Тема 5. Концепція повідомлень у Windows

План

1. Концепція повідомлень.
2. Типи повідомлень.
   2.1. Чергові повідомлення.
   2.2. Позачергові повідомлення.
3. Аналіз та обробка повідомлень.

1. Концепція повідомлень

Повідомлення є реакцією системи Windows на різні події, що відбуваються в системі: рух миші, натискання клавіші, спрацьовування таймера та ін. Характерною ознакою повідомлення є його ***код***, який може набувати значень (для системних повідомлень) від 1 до 0x3FF. Кожному коду відповідає своя символічна константа, ім'я якої достатньо ясно говорить про джерело повідомлення:

при русі миші	WM_MOUSEMOVE	(код 0x200),
при натисканні лівої кнопки миші	WM_LBUTTONDOWN	(код 0x201),
при спрацьовуванні таймера	WM_TIMER	(код 0x113).

Усі повідомлення можна поділити на ***системні*** та ***програмні***. *Системні повідомлення* – це наперед визначені повідомлення, які використовуються при взаємодії прикладної програми і операційної системи. Програмні повідомлення визначаються у програмі і використовуються для взаємодії із компонентами самої програми або з іншими програмами, у яких визначені аналогічні повідомлення.

Системні повідомлення. По ходу створення і виводу на екран головного вікна *Windows посилає в програму* послідовно цілу групу повідомлень, що сигналізують про етапи цього процесу: WM_GETMINMAXINFO для уточнення розмірів вікна, WM_ERASEBKGND при заповненні вікна кольором фону, WM_SIZE при оцінці розмірів робочої зони вікна, WM_PAINT для отримання від програми інформації про вміст вікна і багато інших. Деякі з цих повідомлень Windows обробляє сама; інші зобов'язана обробити прикладна програма.

Може бути і зворотна ситуація, коли *повідомлення створюється у прикладній програмі* по волі програміста і посилається у Windows для того, щоб система виконала необхідні дії (наприклад, заповнила конкретною інформацією вікно із списком або повідомила про стан певного елементу керування).

Повідомлення такого роду також стандартизовані і мають певні номери.

Програмні повідомлення. Програміст може передбачити *власні повідомлення* і направляти їх в різні вікна застосування для сповіщення про ті або інші ситуації.

Для *кожного потоку* у 32-розрядному застосуванні створюється *своя черга* повідомлень. Ці черги не мають фіксованого розміру, а можуть необмежено розширюватися. Повідомлення із системної черги передаються не в застосування в цілому, а розподіляються по його потоках. Повідомлення від миші зазвичай (хоча не завжди) адресовані тому вікну, над яким перебуває її курсор. Дійсно, клацаючи по пункту меню або по кнопці в деякому вікні, ми хочемо спричиняти дію саме для цього вікна. Вікна ж створюються за допомогою функції CreateWindow() тим або іншим потоком застосування. Тому повідомлення від миші прямують в чергу того потоку, який створив дане вікно.

Розглянемо тепер, із чого складається кожне повідомлення. На початку головної функції програми WinMain оголошується структурна змінна Msg. Це найважливіша змінна, за допомогою якої в програму передається вміст повідомлень Windows. Кожним повідомленням є пакет із шести даних, описаних у файлі WINUSER.H за допомогою структури типу MSG:

```
typedef struct tagMSG {
   HWND hwnd;           // дескриптор вікна, якому адресовано повідомлення
   UINT message;        // код даного повідомлення
   WPARAM wParam;       // додаткова інформація
   LPARAM lParam;       // додаткова інформація
   DWORD time;          // час відправлення повідомлення
   POINT pt;            // позиція курсору миші у момент відправлення повідомлення
} MSG;
```

Для повідомлення WM_MOUSEMOVE структура Msg заповнюється наступною інформацією:

`Msg.hwnd` - дескриптор вікна, яке одержує повідомлення;
`Msg.message` - код повідомлення WM_MOUSEMOVE = 0x200;
`Msg.wParam` - комбінація бітових прапорів, що відображають стан клавіш миші (натиснуті/не натиснуті), а також клавіш Ctrl і Shift;
`Msg.lParam` - позиція курсору миші відносно робочої зони вікна;
`Msg.time` - час відправлення повідомлення;
`Msg.pt` - позиція курсору миші відносно границь екрану.

