МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Озерский технологический институт — филиал НИЯУ МИФИ

Кафедра прикладной математики

Вл. Пономарев

ПРАКТИКУМ

по системному программированию

Учебно-методическое пособие

2019 г.

УДК 681.3.06
П 56

Вл. Пономарев. Практикум по системному программированию. Учебно-методическое пособие. Озерск: ОТИ НИЯУ МИФИ, 2019. — 83 с.

В пособии описываются задания практических работ по дисциплине «Системное программирование». Работы включают в себя обработку ошибок и исключений, управление памятью, управление реестром и доступом, статические и динамические загружаемые библиотеки, взаимодействие между процессами (буфер обмена, Data Copy, DDE, почтовые ящики и каналы), сервисы NT.

В качестве основного материала при выполнении практических работ пособие предназначено для студентов, обучающихся по направлению подготовки 09.03.01 «Информатика и вычислительная техника» и специальности 09.05.01 «Применение и эксплуатация автоматизированных систем специального назначения».

Рецензенты:
1.
2.

УТВЕРЖДЕНО
Редакционно-издательским
Советом ОТИ НИЯУ МИФИ

Содержание

Общие цели занятий

В ходе практических работ изучаются следующие темы:

1) обработка ошибок;
2) обработка исключений;
3) реестр Windows;
4) атрибуты безопасности;
5) управление памятью;
6) динамические загружаемые библиотеки;
7) каналы связи между процессами (pipes) и почтовые ящики;
8) сервисы NT.

Программирование ведется в среде Microsoft® Visual Studio®, язык программирования C/C++.

На выполнение каждой работы предположительно отводится 2 академических часа. Успешное усвоение изучаемого материала возможно при условии, что студент самостоятельно находит необходимую техническую информацию, используя конспекты лекций, дополнительную литературу, сеть Интернет.

Каждая выполненная работа должна быть защищена.

Для защиты знаний и навыков, полученных в ходе выполнения практических работ студент должен иметь тетрадь (12-18 листов). Для каждой выполненной работы в тетрадь записывается отчет.

Отчет по работе начинается с заголовка:

1. Фамилия Имя Отчество.
2. Группа.
3. Дата начала выполнения работы.
4. Код работы.
5. Название работы.
6. Цели работы.
7. Задачи работы.

При необходимости при выполнении работы в отчет записываются контрольные значения. Во время защиты тетрадь используется для вопросов преподавателя и ответов студента.

1. Работа SY-101. Строки Windows

Цели:
- изучение символьных и строковых типов данных Windows.
Задачи:
- изучение символьных типов;
- изучение строковых типов;
- макросы преобразований строковых типов;
- функции перекодировки строковых данных.
Опорные документы:
[1, «Символы и строки Windows»]

В настоящей редакции пособия данная работа не описывается.

1.1. Контрольные вопросы и упражнения

1. Назовите и охарактеризуйте символьные типы Windows.
2. Поясните, как расшифровывается тип вида LPCTSTR, из каких элементов он складывается, что означает.
3. Назовите функции преобразования строк многобайтных символов в строки Unicode и наоборот, поясните параметры.

2. Работа SY-102. Обработка ошибок

Цели:
- формирование сообщений об ошибках.
Задачи:
- изучение функции MessageBox;
- изучение функции FormatMessage;
- генерирование системных ошибок.
Опорные документы:
[1, «Обработка ошибок»]

2.1. Шаблон проекта

Получите архив работы SY-102. Извлеките каталог errmsg в корневой каталог диска C:. Откройте проект. Убедитесь, что целевой платформой является x86 или Win32. Работа выполняется в модуле errmsg.cpp. В модуле errmsg.h могут быть подключены необходимые библиотеки. В начале модуля errmsg.cpp укажите сведения об организации, о себе, о проекте, дату начала работы:

```
// ОТИ НИЯУ МИФИ
// 1ПО-00Д
// Студент Имя Отчество
// Системное программирование
// SY-102. Обработка ошибок
// 01.01.2000
```

2.2. Функция MessageBox

Функция MessageBox на самом деле является макросом, который расширяется в функцию MessageBoxA, или в функцию MessageBoxW, в зависимости от настроек проекта. Убедимся, что в настройках проекта выбран многобайтный набор символов.

Тесты функции MessageBox выполняются в отдельных функциях, которые следует включать и выключать в функции main.

2.2.1. Тест 1-1. Вывод простого сообщения

Тест программируется в функции icons модуля errmsg.cpp.

Содержание теста:

- при помощи функции MessageBox выведем сообщения, имеющие следующий вид: «Сообщение номер # - icon».

В сообщении вместо # вставим порядковый номер, а вместо icon вставим слово XXXX константы вида MB_ICONXXXX. Используем все такие константы, которые подскажет среда разработки.

