Тема 8. Обробка виключень у Windows

План
1. Обробка помилок у Windows.
2. Структурна обробка виключень (SEH).
   2.1. Фреймова обробка виключень.
   2.2. Фінальна обробка виключень.
3. Векторна обробка виключень (VEH).

1. Обробка помилок у Windows

Більшість функцій Win32 API повертають код, за яким можна визначити, як завершилася функція: успішно чи ні. Якщо функція завершилася невдачею, то код повернення звичайно дорівнює FALSE, NULL або –1.

У цьому випадку функція Win32 API також встановлює внутрішній код помилки, який називається кодом останньої помилки (last-error code) і підтримується окремо для кожного потоку. Щоб отримати код останньої помилки, потрібно викликати функцію GetLastError, яка має наступний прототип:

DWORD GetLastError(VOID);

Ця функція повертає код останньої помилки, встановленої в потоці. Кожний потік встановлює свій код останньої помилки.

Код останньої помилки потоку можна встановити самостійно за допомогою функції SetLastError, що має наступний прототип:

VOID SetLastError(DWORD dwErrCode**);**

Щоб отримати текстове повідомлення, що відповідає коду останньої помилки, необхідно використовувати функцію FormatMessage, яка має наступний прототип:

```
DWORD FormatMessage(
  DWORD dwFlags,                // опції форматування
  LPCVOID lpSource,             // джерело повідомлень
  DWORD dwMessageId,            // числовий ідентифікатор повідомлення
  DWORD dwLanguageId,           // ідентифікатор мови повідомлення
  LPTSTR lpBuffer,              // вказівник на буфер для повідомлення
  DWORD nSize,                  // максимальний розмір буфера
  va_list *Arguments            // адреса списку параметрів для вставки
 );
```

Приклад використання функції FormatMessage:

```
LPVOID lpMsgBuf;

  FormatMessage(
    FORMAT_MESSAGE_ALLOCATE_BUFFER |
    FORMAT_MESSAGE_FROM_SYSTEM |
    FORMAT_MESSAGE_IGNORE_INSERTS,
    NULL,
    GetLastError(),
    MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT),
    (LPTSTR) &lpMsgBuf,
    0,
    NULL
  );
```

Потім можна вивести повідомлення за допомогою MessageBox() або printf().

1. Структурна обробка виключень (SEH)

Виключенням називається подія, яка сталася під час виконання програми, в результаті чого подальше нормальне виконання програми стає неможливим. Як правило, такі події є помилками в програмі. Тому для подальшої роботи програми потрібне або повер-

нення програми в робочий стан, або її аварійне завершення із звільненням усіх захоплених програмою ресурсів.

В операційних системах Windows для цієї мети призначений механізм структурної обробки виключень (Structured Exception Handling, SEH). SEH надає програмам потужний механізм реагування на непередбачені події:

- Апаратні несправності
- Виключення адресації
- Системні помилки
- Користувацькі виключення

Сенс механізму структурної обробки виключень полягає в наступному. У програмі виділяється блок програмного коду, в якому може відбутися виняток. Такий блок коду називається фреймом, а сам код називається охоронюваним кодом. Потім, після фрейму вставляється програмний блок, який обробляє виключення. Цей блок називається обробником виключення. Після обробки виключення управління передається першої інструкції, яка слідує за обробником виключення.

а) Фреймова обробка виключень

Очевидно, що для того щоб використовувати цей механізм у програмі, в мова програмування С + + потрібно ввести нові ключові слова. Такими ключовими словами є **__try** і **__except**, які розширюють список стандартних ключових слів мови програмування С++ і можуть відрізнятися для різних компіляторів. Ключове слово **__try** позначає фрейм, а ключове слово **__except** позначає обробник виключення. У результаті фрагмент програми, який використовує механізм структурної обробки виключень, виглядає наступним чином:

```
__try
{
      // код, який охороняється
}
__except (вираз-фільтр)
{
      // код обробки виключення
}
```

Тут вираз-фільтр є виразом мови програмування С + + і вказує на те, як повинна виконуватися програма після обробки виключення.

Виразом фільтру може бути:

одна із трьох літеральних констант;

виклик функції фільтру (filter function);

умовний вираз.

Обчислення цього виразу виконується відразу після виникнення виключення і має давати в результаті одне з наступних значень:

EXCEPTION_EXECUTE_HANDLER - управління передається обробнику виключення;

EXCEPTION_CONTINUE_SEARCH - система продовжує пошук обробника виключення (у випадку вкладених блоків);

EXCEPTION_CONTINUE_EXECUTION - система передає управління в точку переривання програми.

