МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
імені ВОЛОДИМИРА ДАЛЯ

МЕТОДИЧНІ ВКАЗІВКИ

до лабораторних робіт з дисципліни
«ОПЕРАЦІЙНІ СИСТЕМИ»
(для здобувачів вищої освіти 2 курсу денної та заочної форми навчання за спеціальностями 122 "Комп'ютерні науки", 123 "Комп'ютерна інженерія")

ЗАТВЕРДЖЕНО
на засіданні кафедри
комп'ютерних наук та інженерії
Протокол №11 від 07.04.2021 р.

Сєвєродонецьк 2021

УДК 004.451

Методичні вказівки до лабораторних робіт з дисципліни «Операційні системи» (для здобувачів вищої освіти 2 курсу денної та заочної форми навчання за спеціальностями 122 "Комп'ютерні науки", 123 "Комп'ютерна інженерія") / Уклад.: М.В. Деркач. – Сєвєродонецьк: вид-во СНУ ім. В. Даля, 2021. – 40 с.

У методичних вказівках викладені матеріали, необхідні для виконання лабораторних робіт з дисципліни «Операційні системи». Методичні вказівки містять в собі необхідні теоретичні відомості по темам лабораторних робіт, приклади рішення типових завдань у середовищі розробки Microsoft Visual Studio, що дозволяють одержати поглиблене уявлення про суть розглянутого питання, приклади для самостійного освоєння матеріалу заняття, а також список літератури і необхідні довідкові матеріали.

Укладач: М.В. Деркач, доц.

Відповідальний за випуск: О.І. Рязанцев, проф.

Рецензент В.С. Кардашук

ЗМІСТ

ВСТУП

Інтерфейс прикладного програмування API (Application Program Interface) - це набір функцій, що належать ядру (або надбудові) операційної системи, який використовується прикладними і системними програмами, як в складі операційної системи, так і в складі системи програмування.

Все API можна розділити на три класи:

- API як інтерфейс високого рівня, що належить до бібліотек RTL (Run Time Library - бібліотека часу виконання);
- API прикладних і системних програм, що входять в поставку операційної системи;
- Інші інтерфейси API.

Бібліотека RTL включає в себе стандартні підпрограми, які система програмування поставляє на етапі компіляції. У загальному випадку це не тільки модулі системи програмування, але і модулі операційної системи.

У Windows API є безліч, як самих непомітних, так і значних відмінностей від інших API, таких як POSIX API, з яким знайомі програмісти, які працюють в UNIX і Linux. Багато системні ресурси Windows представляються у вигляді об'єктів ядра (kernel objects), для ідентифікації і звернення до яких використовуються дескриптори (handles). За змістом ці дескриптори аналогічні дескрипторам (descriptors) файлів й ідентифікаторів (ID) процесів в UNIX.

Існує точка зору, що Windows - це всього лише API операційної системи, що надає набір засобів для вирішення призначених для користувача завдань.

Будь-які маніпуляції з об'єктами ядра здійснюються тільки з використанням Windows API. "Лазівок" для обходу цього правила немає. Подібна організація роботи узгоджується з принципами абстрагування даних, використовуваними в об'єктно-орієнтованому програмуванні, хоча сама система Windows об'єктно-орієнтованою не є.

До об'єктів відносяться файли, процеси, потоки, канали взаємодії між процесами, об'єкти відображення файлів, події і багато іншого. Об'єкти мають атрибути захисту.

Windows - багатий можливостями і гнучкий інтерфейс. По-перше, одні й ті ж або аналогічні завдання можуть вирішуватися за допомогою відразу декількох функцій; так, є допоміжні функції (convenience functions), отримані об'єднанням часто зустрічаються послідовностей функціональних викликів в одну функцію. По-друге, функції часто мають численні параметри і прапори, багато з яких зазвичай ігноруються.

Windows пропонує численні механізми синхронізації і взаємодії, що забезпечують задоволення найрізноманітніших запитів.

Базовою одиницею виконання в Windows є потік (thread). В одному процесі (process) можуть виконуватися один або кілька потоків.

Для функцій Windows використовуються довгі описові імена. Наведені нижче як приклад імена функцій ілюструють не тільки угоди про використання імен, але і багатоликість функцій Windows:

```
WaitForSingleObject
WaitForSingleObjectEx
WaitForMultipleObjects
WaitNamedPipe
```

Існує також кілька угод, що регулюють порядок використання імен типів:

1.Імена визначених типів даних, необхідних API, також є описовими, і в них повинні використовуватися прописні букви. До найбільш поширених належать такі типи даних:

```
BOOL (визначений як 32-бітовий об'єкт, призначений для зберігання одного логічного значення),
HANDLE, DWORD (32-бітове ціле без знаку),
LPTSTR (покажчик на рядок, що складається з 8- або 16-бітових символів) LPSECURITY_ATTRIBUTES.
```

2.В іменах визначених типів покажчиків операція * не використовується, і вони відображають додаткові відмінності між покажчиками різного типу, як, наприклад, в разі типів LPTSTR (визначений як TCHAR *) і LPCTSTR (визначений як const TCHAR *). Тип TCHAR може позначати як звичайний символьний тип char, так і багатобайтовий тип wchar_t.

3.Що стосується використання імен змінних, в прототипах функцій, також є певні угоди. Так, ім'я lpszFileName відповідає "довгому вказівнику на рядок, що завершується нульовим символом", який містить ім'я файлу, dwAccess - подвійне слово (32 біти), що містить прапори прав доступу до файлу, де "dw" означає "double word" - "подвійне слово".

Лабораторна робота №1
Тема. Структура API-програм.

Мета. Ознайомлення зі структурою Windows - додатків, організацією взаємодії програми з операційною системою, отримання практичних навичок зі складання, написання та налагодження простих програм, що містять опис вигляду вікна програми, віконну процедуру.

Теоретичні відомості

ОС Windows складається з компонентів, що працюють у режимі ядра, і компонентів, що працюють у режимі користувача. Незважаючи на міграцію системи у бік монолітного ядра, вона зберегла деяку структуру. Спрощена схема архітектури, зорієнтована на виконання Win32-додатків, показана на рис. 1.1.

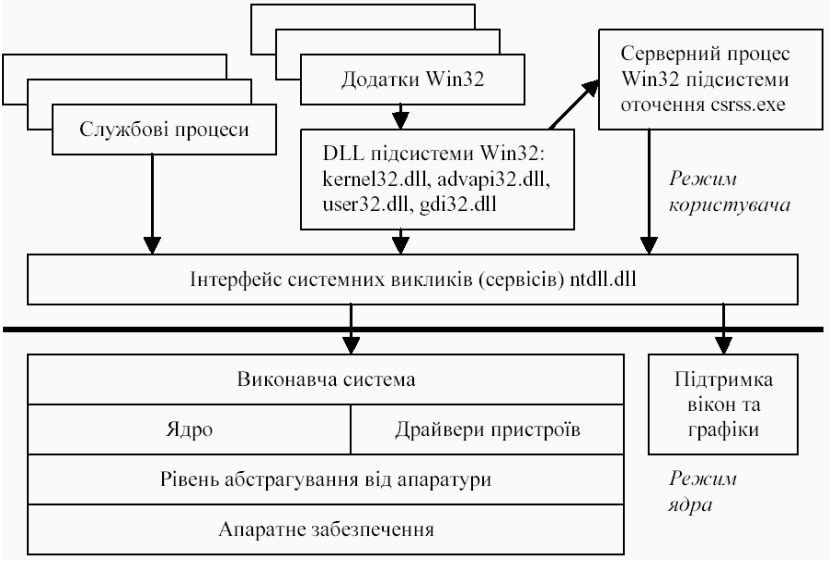

Рис. 1.1. Спрощена архітектурна схема ОС Windows

У вищеподаній схемі виразно видимі кілька функціональних рівнів, кожний з яких користується сервісами нижчого рівня.

Завдання *рівня абстрагування від устаткування (hardware abstraction layer, HAL)* – приховати апаратні відмінності апаратної архітектури для потенційного перенесення системи з однієї платформи на

іншу. HAL надає вищележачим рівням апаратні пристрої в абстрактному вигляді, вільному від індивідуальних особливостей. Це дозволяє ізолювати ядро, драйвери і виконавчу систему ОС Windows від специфіки устаткування (наприклад, від відмінностей між материнськими платами).

Ядром звичайно називають всі компоненти ОС, що працюють у привілейованому режимі роботи процесора або в режимі ядра. Корпорація Microsoft називає *ядром (kernel)* компонент, що знаходиться в невивантажуваній пам'яті і містить низькорівневі функції операційної системи, такі, як диспетчеризація переривань і виключень, планування потоків й ін. Воно також надає набір процедур і базових об'єктів, використовуваних компонентами вищих рівнів.

Ядро і HAL є апаратно-залежними і написані на мовах Сі та асемблера. Верхні рівні написані на мові Сі й є машинно- незалежними.