В окне сообщения только кнопка Ok и заданный значок.

2.2.2. Тест 1-2. Вывод сообщения из буфера

Тест программируется в функции frombuf модуля errmsg.cpp.

Содержание теста:

- Создадим локальный буфер bufa типа char, запишем в него сообщение «Сообщение из буфера ANSI».

- Создадим локальный буфер bufw типа wchar_t, запишем в него сообщение «Сообщение из буфера Unicode».

- Выведем сообщение буфера bufa с заголовком «ANSI» при помощи функции MessageBoxA.

- Выведем сообщение буфера bufw с заголовком «Unicode» при помощи функции MessageBoxW.

В окне сообщения только кнопка Ok и восклицательный знак.

2.2.3. Тест 1-3. Формирование запроса с двумя ответами

Тест программируется в функции yesno модуля errmsg.cpp.

Содержание теста:

- Создадим локальный буфер bufa типа char размером 64.

- Сформируем в буфере bufa сообщение «Удалить файл file-name?», в котором вместо file-name записано содержание буфера file_to_delete. Используем для этой цели функцию sprintf.

- Выведем сообщение буфера при помощи функции MessageBox.

Окно сообщения имеет кнопки «Да» и «Нет», и вопросительный знак.

Кнопкой по умолчанию является кнопка «Нет».

Заголовок окна «Подтвердите».

- Проанализируем результат вопроса.

Если пользователь нажал кнопку «Да», выведем сообщение

«Файл удален»

Если пользователь нажал кнопку «Нет», выведем сообщение

«Отменено пользователем».

Используем функцию MessageBox, заголовок окна «Результат».

2.2.4. Тест 1-4. Формирование запроса с тремя ответами

Тест программируется в функции yesnocancel модуля errmsg.cpp.

Содержание теста:

- Создадим локальный буфер bufa типа char размером 64.

- Сформируем в буфере bufa сообщение «Удалить файл file-name?», в котором вместо file-name записано содержание буфера file_to_delete. Используем для этой цели функцию sprintf. Эту часть кода можно скопировать из предыдущей функции.

- Организуем цикл вывода окна сообщения об удалении файла, такого же, как в предыдущем тесте, но с кнопками «Да», «Нет» и «Отмена». Кнопкой по умолчанию является кнопка «Отмена».

При нажатии кнопки «Да» выводится сообщение «Файл удален», и цикл опроса продолжается. При нажатии кнопки «Нет» выводится сообщение «Отменено пользователем», и цикл опроса продолжается. При нажатии кнопки «Отмена» цикл завершается, сообщений не выводится.

2.3. Функция FormatMessage

Функция FormatMessage формируем буфер с сообщением, которое затем выводится при помощи функции MessageBox. Исходное сообщение, которое подвергается форматированию, может находиться в системном модуле, в модуле пользователя, или в буфере пользователя. Необходимость форматирования исходного сообщения особенно очевидна в случае, когда сообщение имеет подставляемые параметры.

2.3.1. Тест 2-1. Формирование системного сообщения

Тест выполняется в функции format_sys_msg модуля errmsg.cpp. Дополнительно описывается функция format_from_system.

Буфер для форматированного сообщения формирует система. Для этого следует указать флаг FORMAT_MESSAGE_ALLOCATE_BUFFER. Исходное сообщение находится в системной dll, поэтому дополнительно следует указать флаг FORMAT_MESSAGE_FROM_SYSTEM. Функция форматирования вызывается в функции format_from_system, параметрами которой являются код ошибки и идентификатор (код) локали. Эта же функция выводит сформатированное сообщение при помощи функции MessageBox.

Функция format_from_system вызывается в функции format_sys_msg, которая формирует код ошибки и идентификатор локали.

Содержание теста:

1. Функция format_from_system.

- Описываем указатель на буфер типа HLOCAL.

- Описываем вызов функции FormatMessage, используя в качестве примера код, приведенный в опорном документе.

- Описываем результат вызова функции FormatMessage.

Если указатель на буфер разрешен, при помощи функции MessageBox выводим сообщение, окно которого содержит кнопку Ok и значок критической ошибки.

Если указатель на буфер не разрешен, выводим в консоль сообщение с кодом последней ошибки, возвращаемым функцией GetLastError, и при помощи функции MessageBox выводим сообщение «Error FormatMessage».

2. Функция format_sys_msg.

- Сначала устанавливаем код последней ошибки равным константе ERROR_FILE_NOT_FOUND, при помощи функции SetLastError. Таким образом, функция моделирует ситуацию ошибки, возникающей при указании неверного пути или несуществующего файла.

- Описываем переменную dwError типа DWORD и считываем в нее код последней ошибки при помощи функции GetLastError.

