmienne.

Identyfikator	nazwa zmiennej
IDC_EKRAN	m_ekran
IDC_PROCES	m_proces
IDC_STRONA	m_strona
IDC_PRZYCISK1	m_przycisk1
IDC_PRZYCISK2	m_przycisk2
IDC_PRZYCISK3	m_przycisk3

Odczytywanie stanu elementu kontrolnego

W celu odczytania tekstu wpisanego w oknie edycji należy użyć funkcji składowej `GetWindowText` klasy `CEdit`. Funkcja zwraca tekst, który można zamienić na wartość liczbową za pomocą funkcji `sscanf`. Ilustracja 10 pokazuje w jaki sposób uzyskać identyfikator procesu w omawianym przykładzie.

```
DWORD  id;  //identyfikator procesu
CString s;

s = m_proces.GetWindowText();
sscanf((const char *)s,”%i”,&id);
```

Ilustracja 10. Odczytanie wartości z elementu kontrolnego

Dostęp do przestrzeni adresowej innego procesu

W celu uzyskania dostępu do wirtualnej przestrzeni adresowej procesu o znanym identyfikatorze należy użyć funkcji `OpenProcess`. Funkcja zwraca uchwyt procesu, który jest wykorzystywany przez funkcje modyfikujące pamięć wirtualna. Przykład użycia funkcji `OpenProcess` pokazuje ilustracja 11.

```
HANDLE hProces;
DWORD  id;  //identyfikator procesu

hProces = OpenProcess(PROCESS_ALL_ACCESS,0,id);
//operacje na pamięci procesu
//...
//
CloseHandle(hProces);
```

Ilustracja 11. Dostęp do przestrzeni adresowej innego procesu

Modyfikacja praw dostępu stron pamięci innego procesu

W celu ustawienia prawa dostępu strony na "tylko odczyt" należy użyć funkcji `VirtualProtectEx`. Proces, którego przestrzeń adresowa będzie modyfikowana identyfikowany jest przez uchwyt zwrócony przez funkcję `OpenProcess`. Przykład użycia funkcji `VirtualProtectEx` pokazuje ilustracja 12.

```
HANDLE hProces;
DWORD  n;
void *adres;

VirtualProtectEx(hProces,adres,4095,PAGE_READONLY,&n);
```

Ilustracja 12. Modyfikacja praw dostępu strony

Informacje o stronach pamięci innego procesu

W celu odczytania informacji o wirtualnej przestrzeni adresowej procesu należy użyć funkcji `VirtualQueryEx`. Proces, którego przestrzeń adresowa będzie przeglądana identyfikowany jest przez uchwyt zwrócony przez funkcję `OpenProcess`.
W ćwiczeniu można użyć biblioteki dynamicznej `lab2lib.dll`, która zawiera funkcję `ShowVMem`. Funkcja zwraca łańcuch tekstowy zawierający opis wirtualnej przestrzeni adresowej wskazanego procesu.
Powyższa biblioteka została dołączona do projektu. Gdyby było inaczej to należy:
W celu użycia biblioteki skopiować plik `lab2lib.h` do katalogu z plikami źródłowymi projektu, pliki `lab2lib.dll` i `lab2lib.lib` do podkatalogu `debug` (tworzony przez kompilator podczas kompilowania). Z menu *`Project`* wybrać polecenie *`Add To Project`* a następnie `Files` i dodać do projektu pliki `lab2lib.h` i `lab2lib.lib`.

Przykład użycia funkcji `ShowVMem` pokazuje ilustracja 13.

```
HANDLE hProces;
unsigned int  n;
char *s;

n = 0;
ShowVMem(hProces,0,&n);
s = (char *)malloc(n);
ShowVMem(hProces,s,&n);
m_ekran.SetWindowText(s);
free(s);
```

Ilustracja 13. Przeglądanie stron pamięci procesu

Sprawozdanie z ćwiczenia

Powinno zawierać następujące punkty:
- nazwę komputera (zakładka identyfikacja właściwości otoczenia sieciowego),
- opis strony pamięci, na której leży bajt modyfikowany przez program `tester.exe`.
- opis tej samej strony, ale po modyfikacji jej praw dostępu na "tylko odczyt" i "brak dostępu"
- komunikat (razem ze szczegółami), jaki wyświetla system, gdy `tester.exe` próbuje zapisywać stronę ze zmienionymi prawami dostępu.

Opis strony powinien zawierać następujące informacje:
- adres strony,
- adres grupy stron, które `tester.exe` alokował razem z modyfikowaną stroną,
- prawa dostępu do strony, które `tester.exe` nadał stronie podczas alokacji,
- aktualne prawa dostępu do strony,
- rozmiar regionu, który zaczyna się od adresu strony (wszystkie strony w tym regionie mają identyczne atrybuty),
- stan strony.

Pytania na wejściówkę

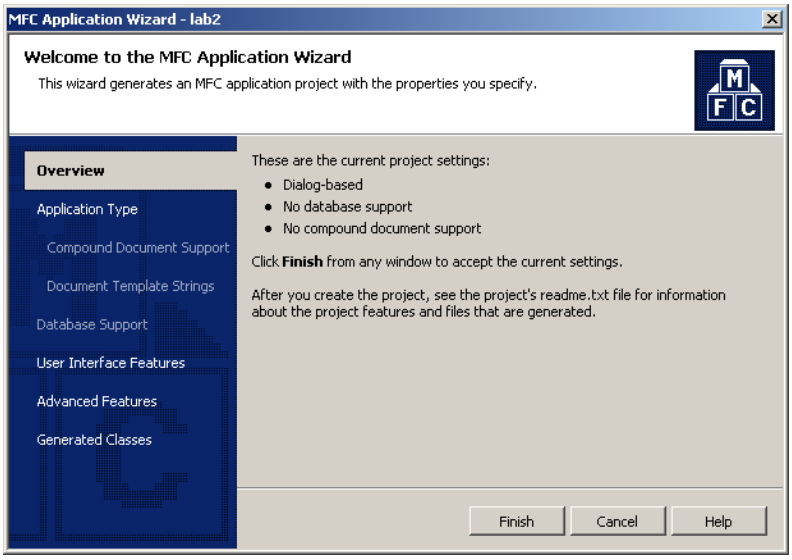
- Jaki jest rozmiar wirtualnej przestrzeni adresowej procesu i jaka jest jej organizacja.
- Omówić stany, w jakich może znajdować się strona.
- Omówić związki między maszyną wirtualną, katalogiem stron, tablicami stron, deskryptorami stron.
- Omówić sposób interpretacji przez system liniowego adresu w wirtualnej przestrzeni adresowej procesu.

Literatura

1. Richard C. Leinecker "Visual C++ 5: narzędzia programowania"
2. Steven Holzner "Visual C++ 5"
3. Al Williams "MFC czarna księga"

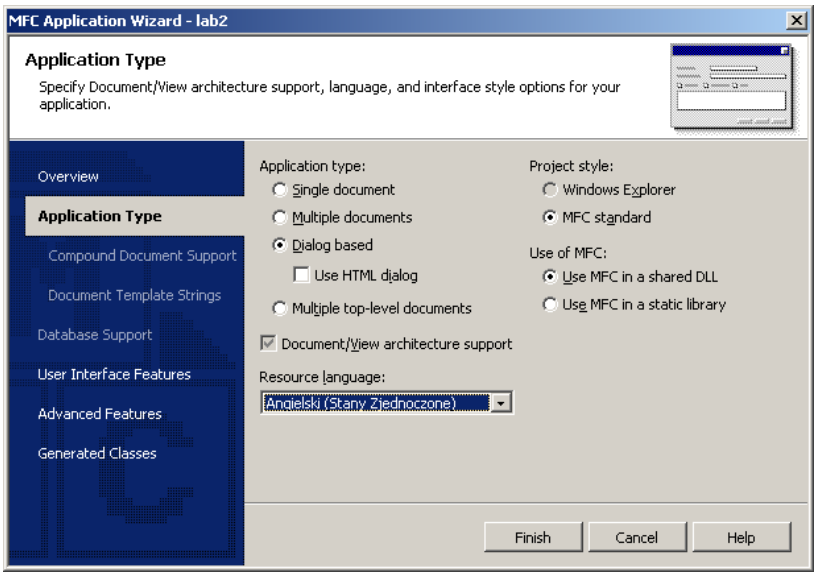
Tworzenie aplikacji graficznej przy użyciu kreatora

1. Z menu File wybrać polecenie New. W oknie dialogowym wybrać zakładkę Projects i z listy dostępnych szablonów wybrać MFC Application. Wpisać nazwę i lokalizację projektu. W prezentowanym przykładzie projekt nazywa się Lab2. Wcisnąć przycisk `Ok`.



Ilustracja 2 Tworzenie nowego projektu

2. Jako typ aplikacji wybrać Dialog based. Okno główne aplikacji będzie wtedy planszą dialogową.

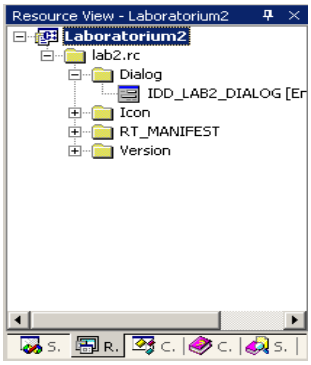


Ilustracja 3 Pierwszy krok tworzenia aplikacji przez kreatora

3. Do kolejnych kroków tworzenia aplikacji przechodzimy wybierając kolejne pola menu po lewej. W krokach 2 i 3 nie trzeba wprowadzać zmian w domyślnych ustawieniach kreatora. W kroku 4 kreator pyta się o nazwy klas aplikacji i okna głównego. W prezentowanym przykładzie klasa aplikacji nosi nazwę `Clab2App`, klasa okna głównego `Clab2Dlg`. Po wciśnięciu przycisku `Finish` i potwierdzeniu przyciskiem `Ok` informacji pokazanych przez kreatora utworzony zostanie szkielet aplikacji.

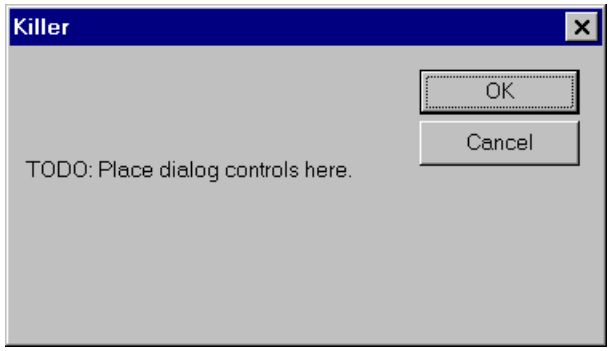
Dodawanie elementów do okna głównego

Wybrać zakładkę Resource View (Ilustracja 4). Pokazana zostanie lista zasobów, jakie utworzył kreator. Z listy zasobów wybrać typ Dialog i dwukrotnie kliknąć na identyfikatorze okna głównego. Wczytana zostanie plansza dialogowa przygotowana przez kreatora (Ilustracja 5).



