МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

Кафедра прикладної математики

ПОВІДОМЛЕННЯ ТА ЧЕРГИ ПОВІДОМЛЕНЬ В ОС WINDOWS

методичні рекомендації до виконання лабораторної роботи №3

з дисципліни:

«Операційні системи»

Київ — 2015

Зміст

ВСТУП

На відміну від аплікацій, написаних для операційної системи (ОС) MS-DOS, аплікації, написані для ОС Microsoft Windows, керовані за допомогою подій. Вони не здійснюють явні виклики функцій (наприклад, виклики бібліотеки мови програмування С) для отримання уведення від користувача. Натомість, вони ждуть, щоб ОС передала їм *повідомлення* у разі, якщо таке уведення матиме місце.

ОС Windows передає все уведення, призначене для деякої аплікації, різним вікнам цієї аплікації. Кожне вікно має функцію, яку називають віконною процедурою, яку Windows викликає, щоразу для вікна з'являється уведення. Віконна процедура опрацьовує уведення та повертає керування ОС.

У даній роботі розглядатимемо засоби роботи з повідомленнями в ОС Windows версії 2000 та вище.

1 ПОСТАНОВКА ЗАДАЧІ

У даній лабораторній роботі потрібно ознайомитися з концепцією повідомлень в ОС Windows, отримати навички використання засобів Windows API для роботи з повідомленнями для поєднання різних компонентів аплікації або різних аплікацій між собою.

У рамках виконання лабораторної роботи потрібно:

а) ознайомитися з теоретичними відомостями, викладеними в розділі 2;

б) написати будь-якою мовою програмування програму, яка повинна:

1) використовувати стандартні повідомлення Windows для уведення з клавіатури та за допомогою миші, обміну даними з елементами керування користувацького інтерфейсу, реагування на команди меню, керування вікнами;

2) використовувати користувацькі повідомлення для обміну даними між компонентами аплікації;

3) використовувати тільки засоби Windows API і не повинна використовувати стандартні процедури MFC тощо;

в) відлагодити програму в ОС Windows версії 2000 та вище;

г) підготувати звіт із лабораторної роботи відповідно до вимог розділу 3.

Кожен студент повинен написати програму з індивідуальними функціональними можливостями. Програми різних студентів не можуть мати однакові функціональні можливості.

Типові приклади програм, які можна написати в рамках даної лабораторної роботи, включають:

- дві аплікації, що комунікують між собою за допомогою користувацького повідомлення: одна аплікація посилає текстові повідомлення, які уводить користувач, інша якимось чином їх відображає;

- аплікація, що здійснює пошук файлів за шаблоном, який уводить користувач;

- текстовий редактор, який дозволяє відкривати, редагувати та зберігати текстові файли;

- аплікація, яка дозволяє малювати чорно-білі рисунки у своєму вікні.

2 ТЕОРЕТИЧНІ ВІДОМОСТІ

2.1 Віконні процедури

Кожне вікно в ОС Windows — об'єкт певного *віконного класу* (window class). Віконний клас визначає *віконну процедуру* (window procedure), яку конкретне вікно використовує для опрацювання повідомлень. Віконна процедура — це функція, котра опрацьовує всі повідомлення, надіслані вікну певного класу. Кожний клас вікна має асоційовану віконну процедуру, і кожне вікно цього класу використовує одну й ту саму віконну процедуру для реагування на повідомлення. Наприклад, ОС визначає віконну процедуру для класу випадного списку (`COMBOMOX`), і всі випадні списки її використовують. Усі аспекти зовнішнього вигляду вікна та його поведінки залежать від того, як віконна процедура опрацьовує повідомлення.

Як правило, аплікація *реєструє* принаймні один новий віконний клас та асоційовану з ним віконну процедуру. Після реєстрації класу, аплікація може створювати багато вікон цього класу, кожне з яких використовуватиме одну й ту саму віконну процедуру. Питання реєстрації віконних класів та написання аплікацій із використанням Windows API виходить за рамки даних методичних рекомендацій. За детальнішою інформацію з цього приводу можна звернутися до [1].

Система надсилає віконній процедурі повідомлення шляхом передачі даних повідомлення як параметрів процедури. Віконна процедура перевіряє ідентифікатор повідомлення та опрацьовує його, використовуючи інформацію, надіслану в якості параметрів повідомлення.

Як правило, віконна процедура не ігнорує повідомлень. Усі пові-

домлення, які віконна процедура не призначена опрацьовувати, повинно бути надіслано ОС для опрацювання за замовчуванням шляхом викликання функції `DefWindowProc`. Ця функція визначає певну мінімальну поведінку, спільну для всіх вікон. Функція `DefWindowProc` виконує дію, задану за замовчуванням, та повертає в процедуру результат опрацювання. Віконна процедура повинна повернути цей результат як власний результат опрацювання повідомлення. Більшість віконних процедур опрацьовують тільки декілька повідомлень, а решту повертають ОС через `DefWindowProc`.