- Вызываем функцию format_from_system. Первым параметром передаем переменную dwError. Чтобы сформировать второй параметр, используем макрос MAKELANGID, как указано в опорном документе.

3. Тестируем написанный код.

- В функции main описываем вызов функции format_sys_msg.

- Запускаем программу, убеждаемся, что выводится сообщение на русском языке. Если вместо этого в консоль выводится сообщение с кодом ошибки, устраняем ошибки в программе.

- В функции format_sys_msg дополнительно описываем два вызова функции format_from_system, подставляя в качестве второго параметра сначала константу 1049, затем константу 1033. Первая константа является десятичным значением идентификатора русской локали, вторая константа является десятичным значением идентификатора американской локали.

- В функции format_sys_msg дополнительно описываем вызов функции format_from_system, подставляя в качестве первого параметра константу 0, а в качестве второго параметра константу 1049.

2.3.2. Тест 2-2. Формирование сообщения с параметрами

Тест выполняется в функции format_sys_msg модуля errmsg.cpp. Дополнительно описывается функция format_with_param.

Целью данного теста является изучение сообщений с параметрами. Сначала найдите файл winerror.h. Откройте его при помощи, например, редактора FAR, найдите константу ошибки ERROR_WRONG_DISK, код ошибки которой равен 34. Обратите внимание на сообщение об ошибке:

```
// MessageId: ERROR_WRONG_DISK
//
// MessageText:
//
// The wrong diskette is in the drive.
// Insert %2 (Volume Serial Number: %3) into drive %1.
//
#define ERROR_WRONG_DISK                 34L
```

В этом сообщении содержится три подставляемых параметра, обозначенных знаком процента и номером. Первым параметром является название диска, который требуется вставить, вторым параметром является имя дисковода, третьим параметром является серийный номер диска. Чтобы сформировать это сообщение, необходимо передать эти три параметра в указанном порядке.

Содержание теста.

1. Функция format_with_param.

- Описываем указатель на буфер типа HLOCAL.

- Описываем вызов функции FormatMessage, используя в качестве примера код, приведенный в опорном документе.

- Описываем результат вызова функции FormatMessage.

Если указатель на буфер разрешен, при помощи функции MessageBox выводим сообщение, окно которого содержит кнопку Ok и значок критической ошибки.

Если указатель на буфер не разрешен, выводим в консоль сообщение с кодом последней ошибки, возвращаемым функцией GetLastError, и при помощи функции MessageBox выводим сообщение «Error FormatMessage».

2. Функция format_sys_msg.

- вызываем функцию format_with_param, вызовы других функций выключаем при помощи комментария.

3. Тестируем формирование сообщения с параметрами.

Примечания.

1. В вызове функции FormatMessage дополнительно следуем указать флаг FORMAT_MESSAGE_ARGUMENT_ARRAY, так как нам необходимо передать вставляемые в сообщение параметры.

2. Поскольку функции format_with_param не передаются параметры:

- в качестве кода ошибки передаем константу ошибки;

- в качестве идентификатора локали передаем константу 1049.

3. Формирование массива параметров для простоты выполняется непосредственно в функции format_with_param, примерно так, как показано в следующем фрагменте кода:

```
DWORD_PTR params[] = {
    (DWORD_PTR) "A",
    (DWORD_PTR) "diskette",
    (DWORD_PTR) "0000-1111",
};
```

4. Параметры подставляются в вызов функции FormatMessage последним параметром, переменную params нужно привести к типу va_list*.

2.3.3. Тест 2-3. Формирование сообщения из модуля

Тест выполняется в функции format_dll_msg модуля errmsg.cpp.

В функции format_dll_msg опишите форматирование и вывод сообщения, номер которого задается параметром этой функции. Есть некоторые особенности в вызове функции FormatMessage.

1. Вместо флага FORMAT_MESSAGE_FROM_SYSTEM нужно использовать флаг FORMAT_MESSAGE_FROM_HMODULE.

2. Перед вызовом функции FormatMessage нужно загрузить в память модуль библиотеки kernel32.dll:

```
HMODULE hModule = LoadLibrary("kernel32.dll");
```

После загрузки нужно убедиться в отсутствии ошибки.

3. Переменная hModule используется в качестве второго параметра вызова функции FormatMessage.

4. Идентификатор локали задается константой 1049.

Для тестирования в основной функции main нужно вызвать функцию format_dll_msg, подставляя в качестве параметра, например, константу 5.

Заметим, что многие значения кодов ошибок этого модуля совпадают с кодами файла winerror.h.

2.3.4. Тест 2-4. Формирование сообщения из буфера

Тест выполняется в функции format_from_string модуля errmsg.cpp.