Ilustracja 4 Okno z listą zasobów

Z planszy usunąć wszystkie elementy. Z menu Toolbox wybrać Edit Control i umieścić w oknie tworzonej aplikacji. Postępując identycznie wybrać przycisk - element Button, etykietę - StaticText i linię edycji – Edit Control, Ułożyć elementy w sposób podany na ilustracji 1.



Ilustracja 5 Okno główne aplikacji

Zmiana właściwości elementów okna głównego

K**Błąd! Nie można odnaleźć źródła odsyłacza.** na elemencie klasy `TextBox` Z menu Toolbox wybrać Properties. Pojawi się okno z właściwościami obiektu. Wybrać zakładkę All i ustawić wartość właściwości odpowiednich właściwości, tak jak pokazano w tabeli.

Właściwość	wartość	znaczenie
MultiLine	true	okno będzie wyświetlać wiele linii tekstu
ScrollBars	both	jeżeli tekst nie zmieści się w oknie, pokazane będą suwadła
WordWarp	false	linie dłuższe niż szerokość okna nie będą zawijane

Identyfikatory elementów w oknie głównym

Dla każdego z elementów kontrolnych wyświetlić okno właściwości i w zakładce General w polu ID wpisać identyfikatory, tak jak pokazano w tabeli.

Element	identyfikator
Edit Control	IDC_EKRAN
	IDC_PROCES
	IDC_STRONA
Button	IDC_PRZYCISK1
	IDC_PRZYCISK2
	IDC_PRZYCISK3

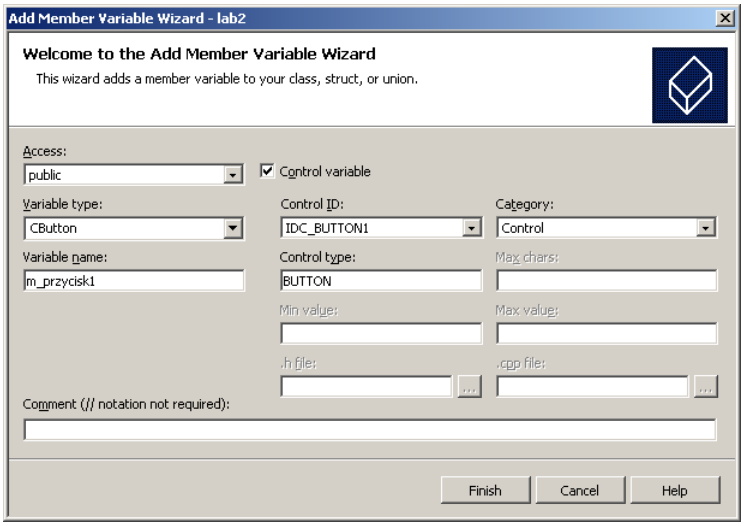
Dodawanie kodu obsługującego elementy kontrolne

K**Błąd! Nie można odnaleźć źródła odsyłacza.** na elemencie który chcemy obsłużyć Z menu Toolbox wybrać Add Variable. W oknie kreatora (Ilustracja 6) należy podać nazwę zmiennej związanej z elementem kontrolnym. Nazwę zmiennej wpisać w oknie `Variable name`. Zmienna ta będzie umieszczona w sekcji public. Każda operacja na tej zmiennej będzie miała wpływ na element kontrolny, np. instrukcja

```
m_przycisk1.SetCaption(”Ustawienia”);
```

zmieni tekst wyświetlany przez przycisk.
Wykorzystując opisaną procedurę dodać zmienne dla elementów kontrolnych, tak jak pokazano w tabeli.

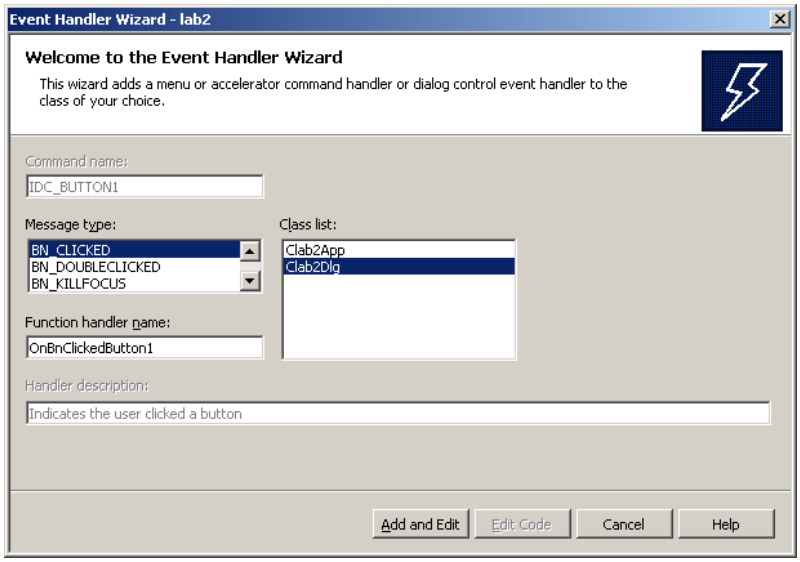
Identyfikator	nazwa zmiennej
IDC_EKRAN	m_ekran
IDC_PROCES	m_proces
IDC_STRONA	m_strona
IDC_PRZYCISK1	m_przycisk1
IDC_PRZYCISK2	m_przycisk2
IDC_PRZYCISK3	m_przycisk3



Ilustracja 6 Variable Wizard

Obsługa zdarzeń

K**Błąd! Nie można odnaleźć źródła odsyłacza.** na elemencie który chcemy obsłużyć Z menu Toolbox wybrać Add Event Handler (Ilustracja 8). W oknie Message type wybrać zdarzenie i wcisnąć przycisk Add and Edit. W okienku Function Handler Name można wpisać nazwę funkcji, która będzie wywołana, gdy zostanie wciśnięty przycisk np. "Do odczytu". Postępując identycznie utworzyć funkcję dla przycisku "Brak dostępu" i ”Wyświetl”.



Ilustracja 7 Kreator informuje o tworzeniu nowej klasy

Systemy operacyjne

Laboratorium

Ćwiczenie 3

Synchronizacja wątków
Wymiana informacji między wątkami

Wstęp

Proces w systemie Windows może uruchomić jeden lub więcej wątków. Wątki mogą być wykorzystane do odczytu lub zapisu danych w tle lub przeprowadzania długotrwałych obliczeń. W aplikacjach wielowątkowych pojawiają się problemy synchronizacji wątków, współdzielenia zasobów i wymiany informacji między pracującymi wątkami.

Wielozadaniowość w Windows

W Windows zrealizowano wielozadaniowość z wywłaszczeniem. Każdy uruchomiony wątek otrzymuje szczelinę czasową procesora. Aktualnie wykonywany wątek jest wstrzymywany, gdy upłynie przydzielony mu czas. Ponieważ każda szczelina czasowa jest krótka (około 20 milisekund), sprawia to wrażenie, że wątki wykonują się równocześnie.
Każdy proces ma określoną klasę priorytetu. Każdy wątek procesu ma określony poziom priorytetu. Klasa i poziom priorytetu są łączone przez system, tworząc priorytet podstawowy wątku. Oprócz priorytetu podstawowego wątek posiada priorytet dynamiczny. Jest to priorytet używany przez algorytm kolejkowania do określenia, który wątek powinien otrzymać czas procesora.
System tworzy kolejki wątków oczekujących na czas procesora. Każda kolejka zawiera wątki o równym priorytecie. Gdy wątek wykorzysta swoją szczelinę czasową system wykonuje następujące czynności:

- zapamiętuje stan wątku,
- umieszcza wstrzymany wątek na końcu kolejki, do której należy,
- znajduje kolejkę o najwyższym priorytecie, w której jest gotowy wątek,
- przesuwa gotowy wątek na czoło kolejki i wykonuje go.

Wątki, które nie są gotowe to:

- wątki utworzone jako wstrzymane,
- wątki zatrzymane w trakcie wykonywania
- wątki zatrzymane przez funkcje blokujące.

W pewnych przypadkach system zwiększa priorytet wątku, aby zapewnić, że zostanie mu przydzielony czas procesora:

- gdy zostanie uaktywnione okno, system zwiększa priorytet wątku, który utworzył to okno, aby był większy niż priorytet okien, które są w tle,
- gdy okno otrzyma meldunek, np. WM_TIMER, WM_MOUSEMOVE lub inny,
- gdy wątek zatrzymany przez funkcję blokującą zostaje wznowiony

Priorytet nie jest zwiększany na stałe. Za każdym razem, gdy wątek wykorzysta swoją szczelinę czasową system redukuje poziom priorytetu, aż osiągnie on początkowy poziom.

Obiekty służące do synchronizacji

Są to obiekty, które mogą być używane przez funkcje blokujące w celu synchronizacji wątków.

Zdarzenie	informuje jeden lub więcej oczekujących wątków, że wystąpiło określone zdarzenie,
Mutex	tylko jeden wątek może być właścicielem określonego obiektu typu Mutex, co pozwala na wzajemnie wykluczający się dostęp do wspólnych zasobów,
Semafor	posiada licznik przyjmujący wartości między 0 a ustaloną wartością maksymalną, ograniczając liczbę wątków, które równocześnie mogą korzystać ze wspólnych zasobów,
Sekcja krytyczna	podobnie jak Mutex pozwala na dostęp do zasobów tylko jednemu wątkowi, ale musi być używana przez wątki jednego procesu

Wszystkie obiekty, za wyjątkiem sekcji krytycznej, posiadają nazwy (tak jak pliki na dysku), co pozwala także na synchronizację między procesami. Obiekty synchronizacji mogą przyjmować dwa stany:

dostępny	wątek, który przekaże dostępny obiekt funkcji blokującej nie zostaje wstrzymany
niedostępny	wątek, który przekaże niedostępny obiekt funkcji blokującej zostaje wstrzymany, aż stan obiektu zostanie zmieniony na dostępny

Wątki, których wykonanie zostało wstrzymane przez funkcję blokującą oczekują na zmianę stanu obiektu synchronizacji na dostępny. Wstrzymany wątek wykorzystuje czas procesora w bardzo niewielkim stopniu.

Funkcje blokujące

Wątek, który chce uzyskać dostęp do wspólnego zasobu lub zaczekać na inny wątek przekazuje obiekt synchronizacji funkcji blokującej. Jeżeli obiekt jest dostępny, wątek kontynuuje działanie, w przeciwnym przypadku zostaje wstrzymany. Są dwa typy funkcji blokujących: funkcje oczekujące na zmianę stanu pojedynczego obiektu synchronizacji i funkcje oczekujące na zmianę stanu kilku obiektów synchronizacji. Nie wolno używać funkcji blokujących w wątku, który utworzył okno główne aplikacji. Aplikacja, której główny wątek został wstrzymany, nie odpowiada na meldunki wysyłane przez system. W skrajnym przypadku doprowadza to do zawieszenia Windows 95.