Виконавча система (executive) забезпечує управління пам'яттю, процесами і потоками, захист, уведення-виведення і взаємодію між процесами. *Драйвери пристроїв* містять апаратно- залежний код і забезпечують трансляцію призначених для користувача викликів у запити, специфічні для конкретних пристроїв. Підсистема підтримки вікон і графіки реалізує функції графічного інтерфейсу користувача (GUI), відоміші як Win32-функції модулів USER і GDI.

У просторі користувача працюють різноманітні сервіси (аналоги демонів в Unix), керовані диспетчером сервісів, що виконують системні завдання. Деякі системні процеси (наприклад, обробка входу в систему) диспетчером сервісів не управляються і називаються фіксованими процесами підтримки системи. Призначені для користувача додатки (user applications) бувають п'яти типів: Win32, Windows 3.1, MS-DOS, POSIX і OS/2 1.2. Середовище для виконання процесів користувача надають три підсистеми оточення: Win32, POSIX і OS/2. Таким чином, додатки користувача не можуть викликати системні виклики ОС Windows безпосередньо, а змушені звертатися до DLL (динамічно зв'язаних бібліотек) підсистем.

Основні компоненти ОС Windows реалізовані в таких системних файлах, що знаходяться в каталозі system32:

- ntoskrnl.exe – виконавча система і ядро;
- ntdll.dll – внутрішні функції підтримки та інтерфейси диспетчера системних сервісів з функціями виконавчої системи;
- hal.dll – рівень абстрагування від устаткування;
- win32k.sys – частина підсистеми Win32, що працює в режимі ядра;
- kernel32.dll, advapi32.dll, user32.dll, gdi32.dll – основні dll

підсистеми Win32.

Взаємодія між додатком і операційною системою здійснюється за допомогою системних викликів (системних сервісів у термінології Microsoft). Проте додаток не може здійснювати системний виклик безпосередньо (більше того, системні виклики не документовані). Натомість додаток повинен скористатися програмним інтерфейсом ОС – Win32 API.

Win32 API (Application Programming Interface) – основний інтерфейс програмування в сімействі операційних систем Microsoft Windows. Функції Win32 API, наприклад, *CreateProcess* або *CreateFile*, – документовані підпрограми, які можуть викликатися і реалізуються Win32 підсистемою.

До складу Win32 підсистеми (див. рис. 1.2) входять: серверний процес підсистеми оточення csrss.exe, драйвер режиму ядра Win32k.sys, dll-модулі підсистем (kernel32.dll, advapi32.dll, user32.dll і gdi32.dll), що експортують Win32-функції і драйвери графічних пристроїв. У процесі еволюції структура підсистеми зазнала зміни. Наприклад, функції вікон і малювання з метою підвищення продуктивності були перенесені з серверного процесу, що працює в режимі користувача, в драйвер режиму ядра Win32k.sys. Проте ці та подібні зміни ніяк не відбилися на працездатності додатків, оскільки існуючі виклики Win32 API не змінюються з новими випусками системи Windows, хоча їх склад постійно поповнюється.

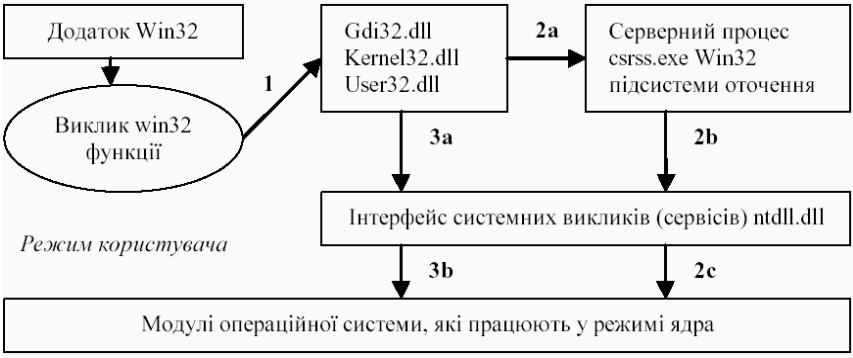

Рис. 1.2. Різні маршрути виконання викликів Win32 API

Додаток, зорієнтований на використання Win32 API, може працювати практично на всіх версіях Windows, незважаючи на те, що

самі системні виклики в різних системах різні. Таким шляхом корпорація Microsoft забезпечує спадкоємність своїх операційних систем.

При запуску процесу всі необхідні динамічні бібліотеки відображаються на його віртуальний адресний простір, а для швидкого виклику бібліотечної процедури використовується спеціальний вектор передачі.

При виклику додатком однієї з Win32-функцій dll-підсистем може виникнути одна з трьох ситуацій (рис. 1.2):

- функція повністю виконується всередині даною dll (крок 1);
- для виконання функції задіюється сервер csrss, для чого йому посилається повідомлення (крок 2a, за яким зазвичай ідуть кроки 2b і 2c);
- даний виклик транслюється в системний сервіс (системний виклик), який зазвичай обробляється в модулі ntdll.dll (кроки 3a і 3b). Наприклад Win32-функція ReadFile виконується за допомогою недокументованого сервісу *NtReadFile*.

Деякі функції (наприклад, CreateProcess) вимагають виконання обох останніх пунктів.

У перших версіях ОС Windows практично всі виклики Win32 API виконувалися, дотримуючись маршруту 2 (2a, 2b, 2c). Після того, як істотна частина коду системи для збільшення продуктивності була перенесена в ядро, виклики Win32 API, як правило, йдуть безпосередньо по 3-му (3a, 3b) шляху, минувши підсистему оточення Win32. У даний час лише невелика кількість викликів виконується по довгому 2-му маршруту.

Окрім перерахованих, найбільш важливих dll-бібліотек, у системному каталозі system32 є велика кількість інших dll-файлів. У даний час кількість викликів API складає кілька десятків тисяч.

Програма у ОС Windows пасивна. Після запуску вона очікує, коли їй приділить увагу операційна система. Операційна система робить це посиланням спеціально оформлених груп даних, які називаються *повідомленнями* (*Messages*). Повідомлення можуть бути різного типу, вони функціонують у системі достатньо хаотично, і програмний додаток не знає, якого типу повідомлення прийде наступним. Звідси випливає, що логіка побудови Windows-додатка повинна забезпечувати коректну і передбачувану роботу при поступленні повідомлень будь-якого типу. Для забезпечення нормального функціонування своєї програми програміст повинен вміти ефективно використовувати функції інтерфейсу прикладного програмування API (Application Program Inter- face) операційної системи.

Windows підтримує два типи програмних додатків:

- *віконний додаток* – будується на базі спеціального набору

функцій (API), які складають графічний інтерфейс користувача GUI (Graphic User Interface). Віконний додаток являє собою програму, яка виведення на екран виконує в графічному вигляді. Першим результатом роботи віконного додатка є відображення на екрані спеціального об'єкта – вікна. Після того як вікно відобразиться на екрані, вся робота додатка спрямована на те, щоб підтримувати його в актуальному стані;

- *невіконний додаток*, який також називають *консольним*, являє собою програму, що працює в текстовому режимі. Консольний додаток забезпечується спеціальними функціями Windows.

Різниця між двома типами додатків Windows визначається тим, з яким типом інформації вони працюють. Основний тип додатків у Windows – віконний.

Будь-який віконний Windows-додаток має типову структуру, основу якої складає так званий *каркасний додаток*. Цей додаток містить мінімально необхідний програмний код для забезпечення функціонування повноцінного Windows-додатка. Дослідження роботи каркасних додатків відображає основні особливості взаємодії програм з операційною системою Windows. Більше того, написавши та налагодивши один раз каркасний додаток, можливо використовувати його в подальшому, як основу для написання будь-якого іншого, значно більш складнішого Windows- додатка.

Мінімальний каркасний додаток Windows складається з трьох частин:

- головної функції;
- циклу обробки повідомлень;
- віконної функції.

Виконання будь-якого віконного Windows-додатка починається з *головної функції*. Вона містить код, що здійснює налаштування (ініціалізацію) додатка в середовищі операційної системи Windows. Видимим для користувача результатом роботи головної функції є поява на екрані графічного об'єкта у вигляді вікна. Останньою дією коду головної функції є створення *циклу обробки повідомлень*. Після його створення додаток стає пасивним і починає взаємодіяти з оточуючим середовищем за допомогою спеціальним чином оформлених даних – *повідомлень*. Обробка повідомлень, які надходять у додаток, здійснюється спеціальною функцією, яка називається *віконною*. Віконна функція унікальна тим, що може бути викликана тільки з операційної системи, а не з додатка, який її містить. Тіло віконної функції має певну структуру. Таким чином, Windows-додаток повинен складатися принаймні з трьох перелічених елементів.

Повідомлення в Win32 являє собою об’єкт особливої структури, що формується Windows. Формування та забезпечення доставки цього об’єкта у необхідне місце в системі дозволяють керувати як роботою самої системи Windows, так і роботою завантажених Windows-додатків. Ініціювати формування повідомлення може кілька джерел: користувач, самий додаток, система Windows, інші додатки. Саме наявність механізму повідомлень дозволяє Windows реалізувати багатозадачність, яка при роботі на одному процесорі є, звичайно ж, псевдомультизадачністю. Windows підтримує чергу повідомлень для кожного додатка. Запуск додатка автоматично передбачає формування для нього власної черги повідомлень, навіть якщо цей додаток і не буде нею використовуватися. Останнє малоймовірно, оскільки в цьому випадку в додатка не буде зв’язку з навколишнім середовищем.