Оскільки віконна процедура спільна для всіх вікон одного класу, вона може отримувати повідомлення, призначені для різних вікон. Для ідентифікації конкретного вікна, якому адресовано повідомлення, віконна процедура може аназілувати *дескриптор вікна*, який передають разом із повідомленням (див. розділ 2.2).

2.2 Повідомлення Windows

ОС Windows передає віконній процедурі уведення у формі *повідомлень* (messages). Повідомлення можуть генерувати як Windows, так і аплікації:

- Windows генерує повідомлення під час кожної події уведення (наприклад, коли користувач уводить дані з клавіатури, рухає мишу, клікає нею по елементу керування інтерфейсу користувача тощо). Також Windows генерує повідомлення у відповідь на зміни в системі, які викликає аплікація (наприклад, зміна набору шрифтів системи, зміна розміру вікна аплікації тощо);

- аплікація генерує повідомлення, щоб змусити власні вікна виконати певні задачі або щоб зв'язатися з вікнами інших аплікацій.

Windows надсилає віконній процедурі повідомлення з чотирма параметрами:

- *дескриптор вікна* (window handle). У своєму ядрі, ОС Windows формує таблицю всіх об'єктів, які вона відтримує. *Дескриптор* — це беззнакова 32-бітове числове значення, яке ОС використовує для ідентифікації об'єктів у цій таблиці. Дескриптори можна розглядати як непрямі вказівники на об'єкти. Windows використовує дескриптор вікна, щоб визначити, у яку віконну процедуру надіслати повідомлення;

- *ідентифікатор повідомлення* (message identifier) — іменована 32-бітова константа, яка ідентифікує призначення повідомлення. Віконна процедура використовує це значення, щоб визначити, як його варто опрацьовувати. Наприклад, ідентифікатор повідомлення `WM_PAINT` вказує віконній процедурі, що клієнтську область вікна змінено, і її потрібно перемалювати;

- два 32-розрядні значення — *параметри повідомлення* (message parameters). Ці параметри визначають дані (або їхнє розташування), які віконна процедура може використовувати під час опрацювання повідомлення. Інтерпретація параметрів залежить від конкретного повідомлення: параметром може бути ціле число, бітові прапорці, указівник на структуру з додатковими даними тощо. Якщо повідомлення не використовує ці параметри, їх зазвичай прирівнюють порожньому вказівнику (`NULL`). Віконна процедура повинна обов'язково перевірити ідентифіктор повідомлення перед інтерпретацією його параметрів.

2.2.1 Типи повідомлень

Повідомлення в ОС Windows бувають двох типів:

- повідомлення, які визначає ОС (system-defined messages);

- повідомлення, які визначає аплікація (application-defined messages).

Windows надсилає повідомлення, які визначає ОС, коли вона зв'язується з аплікацією. Windows використовує ці повідомлення, щоб керувати діями аплікацій та забезпечувати аплікації уведення та іншу інформацію для опрацювання. Аплікація також може надсилати повідомлення, які визначає ОС. Як правило, аплікації використовують ці повідомлення, щоб керувати вікнами елементів керування графічного інтерфейсу, створених із використанням передвизначених віконних класів.

Кожне повідомлення, яке визначає ОС, має унікальний ідентифікатор та відповідну символьну константу (визначену у файлах заголовків SDK (SDK header files)). Наприклад, константа `WM_PAINT` запитує, щоб вікно перемалювало вміст клієнтської області.

Символьні константи визначають клас, якому належать повідомлення, які визначає ОС. Префікс константи ідентифікує тип вікна, яке може інтерпретувати та опрацьовувати відповідне повідомлення. У таблиці 2.1 наведено перелік префіксів та відповідних категорій повідомлень.

Таблиця 2.1 – Категорії повідомлень та префікси відповідних символьних констант

Префікс	Категорія повідомлення
ABM and ABN	Application desktop toolbar
ACM and ACN	Animation control
BCM, BCN, BM, and BN	Button control
CB and CBN	ComboBox control
CBEM and CBEN	ComboBoxEx control
CCM	General control
CDM	Common dialog box
DFM	Default context menu
DL	Drag list box
DM	Default push button control
DTM and DTN	Date and time picker control
EM and EN	Edit control
HDM and HDN	Header control
HKM	Hot key control

Продовження таблиці 2.1

Префікс	Категорія повідомлення
IPM and IPN	IP address control
LB and LBN	List box control
LM	SysLink control
LVM and LVN	List view control
MCM and MCN	Month calendar control
PBM	Progress bar
PGM and PGN	Pager control
PSM and PSN	Property sheet
RB and RBN	Rebar control
SB and SBN	Status bar window
SBM	Scroll bar control
SMC	Shell menu
STM and STN	Static control
TB and TBN	Toolbar
TBM and TRBN	Trackbar control

Продовження таблиці 2.1

Префікс	Категорія повідомлення
TCM and TCN	Tab control
TDM and TDN	Task dialog
TTM and TTN	Tooltip control
TVM and TVN	Tree-view control
UDM and UDN	Up-down control
WM	General

Загальні повідомлення (*General* у таблиці 2.1) покривають широкий діапазон інформації та запитів, уключно з повідомленнями для уведення з клавіатури та миші, уведення через меню та діалогові вікна, створення та керування вікнами, а також динамічного обміну даними (Dynamic Data Exchange, DDE).