Задача этого теста — научиться формировать сообщения, содержание которых задает пользователь при помощи буфера (строки). Отформатировать простое сообщение из буфера не имеет никакого смысла, так как буфер можно вывести в сообщение без его форматирования. Поэтому форматирование сообщения из буфера используется тогда, когда сообщение изменяет свой вид в зависимости от подставляемых параметров.

В самом общем виде вставляемый параметр помечается при помощи знака процента и порядкового номера параметра, обозначаемого числом от единицы до 99, например, %1. За этой меткой может располагаться формат, окруженный восклицательными знаками, например, %1!s!.

Для выполнения теста скопируйте код функции format_with_param в функцию format_from_string. В начале функции объявите буфер для сообщения, изначально содержащий какой-нибудь текст:

```
char buf[] = "Error %1 occurred";
```

Из функции format_with_param скопируйте описание параметров сообщения params, вставьте его в начале функции format_from_string.

Вызов функции FormatMessage нужно немного изменить.

1. Вместо флага FORMAT_MESSAGE_FROM_SYSTEM нужно использовать флаг FORMAT_MESSAGE_FROM_STRING.

2. В качестве источника сообщения укажите буфер buf.

3. Номер ошибки не задается, параметр равен нулю.

4. Идентификатор локали не имеет смысла, параметр равен нулю.

Вызовите функцию format_from_string в основной функции main.

Если все запрограммировано правильно, то запуск программы выведет сообщение «Error A occurred».

Попробуйте заменить номер параметра в буфере с 1 на 2. Запустите программу, убедитесь, что выводится сообщение «Error diskette occurred». Верните назад в буфере buf номер параметра 1.

Поэкспериментируем с форматированием.

Используем формат *.*s. Он означает, что требуется указать два числа, обозначенных *.*, и строку, которая будет форматироваться.

Согласно спецификации %s, первое число обозначает число пробелов в начале вывода. Второе число обозначает количество выводимых знаков.

Пусть эти числа будут 0 и 4, а строкой будет слово diskette.

Тогда нужно изменить как передаваемые параметры, так и спецификацию в буфере:

```
char buf[] = "Error %1!*.*! occurred";
DWORD_PTR params[] = {
    (DWORD_PTR) 0,
    (DWORD_PTR) 4,
    (DWORD_PTR) "diskette",
};
```

В результате форматирования сообщения с указанными параметрами должно выводиться сообщение «Error disk occurred». Следует обратить внимание, что мы передаем три параметра, и все они используются, хотя в буфере указан только параметр 1. Первый параметр подставляется вместо первой звездочки, второй — вместо второй звездочки, третий — вместо s. Фактически задан формат "%0.4s".

Попробуем вывести число по спецификации %d с ведущими нулями. Строка в буфере должна содержать формат "%1!0*d!". Используются два первых передаваемых параметра, равные, например, 3 и 3. При этом должно выводиться сообщение «Error 003 occurred».

Запишите в отчет используемые форматы, подставляемые параметры, содержание буфера и выводимые результаты.

2.4. Контрольные вопросы и упражнения

1. Какие значения могут возвращать системные функции?
2. Опишите структуру кода ошибки.
3. Как установить или прочитать код последней ошибки?
4. Где содержится описание ошибки?
5. Какие параметры могут быть заданы для функции MessageBox?
6. Как задаются сообщения об ошибках с параметрами?
7. Как форматируются сообщения с параметрами?
8. Как форматируются сообщения из буфера пользователя?
9. Сформируйте сообщение «This diskette is a disk», используя два раза подставляемый параметр diskette и дополнительные параметры.

3. Работа SY-103. Обработка исключений

Цели:
- изучение структурированной обработки исключений Windows.

Задачи:
- изучение блока завершения;
- изучение блока обработчика;
- генерирование исключений.

Опорные документы:
[1, «Обработка исключений»]

3.1. Шаблон проекта

Получите архив работы SY-103. Извлеките каталог except в корневой каталог диска C:. Откройте проект. Убедитесь, что целевой платформой является x86 или Win32. Откройте свойства проекта, убедитесь, что структурированная обработка исключений включена. Раздел C/C++, Code Generation, параметр Enable C++ Exceptions, значение Yes with SEH Exceptions.

Работа выполняется в модуле except.cpp. В начале модуля укажите сведения об организации, о себе, о проекте, дату начала работы:

```
// ОТИ НИЯУ МИФИ
// 1ПО-00Д
// Студент Имя Отчество
// Системное программирование
// SY-103. Обработка исключений
// 01.01.2000
```

3.1.1. Обработчик основной функции

В начале модуля except.cpp опишем константу ошибки:

```
#define EXCEPT_MALLOC 0xE0000001
```

В основной функции опишем обработчик исключительных ситуаций, которые могут возникать в ходе выполнения других функций. Блок except использует в качестве фильтра константу EXCEPTION_EXECUTE_HANDLER, предписывающую выполнить код обработчика.