Текст Windows API програми підготовляється на мові C (C++) і може регламентуватиме за допомогою MS Visual Studio. При створенні проекту в середовищі MS Visual Studio генерується вихідний текст, який є працюючою заготовкою програми, формує вікно з меню і вбудованим діалогом «About». Структура цієї заготовки повністю відповідає структурі найпростішої Windows API програми і містить такі основні частини і функції:

- підключення заголовних файлів;
- оголошення глобальних змінних;
- функцію опису та реєстрації класу вікна;
- функцію створення вікна програми;
- «віконну» функцію;
- головну функцію програми.

Директива препроцесора include присутня у кожному додатку. Вона створює копію файлу, ім'я якого в ній зазначено, і додає його в папку програми. Є два варіанти включення директиви include:

#include <ім'я файлу> - пошук цього файлу в кутових дужках буде вестися в папці, де розташовані бібліотечні файли;

#include “ім'я файлу” - пошук файлу в лапках здійснюватиметься в папці, де розташована дана програма.

За замовчуванням помічник створення проектів в MS Visual Studio включає директиву #include “stdafx.h", за якою включаються до проекту заголовні файли, які рідко змінюються або взагалі ніколи не міняються для даного проекту.

Глобальними змінними вважаються ті, які зберігають свій тип і значення у всій програмі, в тому числі і всередині функцій. Для цього перед описувачем додається специфікатор static, що означає, що об'єкт або

змінна розміщується в пам'яті при запуску додатку і видаляється з пам'яті тільки при завершенні роботи програми. Рекомендується до глобальних відносити ті змінні, які не змінюють свого значення протягом роботи або для скорочення кількості аргументів у функціях.

Додаток, що використовує графічний інтерфейс з користувачем, повинен мати, як мінімум, одне вікно - головне вікно програми. Для того, щоб створити вікно, необхідно спочатку зареєструвати клас вікна, Клас вікна визначає набір властивостей, характерних для всіх вікон, створюваних на базі даного класу. У Windows визначені стандартні класи вікон, використовувані для створення стандартних органів управління, Стандартні класи не реєструються додатком, і для них не визначається функція вікна, оскільки ці операції виконуються автоматично при завантаженні Windows. Функція опису та реєстрації класу вікна заповнює спеціальну структуру з ім'ям WNDCLASSEX, У цій структурі розміщується посилання на «віконну» функцію, яка буде обробляти всі події пов'язані з вікном, стиль оформлення, вид курсора і піктограми вікна і ряд інших параметрів. Після заповнення структури WNDCLASSEX, клас необхідно зареєструвати в системі. Це робиться за допомогою функції RegisterClassEx(), передавши їй як аргумент адресу структури WNDCLASSEX.

Функція створення вікна програми CreateWindow серед аргументів отримує геометричні розміри і координати вікна, стиль поведінки вікна, ім'я класу вікна та інші параметри.

Віконна функція викликається, коли в структуру mess потрапляє чергове повідомлення, вибране з вхідної черги. Віконна функція повинна проаналізувати код повідомлення і обробити його. З кожним вікном зв'язується своя віконна функція. Вона отримує 4 параметра. Перший параметр являє собою дескриптор вікна, якому призначено дане повідомлення.

Другий параметр визначає код повідомлення, що надійшло. Оскільки повідомлень багато, то за його кодом визначається і виконується конкретна гілка оператора вибору (switch) в тілі віконної функції. Однак реально в програмі потрібно обробляти не всі повідомлення. Для того щоб програміст міг включити у віконну функцію обробку тільки своїх повідомлень, в Windows передбачена функція DefWindowProc, яка обробляє практично всі повідомлення, за винятком повідомлення WM_DESTROY про знищення вікна. Тому в простому випадку у віконній функції необхідно реалізувати обробку тільки цього повідомлення.

Основною частиною головної функції програми (_tWinMain) є цикл обробки повідомлень, який складається з керуючої структури while. При

кожному проході циклу витягується чергове повідомлення з черги, за допомогою функції GetMessage(). Потім всі повідомлення від віртуальних клавіш транслюються в символьні повідомлення за допомогою функції TranslateMessage(). Після цього отримане повідомлення відсилається на обробку віконній процедурі за допомогою функції DispatchMessage().

Функція GetMessage() повертає ненульове значення, тому цикл не завершується до моменту завершення програми. При завершенні програми вікна надсилається повідомлення WM_QUIT, яке є єдиним сполученням, при отриманні якого функція GetMessage() повертає нуль, і цикл обробки повідомлень завершується, а код виходу з програми, що зберігається в елементі wParam структури MSG, повертається функцією _tWinMain.

```
int APIENTRY WinMain(HINSTANCE hInstance,
                             HINSTANCE hPrevInstance,
                             LPSTR     lpCmdLine,
                             int       nCmdShow)
{
        MSG msg;

        // Регистрация класса окна
        MyRegisterClass(hInstance);

        // Создание окна приложения
        if (!InitInstance (hInstance, nCmdShow))
        {
                return FALSE;
        }
        // Цикл обработки сообщений
        while (GetMessage(&msg, NULL, 0, 0))
        {
                        TranslateMessage(&msg);
                        DispatchMessage(&msg);
        }
        return msg.wParam;
}
```

Завдання.

Розробити програму на базі функцій Win32 API, яка після запуску на виконання виводить у вікно програми два рядки: наприклад, ім'я комп'ютера та ім'я користувача.

Для створення проекту виконується наступна послідовність дій:

- відкрити Visual Studio, якщо це не було зроблено раніше;
- активуйте діалог завдання нового проекту, вид якого показаний на рисунку 1.3;

- в списку шаблонів додатків вибрати Класичний додаток Windows;
- потім ввести ім'я проекту та задати шлях до папки, в якій будуть розташовуватися папки та файли проекту.

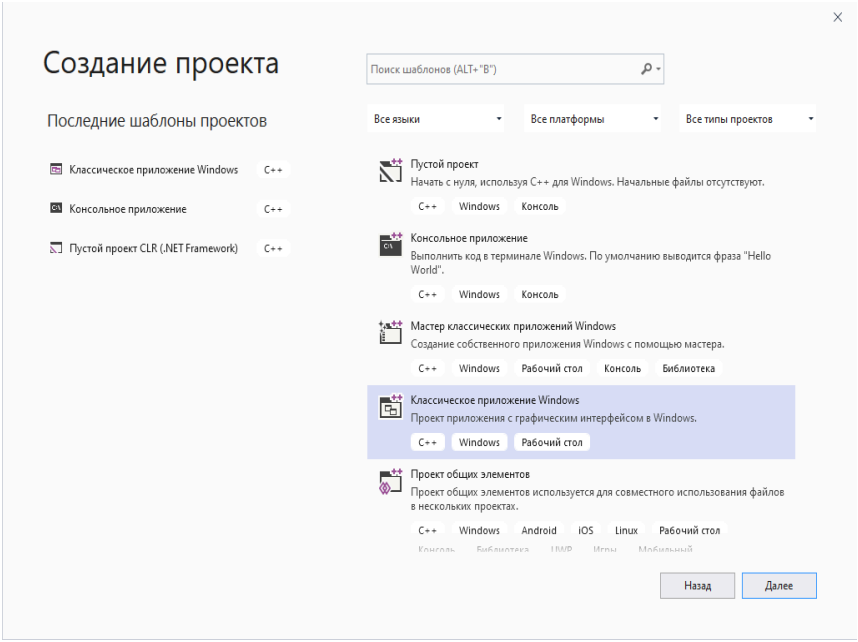

Рис. 1.3. Діалог надання нового проекту

По закінченні налаштувань - натиснути кнопку «ОК».

У результаті створення проекту буде генерований вихідний текст, який містить всі основні функції і додані ресурси.