Параметри повідомлення WM_MOUSEMOVE можна одержати таким чином:

```
fwKeys = wParam;              // прапорці кнопок
xPos = LOWORD(lParam);        // горизонтальна позиція курсору
yPos = HIWORD(lParam);        // вертикальна позиція курсору
```

Для клавіатурного повідомлення WM_KEYDOWN:

```
nVirtKey = (int) wParam;      // віртуальний код клавіші
lKeyData = lParam;            // дані з клавіатури
```

Параметр lParam містить кількість повторень (0-15), скан-код (16-23), прапорець розширеної клавіші (24), код контексту (29), прапорець попереднього стану клавіші (30) та прапорець стану переходу (31).

2. Типи повідомлень

Windows використовує два методи, щоб надсилати повідомлення процедурі вікна:

- посилаючи повідомлення в чергу типу FIFO, названу ***черга повідомлень***, (визначений системою об'єкт пам'яті, який тимчасово зберігає повідомлення);
- відправляючи повідомлення безпосередньо процедурі вікна.

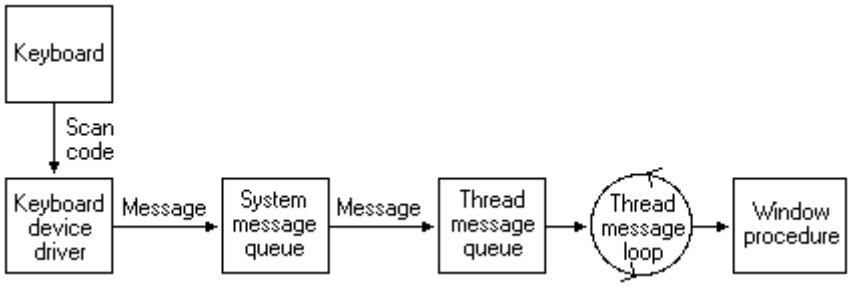
Чергові повідомлення це ті, які розміщуються в черзі повідомлень програми системою Windows. Вони є передусім результатом введення користувачем за допомогою миші або клавіатури, як наприклад WM_MOUSEMOVE, WM_LBUTTONDOWN, WM_KEYDOWN, WM_CHAR. Інші чергові повідомлення включають повідомлення таймера, малювання та виходу: WM_TIMER, WM_PAINT, і WM_QUIT. У циклі повідомлень програми вони дістаються і відсилаються до процедури вікна.

Позачергові повідомлення – це результат безпосередніх викликів Windows процедури вікна. Говорять, що чергові повідомлення "посилаються" до черги повідомлень, а позачергові повідомлення "відправляються" процедурі вікна. У будь-якому випадку, процедура вікна одержує усі повідомлення для вікна – як *чергові* так і *позачергові*.

Windows обслуговує ***єдину чергу повідомлень*** системи і будь-яке число ***черг повідомлень потоків***, один для кожного потоку GUI. Щоб уникати невиправданого створення черги повідомлень для потоків non-GUI, всі потоки створюються спочатку поза чергою повідомлень. Система створює чергу повідомлень потоку лише коли потік здійснює свій перший виклик до однієї із функцій Win32 API User або GDI.

2.1. Чергові повідомлення

Кожного разу користувач переміщає мишу, клацає клавіші миші, або набирає на клавіатурі, драйвер пристрою для миші або клавіатури перетворює подію введення на повідомлення і розміщує їх в черзі повідомлень системи. Windows по черзі дістає повідомлення із черги повідомлень системи, досліджує їх, аби визначити вікно призначення, а потім переносить їх до черги повідомлень потоку, який створив вікно призначення. Черга повідомлень потоку отримує усі повідомлення миші і клавіатури для вікон, створених потоком. Потік виймає повідомлення із своєї черги і направляє Windows, щоб надіслати їх відповідній процедурі вікна для обробки.



За винятком повідомлення WM_PAINT, Windows завжди посилає повідомлення *в кінець черги* повідомлень. Це гарантує, що вікно отримує свої вхідні повідомлення в належному порядку (FIFO). Повідомлення WM_PAINT, проте, зберігається в черзі і пересилається до процедури вікна тільки, коли черга не містить ніяких інших повідомлень. Множинні повідомлення WM_PAINT для того ж вікна комбінуються в єдине повідомлення WM_PAINT, об'єднуючи всі недійсні частини клієнтської області в єдину області. Об'єднання повідомлень WM_PAINT скорочує число разів, коли вікно повинне перемальовувати вміст його клієнтської області.

Система переносить повідомлення у чергу повідомлень потоку, заповнюючи структуру MSG а потім копіюючи її в чергу повідомлень***. Інформація в MSG*** включає: дескриптор вікна, для якого повідомлення призначається, ідентифікатор повідомлення, два параметри повідомлення, час, коли повідомлення було перенесене, і позицію курсору миші. Потік може перенести повідомлення до своєї власної черги повідомлень або до черги іншого потоку, використовуючи функцію **PostMessage** або PostThreadMessage.