Обработчик выполняет следующие действия:

1) При помощи функции GetExceptionCode получает код ошибки в локальную переменную code типа unsigned int.

2) Выводит сообщение об исключении.

Для этого обработчик анализирует код ошибки, и для некоторых из кодов выводит соответствующие сообщения, а для других выводит общее сообщение "Exception raised КОД ОШИБКИ in mainf". Здесь КОД ОШИБКИ — это шестнадцатеричный код ошибки, которая привела к исключению. Используем для вывода спецификацию %X.

Примерный вид обработчика следующий:

```
} __except (EXCEPTION_EXECUTE_HANDLER) {
    unsigned int code = GetExceptionCode();
    switch (code) {
    case EXCEPTION_INT_DIVIDE_BY_ZERO:
        printf("Division by zero raised 0x%X in main\n", code);
        break;
    case EXCEPTION_INT_OVERFLOW:
        printf("Integer overflow raised 0x%X in main\n", code);
        break;
    case EXCEPTION_ACCESS_VIOLATION:
        printf("Access violation raised 0x%X in main\n", code);
        break;
    case EXCEPTION_STACK_OVERFLOW:
        printf("Stack overflow raised 0x%X in main\n", code);
        break;
    default:
        printf("Exception raised 0x%X in main\n", code);
    }
}
```

Для тестирования обработчика в блоке try возбудим исключение с кодом ошибки EXCEPT_MALLOC при помощи функции RaiseException:

```
RaiseException(EXCEPT_MALLOC, 0, 0, 0);
```

Запустим программу, убедимся, что в консоль выводится:

```
Exception raised 0xE0000001 in mainf
```

Обратим внимание на то, чтобы шестнадцатеричная константа была выведена именно так, как показано.

Тестирующие функции, которые описываются в ходе выполнения работы, должны вызываться в блоке try основной функции mainf.

3.2. Обработчик завершения

В этой части изучается использование блока завершения. Так как выполнение блока завершения гарантируется, в нем удобно размещать код, который освобождает выделенную память или другие ресурсы.

Тест выполняется в функции memory_alloc.

В начале модуля опишем переменную mem_alloc типа char* с нулевым начальным значением. Эта переменная предназначена для хранения адреса памяти, выделенного функцией malloc.

Основу функции memory_alloc составляет блок try с блоком finally.

Функции по ссылке передается параметр size типа size_t. В блоке try функция пытается выделить блок памяти размером 1 + size при помощи функции malloc. Это ей удается или не удается.

Если выделить блок памяти удается, функция выводит сообщение "Memory allocated at 0x12345678", где 0x12345678 — адрес блока выделенной памяти. Используйте для вывода адреса спецификацию %p.

Если выделить блок памяти не удается, функция возбуждает исключение EXCEPT_MALLOC при помощи функции RaiseException.

В конце блока try функция выполняет деление значения 1 000 000 000 на параметр size, результат присваивается той же переменной size. Если функции будет передано нулевое значение size, и блок памяти будет успешно выделен, операция вызовет исключение.

Память, выделенная malloc, должна быть освобождена. Для этой цели используется блок завершения finally. Этот блок будет выполнен в любом случае, и при выделении памяти, и при отказе в выделении памяти.

В начале блока finally выводим сообщение "Memory alloc finally".

Затем проверяем, была выделена память, или нет, сравнивая значение переменной mem_alloc с нулем. Если память была выделена, освобождаем ее при помощи функции free, и выводим сообщение «Memory freed».

В начале функции mainf опишем переменную result типа unsigned int, с начальным значением 1 000 000 000. Функцию memory_alloc вызываем в блоке try, передаем функции переменную result. В конце функции mainf выводим сообщение "result = ЗНАЧЕНИЕ", где ЗНАЧЕНИЕ — значение переменной result, используем спецификацию %u.

3.2.1. Тест 3-1. Выделение памяти

Запишем в отчет название теста.

Тест выполняется три раза с разными начальными значениями переменной result. В первом раунде значение равно 1000000000, во втором раунде значение равно нулю, в третьем раунде значение равно 3000000000.

Запускаем программу три раза с разными начальными значениями, записываем в отчет дословно все, что программа выводит в консоль. Поясняем в отчете действия программы отдельно для каждого раунда.

Заметим, что этот вариант освобождения блока памяти или другого ресурса является не единственным. Функция может иметь обработчик исключения, после которого выполняется освобождение ресурсов. Но в этом случае нужно гарантировать, чтобы обработчик не вызывал другого исключения и был обязательно выполнен.

3.3. Тест 3-2. Переполнение стека

Тест выполняется в функции stackov. Вызываем эту функции в блоке try функции main вместо предыдущей функции, передаем ей в качестве параметра значение переменной result. Значение переменной result установим равным нулю.