Аплікація може визначати повідомлення для використання у власних вікнах або для зв'язку з вікнами в інших аплікаціях. Якщо аплікація визначає власні повідомлення, віконна процедура, яка їх отримує, повинна інтерпретувати їх та забезпечувати належне опрацювання.

У Windows використовують такі значення ідентифікаторів повідомлень:

- значення в діапазоні 0000h–03FFh (`WM_USER` – 1) зарезервовано для повідомлень, які визначає ОС. Аплікації не можуть використовувати

ці значення для власних повідомлень;

- значення в діапазоні 0400h (WM_USER)–7FFFh доступні для ідентифікаторів повідомлень, які визначає аплікація;

- якщо аплікацію позначено версією 4.0, то вона може використовувати значення в діапазоні 0800h (WM_APP)–BFFFh для ідентифікаторів повідомлень, які визначає аплікація;

- Windows повертає значення в діапазоні C000h–FFFFh, коли аплікація викликає функцію RegisterWindowMessage для реєстрації ідентифікатора повідомлення. При цьому гарантовано унікальність такого ідентифікатора. Використання цієї функції упереджує конфлікти, які можуть виникати в разі використання одного ідентифікатора різними аплікаціями для різних цілей.

2.3 Маршрутизація повідомлень

ОС Windows використовує два методи надсилання повідомлень віконній процедурі:

- надсилання (*posting*) повідомлень у *чергу повідомлень* (message queue) — системний об'єкт пам'яті, який тимчасово зберігає повідомлення. Повідомлення, поставлені в чергу повідомлень (*чергові повідомлення*, queued messages), — це результати уведення користувача за допомогою миші чи клавіатури (наприклад, повідомлення WM_MOUSEMOVE, WM_LBUTTONDOWN, WM_KEYDOWN, WM_CHAR тощо), а також повідомлення таймера (WM_TIMER), перемалювання (WM_PAINT) та виходу (WM_QUIT);

- надсилання (*sending*) повідомлень віконній процедурі безпосере-

дньо. Такі повідомлення називають *позачерговими* (non-queued messages).

2.3.1 Чергові повідомлення

ОС Windows може одночасно демонструвати довільну кількість вікон. Тому для маршрутизації уведення від миші та клавіатури до потрібних вікон вона використовує черги повідомлень. ОС підтримує одну системну чергу повідомлень та довільну кількість потокових черг повідомлень — по одній для кожного потоку графічного інтерфейсу користувача.

Щоб уникнути створення черг повідомлень для потоків, не пов'язаних із графічним інтерфейсом користувача, усі потоки спочатку створюють без черги повідомлень. Система створює чергу повідомлень потоку, тільки коли потік здійснює виклик однієї з користувацьких функцій.

Щоразу, коли користувач рухає мишу, натискає кнопки на миші чи клавіатурі, драйвер миші чи клавіатури перетворює уведення у відповідні повідомлення то розміщує їх у системній черзі повідомлень. Windows вибирає з черги повідомлення одне за одним, досліджує їх для визначення вікна-адресата та надсилає їх у чергу того потоку, який створив вікно-адресата. Черга повідомлень потоку отримує всі повідомлення миші та клавіатури для вікон, які створив цей потік. Потік вибирає повідомлення з черги одне за одним та просить Windows надсилати їх відповідній віконній процедурі на опрацювання.

Windows завжди розміщує повідомлення наприкінці черги. Це гарантує, що вікно отримує вхідні повідомлення в порядку їх надходження. Єдині повідомлення, які Windows не розміщує наприкінці черги, — це

повідомлення WM_PAINT, WM_TIMER та WM_QUIT. Їх надсилають віконній процедурі тільки в тому разі, якщо черга не містить жодних інших повідомлень. Окрім цього, множинні повідомлення WM_PAINT для одного вікна об'єднують в одне повідомлення, у якому всі призначені для перемалювання області вікна об'єднують в одну. Такий підхід дозволяє зменшити кількість разів, які вікно повинно перемальовувати вміст клієнтської області.

ОС розміщує повідомлення в чергу потоку шляхом заповнення полів структури MSG та копіюючи її в чергу. Структура MSG має наступний формат:

```
typedef struct tagMSG {
    HWND hwnd;
    UINT message;
    WPARAM wParam;
    LPARAM lParam;
    DWORD time;
    POINT pt;
} MSG;
```

Прокоментуймо призначення кожного поля структури:

- hwnd — дескриптор вікна, для якого призначено повідомлення;
- message — ідентифікатор повідомлення;
- wParam і lParam — параметри повідомлення;
- time — час надсилання повідомлення;
- pt — позиція курсора миші у момент надсилання повідомлення.