Komunikacja między wątkami

Wątki można podzielić na dwa typy: wątki robocze, które nie tworzą okna i wątki które utworzyły okno, służące do wizualnej komunikacji z użytkownikiem. Wszystkie wątki jednego procesu współdzielą przestrzeń adresową i mają dostęp do zmiennych globalnych procesu. Wątki mogą więc przekazywać między sobą dane za pomocą zmiennych globalnych lub wskaźników. Dodatkowo, wątki które posiadają okno mogą otrzymywać prywatne meldunki.

Klasy i funkcje wykorzystane w ćwiczeniu

Klasa CEvent

Klasa ta opisuje obiekt synchronizacji typu zdarzenie.

```
CEvent(BOOL StanPocz,BOOL ResetMan,char * Nazwa,SECURITY_ATTRIBUTES *Dostep);
```

Konstruktor, tworzy nowy obiekt. Wszystkie parametry są opcjonalne.

Parametr	opis
`StanPocz`	Jeżeli `TRUE`, pierwszy wątek, który użyje tego obiektu w celu synchronizacji nie zostanie wstrzymany. Domyślnie `FALSE`.
`ResetMan`	Jeżeli `TRUE`, obiekt jest zdarzeniem manualnym, w przeciwnym przypadku jest zdarzeniem automatycznym. Domyślnie `FALSE`. Zdarzenie manualne pozostaje w stanie określonym przez wywołanie funkcji `SetEvent` lub `C`. Zdarzenie automatyczne powraca do stanu *niedostępny*, gdy ostatni wątek, który był na nim zatrzymany, zostanie wznowiony. Domyślnie `FALSE`.
`Nazwa`	Nazwa obiektu dla systemu Windows. W przypadku synchronizacji wątków nie trzeba podawać nazwy (taka jest domyślna wartość).
`Dostep`	Prawa dostępu do zdarzenia. W Windows 9x ignorowane.

```
SetEvent();
```

Funkcja zmienia stan obiektu na *dostępny*. Jeżeli zdarzenie jest manualne, wywołanie funkcji wznowi wszystkie wątki wstrzymane przez ten obiekt. Zdarzenie automatyczne wznowi tylko jeden wątek, system następnie zmieni stan zdarzenia na *niedostępny*. Jeżeli żaden wątek nie jest wstrzymany, zdarzenie pozostaje w stanie *dostępny*.

```
PulseEvent();
```

Funkcja zmienia stan obiektu na *dostępny*, zwalnia czekające wątki, a następnie ustawia stan zdarzenia na *niedostępny*. Jeżeli zdarzenie jest manualne, wywołanie funkcji wznowi wszystkie wątki wstrzymane przez ten obiekt. Zdarzenie automatyczne wznowi tylko jeden wątek, system następnie zmieni stan zdarzenia na *niedostępny*.

```
ResetEvent( );
```

Funkcja zmienia stan obiektu na *niedostępny*. Wątki używające zdarzenia do synchronizacji zostaną wstrzymane. Funkcji nie używa się z zdarzeniach automatycznych.

Klasa CCriticalSection

Klasa ta opisuje obiekt synchronizacji typu sekcja krytyczna. Konstruktor klasy nie przyjmuje żadnych parametrów.

```
Lock();
```

Funkcja zajmuje obiekt synchronizacji. Jeżeli obiekt jest dostępny funkcja wraca natychmiast i wątek kontynuuje działanie, jeżeli obiekt jest niedostępny funkcja wstrzymuje wątek, który ją wywołał.

```
Unlock();
```

Zwalnia obiekt i uruchamia kolejny wątek wstrzymany funkcją `Lock`.

Klasa CSingleLock

Klasa opisuje funkcję blokującą oczekującą na zmianę stanu pojedynczego obiektu synchronizacji.

```
CSingleLock( CSyncObject* pObiekt, BOOL ZajmijObiekt );
```

Konstruktor, tworzy nowy obiekt. Parametr `ZajmijObiekt` jest opcjonalny.

Parametr	opis
`pObiekt`	Wskaźnik do obiektu synchronizacji.
`ZajmijObiekt`	Jeżeli `TRUE`, konstruktor przejmie obiekt synchronizacji wskazany przez pierwszy parametr (tzn. wywoła funkcję `Lock`). Domyślnie `FALSE`.

```
Lock(DWORD TimeOut);
```

Jeżeli obiekt synchronizacji związany z funkcją blokującą jest dostępny funkcja wraca natychmiast i wątek kontynuuje działanie, jeżeli obiekt jest niedostępny funkcja wstrzymuje wątek na podany czas (domyślnie jest to nieskończoność). Jeżeli po upływie podanego czasu obiekt synchronizacji nie będzie dostępny, funkcja zwróci wartość 0.

```
Unlock();
```

Zwalnia obiekt synchronizacji związany z funkcją blokującą i uruchamia kolejny wątek wstrzymany funkcją `Lock`.

Klasa CWnd

Klasa opisuje okno widoczne na ekranie. W ćwiczeniu mogą być potrzebne funkcje wymienione niżej.

```
PostMessage( UINT message, WPARAM wParam, LPARAM lParam);
```

Wysyła meldunek do okna. Funkcja umieszcza meldunek w kolejce meldunków i nie czeka na jego obsługę przez okno.

Parametr	opis
`message`	Liczba naturalna, kod meldunku
`wParam`	Dowolna liczba całkowita, domyślnie 0
`lParam`	Dowolna liczba całkowita, domyślnie 0

```
UINT SetTimer(UINT ID, UINT Czas, void (CALLBACK EXPORT* lpfnTimer)(HWND, UINT,
UINT, DWORD) );
```

Funkcja uruchamia czasomierz.

Parametr	opis
`ID`	Identyfikator czasomierza, niezerowa liczba naturalna
`Czas`	Czas w milisekundach odmierzany przez czasomierz
`lpfnTimer`	Adres funkcji, która będzie wywoływana, gdy upłynie czas określony przez drugi parametr. Jeżeli ten parametr ma wartość 0, okno otrzymuje meldunek `WM_TIMER` co określony czas.

```
CWinThread* AfxBeginThread( AFX_THREADPROC pFunkcja, LPVOID pParametr,
int Priorytet, UINT Stos, DWORD Flagi, LPSECURITY_ATTRIBUTES pPrawaDostepu);
```

Funkcja uruchamia nowy wątek roboczy.

Parametr	opis
`AFX_THREADPROC`	Wskaźnik do funkcji, która będzie wykonywana jako wątek roboczy. Funkcja musi być postaci: `UINT NazwaFunkcji( LPVOID pParam );` i może być funkcją nie należącą do żadnej klasy lub statyczną funkcją dowolnej klasy.
`pParametr`	Wskaźnik, który zostanie przekazany wątkowi roboczemu jako parametr. Może być to dowolna zmienna, trzeba tylko wymusić konwersję typu do `void *`
`Priorytet`	Poziom priorytetu wątku. Domyślnie taki sam jak poziom wątku, który wywołuje `AfxBeginThread`
`Flagi`	Określa, czy wątek ma być wstrzymany po utworzeniu, czy uruchomiony. Domyślnie wątek jest uruchomiony
`Stos`	Wielkość stosu. Domyślnie taki sam jak stos wątku, który wywołuje `AfxBeginThread`
`pPrawaDostepu`	Prawa dostępu do wątku. W Windows 9x ignorowane. Domyślnie 0.

```
CWnd* AfxGetMainWnd( );
```

Funkcja zwraca wskaźnik do okna głównego programu.

Przykłady

Wysłanie meldunku

Poniższe wyrażenie wysyła do okna głównego meldunek o kodzie `WM_WIADOMOSC` z parametrami 5 oraz wartościami dwóch zmiennych: j oraz k. Funkcja odbierająca meldunek musi odpowiednio zinterpretować drugi parametr, aby odzyskać wartości tych zmiennych.

```
char j;
unsigned int k;
AfxGetMainWnd()->PostMessage(WM_WIADOMOSC,5,j+k<<8);
```

Wyłączny dostęp do współdzielonego zasobu

W środowisku wielowątkowym wyświetlanie na wspólnym ekranie może wyglądać tak jak niżej.

```
int wyswietl(char *s)
{
  static CCriticalSection cs();
  int w;

  cs.Lock();
  w=printf(s);
  cs.Unlock()
  return w;
}
```

Oczekiwanie na zdarzenie

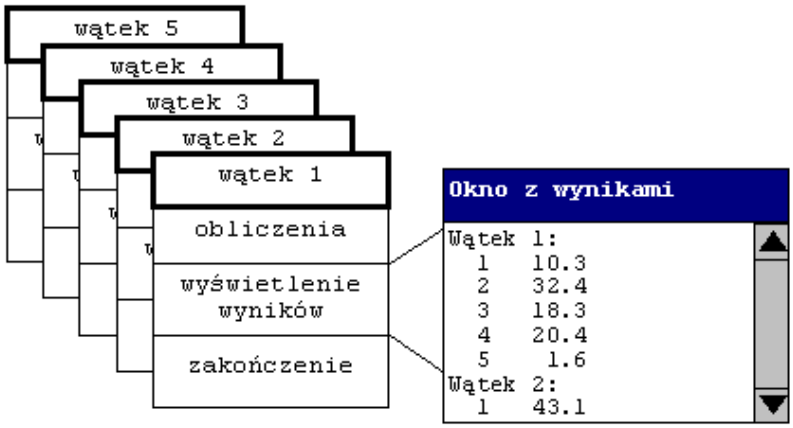
Wątek 1 powinien czekać, aż wątek 2 zakończy pewien proces. Wątek 2 informuje o tym za pomocą obiektu synchronizacji typu zdarzenie.

Wspólne zasoby	Wątek 1	Wątek 2

`CEvent e;`	`UINT w1(void *p) {`
	`  CSingleLock SL(&e);`
	`  SL.Lock();`
	`  obliczaj2();`
	`  return 0;`
	`}`
		`UINT w2(void *p) {`
		`  obliczaj1();`
		`  e.SetEvent();`
		`  return 0;`
		`}`

Konstrukcja programu.

Pięć wątków pracuje równolegle wykonując obliczenia. Czas trwania obliczeń jest różny dla każdego wątku. Po zakończeniu obliczeń wątki wyświetlają wyniki we wspólnym oknie. W czasie, gdy jeden z wątków wyświetla wyniki żaden inny nie powinien mieć dostępu do okna.



Okno główne programu pokazane jest na ilustracji 1. Przycisk *Uruchom* uruchamia wątki, przycisk *Przerwij* zatrzymuje pracujące wątki, przycisk *Koniec* kończy program.