Описываем функцию stackov:

```
void stackov(unsigned int & level) {
    if (++level > 0) stackov(level);
}
```

Записываем в отчет название теста.

Запускаем программу, записываем в отчет все, что программа выводит в консоль.

3.4. Тест 3-3. Нарушение доступа к памяти

Тест выполняется в функции violation. Вызываем эту функции в блоке try функции main вместо предыдущей функции, передаем ей в качестве параметра значение переменной result. Значение переменной result установим равным нулю.

Описываем функцию violation:

```
void violation(unsigned int address) {
    ((char*)address)[0] = 0;
}
```

Записываем в отчет название теста.

Запускаем программу, записываем в отчет все, что программа выводит в консоль.

3.5. Тест 3-4. Фильтрующая функция

Тест выполняется в функции violationf.

Нарушение доступа к памяти является единственным исключением, имеющим параметры. Первый параметр указывает операцию: 0 — попытка чтения, 1 — попытка записи. Второй параметр указывает адрес.

Описываем функцию фильтра следующим образом:

```
int filter(unsigned int code, PEXCEPTION_POINTERS ep) {
    if (code == EXCEPTION_ACCESS_VIOLATION) {
        int op = ep->ExceptionRecord->ExceptionInformation[0];
        int ad = ep->ExceptionRecord->ExceptionInformation[1];
        if (op == 0) {
            printf("Violation to read at 0x%p\n", (void*)ad);
        } else {
            printf("Violation to write at 0x%p\n", (void*)ad);
        }
    }
    return EXCEPTION_EXECUTE_HANDLER;
}
```

Переходим к функции violationf.

В этой функции нужно описать обработчик, который использует фильтрующую функцию. Параметрами фильтрующей функции являются вызовы функций GetExceptionCode и GetExceptionInformation.

В блок try функции violationf вписываем тот же код, что и в функции violation, то есть попытку записи в память.

В функции mainf задаем начальное значение result, равное 16. Вызываем функцию violationf в блоке try функции mainf, передавая ей в качестве параметра переменную result.

Записываем в отчет теста и условия теста.

Запускаем программу.

Записываем в отчет все, что программа выводит в консоль.

3.6. Целочисленные операции

Обычным образом выявить переполнение целочисленной арифметической операции нельзя. Процессор генерирует прерывание 4 в ситуации, когда получает результат, не умещающийся в целевой регистр. Это прерывание ведет к исключению EXCEPTION_INT_OVERFLOW. Фактически прерывание 4 возникает только при выполнении машинных команд DIV или IDIV, когда в регистрах задано неправильное соотношение делимого и делителя.

Во всех других случаях прерывание 4 не генерируется, но, если нужно проконтролировать переполнение, то его можно вызвать, используя машинную команду INTO. Если флаг переполнения установлен, эта команда вызывает прерывание 4, что ведет к исключению.

Тест выполняется в функции integer_overflow.

Эту функцию вызываем в блоке try функции mainf вместо предыдущей функции, и передаем в качестве параметра переменную result.

3.6.1. Тест 3-5. Целочисленное переполнение

Рассмотрим обычное сложение. Сейчас в функцию передается значение 3000000000, если это не так, то установим значение переменой result функции mainf, равное 3000000000.

Запишем в отчет название теста.

Раунд 1.

Запишем условия теста: сложение двух значений, сумма которых превышает диапазон представления unsigned int.

В функции integer_overflow описываем сложение числа number с самим собой, результат присваиваем переменной number.

Запускаем программу, записываем в отчет все, что выводит программа в консоль, анализируем результат, и в отчете даем ему объяснение.

Раунд 2.

Запишем в отчет условия теста: попытка вызвать прерывание 4.

После выполнения операции сложения пробуем вызывать прерывание 4 при помощи машинной инструкции INTO:

```
number += number;
_asm {
    into;
}
```

Запускаем программу, записываем в отчет все, что выводит программа в консоль, анализируем результат, и в отчете даем ему объяснение.

Раунд 3.

Следующий код записываем перед имеющимся. Цель — попробовать выявить переполнение при сложении двух чисел диапазона short.

Объявим в начале функции переменную x типа short, начальное значение равно 32000. Следующий оператор вычисляет новое значение x как сумму двух значений x и x.

Следующий оператор — вставка кода на ассемблере, команда INTO.

Следующий оператор присваивает переменной number значение переменной x. Далее в коде функции идет написанный ранее код.

Запишем в отчет условия теста: попытка выявить переполнение при сложении целых чисел диапазона short.

Запускаем программу, записываем в отчет все, что она выводит в консоль.

Выполняем код функции integer_overflow шаг за шагом, все вычисляемые значения записываем в отчет, пробуем найти какое-нибудь объяснение наблюдаемым вычислениям.