2.3.2 Позачергові повідомлення

Позачергові повідомлення надсилають віконній процедурі вікна-адресата миттєво, повз системну та потокові черги повідомлень. Як правило, Windows надсилає позачергові повідомлення, щоб поінформувати вікно про події, які впливають на нього. Наприклад, коли користувач активізує нове вікно аплікації, Windows надсилає вікну низку повідомлень, у тому числі `WM_ACTIVATE`, `WM_SETFOCUS` та `WM_SETCURSOR`, які інформують вікно, що воно активізовано, що уведення з клавіатури тепер направлятимуть у це вікно та що курсор миші перемістився всередину вікна, відповідно.

Позачергові повідомлення також можуть бути результатом виклику з аплікації деяких функцій Windows. Наприклад, Windows надсилає повідомлення `WM_WINDOWPOSCHANGED` після того, як аплікація викликає функцію `SetWindowPos` для переміщення вікна.

2.3.3 Надсилання чергових повідомлень з аплікації

Як правило, аплікація надсилає чергове повідомлення (posts a message), щоб змусити певне вікно виконати деяку задачу. Потік аплікації може надсилати повідомлення у власну чергу за допомогою функції `PostMessage`. Функція `PostMessage` має наступний формат:

```
BOOL WINAPI PostMessage(
    HWND hWnd,
    UINT Msg,
```

```
    WPARAM wParam,
    LPARAM lParam
);
```

Ця функція створює структуру `MSG` для повідомлення на основі переданих у функцію даних, а також системного часу та позиції курсора (для полів структури `time` та `pt`, відповідно), та копіює її в чергу. Аргументи функції аналогічні однойменним полям структури `MSG`. Функція повертає ненульове значення у разі успішного завершення, та 0, якщо сталася помилка.

Аплікація може надсилати чергове повідомлення, не вказуючи конкретне вікно, тобто передавши в якості дескриптора вікна значення `NULL`. У такому разі повідомлення буде розміщено в чергу, асоційовану з поточним потоком, і аплікації потрібно буде самостійно його опрацювати в циклі опрацювання повідомлень (див. розділ 2.4). У такий спосіб можна створювати повідомлення, які стосуються всієї аплікації, а не конкретного вікна.

Аплікація також може надіслати чергове повідомлення, яке стосується всіх вікон найвищого рівня. Для цього потрібно передати в якості дескриптора вікна значення `HWND_TOPMOST`.

Варто окремо зазначити, що типовою програмістською помилкою є припущення, що функція `PostMessage` завжди надсилає повідомлення в чергу. Якщо черга повна, то це не відповідає дійсності. Тому аплікація повинна перевіряти значення, яке повертає `PostMessage`, щоб визначити, чи було надіслано повідомлення, і перенадіслати його у випадку помилки.

Потік аплікації також може надсилати повідомлення в чергу іншого потоку за допомогою функції `PostThreadMessage`. Формат цієї фун-

кції наступний:

```
BOOL WINAPI PostThreadMessage(
    DWORD idThread,
    UINT Msg,
    WPARAM wParam,
    LPARAM lParam
);
```

Аргумент `idThread` цієї функції — це ідентифікатор потоку, якому потрібно надіслати відповідне повідомлення. Інші параметри цієї функції аналогічні однойменним параметрам функції `PostMessage`.

2.3.4 Вибирання чергових повідомлень з аплікації

Аплікація може вибирати повідомлення з черги, використовуючи функцію `GetMessage`. Формат цієї функції наступний:

```
BOOL WINAPI GetMessage(
    LPMSG lpMsg,
    HWND hWnd,
    UINT wMsgFilterMin,
    UINT wMsgFilterMax
);
```

Аргументи цієї функції мають таку інтерпретацію:

- `lpMsg` — указівник на структуру `MSG`, у яку буде записано інформацію про повідомлення з черги потоку;

- hWnd — дескриптор вікна, чиє повідомлення потрібно вибрати з черги; вікно повинно належати поточному потоку;

- wMsgFilterMin — нижній поріг значень ідентифікаторів повідомлень, які потрібно вибрати;

- wMsgFilterMax — верхній поріг значень ідентифікаторів повідомлень, які потрібно вибрати. Якщо wMsgFilterMin та wMsgFilterMax одночасно дорівнюють 0, то GetMessage повертає всі доступні повідомлення.

Фактично, за допомогою параметрів hWnd, wMsgFilterMin та wMsgFilterMax можна здійснювати *фільтрацію повідомлень*, тобто вибирати з черги тільки ті повідомлення, які задовольняють відповідним умовам. Це може бути корисно, коли аплікація шукає в черзі повідомлення, які надійшли раніше, або коли аплікації потрібно опрацювати повідомлення від апаратних пристроїв раніше за надіслані програмно. При цьому аплікація повинна задавати фільтри таким чином, щоб гарантувати появу в черзі повідомлення, яке б їм відповідало. Наприклад, якщо вікно не отримує уведення від клавіатури, а аплікація намагається вибрати з черги повідомлення WM_CHAR, фукнція GetMessage зависне.