При створенні програми вводяться глобальні змінні для дескриптора додатки, для завдання імені класу і заголовка вікна.

```
HINSTANCE hInst = NULL; / / Дескриптор додатку
TCHAR szTitle [MAX_LOADSTRING]; / / ім'я класу
TCHAR szWindowQass [MAX_LOADSTRING];//заголовок вікна
```

Функція опису та реєстрації класу вікна генерується при створенні програми в Visual Studio. Завдання полягає у визначенні значень і заповненні спеціальної структури з ім'ям *WNDCLASSEX*.

```
ATOM MyRegisterClass(HINSTANCE hInstance)
{
  WNDCLASSEX wcex;
  wcex.cbSize = sizeof(WNDCLASSEX);
  wcex.style            = CS_HREDRAW | CS_VREDRAW;    //
стиль окна
  wcex.lpfnWndProc      =    (WNDPROC)WndProc;   //   оконная
процедура
  wcex.cbClsExtra= 0;
wcex.cbWndExtra= 0;
  wcex.hInstance        = hInstance;          //   указатель
приложения
  wcex.hIcon            = MyIcon1;            // определение
иконки
  wcex.hCursor          = LoadCursor(NULL, IDC_ARROW);
// определение курсора
  wcex.hbrBackground    = GetSysColorBrush(COLOR_BTNFACE);
// установка фона
  wcex.lpszMenuName     = NULL;        // определение меню
  wcex.lpszClassName    = szWindowClass;       // имя класса
  wcex.hIconSm          = NULL;
  return RegisterClassEx(&wcex); // регистрация класса окна
}
```

Таблиця 1.1 - Прапори стилів класу вікна

Прапори стилів класу	Значення	Опис
CS_VREDRAW	0x0001	Перемалювати вікно при зміні висоти вікна
CS_HREDRAW	0x0002	Перемалювати вікно при зміні ширини вікна
CS_DBLCLKS	0x0008	Посилати повідомлення віконній функції при подвійному натисканні мишею, якщо курсор знаходиться в межах вікна
CS_OWNDC	0x0020	Для кожного вікна класу виділяється власний контекст
CS_CLASSDC	0x0040	Один і той же контекст пристрою розділяється всіма вікнами цього класу
CS_PARENTDC	0x0080	Дочірні вікна успадковують контекст батьківського вікна
CS_NOCLOSE	0x0200	Прибрати команду « Close » з системного меню
CS_SAVEBITS	0x0800	Зберігати частину області екрану, закриту вікном, як bitmap, при видаленні відновлювати перекриту область

CS_BYTEALIGNCLIENT	0x1000	Вирівнює кордон робочої області вікна (в горизонтальному напрямку) таким чином, щоб для відображення рядка вимагалося ціле число байтів
CS BYTEALIGNWINDOW	0x2000	Те ж, але дія зачіпає все вікно
CS_GLOBALCLASS	0x4000	Дозволяється створювати клас, що не залежить від поточного hInstance

Таблиця 1.2-Коди вбудованих «іконок»

Код	Значення
IDI APPLICATION	стандартна піктограма для додатка
IDI HAND	"стоп" (хрестик на червоному тлі)
IDI QUESTION	"знак питання" в синьому колі
IDI EXCLAMATION	знак оклику в трикутнику
IDI ASTERISK	«I" в синьому колі
IDI WINLOGO	стандартна піктограма для додатка
IDI WARNING	знак оклику в трикутнику
IDI ERROR	хрестик у червоному колі
IDI_INFORMATION	«I» в синьому колі

Таблиця 1.3- Коди вбудованих курсорів

Код	Значення
IDC ARROW	стрілка
IDC IBEAM	вертикальна риса
IDC WAIT	"пісочний годинник" або інше
IDC CROSS	хрестик
IDC UPARROW	вертикальна стрілка
IDC SIZE	4-стрілки, що вказують в різні сторони
IDC ICON	курсор, використовуваний при drag & drop.
IDC SIZENWSE	стрілка "північний захід / південний схід"
IDC SIZENESW	стрілка "північний схід / південний захід"
IDC SIZEWE	двонаправлена стрілка "схід-захід"
IDC_SIZENS	двонаправлена стрілка "північ-південь"

Таблиця 1.4- Коди вбудованих пензлів зафарбування фону

Код	Значення
WHITE BRUSH	Білий фон
LTGRAY BRUSH	Світло-сірий фон
GRAY BRUSH	Сірий фон
DKGRAY BRUSH	Темно-сірий фон
BLACK BRUSH	Чорний фон
NULL BRUSH	Відсутність фону (прозорий)
HOLLOW BRUSH	Відсутність фону (прозорий)

Для створення вікна програми використовується функція CreateWindow, яка в якості аргументів отримує геометричні розміри і координати вікна, стиль поведінки вікна, ім'я класу вікна та інші параметри.

```
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
   HWND hWnd;
      hInst = hInstance; // сохраняет указатель приложения в
переменной hInst
   hWnd=CreateWindow(szWindowClass, // имя класса окна
 szTitle,   // имя приложения
 WS_OVERLAPPEDWINDOW, // стиль окна
 CW_USEDEFAULT,// положение по X
 CW_USEDEFAULT,        // положение по Y
 CW_USEDEFAULT,    // размер по X
 CW_USEDEFAULT,    // размер по Y
 NULL,  // описатель родительского окна
 NULL,       // описатель меню окна
 hInstance,  // указатель приложения
 NULL);     // параметры создания.

   if (!hWnd) // Если окно не создалось, функция возвращает FALSE
   {
      return FALSE;
   }
   ShowWindow(hWnd, nCmdShow);// Показать окно
   UpdateWindow(hWnd);                // Обновить окно
   return TRUE;               //Успешное завершение функции
}
```

Таблиця 1.5-Коди прапорів стилю вікна

Код	Значення
WS BORDER	Створення вікна з рамкою.
WS CAPTION	Додавання заголовка до вікна з рамкою.
WS_CHILD	Створення дочірнього вікна. Не викор. з вікнами WS POPUP.
WS_CHILDWINDOW	Створення дочірнього вікна стилю WS CHILD.
WS_CLIPCHILDREN	Використовується при створ. нового вікна; забороняє малювання нового вікна в області, зайнятої будь-яким дочірнім вікном.
WS_CLIPSIBLINGS	Вик. тільки зі стилем WS_CHILD. Ці вікна викл. області, зайняті іншими дочірніми вікнами, при перемальовуванні.
WS DISABLED	Створення спочатку неактивного вікна.
WS_DLGFRAME	Створення вікна з подвійною рамкою без заголовку.
WS_GROUP	Вик. тільки в діалогових вікнах. Цей стиль вказ. для 1- ого елемента керування в групі елементів. Можна переходити від одного елемента керування до іншого за допомогою клавіш стрілок.
WS_HSCROLL	Створення вікна з горизонтальною смугою прокрутки.
WS MAXIMIZE	Створення вікна максимального розміру.
WS_MAXIMIZEBOX	Створення вікна, яке має кнопку максимізації.
WS MINIMIZE	Створення вікна мінімального розміру.
WS_MINIMIZEBOX	Створення вікна, яке має кнопку мінімізації.
WS OVERLAPPED	Створення вікна, що буде перекриватись.
WS_OVERLAPPEDWINDOW	Використовуються стилі WS OVERLAPPED, WS THICKFRAME і WS SYSMENU.
WS_POPUP	Створення тимчасового вікна. Не використовується з вікнами стилю WS CHILD.
WS_POPUPWINDOW	Для створення тимчасового вікна, що викор. стилі WS BORDER, WS POPUP і WS SYSMENU.
W S_SY SMENU	Створення вікна з систем. меню. Для дочірнього вікна - кнопка " закрити вікно". Стиль WS_SYSMENU використовується тільки для вікон, що мають заголовок.

WS_TABSTOP	Вик. тільки в діалогових вікнах. За допомогою натискань Tab можна переміщатися між елементами управління зі стилем WS TABSTOP.
W S_THICKFRAME	Створення вікна з товстою рамкою. Рамка використовується для зміни розмірів вікна.
WS_VISIBLE	Створення вікна, видимого відразу після створення. Використовується для вікон, що перекриваються і тимчасових вікон.
WS_VSCROLL	Створення вікна з вертикальною смугою прокрутки.

Visual Studio при створенні проекту додатка генерує текст віконної процедури з ім'ям WndProc, яка містить обробку повідомлень WM_COMMAND, WM_PAINT, WM_DESTROY.

Windows стежить за зміною розташування вікон і при зміні розмірів вікна, його положення на екрані або часткового перекриття іншим вікном посилає функції вікна повідомлення WM_PAINT. Щоб при запуску програми виконати вивід заданої інформації в її вікно, необхідно оператори, що формують цю інформацію, помістити в цю частину віконної процедури. За умовою задачі необхідно запросити у системи ім'я користувача та ім'я комп'ютера і вивести їх по центру екрана. Для зберігання і виведення інформації буде використовуватися буферний масив: char infoBuf [MAX_COMPUTERNAME_LENGTH+1] і змінна DWORD bufSize = MAX_COMPUTERNAME_LENGTH+1, для зберігання кількості символів в буфері. Відповідно до документації, виконання функції GetComputerName завершиться невдачею, якщо розмір буфера вхідних даних менше, ніж величина константи MAX_COMPUTERNAME_LENGTH+ 1. Для формування рядка, що виводиться на екран, вводиться буферний масив char outBuf [1024].

Обробка повідомлення WM_PAINT завжди починається з виклику функції BeginPaint:

```
hdc = BeginPaint(hWnd, &ps);
```

де hWnd - дескриптор вікна, отриманий через аргумент віконної процедури; ps - адреса структури типу PAINTSTRUCT. Поля цієї структури, що заповнюються в результаті виконання функції BeginPaint, надалі використовуються операційною системою. hdc - дескриптор контексту пристрою.

Контекст пристрою (device context) описує фізичний пристрій виведення інформації, наприклад дисплей або принтер. Контекст

пристрою - це внутрішня структура даних, яка зберігає часто використовувані графічні атрибути, такі як колір фону, перо, пензель, шрифт і їм подібні параметри. Ці атрибути використовуються при виклику всіх малюють функцій, які отримують дескриптор hdc як параметр.