3.7. Вещественные операции

Получить исключение вещественной операции не так просто, потому что вещественные операции выполняются сопроцессором. Для управления сопроцессором используются две функции.

Первая функция, _controlfp, устанавливает слово состояния сопроцессора и маску, задающую исключения и режимы работы сопроцессора.

Вторая функция, _clearfp, очищает слово состояния и возвращает его текущее состояние. Биты в возвращаемом слове показывают исключительные ситуации в сопроцессоре.

Кроме всего прочего, нужно понимать, какие состояния могут возникнуть при вычислениях, а их немало: переполнение и антипереполнение (underflow), денормализация и некорректный результат, множественные ошибки, переполнение стека и другие. Выбрасываемые исключения зависят также от настроек и версии компилятора, платформы и процессора. Современная практика вещественных вычислений не подразумевает обработку исключений вещественных операций. Вместо этого в случае ошибок вещественные переменные получают специальные значения inf и nan, обозначающие бесконечность (переполнение) и нечисловое значение, то есть значение, которое нельзя использовать как числовое в дальнейших вычислениях. В любом случае обработка исключений вещественных операций является чрезвычайно сложной темой.

Наиболее простые исключительные состояния вычислений с плавающей запятой — это переполнение, антипереполнение и денормализация. Переполнение означает выход порядка за границы разрядной сетки, либо в положительную, либо в отрицательную область. Денормализация означает потерю значащих цифр мантиссы.

Исключительные ситуации в сопроцессоре возникают, когда числа приближаются к границам диапазонов. Вычисления должны быть поистине астрономическими, чтобы оперировать с числами, десятичный порядок которых превышает три сотни. Практически такой результат может возникнуть, если используются вычисления в неправильно построенном цикле, или же с неверно заданными исходными данными.

3.7.1. Функция анализа результата операции

Следующие константы определяют биты результата, возвращаемого функцией _clearfp:

_EM_INEXACT — некорректный результат
_EM_UNDERFLOW — антипереполнение
_EM_OVERFLOW — переполнение
_EM_ZERODIVIDE — деление на ноль
_EM_DENORMAL — денормализация результата
_EM_INVALID — неправильная операция

Проверить один бит можно при помощи следующего кода:

```
if (result & _EM_INEXACT) {
    printf("Inexact\n");
}
```

Такая конструкция должна быть написана для каждой из констант.

Описываем код анализа в функции fp_result, параметром result которой является результат функции _clearfp.

Сначала в этой функции выводим значение параметра result в шестнадцатеричном виде, затем по очереди проверяем каждую из констант, и выводим соответствующий константе текст.

3.7.2. Тест 3-6. Ошибки вещественных операций

Тесты выполняются в функции float_error, которую нужно вызвать в функции main вместо функции mainf.

Запишем в отчет название теста.

Опишем переменную z типа float, начальное значение равно FLT_MAX, переменную y типа float, начальное значение равно FLT_MIN. Это границы диапазона представления. Опишем переменную x типа float, начальное значение равно 0.0f.

В каждом раунде выполняется одна операция, результат которой записывается в переменную x.

После операции вызываем функцию fp_result, в нее подставляем вызов функции _clearfp.

После этого в консоль выводим значение x по спецификации %g.

Раунд 1.

x = z + z

Запишем в отчет условия теста.

Запускаем программу F5, записываем в отчет все, что программа выводит в консоль.

В этом раунде функция fp_result должна вывести два текстовых значения, Inexact и Overflow. Если это не так, возможно, неправильно выполнен анализ в функции fp_result.

Раунд 2.

x = –z – z

Запускаем программу F5, записываем в отчет вывод в консоль.

Раунд 3.

x = y * y

Запускаем программу F5, записываем в отчет вывод в консоль.

Раунд 4.

x = y * 1.f

Запускаем программу F5, записываем в отчет вывод в консоль.

Раунд 5.

x = sqrt(–1)

Запускаем программу F5, записываем в отчет вывод в консоль.

3.7.3. Тест 3-7. Исключения вещественных операций

Тесты выполняются в функции float_except.

В начале этой функции описываем те же переменные, что и в функции float_error. Их можно просто скопировать.

Далее нужно установить слово состояния сопроцессора при помощи функции _controlfp. В качестве маски используем константу _MCW_EM, которая равна сумме всех использованных ранее констант:

```
_controlfp(0, _MCW_EM);
```

Далее нужно описать два совершенно одинаковых обработчика:

```
__try {
} __except(EXCEPTION_EXECUTE_HANDLER) {
    fp_result(_clearfp());
    DWORD code = GetExceptionCode();
    switch (code) {
    case STATUS_FLOAT_OVERFLOW:
        printf("Except Overflow\n");
        break;
        . . .
    default:
        printf("Exception raised 0x%X in test\n", code);
    }
}
```

В качестве примера здесь описано только исключение переполнения.