У випадку успішного завершення, функція GetMessage повертає 0 (якщо було вибрано повідомлення WM_QUIT) або інше ненульове значення (якщо було вибрано інше повідомлення). У випадку помилки функція повертає –1.

Іноді аплікація повинна аналізувати вміст черги повідомлень, не вибираючи з неї повідомлень. Наприклад, якщо віконна процедура повинна виконувати тривалу операцію малювання, може бути корисно передбачити можливість переривання виконання такої операції з боку користувача. Якщо аплікація не аналізуватиме чергу повідомлень на наявність повідом-

лень від клавіатури чи миші, вона не зможе на них вчасно відреагувати.

Для того, щоб проаналізувати повідомлення, не вибираючи його з черги, аплікація може використовувати функцію `PeekMessage`. Розгляд цієї функції виходить за межі даних методичних рекомендацій. За детальнішою інформацію можна звернутися до [1].

2.3.5 Надсилання чергових повідомлень віконним процедурам з аплікації

Після вибирання повідомлення з черги, аплікація може за допомогою функції `DispatchMessage` попросити ОС надіслати це повідомлення віконній процедурі для опрацювання. Формат цієї функції наступний:

```
LRESULT WINAPI DispatchMessage(const MSG *lpmsg);
```

Функція приймає на вхід указівник на структуру `MSG` з повідомленням, яку попередньо було заповнено за допомогою функції `GetMessage` (`PeekMessage`). У якості результату функція повертає результат виконання віконної процедури, яка опрацювала надіслане повідомлення.

Функція `DispatchMessage` передає у віконну процедуру дескриптор вікна, ідентифікатор повідомлення та два його параметри. Час створення повідомлення та позицію курсора миші функція `DispatchMessage` у віконну процедуру не передає.

Якщо в черзі потоку немає повідомлень, він може передати керування іншим потокам за допомогою функції `WaitMessage`, яка призупиняє виконання потоку до моменту появи в черзі нового повідомлення. Ця функція не має жодних аргументів:

```
BOOL WINAPI WaitMessage(void);
```

2.3.6 Надсилання позачергових повідомлень з аплікації

Як правило, аплікація надсилає позачергове повідомлення (sends a message), щоб змусити певне вікно виконати деяку задачу *негайно*. Аплікація може надіслати позачергове повідомлення за допомогою функції `SendMessage`. Формат цієї функції наступний:

```
LRESULT WINAPI SendMessage(
    HWND hWnd,
    UINT Msg,
    WPARAM wParam,
    LPARAM lParam
);
```

Аргументи цієї функції можна інтерпретувати так само, як і аргументи функції `PostMessage`. Функція повертає результат опрацювання повідомлення, який залежить від типу повідомлення.

Функція `SendMessage` безпосередньо надсилає повідомлення `Msg` віконній процедурі, що відповідає вікну з дескриптором `hWnd`, оминаючи відповідну чергу повідомлень. Функція жде, поки віконна процедура завершить опрацювання повідомлення та поверне результат опрацювання.

Часто батьківські та дочірні вікна спілкуються між собою шляхом надсилання позачергових повідомлень. Наприклад, батьківське вікно з полем редагування, яке є його дочірнім вікном, може задати текст поля шляхом надсилання йому позачергового повідомлення. У свою чергу, по-

ле редагування може проінформувати батьківське вікно у випадку зміни тексту з боку користувача шляхом надсилання позачергового повідомлення батьківському вікну.

Низку повідомлень може бути надіслано елементам керування в діалоговому вікні. Ці повідомлення установлюють загальний вигляд, поведінку та вміст елементів керування, а також повертають інформацію про стан елемента керування. Надсилання таких повідомлень здійснюють за допомогою функції `SendDlgItemMessage`, формат якої має вигляд:

```
LRESULT WINAPI SendDlgItemMessage(
    HWND hDlg,
    int nIDDlgItem,
    UINT Msg,
    WPARAM wParam,
    LPARAM lParam
);
```

Аргумент `hDlg` — дескриптор діалогового вікна, `nIDDlgItem` — ідентифікатор елемента керування, якому потрібно надіслати повідомлення. Інші аргументи функції можна інтерпретувати аналогічно однойменним аргументам функції `SendMessage`.

Приклад 2.1.