Далі слід викликати функцію GetClientRect, яка призначена для отримання розмірів клієнтської області вікна:

```
GetClientRect (hWnd, & rect);
```

де змінна rect повинна мати тип структури RECT (прямокутник (rectangle)). Структура RECT має такі поля: left, top - координати лівого верхнього кута; right, bottom - координати правого нижнього кута прямокутника, що обмежує клієнтську область вікна.

Після виконання функції GetClientRect поля left і top завжди отримують нульове значення, а поля right і bottom містять ширину і висоту клієнтської області у точках. Інформація про розміри клієнтської області, збережена у змінній rect, буде використана у функції промальовування тексту DrawText. Для того, щоб вивести інформацію в два рядки, треба знати висоту шрифту і міжрядкову відстань. Щоб використовувати функцію GetTextMetrics, спочатку потрібно визначити змінну типу структура: TEXTMETRIC tm, викликати функцію і розрахувати відстань між рядками тексту. Для зберігання відстані між рядками тексту буде використовуватися змінна HeightString:

```
int HeightString;
…
GetTextMetrics (hDC, & tm);
HeightString = tm.tmHeight+ tm.tmExternalLeading;
```

Функція GetComputerName використовується для отримання поточного імені комп'ютера:

```
GetComputerName(infoBuf, &bufSize);
sprintf(outbuf, "Имя компьютера = %s", infoBuf);
DrawText(hDC, outBuf, -1, &rectt, DT_SINGLELINE | DT_LEFT);
```

Функція sprintf переводить в текстовий формат дані, перераховані в списку виведення, і з'єднує їх в один рядок. Відповідність між елементами рядка формату і списком змінних встановлюється однозначно зліва направо. Для використання цієї функції треба додати директиву #include <stdio.h>.

Таблиця 1.6-Форматні коди

Код	Опис
%c	символ
%d	Десяткове ціле число зі знаком
%i	Десяткове ціле число зі знаком
%e	Експоненціальне представлення числа (у вигляді мантиси і порядку)
%f	Десяткове число з плаваючою точкою
% g	Використовує більш короткий з форматів %e або %f
%o	Вісімкове число без знаку
%s	символьний рядок
%u	Десяткове ціле число без знаку
%x	Шістнадцяткове без знаку (малі літери)
%p	виводить покажчик
%%	Виводить знак відсотка

Призначення параметрів функції: outBuf - покажчик на вихідний рядок, "Ім'я комп'ютера =% s" - рядок с кодами форматування, infoBuf - покажчик на вхідний рядок. Ознака форматного коду - знак відсотка (%). Список форматних кодів дано в таблиці 1.6.

Вивід інформації виконується за допомогою функції DrawText, що має наступний формат:

```
DrawText (
hdc, / / дескриптор контексту пристрою
outBuf, / / покажчик на вихідний рядок
-1, / / Довжина тексту. Якщо -1, то система сама визначить
/ / довжину рядка
& rect, / / Покажчик на прямокутник, що обмежує вікно
DT_SINGLELINE | DT_LEFT / / прапори форматування тексту );
```

Функція виводить текст з рядка outBuf в прямокутну область, задану структурою типу RECT, використовуючи метод форматування, заданий прапорами форматування. На місці останнього параметра функції заданий набір прапорів DT_SINGLELINE | DT_LEFT.

Прапори форматування тексту:

- DT_BOTTOM - Виведений текст вирівнюється по нижньому краю.
- DT_CENTER - Виведений текст вирівнюється по правому краю.
- DT_END_ELLIPSIS - Якщо рядок не вміщується в rect, його

кінець замінюється на три точки.

- DT_LEFT - Виведений текст вирівнюється по лівому краю.
- DT_PATH_ELLIPSIS - Використовується з шляхами, що містять символ '\ '. Якщо рядок не вміщується в rect, то частина його з середини замінюється на три точки.
- DT_RIGHT - Виведений текст вирівнюється по центру.
- DT_SINGLELINE - Текст виводиться в один рядок, при цьому ігноруються символи переходу на початок рядку і повернення каретки.
- DT_TOP - Виведений текст вирівнюється по верхньому краю.
- DT_WORDBREAK - Текст розбивається на рядки так, щоб він помістився по ширині в прямокутнику rect. Символи повернення каретки і переведення рядка також призводять до переходу на новий рядок.

Для того, щоб рядок з ім'ям користувача вивести під рядком з ім'ям комп'ютера, треба змістити положення верхньої кромки би прямокутника на значення висоти рядку:

```
rect.top = rect.top+ HeightString;
bufSize = MAX_COMPUTERNAME_LENGTH+ 1;
GetUserName (infoBuf, & bufSize);
sprintf (outBuf, " Ім'я користувача = % s ", infoBuf);
DrawText (hdc, outBuf, -1, & rect, DT_SINGLELINE | DT_LEFT);
```

Перед зверненням до функції GetUserName обов'язково треба встановити значення розміру буфера bufSize не менше MAX_COMPUTERNAME_LENGTH+ 1.

Завдання на самостійну роботу

Для всіх наступних завдань потрібно розробити додаток на базі функцій Win32 API, в якому:

1) після запуску на виконання у вікно програми виводяться два рядки: ім'я системної директорії і директорії, в якій знаходиться Windows;
2) після запуску на виконання у вікно програми виводяться два рядки: ім'я директорії, в якій знаходяться тимчасові файли, й ім'я користувача;
3) після запуску на виконання у вікно програми виводяться два рядки: ім'я комп'ютера і номер версії Windows;
4) після запуску на виконання у вікно програми виводяться два рядки: розміри екрану - ширина і висота;
5) після запуску на виконання у вікно програми виводяться два рядки системні параметри клавіатури: затримка часу початку генерації

символів і швидкість повторення символів при утриманні клавіші.

Контрольні питання

1.Яке призначення віконної процедури програми?

2.Опишіть основні етапи створення вікна програми.

3.Яким чином відбувається обробка повідомлень в додатках Windows?

4.Поясніть особливості формування повідомлення WM_PAINT.

5. Які системні повідомлення ви знаєте?

Лабораторна робота №2

Тема. Органи управління додатком.

Мета. Отримання навичок використання елементів управління загального користування: кнопок, перемикачів, прапорців, рамок і написів.

Теоретичні відомості

Діалогові вікна бувають модальні (modal) і немодальні (modeless). Модальне діалогове вікно - очікує виконання деякої дії з боку користувача, перш ніж програма зможе продовжити своє виконання. Немодального діалогове вікно не зупиняє виконання програми. Воно може отримувати і втрачати фокус вводу. Всі версії Windows підтримують так звані базові елементи управління. В системі використовується бібліотека елементів управління загального користування (common control library).

Таблиця 2.1- Базові елементи управління

Елемент управління	Опис	Ім'я класу
Рисунок (Picture control)	Елемент управління, що відображає порожню прямокутну рамку, закрашеної прямокутну область або растровий образ	STATIC
Напис (Static text)	Текстовий рядок. Зазвичай використовується як мітка (пояснюється напис) поряд з полем вводу або елементом управління іншого типу. Може застосовуватися як самостійна інформаційна напис	STATIC
Рамка (Group box)	Прямокутна рамка з написом, використовувана для групування набору пов'язаних елементів управління	BUTTON
Кнопка (Button)	Елемент, який користувач « натискає », щоб виконати будь-яку дію	BUTTON
Прапорець (Check box)	Елемент, який може бути або встановлений, або скинутий для вибору або скасування опції, яка не пов'язана з іншими опціями	BUTTON
Перемикач (Radio button)	Елемент, що використовується для вибору однієї з групи взаємовиключних опцій. У групі перемикачів може бути вибраний тільки один з них	BUTTON
Список (List box)	Прямокутне вікно зі списком елементів (рядків), з якого користувач може вибрати будь-який елемент	LISTBOX

Вікно редагування (Edit box)	Прямокутне вікно для введення тексту з клавіатури. Елемент надає певні засоби редагування тексту	EDIT
Комбінований список (Combo box)	Елемент, який об'єднує список з вікном редагування	COMBOBOX
Смуга прокрутки (Scroll bar)	Елемент управління лінійкою прокрутки	SCROLLBAR

Кожен елемент управління, описаний в шаблоні діалогу, реалізується Windows у вигляді вікна відповідного класу. Як всяке вікно, воно ідентифікується своїм дескриптором типу HWND і створюється за допомогою функції CreateWindow. Додаток може взаємодіяти з елементами управління за допомогою функції SendMessage. Відмінні риси органів управління:

- для них вже описані класи вікон;
- всі вони дочірні вікна (стилю ws_child);
- для них описані додаткові стилі і списки оброблюваних і одержуваних повідомлень;
- для них майже завжди потрібно описувати ідентифікатори.

Для створення органів управління викликається функція CreateWindow, для вибору аргументів якій існують наступні рекомендації:

- в якості lpClassName вказують ім'я класу BUTTON, COMBOBOX, EDIT, LISTBOX, MDICLIENT, SCROLLBAR, STATIC;
- аргумент lpWindowName визначає текст на кнопці;
- аргумент dwStyle задає стиль кнопки - комбінація констант WS_CHILD, WS_VISIBLE. Винятком є константи BS_LEFTTEXT, BS_RIGHTBUTTON;
- значення hWndParent - дескриптор вікна-батька (вказують обов'язково);
- значення hMenu має дорівнювати ідентифікатору кнопки;
- для органів управління в якості lpParam вказують NULL. Приклади створення і використання елементів управління будуть дані при вирішенні типової задачі.