Komunikacja między wątkami

W programie istnieje sześć niezależnych wątków: jeden wątek główny i pięć wątków roboczych. Wątki robocze pokazują informacje o swoim stanie w swoich oknach. Wynik obliczeń pokazywany jest na ekranie (największy element typu *Edit Control* na ilustracji 1). Wątek główny po wciśnięciu przycisku *Przerwij* informuje o tym wątki robocze, które powinny się jak najszybciej skończyć.
Korzystając z MFC trzeba przestrzegać zasady, że wątek może używać tylko tych obiektów, które sam utworzył. W związku z tym, wyświetlać wiadomości i wyniki obliczeń może tylko wątek główny.
Wątki robocze muszą te dane w pewien sposób przekazać do wątku głównego.

Wynik obliczeń

Wynik obliczeń może być przekazywany przez zmienną globalną, do której dostęp mają wątki robocze i wątek główny. Dostęp do tej zmiennej musi być synchronizowany, ponieważ może się zdarzyć, że w trakcie wyświetlania wyniku obliczeń przez wątek główny, zakończy się kolejny wątek roboczy i zamaże wynik obliczeń poprzedniego wątku.

Stan wątku roboczego

Wątek roboczy może znajdować się w następujących stanach:

1 przeprowadza obliczenia
2 oczekuje na dostęp do zmiennej globalnej, w której umieści wynik obliczeń
3 oczekuje, aż wątek główny wyświetli wynik obliczeń na ekranie
4 zakończył się

Z każdym stanem można związać pewną wiadomość, która powinna pojawić się w oknie wątku roboczego. Wątek roboczy może przesyłać informacje o swoim stanie za pomocą meldunków. Wysłanie meldunku odbywa się w dwóch krokach:

- korzystając z funkcji `AfxGetMainWnd` uzyskać wskaźnik okna głównego
- funkcją `PostMessage` wysłać do okna głównego meldunek

Funkcja `PostMessage` może przekazać dwa parametry, pierwszym może być numer wątku, drugim jego stan.

Przerwanie wątku roboczego

Wątek główny po wciśnięciu przycisku *Przerwij* może zapisać odpowiednią wartość do zmiennej globalnej, która jest sprawdzana przez wątki robocze. Wątki robocze mogą sprawdzać wartość tej zmiennej podczas przeprowadzania obliczeń i w momencie przejścia z jednego stanu do następnego. Wątek główny musi wiedzieć, kiedy wszystkie wątki zakończyły się. Można do tego celu użyć zmiennej globalnej, którą każdy wątek zwiększa o 1 w momencie rozpoczęcia i zmniejsza o 1 w momencie zakończenia.

Wątek roboczy - algorytm

- Wykonywać obliczenia, co 0,5 sekundy wysyłać meldunek do okna głównego. Obliczenia są symulowane, wątek przez czas z przedziału 5..20 sekund nic nie robi, tylko wysyła meldunki i sprawdza, czy powinien się zakończyć
- Uzyskać dostęp do zmiennej globalnej z wynikami
- Zapisać wynik obliczeń w zmiennej globalnej
- Zaczekać, aż wątek główny wyświetli wynik obliczeń
- Zwolnić dostęp do zmiennej z wynikami

Zadania wątku głównego

- Po wciśnięciu przycisku *Uruchom* należy wygenerować czasy pracy dla wątków roboczych i uruchomić wątki. Czasy pracy wątków można przechowywać w zmiennych globalnych, do których dostęp mają zarówno wątki robocze, jak i wątek główny.
- Obsługiwać meldunki od wątków roboczych
- Co 4 sekundy sprawdzać stan zmiennej globalnej, do której wątki robocze zapisują wynik obliczeń. Gdy pojawi się w niej niezerowa wartość wyświetlić ją na ekranie i korzystając z jednego z obiektów synchronizacji poinformować wątek roboczy o wyświetleniu wyniku
- Po wciśnięciu przycisku *Przerwij* należy ustawić zmienną globalną, która poinformuje wątki robocze o konieczności zakończenia pracy. Zaczekać, aż wszystkie wątki robocze się zakończą.

Przebieg ćwiczenia

W dostępnym na laboratorium szablonie znajdują się etykiety `TODO:` które wskazują miejsca które należy uzupełnić. Dla szybszego ich wyszukania można wybrać z menu View > Show Tasks > All..

Zmienne globalne

W pliku Okno.cpp dodać zmienne globalne, tak jak pokazano w tabeli.

Typ	opis
`całkowity`	Zmienna powinna być ustawiana na 1 w odpowiedzi na wciśnięcie przycisku *Przerwij*. Wątki robocze powinny sprawdzać stan tej zmiennej i odpowiednio reagować
`tekst`	Do tej zmiennej wątki robocze zapisują wynik obliczeń. Zmienna jest sprawdzana co 4 sekundy przez wątek główny i jej wartość jest wyświetlana na ekranie. Wątki robocze powinny korzystać z obiektów synchronizacji podczas dostępu do tej zmiennej
`całkowity`	Tablica z czasami pracy wątków, wypełniana po wciśnięciu przycisku *Uruchom*. Jednostką jest 500 ms.
`całkowity`	Licznik uruchomionych wątków
`x`	obiekt synchronizacji zapewniający dostęp do zmiennej, w której wątki robocze zapisują wynik obliczeń
`x`	obiekt synchronizacji za pomocą którego wątek główny informuje wątki robocze, że wyświetlił wynik obliczeń na ekranie
`x`	funkcja blokująca używana przez wątki robocze podczas oczekiwania na potwierdzenie wyświetlenia wynik obliczeń na ekranie. Może być również umieszczona jako zmienna lokalna funkcji wątku roboczego

Uruchomienie wątków

Wątki uruchamia się po wciśnięciu przycisku *Uruchom*. Wywoływana jest wówczas funkcja `OnBnClickedUruchom`. Funkcja blokuje przycisk *Uruchom*, odblokowuje przycisk *Przerwij* i za pomocą funkcji `AfxBeginThread` powinna uruchomić wszystkie wątki. Wątkowi przekazywany jest jeden parametr - jego numer.

```
void Cokno::OnBnClickedUruchom()
{
 m_uruchom.EnableWindow(FALSE);
 m_przerwij.EnableWindow();

  // wyzerować zmienną, która informuje wątki
  // robocze o wciśnięciu przycisku Przerwij
  for(int i=0;i<5;i++)  {
    // wygenerować czasy pracy wątków:
    // 10+(30*rand())/RAND_MAX;
    AfxBeginThread(WatekRoboczy,(void *)i);
  }
}
```

Implementacja wątku roboczego

Pętla `do - while` odpowiada za fazę obliczeń wątku. Linie zawierające komentarz należy zastąpić odpowiednim kodem w języku C++. Należy pamiętać, że przed zakończeniem wątku trzeba zmniejszyć licznik wątków o 1. Wysyłając meldunek przekazać dwa parametry: numer wątku i jego stan. Wynikiem obliczeń może być dowolny tekst, np.
`"Wątek 1 zakończył się\n".`
Przejście do nowej lini w elemencie typu *Edit Control* wymaga użycia „\r\n”

```
UINT COkno:: WatekRoboczy(void * pParam)  {
 int i = (int )pParam;  // numer wątku

 // zwiększyć licznik wątków
 do {
  // sprawdzić, czy zakończyć wątek
  // wysłać meldunek o stanie wątku (stan 1)
  Sleep(500);  // to są obliczenia
  // jeżeli upłynął czas pracy wątku przerwać
  // pętlę
 } while(1);
 // sprawdzić, czy zakończyć wątek
 // wysłać meldunek o stanie wątku (stan 2)
 // uzyskać dostęp do zmiennej z wynikami
 // sprawdzić, czy zakończyć wątek
 // zapisać wynik obliczeń w zmiennej
 // wysłać meldunek o stanie wątku (stan 3)
 // zaczekać, aż wątek główny wyświetli wynik
 // zwolnienie dostępu do zmiennej z wynikami
 // sprawdzić, czy zakończyć wątek
 // wysłać meldunek o stanie wątku (stan 4)
 // zmniejszyć licznik wątków
return 0;
}
```

Obsługa meldunku WM_TIMER

Funkcja `OnTimer` powinna wyświetlić tekst z wynikiem obliczeń i poinformować o tym fakcie oczekujący wątek roboczy.
Nie trzeba analizować parametru `nIDEvent`, ponieważ uruchomiony był tylko jeden czasomierz.

```
void COkno::OnTimer(UINT nIDEvent)
{
 CString s;
 char buffer[1024];

 CDialog::OnTimer(nIDEvent);
 // jeżeli nie ma wyniku koniec funkcji
 m_ekran.GetWindowText(buffer,1024);
 s = buffer
 s = s + <zmienna z wynikiem>;
 m_ekran.SetWindowText(s);
 // zaznaczyć, że nie ma wyniku
 // ustawienie stanu odpowiedniego
 // obiektu synchronizacji
 // (informacja dla czekającego wątku)
}
```

Przerwanie wątków

Po wciśnięciu przycisku *Przerwij* należy ustawić wartość zmiennej, która informuje wątki robocze, że powinny się zakończyć. Ponieważ niektóre wątki mogą czekać na potwierdzenie wyświetlenia wyników obliczeń, trzeba wywołać funkcję `OnTimer`, która zwolni takie wątki.

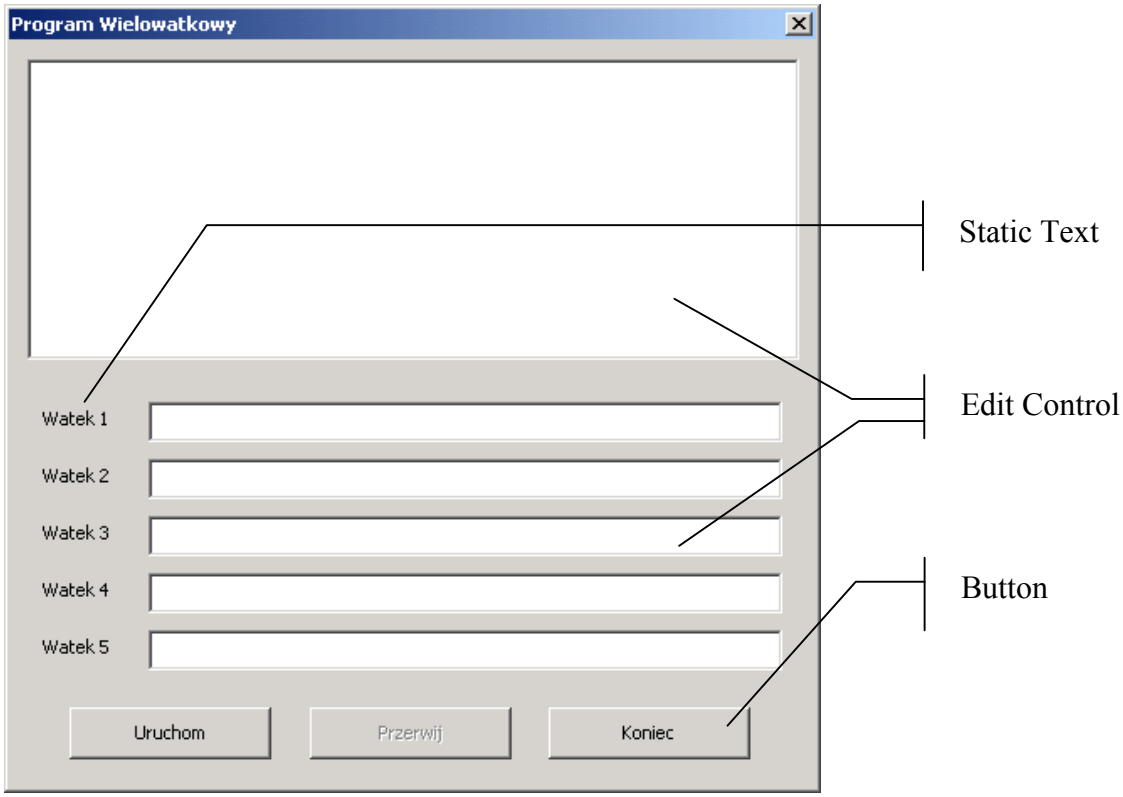
```
void COkno:: OnBnClickedPrzerwij()
{
// ustawić zmienną, która informuje wątki
// robocze o wciśnięciu przycisku Przerwij
 m_uruchom.EnableWindow();
 m_przerwij.EnableWindow(FALSE);
 OnTimer(1);
// zaczekać na koniec wszystkich wątków
}
```

Tworzenie aplikacji

Na zajęciach udostępniony zostanie projekt wstępny gotowy do oprogramowania. Poniżej podana jest procedura w jaki sposób był on tworzony.