Розгляньмо надсилання повідомлення елементу керування діалогового вікна на прикладі повідомлення `CB_ADDSTRING`, яке додає рядок до випадного списку (combo box). У нижченаведеному вихідному коді, написаному мовою програмування C, рядок із текстового поля випадного списку копіюють у поле списку.

```
HWND hwndCombo;
```

```
int cTxtLen;
PSTR pszMem;
switch(uMsg)
{
    case WM_COMMAND:
    switch(LOWORD(wParam))
    {
        case IDD_ADDCBITEM:
            //отримуємо дескриптор випадного списку
            //та довжину рядка
            hwndCombo = GetDlgItem(hwndDlg, IDD_COMBO);
            cTxtLen = GetWindowTextLength(hwndCombo);

            //виділяємо пам'ять під рядок
            pszMem = (PSTR) VirtualAlloc((LPVOID) NULL,
                (DWORD) (cTxtLen + 1), MEM_COMMIT,
                PAGE_READWRITE);

            //та копіюємо в нього вміст поля випадного списку
            GetWindowText(hwndCombo, pszMem,
                cTxtLen + 1);

            //додаємо рядок до випадного списку
            if(*pszMem != NULL)
            {
                SendDlgItemMessage(hwndDlg, IDD_COMBO,
                    CB_ADDSTRING, 0,
                    (DWORD) ((LPSTR) pszMem));

                //очищуємо поле випадного списку
                SetWindowText(hwndCombo, (LPSTR) NULL);
            };

            //очищуємо пам'ять
            VirtualFree(pszMem, 0, MEM_RELEASE);

            return TRUE;
    }
}
```

Окрім функції `SendMessage`, аплікація також може використовувати функцію `SendMessageCallback`. Остання функція, на відміну від першої, повертає керування аплікації негайно. Формат функції `SendMessageCallback` наступний:

```
BOOL WINAPI SendMessageCallback(
    HWND hWnd,
    UINT Msg,
    WPARAM wParam,
    LPARAM lParam,
```

```
    SENDASYNCPROC lpCallBack,
    ULONG_PTR dwData
);
```

Аргумент `lpCallBack` — це вказівник на функцію, яку викличе ОС після того, як віконна процедура завершить опрацювання надісланого їй повідомлення. Аргумент `dwData` — це дані, які передають функції, визначеній за допомогою вказівника `lpCallBack`.

Як і у випадку з надсиланням чергових повідомлень, в окремих випадках доречно надсилати позачергові повідомлення всім вікнам найвищого рівня. Для цього потрібно передати в якості дескриптора вікна значення `HWND_TOPMOST`. Наприклад, якщо аплікація змінює системні налаштування часу, вона повинна проінформувати про це всі вікна найвищого рівня шляхом надсилання позачергового повідомлення `WM_TIMECHANGE`.

2.4 Цикл опрацювання повідомлень

Аплікація повинна вибирати повідомлення з черг своїх потоків та опрацьовувати їх. Однопотокова аплікація зазвичай використовує для цього *цикл опрацювання повідомлень* у своїй функції `WinMain` (`WinMain` — це точка входу в аплікацію Windows). Багатопотокова аплікація може містити такий цикл у кожному своєму потоці, який створює вікно.

Головний потік аплікації починає цикл опрацювання повідомлень після ініціалізації аплікації та створення щонайменше одного вікна. На кожній ітерації циклу вибирають одне повідомлення з черги та направляють його потрібній віконній процедурі. Цикл завершує свою роботу, коли `GetMessage` вибирає з черги повідомлення `WM_QUIT` (див. нижче).

Найпростіший цикл опрацювання повідомлень містить по одному виклику функцій `GetMessage`, `TranslateMessage`, и `DispatchMessage` у вказаній послідовності:

```
MSG msg;
```

```
BOOL bRet;

while((bRet = GetMessage(&msg, NULL, 0, 0)) != 0)
{
    if (bRet == -1)
    {
        //handle the error and possibly exit
    }
    else
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
}
```

У вищенаведеному циклі окремо виконано перевірку на рівність значення функції GetMessage –1, оскільки в цьому випадку потрібно виконувати опрацювання помилки під час виконання цієї функції.

Прокоментуймо роботу цього циклу. Функція GetMessage вибирає з черги повідомлення та копіює його в структуру типу MSG. Функція повертає значення, що дорівнює 0, тільки у випадку вибирання повідомлення WM_QUIT. У такому разі аплікація вийде з циклу. Для однопотокових аплікацій вихід із циклу опрацювання повідомлень часто є першим кроком до закриття. Аплікація може самостійно завершити виконання власного циклу опрацювання повідомлень за допомогою функції PostQuitMessage (надсилання в чергу повідомлення WM_QUIT), як правило, у результаті опрацювання повідомлення WM_DESTROY у віконній процедурі головного вікна аплікації. Це показано в нижченаведеному фрагменті коду мовою програмування C:

```
case WM_DESTROY:
    //perform cleanup tasks
    PostQuitMessage(0);
    break;
```

Якщо потік отримує символьне уведення з клавіатури, у циклі опрацювання повідомлень повинно бути передбачено функцію TranslateMessage. ОС генерує повідомлення натискання та відпускання віртуальних клавіш (WM_KEYDOWN та WM_KEYUP, відповідно), щоразу користувач натискає деяку клавішу. Повідомлення віртуальних клавіш містять код віртуальної клавіші, яку було натиснуто, але не її символьне значення. Отримати символьне значення можна за допомогою функції TranslateMessage, яка транслює повідомлення віртуальної клавіші у символьне повідомлення (WM_CHAR) та повертає його в чергу. Символьне повідомлення може бути вибрано та опрацьовано належним чином на наступній ітерації циклу.