Завдання.

Розробити з використанням функцій Windows API програму, в робочому вікні якої створюється група з трьох прапорців для вибору прямокутника, який потрібно зафарбовувати, група з двох перемикачів для вибору синього або червоного кольору зафарбовування прямокутників,

кнопки для скидання перемикачів кольору і прапорців вибору прямокутника. Якщо всі перемикачі і / або прапорці скинуті, то прямокутники повинні зафарбовуватися в білий колір.

Насамперед розробляється ескіз робочого вікна програми, який показаний на рисунку 2.1. Створити проект програми. Параметри вікна програми беруться такими: ширина вікна - 400, висота - 300.

Рисунок 2.1 - Ескіз робочого вікна

Необхідно визначити ідентифікатори кнопок. За допомогою цих ідентифікаторів функція вікна-батька буде "впізнавати" кнопки:

```
#define IDB_BOX1 37710 // первый флажок
#define IDB_BOX2 37720 // второй флажок
#define IDB_BOX3 37730 // третий флажок
#define IDB_REDBACK 37740//верхний переключатель ("Красный")
#define IDB_BLUEBACK 37750//нижний переключатель ("Синий")
#define IDB_CLEARBOXES 37770 // верхняя нажимная кнопка
#define IDB_CLEARBACK 37780 // нижняя нажимная кнопка
#define  IDB_CHANGEBGCOLOR  37790  //кнопка  изменения  цвета
фона
 //описать кисти для закрашивания
HBRUSH  hbrRed,  hbrBlue,  hbrWhite,  hBlack,  hbr;  //hbr  -
значение выбранной кисти
BOOL fc1 = FALSE, fc2 = FALSE, fc3 = FALSE; //флажки выбора
рамки для закраски
RECT rc, rc1, rc2, rc3; //задать структуры для прорисовки
рамок
```

Доцільно задати опис функції з ім'ям InitTool для ініціалізації структур даних пензлів і прямокутників (рамок).

```
      void InitTool()
      {
            rc1.left = 40;   rc1.top = 140;   rc1.right = 80;
rc1.bottom = 180;
            rc2.left = 100;  rc2.top = 140;   rc2.right = 140;
rc2.bottom = 180;
            rc3.left = 160;  rc3.top = 140;   rc3.right = 200;
rc3.bottom = 180;
           hbrRed = CreateSolidBrush(RGB(255, 0, 0)); //красная
кисть
        hBlack = CreateSolidBrush(RGB(0, 0, 0)); // черная кисть
        hbrBlue = CreateSolidBrush(RGB(0, 0, 255)); //синяя кисть
         hbrWhite = CreateSolidBrush(RGB(255, 255, 255)); //белая
кисть
        hbr = hbrWhite; // текущая кисть
      }
```

Ця функція повинна викликатися тільки один раз при запуску програми. Тому її виклик можна пов'язати з подією WM_CREATE.

Другою допоміжною функцією є функція зафарбовування рамок заданим кольором:

```
      void Draw(HDC hdc) //значения fc1,fc2,fc3 опред. состоянием
флага
      {
          if (fc1)
                FillRect(hdc, &rc1, hbr); // hbr - дескриптор тек.
кисти
          else
                        FillRect(hdc,  &rc1,  hbrWhite);//закраска
прямоугольника 1
          FrameRect(hdc, &rc1, hBlack);//контур прямоугольника 1
          if (fc2)
              FillRect(hdc, &rc2, hbr);
          else
                        FillRect(hdc,  &rc2,  hbrWhite);//закраска
прямоугольника 2
          FrameRect(hdc, &rc2, hBlack);//контур прямоугольника 2
          if (fc3)
              FillRect(hdc, &rc3, hbr);
          else
                      FillRect(hdc,  &rc3,  hbrWhite);  //закраска
прямоугольника 3
          FrameRect(hdc, &rc3, hBlack);//контур прямоугольника 3
```

```
}
```

У віконну функцію додатка вставити обробку події WM_CREATE, в якому:

- викликати функцію визначення пензлів і рамок: InitTool ();
- створити кнопки очищення кольору і скидання прапорців:

```
hCLEARBOX = CreateWindow(L"BUTTON", L"Сброс флагов",
            WS_CHILD | WS_VISIBLE | BS_DEFPUSHBUTTON,
              270, 40, 110, 24, hWnd, (HMENU)IDB_CLEARBOXES,
hInst, NULL);
               //аналогично для кнопки очистки цвета.
```

- задати рамку навколо перемикачів:

```
hGroup = CreateWindow(L"button", L"Цвет фона",
            WS_CHILD | WS_VISIBLE | BS_GROUPBOX,
            140, 30, 120, 85, hWnd, NULL, hInst, NULL);
```

- задати перемикач вибору червоного кольору:

```
hREDBACK = CreateWindow(L"button", L"Красный",
            WS_CHILD | WS_VISIBLE | BS_AUTOCHECKBOX,
            144, 52, 100, 24, hWnd, (HMENU)IDB_REDBACK,
hInst, NULL);
//аналогично задается переключатель выбора синего цвета.
```

- задати прапор вибору 1-й рамки для зафарбовування:

```
hBOX1 = CreateWindow(L"button", L"Первая рамка",
            WS_CHILD | WS_VISIBLE | BS_AUTORADIOBUTTON,
            10, 30, 120, 24, hWnd, (HMENU)IDB_BOX1, hInst,
NULL);
//аналогично задаются флаги выбора 2-й и 3-й рамки.
```

- викликати перерисовку вікна:

```
InvalidateRect(hWnd, NULL, TRUE);
```

Для того, щоб при змінах робочого вікна автоматично перемальовувалися і зафарбовувати рамки, необхідно у віконній функції в гілці обробки події WM_PAINT вставити виклик функції Draw (hdc).

Для обробки натискань органів управління у віконну функцію необхідно вставити додаткові гілки:

```
      LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM
wParam, LPARAM lParam)
      {…
               switch (wmId)
               {
               case IDB_CLEARBACK:
                   //сброс переключателей
                   hbr = hbrWhite;
                                SendMessageA(hREDBACK, BM_SETCHECK,
(WPARAM)BST_UNCHECKED, NULL);
                               SendMessageA(hBLUEBACK, BM_SETCHECK,
(WPARAM)BST_UNCHECKED, NULL);
                   InvalidateRect(hWnd, NULL, TRUE);
                   break;
               case IDB_CLEARBOXES:
                   // сброс флажков:
                   fc1 = FALSE;
                   fc2 = FALSE;
                   fc3 = FALSE;
                                    SendMessageA(hBOX1, BM_SETCHECK,
(WPARAM)BST_UNCHECKED, NULL);
                                    SendMessageA(hBOX2, BM_SETCHECK,
(WPARAM)BST_UNCHECKED, NULL);
                                    SendMessageA(hBOX3, BM_SETCHECK,
(WPARAM)BST_UNCHECKED, NULL);
                   InvalidateRect(hWnd, NULL, TRUE);
                   break;
      case IDB_BLUEBACK:      //  включить  переключатель  синего
цвета:
                   hbr = hbrBlue;
                   InvalidateRect(hWnd, NULL, TRUE);
                   break;
        // аналогично для переключателя красного цвета
      case IDB_BOX1:
                   //- щелчок на флажке 1-й рамки

                   fc1 = !fc1;
                   InvalidateRect(hWnd, NULL, TRUE);
                   break;      //аналогично  обрабатываются  щелчки
на других флажках
```

Завдання на самостійну роботу

Для всіх наступних завдань використовується умова типової задачі, в якій змінено вимоги:

1) вибір рамки для зафарбовування виконувати за допомогою перемикачів;

2) вибір кольору виконувати за допомогою прапорців;

3) ввести кнопку виклику діалогу вибору кольору для фону вікна.

Контрольні питання

1) Як у процесі роботи можна змінити написи на кнопках?
2) Перерахуйте відмінні риси органів управління.
3) Які елементи управління допускаються в робочому вікні програми?

Лабораторна робота №3
Тема. Робота зі списками.

Мета. Отримання навичок розробки додатків зі списками та закріплення навичок формування обмінів між вікнами за допомогою функції посилки повідомлень SendMessage.

Теоретичні відомості

Список (Listbox) являє собою прямокутне вікно, в якому відображається набір елементів, з яких користувач може робити вибір. Елементи списку можуть бути представлені рядками, растровими образами або комбінацією тексту та зображення. Якщо розміри вікна не дозволяють показати всі елементи списку, то Listbox створює смугу прокрутки.

Розрізняють списки з одиничним вибором, в яких користувач може вибрати тільки один пункт списку, і списки з множинним вибором, що допускають вибір більше одного пункту списку. Windows показує вибраний елемент у списку, з інвертованим кольором тексту і фону для символів.