Програма може дістати повідомлення із своєї черги, використовуючи функцію **GetMessage**. Аби дослідити повідомлення без видалення його із черги, програма може використовувати функцію **PeekMessage**. Ця функція заповнює структуру MSG інформацією про повідомлення.

Після видалення повідомлення із своєї черги, програма може використовувати функцію **DispatchMessage**, щоб змусити Windows відправити повідомлення процедурі вікна для обробки. DispatchMessage бере вказівник на MSG, яка була заповнена попереднім викликом функції GetMessage або PeekMessage. DispatchMessage передає дескриптор вікна, ідентифікатор повідомлення, і два параметри повідомлення, в процедуру вікна, але не передає час, коли повідомлення було передане або позицію курсору миші. Програма може отримати цю інформацію, викликаючи функції GetMessageTime і GetMessagePos під час обробки повідомлення.

Потік може використовувати функцію WaitMessage, аби передати керування іншим потокам, коли він не має жодних повідомлень у своїй черзі. Функція призупиняє потік і не повертається, доки нове повідомлення не потрапить у чергу повідомлень потоку.

Програма повинна дістати і обробити повідомлення, занесені до черг повідомлень його потоків. *Однопотокова програма* зазвичай використовує цикл повідомлень у його функції WinMain, щоб діставати і відправляти повідомлення відповідним процедурам вікон для обробки. *Програми із кількома потоками* можуть включати цикл повідомлень у кожний потік, який створює вікно.

2.2. Позачергові повідомлення

Позачергові повідомлення відправляються негайно процедурі вікна призначення, оминаючи системну чергу повідомлень і чергу повідомлень потоку (викликається функція WindowProc). Windows зазвичай посилає позачергові повідомлення, щоб повідомити вікно про події, які впливають на нього. Наприклад, коли користувач активізує нове прикладне вікно, Windows посилає вікну серію повідомлень, у тому числі WM_ACTIVATE, WM_SETFOCUS, і WM_SETCURSOR. Ці повідомлення сповіщають вікно, що воно було активоване, що введення даних з клавіатури спрямоване вікну, і що курсор миші був переміщений в межах границь вікна. Позачергові повідомлення можуть також виникнути, коли програма викликає певну функцію Windows. Наприклад, Windows посилає повідомлення WM_WINDOWPOSCHANGED після того, як програма використовує функцію SetWindowPos, щоб перемістити вікно.

Існує також можливість відправити *повідомлення* безпосередньо *процедурі вікна*. Для цього використовують функцію **SendMessage()**, яка викликає процедуру вікна і чекає на результат. Повідомлення може бути відправлене будь-якому вікну в системі; все що потрібно це дескриптор вікна. Якщо дескриптор вікна невідомий, його можна знайти за допомогою функції **FindWindow()**, але потрібно знати ім'я класу вікна та ім'я вікна.

3. Аналіз та обробка повідомлень у WindowProc()

Процес декодування повідомлення, яке посилає Windows, зазвичай виконується за допомогою оператора вибору **switch** у функції **WindowProc()**, виходячи із значення повідомлення. Вибір необхідного повідомлення здійснюється за допомогою оператора **case** для кожного типу повідомлення. Типова структура такої конструкції подана нижче:

```
LRESULT CALLBACK WindowProc(HWND hwnd, UINT msg,
                            WPARAM wParam, LPARAM lParam)
{
      switch(msg)
      {
      case WM_LBUTTONDOWN:
            MessageBox(hWnd, "Left Button Pressed", "Mouse Message", MB_OK);
            break;
      case WM_CLOSE:
            DestroyWindow(hwnd);
            break;
      case WM_DESTROY:
            PostQuitMessage(0);
            break;
      default:
            return DefWindowProc(hwnd, msg, wParam, lParam);
      }
      return 0;
}
```

Кожна програма Windows має щось подібне у своєму коді, хоча це може бути прихованим від очей, наприклад, якщо ви пишете програми із використанням MFC або інших механізмів. У кожному випадку програма сприймає конкретне значення повідомлення і забезпечує відповідну обробку для цього повідомлення. Будь-які повідомлення, з якими програма не хоче мати справи, повинні бути повернуті назад у Windows, яка обробить їх за допомогою процедури, заданої за умовчанням. Повернення процедури у Windows здійснюється функцією **DefWindowProc()**.