Формат функції TranslateMessage наступний:

```
BOOL WINAPI TranslateMessage(const MSG *lpMsg);
```

Функція DispatchMessage, як було зазначено в розділі 2.3.1, надсилає повідомлення віконній процедурі, асоційованій із дескриптором вікна, зазначеним у структурі MSG, яку передано всередину функції. Якщо дескриптор вікна дорівнює HWND_TOPMOST, DispatchMessage надсилає повідомлення віконним процедурам усім вікнам найвищого рівня. Якщо дескриптор дорівнює NULL, DispatchMessage не надсилає повідомлення взагалі.

Навіть якщо аплікація має багато вікон, для опрацювання черги повідомлень достатньо одного циклу. Функція DispatchMessage завжди направляє повідомлення правильному вікну-адресату шляхом аналізу дескриптора вікна у відповідній структурі MSG.

2.4.1 Модифікації циклу опрацювання повідомлень

Базовий цикл опрацювання повідомлень, розглянутий вище, можна модифікувати у декілька способів:

- можна вибирати повідомлення з черги, не направляючи їх жодному

вікну. Це може бути корисно для аплікацій, які працюють із повідомленнями, які не стосуються жодного вікна;

- можна використати фільтрацію повідомлень у функції `GetMessage`, щоб змінити порядок опрацювання повідомлень у черзі;

- якщо аплікація використовує клавіші швидкого доступу, потрібно передбачити трансляцію повідомлень клавіатури у повідомлення команд. Для цього в циклі опрацювання повідомлень повинно бути передбачено виклик функції `TranslateAccelerator`;

- якщо потік використовує немодальне діалогове вікно, цикл опрацювання повідомлень повинен містити функцію `IsDialogMessage`, щоб уможливити отримання з боку цього вікна уведення клавіатури.

Приклад 2.2.

Розгляньмо вихідний текст програми на мові програмування С, що демонструє використання циклу опрацювання повідомлень у функції `WinMain` простої аплікації, головне вікно якої представлено на рисунку 2.1. У відповідь на повідомлення `WM_PAINT` програма малює у верхньому лівому куті еліпс. Також, у цій програмі натискання клавіші «Enter» веде до появи повідомлення з текстом «Hello World!» (рисунок 2.2).