Przygotowanie okna głównego

Utworzyć nowy projekt *Visual C++ Projects > MFC Application* . Jako typ aplikacji wybrać *Dialog based*. Klasę okna głównego nazwać `COkno`. Odpowiednio zmienić także nazwy plików okna głównego (..Dlg.h na Okno.h itd.). Utworzyć okno główne według Ilustracji 1.



Ilustracja 1 Okno główne programu

Przyciskom nadać identyfikatory `IDC_URUCHOM`, `IDC_PRZERWIJ`, `IDC_KONIEC` (kliknąć prawym przyciskiem myszy, z menu podręcznego wybrać *Properties* i w oknie ID: wpisać `IDC_...`). Elementowi głównemu *Edit Control* nadać identyfikator `IDC_EKRAN` i ustawić właściwość: *MultiLine* oraz *Vertical Scroll* na *True*. Pięciu pozostałym oknom edycyjnym nadać identyfikatory `IDC_W1...IDC_W5`. Wszystkim elementom typu *Edit Control* ustawić właściwość *Read Only* na *True*. Zablokować przycisk *Przerwij*.

Dla każdego elementu, dla którego zmieniane były identyfikatory dodać zmienną o nazwie takiej jak identyfikator z przedrostkiem `m_`, np. zmienna związana z przyciskiem *Uruchom* będzie nazywać się `m_uruchom`.

W pliku stdafx.h dopisać linię `#include <afxmt.h>`. Spowoduje to dołączenie deklaracji obiektów synchronizacji i funkcji blokujących.

W funkcji `OnInitDialog` klasy `COkno` dopisać linie:

```
srand((unsigned )time(0));
SetTimer(1,4000,0);
```

Pierwsza funkcja inicjuje generator liczb losowych, druga uruchamia czasomierz o identyfikatorze 1, który co 4000 ms będzie wysyłać do okna głównego meldunki `WM_TIMER`.

Wątek roboczy

Wątek roboczy może być globalną funkcją lub statyczną funkcją klasy okna głównego. W oknie *Class View* kliknąć prawym przyciskiem mys zy na klasie *COkno* i menu podręcznego wybrać *Add* > *Add Function*. W wyświetlonym okienku wprowadzić deklarację funkcji wątku roboczego:

```
UINT WatekRoboczy(void * pParametr);
```

Zaznaczyć, że funkcja ma być statyczna. Typ wartości zwracanej przez funkcję musi być `UINT`, typ parametru musi być `void *`.

Na początku pliku Okno.cpp dodać deklarację meldunku, który będzie wysyłany przez wątki robocze:

```
#define WM_WIADOMOSC (WM_USER+100)
```

Dodać funkcję obsługującą zdarzenie `BN_CLICKED` przycisku *Uruchom* (kliknąć prawym przyciskiem myszy na przycisku Uruchom > Add Event Handler)

```
void Cokno::OnBnClickedUruchom()
```

Meldunek WM_WIADOMOSC

Do klasy *COkno* dodać funkcję (identycznie jak w punkcie *Wątek roboczy*)

```
afx_msg LRESULT OnWiadomosc(WPARAM n,LPARAM m);
```

która będzie obsługiwać meldunek `WM_WIADOMOSC`. Typ wartości zwracanej przez funkcję musi być `afx_msg LRESULT`, typy parametrów muszą być `WPARAM` i `LPARAM`.

Aby związać meldunek z funkcją należy odszukać w pliku Okno.cpp mapę meldunków utworzoną przez Kreatora:

```
BEGIN_MESSAGE_MAP(COkno, CDialog)
//{{AFX_MSG_MAP(COkno)
```

i dopisać makro:

```
ON_MESSAGE(WM_WIADOMOSC, OnWiadomosc)
```

Obsługa meldunku `WM_WIADOMOSC`

Funkcja obsługująca meldunek otrzymuje dwa parametry: numer wątku i jego stan. Ilustracja pokazuje przykładowy sposób wyświetlenia przekazanej wiadomości.

```
LRESULT COkno::OnWiadomosc(WPARAM n,LPARAM m)
{
  static char* w[4] = {
    "Oblicza...", "Czeka na ekran...",
    "stan 3", " stan 4"};
  switch(n) {
    case 0: m_w1.SetWindowText(w[m]);
          break;
   ...
  }
  return 0;
}
```

Obsługa meldunku `WM_TIMER`

Wybrać zakładkę *Class View*. Kliknąć prawym przyciskiem myszy na klasie *COkno* i menu podręcznego wybrać *Properties*. W wyświetlonym okienku właściwości wybrać zakładkę *Messages*. Obok meldunku `WM_TIMER` wybrać <*Add*> *OnTimer*.

Podobnie jak w przypadku przycisku *Uruchom* dodać funkcje

```
void COkno::OnBnClickedPrzerwij();
void COkno::OnBnClickedKoniec();
```

Zakończenie programu

Program może być zakończony na trzy sposoby: przyciskiem *Koniec*, skrótem z klawiatury *Alt+F4* i poleceniem *Zakończ zadanie* z okna Menedżera zadań. W celu prawidłowej obsługi tych trzech zdarzeń należy oprogramować dwie funkcje. Pierwsza funkcja, odpowiedź na meldunek `BN_CLICKED` przycisku *Koniec*, wywołuje funkcję `EndDialog`. Drugą funkcją jest funkcja obsługująca meldunek `WM_DESTROY`. Funkcja ta jest wywoływana automatycznie, gdy system zamyka okno aplikacji. Z funkcji tej należy wywołać funkcję `OnBnClickedPrzerwij` w celu zakończenia wszystkich wątków.

```
void COkno::OnDestroy()
{
      KillTimer(1);
      OnBnClickedPrzerwij();
      CDialog::OnDestroy();
}

void COkno:: OnBnClickedKoniec()
{
      EndDialog(1);
}
```

Sprawozdanie z ćwiczenia

W sprawozdaniu należy wyjaśnić użycie mechanizmów synchronizacji zastosowanych w ćwiczeniu.
Zaproponować rozwiązanie następującego problemu:
Wątki robocze w stanie 1 nie wysyłają informacji, jak długo jeszcze będą trwały obliczenia. W jaki sposób zmodyfikować wysyłanie meldunku o stanie wątku roboczego i funkcję `OnWiadomosc`, aby wyświetlany był pozostały czas obliczeń.

Pytania na wejściówkę

- Omówić sposób wyznaczania priorytetu wątku przez algorytm kolejkowania w Windows.
- Omówić algorytm kolejkowania w Windows.
- Wymienić przypadki, kiedy system zwiększa priorytet wątku.
- Omówić obiekty służące do synchronizacji wątków.
- Omówić funkcje blokujące. Jakie są ograniczenia w ich stosowaniu.
- Scharakteryzować sposoby komunikacji między wątkami.

Literatura:
1. Richard C. Leinecker “Visual C++ 5: narzêdzia programowania”
2. Steven Holzner “Visual C++ 5”
3. Al Williams “MFC czarna ksiêga”
4. Win32 Software Development Kit: Processes and Threads, Synchronization.

Systemy operacyjne

Laboratorium

Ćwiczenie 4

Synchronizacja procesów
Komunikacja między procesami

Wstęp

System Windows ma wbudowany zestaw funkcji służących do komunikacji i współdzielenia danych między procesami. Ogólnie, operacje udostępniane przez te funkcje nazywane są komunikacją między procesami (interprocess communications - IPC). Oprócz ułatwiania podziału pracy pomiędzy kilka wyspecjalizowanych procesów, niektóre formy komunikacji między procesami potrafią rozdzielić obciążenie obliczeniami pomiędzy komputery współpracujące ze sobą w sieci.
Zwykle współpracujące i komunikujące się aplikacje są określane jako klient i serwer. Klient jest aplikacją lub procesem, który żąda pewnej usługi od innego procesu. Serwer jest aplikacją lub procesem, który odpowiada na żądanie klienta. Wiele aplikacji jest zarówno klientem i serwerem, w zależności od sytuacji. Na przykład aplikacja przetwarzająca dokumenty tekstowe może być klientem żądającym tabeli podsumowującej koszty produkcji od arkusza kalkulacyjnego, który jest serwerem. Arkusz kalkulacyjny może być klientem żądającym od aplikacji zarządzającej magazynem danych o stanie zapasów.

Potoki nazwane

Potok jest mechanizmem komunikacji między procesami. Są dwa typy potoków:

potok bez nazwy	służy do jednokierunkowej wymiany danych między procesami na komputerze lokalnym; używając potoku bez nazwy nie można komunikować się przez sieć
potok nazwany	służy do dwukierunkowej wymiany danych między procesami; serwer potoku nazwanego może komunikować się z klientami zarówno na komputerze lokalnym jak i na komputerze zdalnym

Podczas tworzenia potoku nazwanego serwer określa jego nazwę i poziom zabezpieczeń - nazwy użytkowników lub grup użytkowników, którzy mogą korzystać z potoku. Z punktu widzenia klienta, potok jest plikiem, do którego można zapisywać i czytać dane. Operację zapisu do potoku nazwanego może rozpocząć zarówno klient jak i serwer. System Windows posiada także drugi, znacznie prostszy sposób wykorzystania potoków. Transakcja potokiem nazwanym jest sposobem komunikacji między serwerem a klientem, który łączy operację zapisu i odczytu w pojedynczą operację sieciową. Transakcje zwiększają wydajność komunikacji między klientem, a serwerem na komputerze zdalnym. W tym trybie pracy klient rozpoczyna komunikację zapisem wiadomości dla serwera, po czym odczytuje odpowiedź. Transakcja realizowana jest przez wywołanie jednej funkcji systemu.
Serwer potoku nazwanego musi być uruchomiony na komputerze z systemem operacyjnym Windows NT (może być Windows NT Workstation).
Serwer potoku bez nazwy może pracować z systemem Windows 95.