У списку з одиничним вибором користувач може вибрати пункт списку клацанням миші або натисканням клавіші пробілу після отримання списком фокусу введення. Клавіші управління курсором переміщують як курсор, так і поточну вибірку. У списку з множинним вибором перше клацання на пункті вибирає його, а наступне клацання скасовує попередню операцію.

Завдання.

Розробити додаток для ведення відомості оцінок у формі списку.

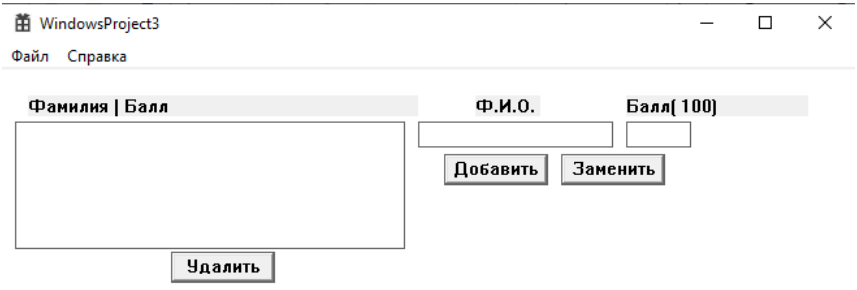

Рисунок 3.1 - Ескіз робочого вікна

Структура рядку:<П.І.Б.> <Бали (100)>. Забезпечити: додавання, видалення і заміну рядків списку.

Насамперед розробляється ескіз робочого вікна програми, який показаний на рисунку 3.1. Створити проект програми. Параметри вікна програми: ширина вікна - 640, висота - 240.

На початок вихідного тексту додати директиви:

```
#include <stdio.h> //стандартный ввод / вывод
```

Вставити визначення ідентифікаторів кнопок і списку:

```
#define ID_LIST 37780
#define IDB_ADD 37781
#define IDB_DEL 37782
#define IDB_RPL 37783
```

Оголосити змінні:

```
HWND lb, //дескриптор списка
btn_Add, //дескриптор кнопки "Добавить"
btn_Del, //дескриптор кнопки "Удалить"
btn_Rpl, //дескриптор кнопки "Заменить"
hwFIO, //дескриптор поля редактирования Ф.И.О.
hwBal100; //дескриптор поля редактирования балла
char FIO[20], Bal100[3], res[30]; //массивы для хранения
Ф.И. О., балла и строки списка
DWORD len = 0; //переменная для задания длины строки
int indx = 0; //переменная для задания номера выделенной в
списке строки
HFONT hfnt; //дескриптор шрифта
```

У віконну процедуру (WndProc) вставити обробку події створення вікна програми, де помістити функції створення елементів управління і допоміжних об'єктів:

```
case WM_CREATE:
    {
        CreateWindow(L"STATIC", L"Фамилия | Балл", WS_CHILD
| WS_VISIBLE, 20, 20, 300, 16, hWnd, NULL, hInst, NULL);
        //Создание окна списка. LBS_NOTIFY- разрешение
обработки событий
        lb = CreateWindow(L"Listbox", NULL, WS_CHILD |
WS_VISIBLE | LBS_NOTIFY | WS_BORDER, 10, 40, 300, 100, hWnd,
(HMENU)ID_LIST, hInst, NULL);
        //Дескриптор моноширинного шрифта
        hfnt=(HFONT)GetStockObject(SYSTEM_FIXED_FONT);
```

```
            //Установка шрифта
            SendMessage(lb, WM_SETFONT, (WPARAM)hfnt,
MAKELPARAM(true, 0));
            //Создание кнопки "Удалить"
            btn_Del = CreateWindow(L"BUTTON", L"Удалить",
WS_CHILD | WS_VISIBLE | WS_BORDER, 130, 140, 80, 24, hWnd,
(HMENU)IDB_DEL, hInst, NULL);
                //создание кнопки "Добавить"
                btn_Add = CreateWindow(L"BUTTON", L"Добавить",
WS_CHILD | WS_VISIBLE | WS_BORDER, 340, 65, 80, 24, hWnd,
(HMENU)IDB_ADD, hInst, NULL);
                //создание кнопки "Заменить"
                btn_Rpl = CreateWindow(L"BUTTON", L"Заменить",
WS_CHILD | WS_VISIBLE | WS_BORDER, 430, 65, 80, 24, hWnd,
(HMENU)IDB_RPL, hInst, NULL);
                //Надпись над полем ввода ФИО
                CreateWindow(L"STATIC", L"Ф.И.О.", WS_CHILD |
WS_VISIBLE, 364, 20, 50, 16, hWnd, NULL, hInst, NULL);
                //Надпись над полем ввода балла
                CreateWindow(L"STATIC", L"Балл( 100)", WS_CHILD
| WS_VISIBLE, 480, 20, 140, 16, hWnd, NULL, hInst, NULL);
                //создание поля редактирования Ф.И.О.
                hwFIO = CreateWindow(L"EDIT", L"", WS_CHILD |
WS_VISIBLE | WS_BORDER | ES_LEFT | ES_AUTOHSCROLL, 320, 40, 150,
20, hWnd, NULL, hInst, NULL);
                    //Ограничение количества символов <=20
                SendMessage(hwFIO, EM_SETLIMITTEXT, 20, 0);
                    //создание поля редактирования баллов
                hwBal100 = CreateWindow(L"EDIT", L"", WS_CHILD |
WS_VISIBLE | WS_BORDER | ES_LEFT | ES_AUTOHSCROLL, 480, 40, 50,
20, hWnd, NULL, hInst, NULL);
                        //Ограничение к-ва символов <=2
                SendMessage(hwBal100, EM_SETLIMITTEXT, 2, 0);
        }
        break;
```

У віконній процедурі (WndProc) у частині аналізу коду команди додати оператори:

```
    switch (wmId)
        {
            case IDB_ADD: //обработка нажатия кнопки 'Добавить"
            //Получение текста из поля ФИО
            len=SendMessage(hwFIO, WM_GETTEXTLENGTH, 0, 0);
            SendMessage(hwFIO,    WM_GETTEXT,    (WPARAM)len+1,
            (LPARAM)FIO);
            //Получение текста из поля Балл(100)
            len=SendMessage(hwBal100, WM_GETTEXTLENGTH, 0, 0);
             SendMessage(hwBal100, WM_GETTEXT, (WPARAM)len+1,
    (LPARAM)Bal100);
```

```
      //Объединение текстов в одну строку и добавление к списку
      sprintf(res, "%-20s\|%-3s",FIO, Bal100);
      SendMessage(lb, LB_ADDSTRING, 0, (LPARAM)res);
                  break;
      case IDB_DEL://обработка нажатия кнопки "Удалить"
      //получить номер выделенной строки списка
      indx = SendMessage(lb, LB_GETCURSEL, 0, 0);
      //если номер >= 0, то удалить строку из списка
      if    (indx    >=    0)    SendMessage(lb,    LB_DELETESTRING,
(WPARAM)indx, 0);
                  break;
      case IDB_RPL://обработка нажатия кнопки "Заменить"
      //получить номер выделенной строки списка
      indx = SendMessage(lb, LB_GETCURSEL, 0, 0);
      //если номер >= 0, то удалить строку из списка
      if(indx >= 0)SendMessage(lb,LB_DELETESTRING,(WPARAM)indx,0);
      //Получение текста из поля ФИО
      len=SendMessage(hwFIO, M_GETTEXTLENGTH, 0,0);
      SendMessage(hwFIO, WM_GETTEXT, (WPARAM)len+1, (LPARAM)FIO);
      //Получение текста из поля Балл(100)
      len=SendMessage(hwBal100,WM_GETTEXTLENGTH,0,0);
      SendMessage(hwBal100, WM_GETTEXT, (WPARAM)len+1,
(LPARAM)Bal100);
      //Объединение текстов в одну строку и вставка в список
      sprintf(res, "%-20s\|%-3s",FIO, Bal100);
      SendMessage(lb,LB_INSERTSTRING,(WPARAM)indx,(LPARAM)res);
                        break;
      …
      }
```

Завдання на самостійну роботу

Для всіх наступних завдань використовується умова типової задачі, в якій змінено вимоги:

1) забезпечити завдання і зберігання в списку крім прізвища та оцінки, ще й номера групи;

2) забезпечити завдання і зберігання у списку даних про комп'ютери в лабораторії;

3) забезпечити розрахунок середнього значення оцінки за всіма студентам, які внесені до списку.

Контрольні питання

1) Як додати рядок в список типу Listbox?

2) Як визначити номер рядка, яка виділена у списку?

3) За допомогою якої функції заноситься інформація в список типу Listbox?

Лабораторна робота №4
Тема. Елементи управління вікна.

Мета. Отримання навичок розробки додатків для аналізу елементів управління, з якими пов'язана обробка подій.

Теоретичні відомості

Кнопка - маленьке прямокутне дочірнє вікно, яке представляє собою кнопку, по якій користувач може клацати мишею, щоб включити або вимкнути її. Кнопки управління можуть використовуватися самостійно або в групах, і вони можуть або бути підписані або з'являтися без тексту. Кнопки управління зазвичай змінюють свій вигляд, коли користувач клацає мишею по ним.