Для точного визначення типу виключення викликають функцію:

DWORD GetExceptionCode(VOID);

Ця функція повинна викликатись одразу після виникнення виключення – у виразі фільтру або обробнику виключення. GetExceptionCode() не може викликатись всередині функції фільтру, але може передаватись цій функції як один із параметрів, наприклад:

```
__except(FilterFunc(GetExceptionCode(), a)) {
…
}
```

Кількість кодів, які повертає функція GetExceptionCode() дуже велика, але їх можна умовно поділити на кілька категорій:

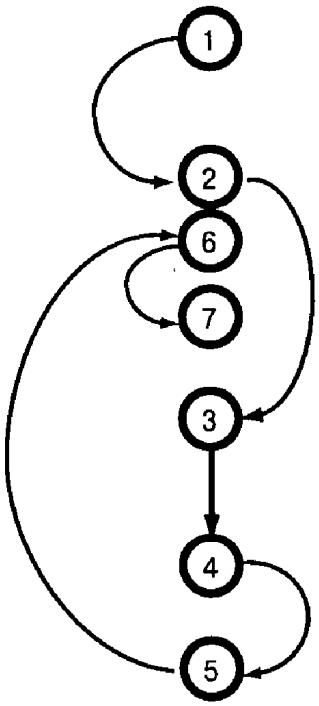
- Виконання програмою некоректних дій;
  EXCEPTION_ACCESS_VIOLATION (спроба читання/запису забороненої адреси),
  EXCEPTION_DATATYPE_MISALIGNMENT (порушення вирівнювання даних)
- Виключення, пов'язані з розподілом пам'яті;
  STATUS_NO_MEMORY, EXCEPTION_ACCESS_VIOLATION
- Коди виключень користувача, які генеруються функцією RaiseException();
- Коди арифметичних виключень;
  EXCEPTION_INT_DIVIDE_BY_ZERO
  EXCEPTION_FLT_OVERFLOW
- Виключення, які використовують відлагоджувальники;
  EXCEPTION_BREAKPOINT
  EXCEPTION_SINGLE_STEP

Більш детальну інформацію про виключення можна одержати за допомогою функції
LPEXCEPTION_POINTERS GetExceptionInformation (VOID);

Ця функція повертає вказівник на структуру EXCEPTION_POINTERS, яка, в свою чергу містить два вказівники на структури типу EXEPTION_RECORD та CONTEXT. Ці структури містять таку інформацію: код виключення, прапорець можливості продовження програми, адреса віртуальної пам'яті, контекст або вміст специфічних для процесора регістрів.

Послідовність обробки виключення

```
_try {
        ...
        i = j / 0;
        ...
}
_except (Filter (GetExceptionCode ())) {
        ...
}
        ...
DWORD Filter (DWORD ExCode)
{
        switch (ExCode) {
        ...
        case EXCEPTION_INT_DIVIDE_BY_ZERO:
        ...
        return EXCEPTION_EXECUTE_HANDLER;
        case ...
        }
}
```



1. Виникнення виключення.
2. Передача керування обробнику виключення. При обчисленні виразу фільтра спочатку викликається функція GetExeptionCode(), а її значення передається функції Filter.
3. Виконується функція фільтра.
4. Аналізується код виключення (EXCEPTION_INT_DIVIDE_BY_ZERO).
5. Функція фільтра повертає значення (EXCEPTION_EXECUTE_HANDLER).
6. Виконується код обробника виключень.
7. Керування передається за межі блоків **try** і **exept**.

b) Фінальна обробка виключень

В операційних системах Windows існує ще один спосіб обробки виключень, суть якого полягає в наступному. Код, при виконанні якого можливий викид виключення, як і у випадку з фреймовою обробкою виключень, включається в блок **__try**. Але тільки тепер за блоком __try слідує фінальний блок, позначений ключовим словом **__finally**. Цей блок виконується після блоку __try, незалежно від того, відбулося виключення чи ні. Такий спосіб обробки виключень називається фінальною обробкою виключень. Структурно фінальна обробка винятків виглядає наступним чином:

```
_try
{
// код, який охороняється
}
finally
{
// фінальный код
}
```

Фінальна обробка виключень використовується для звільнення ресурсів, зайнятих блоком __try (пам'ять, файли, дескриптори та ін.). З іншого боку фінальний обробник виконується завжди, а тому невідомо, чи сталось виключення чи ні. Для перевірки способу завершення попереднього блоку використовують функцію AbnormalTermination()

BOOL AbnormalTermination (VOID) ;

Завершення try-блоку може відбуватися з однієї з причин:

- досягнення кінця блоку (без виключення);
- виконання одного з операторів: return, break, goto, longjmp, continue, __leave;
- відбулося виключення.

Кожний try-блок може мати лише один обробник виключення: або __exept або __finally.

3. Векторна обробка виключень

Починаючи з Windows XP з'явилась векторна обробка виключень (Vectored Exeption Handling або VEH). VEH є розширенням структурної обробки виключень. Програма може зареєструвати функцію для спостереження або обробки усіх виключень для програми. Векторні обробники не засновані на фреймах, тому можна додати обробник, який буде викликатися незалежно від того, чи ви знаходитесь всередині певного фрейму. Векторні обробники викликаються в тому порядку, в якому вони були додані в систему за допомогою спеціальної функції. Векторні обробники викликаються ще до того як система почне шукати відповідний структурний обробник.

Кілька векторних обробників можуть бути зв'язані в ланцюжки. Векторні обробники додаються за допомогою функції PVOID WINAPI **AddVectoredExceptionHandler**(

ULONG FirstHandler, PVECTORED_EXCEPTION_HANDLER VectoredHandler);