Skrzynki pocztowe

Skrzynka pocztowa jest mechanizmem jednokierunkowej wymiany danych między procesami. Dowolny proces może utworzyć skrzynkę pocztową i stać się serwerem skrzynki pocztowej. Inny proces, zwany klientem skrzynki pocztowej, może uzyskać dostęp do skrzynki pocztowej poprzez jej nazwę i przesyłać wiadomości do serwera. Proces może być jednocześnie serwerem i klientem skrzynki pocztowej, więc możliwa jest dwukierunkowa komunikacja, ale przy użyciu dwóch skrzynek pocztowych.
Przychodzące wiadomości są zawsze dołączane do skrzynki pocztowej, która przechowuje je dotąd, aż serwer je przeczyta.

Skrzynki pocztowe są podobne do potoków nazwanych, ale ich obsługa jest uproszczona (wydajny serwer potoku nazwanego jest procesem wielowątkowym) i dodano możliwość rozgłaszania wiadomości do wszystkich komputerów w domenie lub grupie roboczej klienta. Klient skrzynki pocztowej może wysłać wiadomość do skrzynki na komputerze lokalnym, do skrzynki na innym komputerze lub do wszystkich skrzynek o takiej samej nazwie na wszystkich komputerach w domenie lub grupie roboczej. Wiadomości rozgłaszane nie mogą być dłuższe niż 400 bajtów. Rozmiar wiadomości wysyłanych do pojedynczej skrzynki pocztowej jest ograniczony przez serwer podczas tworzenia skrzynki - wiadomości takie mogą być nieograniczonego rozmiaru.
Serwer skrzynki pocztowej może pracować z systemem Windows 95.

Meldunki

Procesy, które utworzyły okna, mogą się komunikować za pomocą meldunków. Meldunki wykorzystywane do komunikacji między procesami powinny być zarejestrowane w systemie przy użyciu funkcji `RegisterWindowMessage`.

Bezpieczeństwo komunikacji między procesami

W Windows 95 nie istnieją mechanizmy zabezpieczające przed niepowołanym dostępem do skrzynki pocztowej lub potoku bez nazwy. Windows NT ma wbudowane takie mechanizmy. Serwer tworząc potok lub skrzynkę pocztową określa listę użytkowników, lub listę grup użytkowników, którzy mają zapewniony dostęp do powstającego obiektu komunikacji. Serwer ustawia również prawa użytkowników, pozwalając im na zapis lub odczyt. Klient próbujący uzyskać dostęp do potoku lub skrzynki pocztowej poddawany jest weryfikacji. Klient jest procesem uruchomionym na komputerze, na którym zalogował się pewien użytkownik. Zarówno w Windows 95 jak i Windows NT podczas logowania podaje się nazwę użytkownika i hasło. System odbierając żądanie dostępu do obiektu komunikacji sprawdza, czy użytkownik komputera, na którym pracuje klient, znajduje się na liście uprawnionych osób, określonej przez serwer. Jeżeli użytkownika nie ma na liście, lub hasło jest nieprawidłowe, system nie przekaże serwerowi żądania wykonania usługi. Transakcja potokiem nazwanym lub przesłanie wiadomości do skrzynki pocztowej zakończy się błędem “Brak dostępu”.

Synchronizacja procesów

Do synchronizacji procesów służą te same obiekty i funkcje blokujące , co do synchronizacji wątków, za wyjątkiem sekcji krytycznej. Ten obiekt służy wyłącznie do synchronizacji wątków. Obiekty używane do synchronizacji procesów muszą posiadać nazwy. Nazwa musi być unikalna dla danej klasy obiektów synchronizacji, tzn. mogą istnieć dwa zdarzenia o tej samej nazwie, ale próba utworzenia semafora o nazwie identycznej z nazwą zdarzeń nie powiedzie się. W nazwie nie może wystąpić znak ”\”.
W Windows 95 i Windows NT 4.0 obiekty synchronizacji używane są do zarządzania procesami tylko na komputerze lokalnym.

Klasy i funkcje wykorzystane w ćwiczeniu

Potoki nazwane - klient

Następująca funkcja realizuje transakcję potokiem nazwanym

```
int CallNamedPipe(char * Nazwa, void * Wiad, long RozmWiad,
      void * Odp, long RozmBufOdp, unsigned long *RozmOdp, long Timeout);
```

parametr	znaczenie
Nazwa	nazwa potoku nazwanego, musi być postaci \\[nazwa komputera]\pipe\[nazwa potoku] jeżeli klient i serwer są na tym samym komputerze, [nazwa komputera] może być zastąpiona przez kropkę
Wiad	wskaźnik do bufora przechowującego wiadomość do wysłania, mogą to być dowolne dane, które potrafi zinterpretować serwer
RozmWiad	rozmiar wiadomości w bajtach
Odp	wskaźnik do bufora przygotowanego na odpowiedź od serwera
RozmBufOdp	rozmiar bufora przygotowanego na odpowiedź w bajtach
RozmOdp	wskaźnik do zmiennej, w której funkcja zapisze rzeczywisty rozmiar odpowiedzi zwróconej przez serwer
Timeout	czas w milisekundach, przez który funkcja będzie czekać aż serwer przygotuje się do komunikacji z klientem, najlepiej użyć stałej NMPWAIT_USE_DEFAULT_WAIT, która poleca funkcji czekać przez czas określony przez serwer podczas tworzenia potoku

Jeżeli transakcja powiedzie się, funkcja zwróci liczbę różną od zera.

Skrzynki pocztowe - serwer

Następująca funkcja używana jest przez serwer skrzynki pocztowej do utworzenia skrzynki

```
HANDLE CreateMailslot(char *Nazwa, long MaksRozmWiad,
      long Timeout, SECUTITY_ATTRIBUTES * Zabezpieczenia);
```

parametr	znaczenie
Nazwa	nazwa skrzynki pocztowej, musi być postaci \\[nazwa komputera]\mailslot\[scieżka]\[nazwa] serwer nie może utworzyć skrzynki pocztowej na komputerze zdalnym, więc [nazwa komputera] musi być zastąpiona przez kropkę, [ścieżka] może być dowolną ścieżką (tak jak w przypadku plików), [nazwa] podobnie jak w plikach może zawierać rozszerzenie po kropce
MaksRozmWiad	maksymalny rozmiar wiadomości w bajtach, aby określić, że wiadomość może mieć dowolny rozmiar należy podać 0
Timeout	czas w milisekundach, przez który operacja odczytu będzie czekać na pojawienie się wiadomości w skrzynce, dwie wartości mają specjale znaczenie: 0 — funkcja odczytująca wróci natychmiast, gdy nie będzie wiadomości MAILSLOT_WAIT_FOREVER — funkcja czeka, aż w skrzynce pojawi się wiadomość
Zabezpieczenia	w Windows 95 wartość 0, w Windows NT określa nazwy użytkowników lub grup użytkowników, którzy mogą zapisywać wiadomości do skrzynki pocztowej oraz inne rodzaje ograniczeń dostępu do skrzynki

Funkcja zwraca identyfikator skrzynki.

Jednym z błędów Windows 95 jest obcinanie nazw skrzynek pocztowych do formatu 8.3. Jeżeli serwer na komputerze z systemem operacyjnym Windows 95 nazwie skrzynkę "\\.\mailslot\fd3\systemy_operacyjne\lab.sprawozdania", to system zamieni tę nazwę na "\\.\mailslot\fd3\systemy_\lab.spr". Klient na komputerze z Windows NT używający poprawnej nazwy nie będzie mógł połączyć się z serwerem. Klient na komputerze z Windows 95 połączy się, ponieważ w jego przypadku nazwa skrzynki pocztowej zostanie obcięta do formatu 8.3. Podobny problem wystąpi, jeżeli serwer pracujący z Windows NT utworzy skrzynkę z nazwami dłuższymi niż 8 znaków. Żaden klient pracujący z Windows 95 nie prześle wiadomości do takiej skrzynki. Rozwiązaniem problemu jest używanie nazw nie dłuższych niż 8 znaków i trzyznakowych rozszerzeń.

Serwer używa poniższej funkcji do sprawdzenia, czy w skrzynce są wiadomości

```
int GetMailslotInfo(HANDLE id, unsigned long *pMaksRozmWiad,
      unsigned long *pRozmNast, unsigned long *pLiczbaWiad,
      unsigned long *pTimeout);
```

parametr	znaczenie
id	identyfikator skrzynki zwrócony przez `CreateMailslot`
pMaksRozmWiad	wskaźnik do zmiennej, która otrzyma maksymalny rozmiar wiadomości w bajtach
pRozmNast	wskaźnik do zmiennej, która otrzyma rozmiar następnej wiadomości oczekującej w skrzynce na odczytanie
pLiczbaWiad	wskaźnik do zmiennej, która otrzyma liczbę wiadomości oczekujących w skrzynce na odczytanie
pTimeout	wskaźnik do zmiennej, która otrzyma czas w milisekundach, przez który operacja odczytu będzie czekać na pojawienie się wiadomości w skrzynce

Zamiast wskaźników można podać wartość 0, funkcja nie zwróci wtedy tak określonego parametru. Serwer odczytuje wiadomości używając funkcji składowej `Read` klasy `CFile`.

Skrzynki pocztowe - klient

Z punktu widzenia klienta skrzynka pocztowa jest plikiem. Do operacji na plikach można użyć klasy `CFile`. Jeżeli klient chce rozgłosić wiadomość (wysłać ją do wszystkich komputerów w domenie lub grupie roboczej), to jako nazwę komputera trzeba podać gwiazdkę. Należy pamiętać o ograniczeniu rozmiaru rozgłaszanych wiadomości - długość nie może przekroczyć 400 bajtów.

Meldunki

Następująca funkcja używana jest do rejestracji meldunku w systemie.

```
UINT RegisterWindowMessage(const char *nazwa);
```

Funkcja zwraca kod meldunku. Nazwa meldunku jest dowolnym ciągiem znaków. Jeżeli dwa różne procesy rejestrują tę samą nazwę, funkcja zwróci identyczny kod meldunku.