Поле редагування - прямокутне дочірнє вікно, всередині якого користувач може надрукувати з клавіатури текст. Користувач вибирає орган управління і дає йому фокус клавіатури, клацаючи по ньому мишею або переміщаючи в нього, каретку шляхом натискання клавіші табуляції (TAB). Користувач може вводити текст, коли вікно редагування тексту відображає миготливу каретку.

Для зчитування інформації з поля редагування використовується функція:

```
int WINAPI GetWindowText (
_In_ HWND hWnd, // дескриптор поля
_Out_ LPTSTR lpString, // покажчик на текстовий рядок
_In_ int nMaxCount); // максимальна кількість символів
```

Значення, що повертається - довжина ліченої текстового рядка.

Щоб зберегти текст в поле редагування використовується функція:

```
BOOL WINAPI SetWindowText (
_In_ HWND hWnd, // дескриптор поля
_In_opt_ LPCTSTR lpString); // покажчик на текстовий рядок
```

У разі успішного завершення функція повертає нульове значення.

Статичний текст - текстове поле, вікно або прямокутник, який використовується для написів, що не підлягають редагуванню.

Завдання.

Розробити додаток, що створює два поля редагування для введення чисел і кнопку «Розрахувати», при натисканні на яку виводиться статичний текст, відповідній сумі. Насамперед розробляється ескіз робочого вікна програми, який показаний на рисунку 4.1.

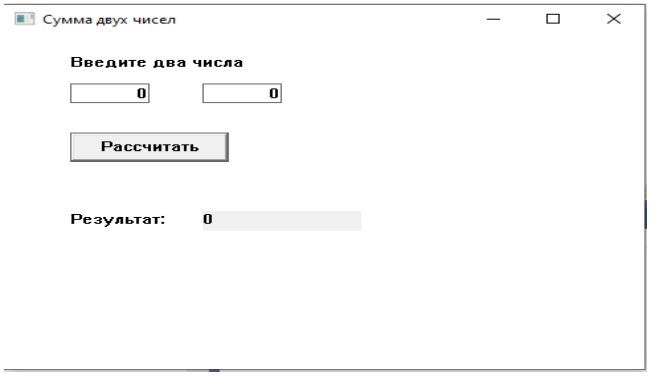

Рисунок 4.1 - Ескіз робочого вікна

Створити проект програми. Параметри вікна програми: ширина вікна - 640, висота - 240. Оголосити змінні:

```
static HWND hBtn; // дескриптор кнопки
static HWND hEdt1, hEdt2; // дескрипторы полей редактирования
static HWND hStat; // дескриптор статического текста
wchar_t StrA[20], StrB[20], StrR[20];
int a, b, sum;
```

Також попередньо необхідно реалізувати функцію перетворення масиву символів у число:

```
int CharToInt(wchar_t a[20])
{
  char st[20] = {0};
  for (int i = 0; a[i]; ++i) {
         wchar_t wc = a[i];
         st[i] = (char) wc;
  }
  return atoi(st);
}
```

У віконну процедуру (WndProc) вставити обробку події створення вікна програми, де помістити функції створення елементів управління і допоміжних об'єктів, а також обробку подій:

```
case WM_CREATE: // сообщение создания окна
```

```
            hInst = ((LPCREATESTRUCT)lparam)->hInstance; //
дескриптор приложения
        // Создаем и показываем первое поле редактирования
            hEdt1 = CreateWindow(L"edit", L"0", WS_CHILD |
WS_VISIBLE | WS_BORDER | ES_RIGHT, 50, 50, 60, 20, hwnd, 0,
hInst, NULL);
        ShowWindow(hEdt1, SW_SHOWNORMAL);
        // Создаем и показываем второе поле редактирования
            hEdt2 = CreateWindow(L"edit", L"0", WS_CHILD |
WS_VISIBLE | WS_BORDER | ES_RIGHT, 150, 50, 60, 20, hwnd, 0,
hInst, NULL);
        ShowWindow(hEdt2, SW_SHOWNORMAL);
        // Создаем и показываем кнопку
            hBtn = CreateWindow(L"button", L"Рассчитать",
WS_CHILD | WS_VISIBLE | WS_BORDER, 50, 100, 120, 30, hwnd,
0, hInst, NULL);
        ShowWindow(hBtn, SW_SHOWNORMAL);
        // Создаем и показываем поле текста для результата
           hStat = CreateWindow(L"static", L"0", WS_CHILD |
WS_VISIBLE, 150, 180, 120, 20, hwnd, 0, hInst, NULL);
        ShowWindow(hStat, SW_SHOWNORMAL);
        break;
case WM_COMMAND:  // сообщение о команде
          if (lparam == (LPARAM)hBtn)     // если нажали на
кнопку
        {
            GetWindowText(hEdt1, StrA, 20);
            GetWindowText(hEdt2, StrB, 20);
            a = CharToInt(StrA);
            b = CharToInt(StrB);
            sum = a + b;
            swprintf_s(StrR, L"%d", sum);
              SetWindowText(hStat, (LPWSTR)StrR); // выводим
результат в статическое поле
        }
        break;
    case WM_PAINT: // перерисовка окна
        hdc = BeginPaint(hwnd, &ps); // начало перерисовки
          TextOut(hdc, 50, 20, L"Введите два числа", 18); //
вывод текстовых сообщений
        TextOut(hdc, 50, 180, L"Результат:", 10);
        EndPaint(hwnd, &ps); // конец перерисовки
        break;
```

Завдання на самостійну роботу

Для всіх наступних завдань використовується умова типової задачі, в якій змінено вимоги: забезпечити виконання завдання, передбачивши окрім складання усі елементарні арифметичні операції багаторозрядних чисел.

Контрольні питання

1) Назвіть елементи управління вікна.

2) Як функція може використовуватися для зчитування інформації з поля редагування?

3) Як функція може використовуватися, щоб зберегти текст в поле редагування?

ПЕРЕЛІК ЛІТЕРАТУРИ

1.Джеффрі Ріхтер WINDOWS. Створення ефективних WIN32-додатків.

2.Шеховцов В.А. Операційні системи. – К.: Видавнича група BHV, 2005. –576 с.

3.Щупак Ю. А. Win32 API. Эффективная разработка приложе-ний. –СПб.: Питер, 2007. –572 с.

4.Ю. Щупак. Win32 API. Ефективна розробка додатків. Питер 2007.

5.Бондаренко М.Ф., Качко О.Г. Операційні системи. – Х.: СМІТ, 2008. –432 с.

6.Олифер В.Г., Олифер Н.А. Сетевые операционные системы. – СПб.: Питер, 2009. –672 с.

7.Дейтел, Х., М. Операционные системы. Основы и принципы. Т. 1 / Х. М. Дейтел, Д.Р. Чофнес. - М.: Бином, 2016. - 1024 с.

8.Рейчард, К. UNIX: справочник; СПб: Питер - М., 2017. - 384 с.

9.Спилман, Джилл MCSE Официальный тест Windows NT 4.0; Microsoft Press. Русская Редакция - М., 2017. - 430 с.

10. Skarga-Bandurova I. A Framework for Real-Time Public Transport Information Acquisition and Arrival Time Prediction Based on GPS Data / I. Skarga-Bandurova, M. Derkach, A. Velykzhanin // Dependable IoT for Human and Industry: Modeling, Architecting, Implementation (Eds. V. Kharchenko, Ah L. Kor, A. Rucinski). – River Publishers Series in Information Science and Technology, 2018. – P. 411-431.

11. Таненбаум, Э. Современные операционные системы / Э. Таненбаум. - СПб.: Питер, 2019. - 1120 с.

12. Internet of Things for Industry and Human Applications. Volume 3. Assessment and Implementation. Intelligent Transportation Systems and IoT. Section 41. / Ed. V. S. Kharchenko.– Ministry of Education and Science of Ukraine, National Aerospace University KhAI, 2019, pp. 373-401.

13. Derkach M. Parking Guide Service for Large Urban Areas / M. Derkach, V. Lysak, I. Skarga-Bandurova, I. Kotsiuba // 10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS). - Metz, France, 2019. – pp. 567 – 571.

Навчальне видання

МЕТОДИЧНІ ВКАЗІВКИ
до лабораторних робіт з дисципліни
«ОПЕРАЦІЙНІ СИСТЕМИ»
(для здобувачів вищої освіти 2 курсу денної та заочної форми навчання за спеціальностями 122 "Комп'ютерні науки", 123 "Комп'ютерна інженерія")

Укладач:
М.В. ДЕРКАЧ

Оригінал-макет *М.В. Деркач*

Підписано до друку _____
Формат 60x84 $^{1}/_{16}$. Папір типогр. Гарнітура Times.
Друк офсетний. Умов. друк. арк. ___. Обл.-вид. арк.___.
Тираж ___ екз. Вид. № ___. Замов. № . Ціна договірна.

Видавництво Східноукраїнського національного університету імені Володимира Даля

Свідоцтво про реєстрацію: серія __ № ____ від _____ р.
Адреса університету: просп. Центральний 59-А
м. Сєвєродонецьк, 93400, Україна
e-mail: vidavnictvoSNU.ua@gmail.com.