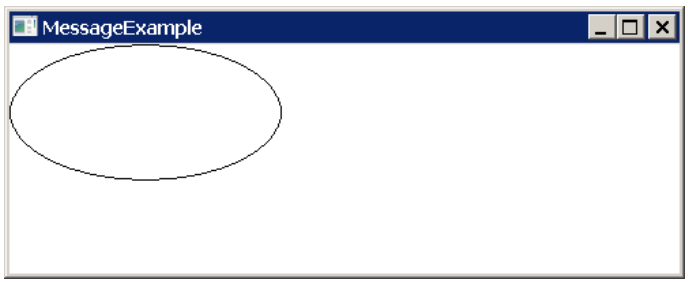

Рисунок 2.1 – Головне вікно програми з Прикладу 2.2

```
#include "stdafx.h"

//глобальні змінні
HINSTANCE hInst; //поточний екземпляр аплікації
TCHAR szTitle[] = L"MessageExample"; //заголовок головного вікна
TCHAR szWindowClass[] = L"MessageExampleWindow"; //ім'я класу головного вікна

ATOM MyRegisterClass(HINSTANCE hInstance);
BOOL InitInstance(HINSTANCE, int);
```

```
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

int WINAPI WinMain(HINSTANCE hInstance,
                   HINSTANCE hPrevInstance,
                   LPSTR lpCmdLine,
                   int nCmdShow)
{
    UNREFERENCED_PARAMETER(hPrevInstance);
    UNREFERENCED_PARAMETER(lpCmdLine);
    MSG msg;
    HACCEL hAccelTable;

    //реєструємо віконний клас головно вікна аплікації
    MyRegisterClass(hInstance);

    //виконуємо ініціалізацію аплікації
    if(!InitInstance (hInstance, nCmdShow))
    {
        return FALSE;
    }

    //завантажуємо клавіші швидкого доступу
    hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC_WIN32PROJECT1));

    //цикл опрацювання повідомлень
    BOOL bRet;
    while((bRet = GetMessage(&msg, (HWND) NULL, 0, 0)) != 0)
    {
        if(bRet == -1)
        {
            //якщо сталася помилка
            return FALSE;
        }
        else
        {
            if(!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
            {
                TranslateMessage(&msg);
                DispatchMessage(&msg);
            }
        }
    }

    //повертаємо код виходу
    return (int) msg.wParam;
}

//функція для реєстрації нового віконного класу
ATOM MyRegisterClass(HINSTANCE hInstance)
{
    WNDCLASSEX wcex;

    wcex.cbSize = sizeof(WNDCLASSEX);

    wcex.style = CS_HREDRAW | CS_VREDRAW;
    wcex.lpfnWndProc = WndProc;
    wcex.cbClsExtra = 0;
```

```
    wcex.cbWndExtra = 0;
    wcex.hInstance = hInstance;
    wcex.hIcon = NULL;
    wcex.hCursor = LoadCursor(NULL, IDC_ARROW);
    wcex.hbrBackground = (HBRUSH)(COLOR_WINDOW+1);
    wcex.lpszMenuName = NULL;
    wcex.lpszClassName = szWindowClass;
    wcex.hIconSm = NULL;

    return RegisterClassEx(&wcex);
}

//функція для створення вікна
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
    HWND hWnd;

    //зберігаємо дескриптор екземпляра програми в глобальній змінній
    hInst = hInstance;

    //створюємо вікно засобами Windows API
    hWnd = CreateWindow(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, NULL, NULL, hInstance, NULL);

    if(!hWnd)
    {
        //якщо сталася помилка
        return FALSE;
    }

    //виводимо вікно на екран
    ShowWindow(hWnd, nCmdShow);
    UpdateWindow(hWnd);

    return TRUE;
}

//віконна процедура
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    PAINTSTRUCT ps;
    HDC hdc;

    //аналізуємо тип повідомлення
    switch(message)
    {
        case WM_CHAR:
            //якщо треба опрацювати уведення з клавіатури
            if(wParam == 13)
            {
                MessageBox(
                    NULL,
                    L"Hello World!",
                    L"Simple message box",
                    MB_ICONWARNING | MB_OK
                );
            }
```

```
        case WM_PAINT:
            //якщо треба опрацювати повідомлення перемалювання
            hdc = BeginPaint(hWnd, &ps);
            Ellipse(hdc, 0, 0, 200, 100);
            EndPaint(hWnd, &ps);
            break;
        case WM_DESTROY:
            //якщо треба опрацювати повідомлення знищення вікна
            PostQuitMessage(0);
            break;
        default:
            //усі інші повідомлення даємо можливість опрацювати ОС
            return DefWindowProc(hWnd, message, wParam, lParam);
    }

    return 0;
}
```

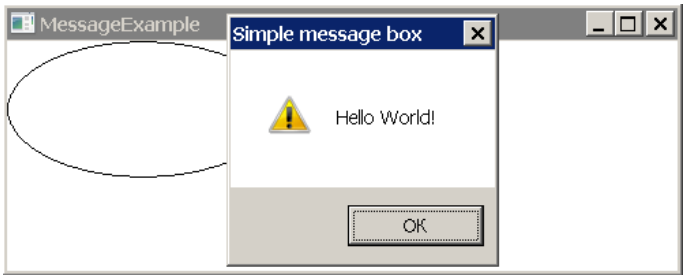

Рисунок 2.2 – Поява повідомлення в програмі з Прикладу 2.2

3 ПОРЯДОК ЗДАЧІ ЛАБОРАТОРНОЇ РОБОТИ ТА ВИМОГИ ДО ЗВІТУ

Здача лабораторної роботи передбачає:

- демонстрацію працездатності розробленої в рамках роботи програми в ОС Windows версії, не нижчої за 2000;
- здачу викладачеві звіту з лабораторної роботи.

Під час здавання лабораторної роботи викладач має право ставити студентові питання за матеріалами розділу 2.

Під час демонстрації роботи програми викладач має право вимагати від студента внесення змін у програму.

Звіт про виконання лабораторної роботи повинен включати:

а) вступ;
б) постановку задачі;
в) теоретичні відомості;
г) опис розробленої програми;
д) результати випробування розробленої програми;
е) висновки;
є) посилання на літературні джерела;
ж) додаток, у якому повинно бути розміщено лістинг розробленої програми.

Детальніші вимоги до звіту викладено в документі «Вимоги до оформлення звіту». Окрім вимог, викладених у ньому, у розділі «Опис розробленої програми» потрібно додатково подати:

а) перелік використаних повідомлень, які визначає ОС;
б) детальний опис використаних повідомлень, які визначає аплікація.

Текстову частину повинно бути оформлено відповідно до вимог ДСТУ 3008-95 «Документація. Звіти у сфері науки і техніки. Структура і правила оформлення» із застосуванням системи підготовки документів LaTeX.

ПЕРЕЛІК ПОСИЛАНЬ

1. Microsoft Library [Електронний ресурс]. — Режим доступу: https://msdn.microsoft.com/en-us/library/

2. Мешков А. Visual C++ и MFC / А. Мешков, Ю. Тихомиров. — [2-е изд.] — С.-Пб. : BHV, 2001. — 1040 с.

3. Petzold C. Programming Windows / C. Petzold. — [5th ed.] — Microsoft Press, 1998. — 1100 p.

4. Simon R. Windows NT Win32 API SuperBible / R. Simon. — Waite Group Press, 1997. — 1510 p.