Pliki

Pliki i operacje na plikach opisuje klasa `CFile`.

```
CFile(const char * nazwa, unsigned int tryb);
```

Konstruktor, tworzy obiekt w podanym trybie. Klient skrzynki pocztowej jako tryb podaje `CFile::modeWrite + CFile::shareDenyNone`.

```
CFile(int hPlik);
```

Konstruktor, wiąże z obiektem istniejący plik. Parametrem jest identyfikator pliku, taki jak zwraca funkcja `CreateMailslot`

```
Read(void * bufor, unsigned int n);
```

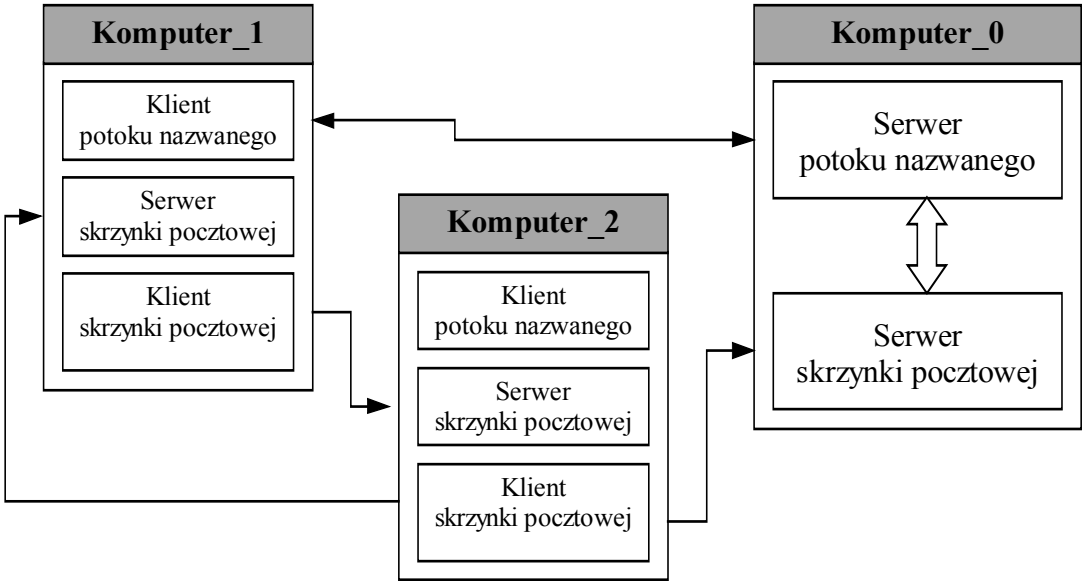
Funkcja czytająca `n` bajtów z pliku, pierwszy parametr jest wskaźnikiem do bufora na odczytane dane. Funkcja zwraca liczbę przeczytanych bajtów

```
Write(void * bufor, unsigned int n);
```

funkcja zapisująca dane do pliku, pierwszy parametr jest wskaźnikiem do bufora zawierającego zapisywane dane, drugi określa ile bajtów funkcja powinna zapisać

Konstrukcja programu

Na komputerze podłączonym do sieci uruchomiony jest serwer potoku nazwanego i serwer skrzynki pocztowej. Serwer potoku nazwanego przechowuje w tablicy listę skrzynek pocztowych. Aby skrzynka została wpisana na listę musi zostać zarejestrowana przez klienta potoku nazwanego. Klient potoku nazwanego może uzyskać od serwera listę zarejestrowanych serwerów skrzynek pocztowych.



Ilustracja 1 Wymiana danych między procesami

Należy napisać klienta potoku nazwanego, serwer skrzynki pocztowej i klienta skrzynki pocztowej. Zadaniem klienta potoku nazwanego jest zarejestrować skrzynkę pocztową obsługiwaną przez serwer i uzyskać od serwera potoku listę skrzynek pocztowych. Następnie klient skrzynki pocztowej powinien wysłać do każdej skrzynki pocztowej z listy przysłanej przez serwer potoku nazwanego wiadomości postaci

```
[użytkownik]@[nazwa komputera]:[dowolny tekst]
```

Po wysłaniu wszystkich wiadomości utworzyć obiekt synchronizacji typu zdarzenie o nazwie ”koniec ćwiczenia” i ustawić jego stan na dostępny. Dołączyć obsługę globalnego meldunku o nazwie `WM_SYSTOPER_LAB4`.

Polecenia rozpoznawane przez serwer potoku nazwanego:

polecenie	opis

`GET`	Serwer zwróci listę zarejestrowanych skrzynek pocztowych w postaci:
	`[liczba skrzynek]`
	`[skrzynka 1]`
	`...`
	`[skrzynka n]`

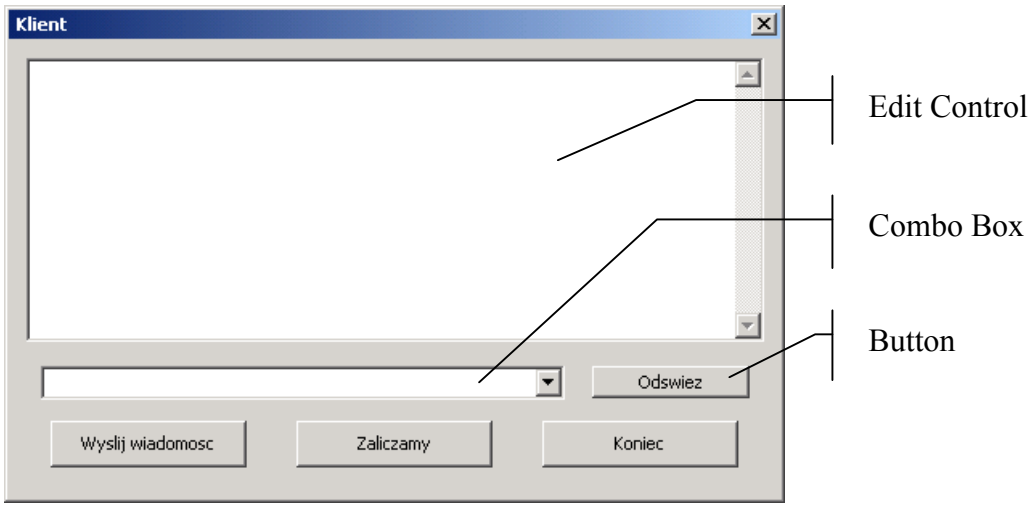
`PUT [nazwa skrzynki]`	Serwer rejestruje skrzynkę pocztową w tablicy, zwraca numer wiersza. Nazwa skrzynki pocztowej musi być prawidłową nazwą sieciową, nie może być kropki zamiast nazwy komputera

Przed rozpoczęciem ćwiczenia należy uruchomić program tester.exe, który sprawdza poprawność wykonania ćwiczenia.

Przebieg ćwiczenia

W dostępnym na laboratorium szablonie znajdują się etykiety `TODO:` które wskazują miejsca które należy uzupełnić.

Okno główne aplikacji wygląda jak pokazuje poniższa ilustracja



Ilustracja 2 Okno główne programu

Obiekt typu Edit Control identyfikuje zmienna `m_ekran`, listę - zmienna `m_lista`. Do wyświetlania tekstu należy użyć funkcji `m_ekran.SetWindowText(tekst);` Przejście do nowej linii odbywa się przez „\r\n”.

W funkcji `OnInitDialog` klasy okna głównego utworzyć skrzynkę pocztową (Skrzynki pocztowe - serwer) i zarejestrować ją na liście skrzynek za pomocą transakcji potokiem nazwanym (Potoki nazwane - klient). Nazwa potoku – „`\\\\.\\pipe\\sysopnp`”. Identyfikator skrzynki zapamiętać w zmiennej składowej klasy opisującej okno.
Utworzoną skrzynkę pocztową można obsługiwać na dwa sposoby: utworzyć wątek sprawdzający, czy przyszły nowe wiadomości, lub sprawdzać stan skrzynki co pewien okres. W celu użycia drugiego sposobu w funkcji `OnInitDialog` klasy okna głównego uruchamiany jest czasomierz wysyłający meldunki co 0,5 sekundy. W funkcji `OnTimer` sprawdzić należy, czy jest nowa wiadomość w skrzynce (Skrzynki pocztowe - klient) i jeżeli jest odczytać ją i wyświetlić.
Po wciśnięciu przycisku *Odśwież* należy za pomocą transakcji potokiem nazwanym pobrać listę zarejestrowanych skrzynek pocztowych (funcja `OnBnClickedOdswiez()`). Dołączanie elementu do listy rozwijanej typu *Combo Box* odbywa się przy wykorzystaniu funkcji `m_lista.AddString(...)`.
Po wciśnięciu przycisku *Wyślij wiadomość* (funcja `OnBnClickedWyslij()`) należy odczytać z listy rozwijanej wybraną skrzynkę pocztową i wysłać do niej wiadomość. Odczytanie wybranego elementu z listy jest dostępne przez funkcję `m_lista.GetWindowText(...);`

W celu zarejestrowania globalnego meldunku dodano:

```
static UINT NEAR WM_LAB4 = RegisterWindowMessage("WM_SYSTOPER_LAB4");
```

oraz utworzono funkcję `OnWiadomosc` obsługującą meldunek (tak jak w Ćwiczeniu 3) związaną ze zmienną `WM_LAB4` za pomocą makrodefinicji `ON_REGISTERED_MESSAGE` (tak jak w Ćwiczeniu 3).

Wciśnięcie przycisku *Zaliczamy* (funkcja `OnBnClickedZaliczamy()`) powinno spowodować utworzenie obiektu synchronizacji typu zdarzenie o nazwie ”koniec ćwiczenia” i ustawienie jego stanu na dostępny. Okno główne otrzyma wtedy od programu tester.exe meldunek zawierający kod, który należy umieścić w sprawozdaniu.

Skrzynki pocztowej można nie zamykać, system zamknie ją automatycznie z chwilą zakończenia głównego wątku aplikacji.

Sprawozdanie z ćwiczenia

Meldunek `WM_SYSTOPER_LAB4` przekazuje procesowi 32-bitowe parametry `wParam` oraz `lParam`. W sprawozdaniu umieścić liczę, którą powstanie przez połączenie ze sobą szesnastkowych wartości tych parametrów, np.

```
wParam=0x1234, lParam=0x567
```

w sprawozdaniu umieszczamy 0000123400000567

Pytania na wejściówkę

- Wymienić mechanizmy komunikacji między procesami.
- Omówić potok nazwany (format nazwy, rodzaje operacji, ograniczenia serwera).
- Omówić skrzynki pocztowe (format nazwy, operacje serwera, operacje klienta, ograniczenia)
- Omówić bezpieczeństwo komunikacji między procesami.
- Omówić obiekty służące do synchronizacji procesów.

Literatura:

Win32 Software Development Kit:	Processes and Threads, Synchronization.
	Mailslots, Pipes
	Security

Systemy operacyjne

Laboratorium

Ćwiczenie 5

Technologia ActiveX

1. Wprowadzenie

Mianem ActiveX firma Microsoft określiła w 1996r. kolejną edycję technologii OLE (po OLE 3.0). W zamierzeniach, zmiana nazwy miała podkreślić zastosowanie do aplikacji internetowych.
ActiveX umożliwia współpracę modułów programowych, nawet wtedy, gdy zostały one utworzone przez niezależnych od siebie twórców, za pomocą różnych narzędzi programistycznych lub w innych językach programowania. Współpraca ta jest możliwa w ramach jednego procesu, między procesami na tym samym stanowisku lub w środowisku rozproszonym.
Termin ActiveX dotyczy kilku zagadnień, wśród których najważniejszymi są:

- Kontrolki ActiveX – interaktywne elementy wyświetlane w oknie aplikacji (stronie WWW)
- Dokumenty ActiveX – praca z nimi jest możliwa z poziomu dowolnej aplikacji będącej tzw. kontenerem dokumentów ActiveX (np. Internet Explorer)
- Zastosowania ściśle internetowe – skrypty (JScript, VBScript), współpraca serwera Web z innymi modułami.

W ćwiczeniu uwagę położono na kontrolki ActiveX.

2. Podstawy ActiveX

ActiveX bazuje na modelu COM (*Component Object Model*), który definiuje binarny interfejs między modułami programowymi. COM jest zorientowany obiektowo i wspiera takie cechy programowania obiektowego jak kapsułkowanie, abstrahowanie czy poliformizm. Obiekty COM mogą być wielokrotnie wykorzystywane na poziomie binarnym (a nie poziomie kodu, jak w tradycyjnym programowaniu) bez ponownej kompilacji – ich użycie nie wymaga dostępności kodu źródłowego. Programy wykorzystujące obiekt COM mogą być utworzone w dowolnym języku programowania. Zmiana wersji obiektu na nowszą, np. z nowymi funkcjami nie powoduje konieczności modyfikacji dotychczasowych programów. Obiekty COM (ale nie ActiveX) mogą również być wykorzystywane między stanowiskami w środowisku rozproszonym (DCOM – *Distributed COM*).
Obiekt COM (w tym kontrolka ActiveX) może mieć zaimplementowanych kilka *interfejsów* (tablic wskaźników do funkcji). Każdy z nich reprezentuje pewną funkcjonalność obiektu, udostępnianą klientom. Interfejs można porównać z abstrakcyjną klasą C++, w której wszystkie funkcje składowe są czystymi funkcjami wirtualnymi, tzn. takimi o których nic nie wiadomo. Interfejs obiektu COM jest *niemodyfikowalny* –nie można do niego dodawać ani usuwać funkcji po opublikowaniu obiektu. W kolejnych wersjach komponentu tworzy się dodatkowe interfejsy, które udostępniają nową funkcjonalność (posiadają dodatkowe funkcje). Można również opublikować całkowicie nowy komponent z nową funkcjonalnością.
Każdy obiekt COM musi mieć zaimplementowany interfejs o nazwie *IUnknown*, z którego są dziedziczone inne interfejsu obiektu. *IUnknown* udostępnia 3 funkcje: **QueryInterface**, **AddRef** oraz **Release**. Pierwsza z nich jest najistotniejszą funkcją całego obiektu, gdyż z jej pomocą klient sprawdza, czy dany interfejs jest udostępniany przez obiekt i uzyskuje do niego wskaźnik. Dwie pozostałe służą do zwiększenia (AddRef) lub zmiejszenia (Release) licznika odwołań do obiektu. Jeżeli licznik wynosi 0, obiekt jest usuwany z pamięci.
Każdy obiekt ActiveX posiada unikalny 128-bitowy identyfikator. Oprócz tego, każdy z interfejsów posiada taki identyfikator. Identyfikatory noszą nazwę CLSID, GUID lub UUID. Występują one w rejestrze systemowym i służą do jednoznacznej identyfikacji obiektu i jego interfejsów.

3. Cel ćwiczenia

Celem ćwiczenia jest utworzenie kontrolki ActiveX wyświetlającej animowany plik AVI. Kontrolkę można wykorzystać w innych programach oraz na stronie WWW. Animacją można sterować (zatrzymać i uruchomić). Można również zmienić animowany plik na inny.
Do materiałów dołączono przykładowe pliki animacji oraz stronę do testowania kontrolki.

4. Przebieg ćwiczenia

1. Utworzyć nowy projekt w VC++ wybierając typ *MFC ActiveX Control*.W zakładce *Control Names* należy zmienić nazwę klasy kontrolki (pole *Control Name*) na SOAnimacja. Pozostałe opcje należy pozostawić bez zmian.
2. W oknie widoku klas zaobserwować, jakie klasy i interfejsy zostały utworzone przez kreatora. Znaleźć w kodzie wygenerowane identyfikatory kontrolki i interfejsów. Podać jakie metody są udostępniane standardowo przez interfejsy kontrolki wygenerowane w VC++.
3. Dodać do klasy reprezentującej kontrolkę pole wskaźnikowe do typu *CAnimateCtrl* o nazwie *m_pMyAnimateCtrl*. Będzie ono reprezentowało okno z animacją. W konstruktorze zainicjować ją wartością NULL. Dodać obsługę komunikatu WM_CREATE (dla klasy SOAnimacja wybrać z paska przycisków w okienku *Properties* przycisk *Messages*). Utworzona zostanie funkcja *OnCreate(..)*. Dopisać do niej następujące linie:

```
m_pMyAnimateCtrl = new CAnimateCtrl;      // Utworzenie klasy okna
animacyjnego
CRect wndSize;
this->GetClientRect(&wndSize);      // Pobranie rozmiarów okna
kontrolki
// Utworzenie okna animacyjnego
#define ID_SOANIMCTRL   1001
UINT styles = WS_CHILD | ACS_TRANSPARENT | ACS_AUTOPLAY | ACS_CENTER;
m_pMyAnimateCtrl->Create(styles, wndSize, this,ID_SOANIMCTRL);
// Wyświetlenie okna i wskazanie pliku z animacją
m_pMyAnimateCtrl->ShowWindow(SW_SHOW);
m_pMyAnimateCtrl->Open("C:\\SOLAB\\CW5\\AVI\\KULA.AVI");  // Zmienić
zgodnie z faktycznym położeniem plików
```

4. W destruktorze kontrolki dopisać linie:
```
m_pMyAnimateCtrl->DestroyWindow();
delete m_pMyAnimateCtrl;
```
5. Zbudować projekt i uruchomić *OLE View*. Sprawdzić, czy kontrolka została zarejestrowana (VC++ robi to automatycznie przy budowaniu projektu).
6. W pliku *test\test.htm* przy znaczniku <*OBJECT*> zmienić identyfikator CLSID na taki, jaki posiada utworzona kontrolka:

```
<OBJECT id=SOAnim1 style="BORDER-LEFT-COLOR: black; LEFT: 0px;
BORDER-BOTTOM-COLOR: black; COLOR: black; BORDER-TOP-COLOR: black;
TOP: 0px; HEIGHT: 160px; BACKGROUND-COLOR: black; BORDER-RIGHT-COLOR:
black" name=SOAnimation classid=clsid:XXXXXXXX-XXXX-XXXX-XXXX-
XXXXXXXXXXXX width=160 height=160>
```

   Otworzyć plik *test\test.htm* w przeglądarce Internet Explorer. Sprawdzić, czy animacja działa. Czy następuje reakcja na naciśnięcie przycisku *O kontrolce*, który wywołuje metodę *AboutBox* kontrolki.

7. Dodać dwie nowe metody do interfejsu kontrolki: *Uruchom()* oraz *Zatrzymaj()*. Można to zrobić za pomocą widoku klas w oknie głównym (prawy przycisk myszy) lub za pomocą *Class wizard* w zakładce *Automation*. W okienku dialogowym *Add method* należy podać nazwę zewnętrzną metody, za pomocą której programy mogą komunikować się z kontrolką (*Uruchom/Zatrzymaj*) oraz nazwę funkcji składowej klasy kontrolki, w której znajdzie się implementacja metody. Jako zwracany typ podać *void* w obu przypadkach.
8. Dodać definicję funkcji składowych *Uruchom* i *Zatrzymaj* klasy kontrolki. W tym celu wykorzystać następujące funkcje klasy *CAnimateCtrl*, do której wskaźnik został utworzony w p. 3:
```
m_pMyAnimateCtrl->Play(0,-1,-1);
m_pMyAnimateCtrl->Stop();
```
Funkcje te powodują cykliczne odtwarzanie animacji oraz jego zatrzymanie. Za pomocą pliku testowego sprawdzić w przeglądarce działanie przycisków, których naciśnięcie powoduje wywołanie utworzonych metod kontrolki.
9. Dodać właściwość *FileName* do interfejsu kontrolki, za pomocą której programy będą mogły odczytywać i ustawiać ścieżkę do odgrywanego pliku. Wybrać rodzaj implementacji *Get/Set functions* oraz ustawić typ zmiennej jako łańcuchowy *BSTR*.
10. W implementacji funkcji *GetFileName* należy zwrócić wartość zmiennej łańcuchowej klasy kontrolki przechowującej nazwę pliku z animacją (np. *m_fileName*). Zmienna ta może być typu łańcuchowego *CString* i jej wartość należy przypisać do zmiennej lokalnej *strResult*, utworzonej przez *Class wizard*. Należy pamiętać o zainicjowaniu tej zmiennej w konstruktorze ścieżką do pliku podaną w funkcji *OnCreate* (patrz p. 3).
11. W funkcji *SetFileName* należy przypisać przekazywany łańcuch znakowy do zmiennej *m_fileName* oraz wywołać funkcję *Open* okna animacji (jak w kodzie z p. 3).
12. Sprawdzić, czy nazwa pliku jest poprawnie wyświetlana i czy istnieje możliwość jej zmiany za pomocą strony testowej. Można w tym celu wykorzystać plik *ZEGAR.AVI*, znajdujący się w tym samym katalogu, co poprzedni plik animacji.
13. Została utworzona testowa dialogowa aplikacja MFC, odpowiadająca funkcjonalnością stronie WWW. W głównym oknie dialogowym znajduje się kontrolka animacyjna oraz przyciski *Uruchom* i *Zatrzymaj*. Za pomocą Class Wizard dodano zmienną *m_animacja*, reprezentującą kontrolkę. (Na pytanie, czy utworzyć klasy C++ odpowiadające obiektowi ActiveX (*wrapper classes*) należy odpowiedzieć twierdząco.) Zaobserwować, jakie funkcje ma utworzona klasa reprezentująca kontrolkę oraz jaka jest ich implementacja. Wywołać odpowiednie funkcje tej klasy po naciśnięciu przycisków *Uruchom/Zatrzymaj*. Zbudować i przetestować aplikację.

Pytania na wejściówkę

1. Co to jest interfejs (*Interface*) ?
2. Co to jest koklasa (*CoClass*) ?
3. Jakie wyróżniamy składniki klasy ?
4. Jakie wyróżniamy składniki interfejsu ?
5. Wymień różnice pomiędzy polem (*field*) a właściwością (*property*).

Źródła

- Microsoft MSDN Library
- Al Williams *MFC Czarna księga*, Helion