Дополнительно нужно описать антипереполнение, денормализацию и неточный результат. Соответствующие константы ошибок:

STATUS_FLOAT_UNDERFLOW — антипереполнение.

STATUS_FLOAT_DENORMAL_OPERAND — денормализация.

STATUS_FLOAT_INEXACT_RESULT — неточный результат.

В блоке try второго обработчика описываем вывод значения переменной x по спецификации %g. В блоке try первого обработчика по очереди будем подставлять операции, использованные в предыдущем тесте.

Запишем в отчет название теста.

Выполняем 4 первых раунда предыдущего теста.

Записываем в отчет условия раунда и вывод в консоль.

3.8. Контрольные вопросы и упражнения

1. Что называется исключительной ситуацией?
2. Что называется исключением?
3. Что такое структурная обработка исключений?
4. Приведите примеры аппаратных исключений.
5. Как генерируются программные исключения.
6. Чем блок завершения finally отличается от блока except?
7. Что называют фильтром исключения?
8. Какие значения может возвращать фильтр исключения? Что означают эти значения?
9. Какие формы может принимать фильтр исключения?
10. Когда возникает переполнение целочисленной операции?
11. Какие исключения выбрасывает процессор плавающей запятой?
12. Что означают: переполнение, антипереполнение, денормализация?
13. Каков диапазон представления вещественных чисел одинарной точности, двойной точности?

4. Работа SY-104. Реестр Windows

Цели:
- изучение реестра Windows.

Задачи:
- запись и чтение информации из файлов инициализации;
- запись и чтение информации из реестра.

Опорные документы:
[1, «Реестр Windows»]

4.1. Шаблон проекта

Войдите в систему под ограниченной учетной записью (student).

Получите архив работы SY-104. Извлеките каталог regina в корневой каталог диска C:. Откройте проект. Убедитесь, что целевой платформой является x86 или Win32. В проекте несколько модулей.

Модуль regina.cpp описывает оконное приложение. Модуль regina.h описывает константы, переменные, типы. Модуль util.h содержит вспомогательные функции. Другие модули используются в других работах.

Основным модулем этой работы является модуль rega.h. Укажите в начале этого модуля сведения об организации, о себе, о проекте, дату начала работы:

```
// ОТИ НИЯУ МИФИ
// 1ПО-00Д
// Студент Имя Отчество
// Системное программирование
// SY-104. Реестр Windows
// 01.01.2000
```

4.2. Проект regina

Программа regina представляет собой оконное приложение диалогового типа. С программой связан документ — сведения, которыми управляет программа. Документ представлен на диске файлом двоичного типа. Программа сохраняет документ в файл, и читает данные из файла.

Данные программы описывает следующая структура, модуль regina.h:

```
// структура документа приложения regina
struct ReginaRecord {
    DWORD locale;   // ид локали
    DWORD year;     // год
    DWORD month;    // месяц
    DWORD day;      // день
    DWORD integral; // целая часть суммы
    DWORD fraction; // дробная часть суммы
    DWORD wleft;    // положение окна X
    DWORD wtop;     // положение окна Y
};
```

Эти данные включают в себя идентификатор локали, дату в виде трех элементов данных, и вещественное число, называемое суммой, заданное в виде целой и дробной частей числа. Кроме этого, данные включают в себя положение окна программы на экране.

Данные инициализируются следующим образом. Идентификатор локали по умолчанию соответствует России. Дата соответствует системной дате. Целая часть суммы соответствует миллисекундам, а дробная часть соответствует секундам системного времени. Наша задача заключается в том, чтобы вывести дату и сумму в формате, соответствующем локали.

Информация о формате хранится в реестре Windows. В предыдущих версиях Windows эта информация частично хранилась в файле win.ini. Эту информацию нужно извлечь и использовать для форматирования и вывода в соответствующие поля окна программы.

Информация может быть сохранена в файл или считана из файла. Эта часть кода программы реализована шаблоном. Кроме этого, информацию можно записать в файл инициализации программы, или в реестр, а также считать информацию из этих мест сохранения.

Тип файла ".regina" нужно зарегистрировать в системе с тем, чтобы программу можно было открыть при помощи щелчка на документ. С документом нужно связать значок документа. Для этого в реестр Windows записывается определенная информация.

4.3. Реестр Windows

Реестр Windows — это системная база данных, хранящая множество настроек. Для просмотра и редактирования реестра используется редактор реестра regedit, который можно открыть при помощи окна «Выполнить».

Следует помнить, что неосторожные действия в редакторе могут вывести операционную систему из строя, или нарушить нормальную работу установленных приложений. Поэтому будем предельно внимательны при удалении узлов реестра или изменении их значений.

В 64-х битных системах для управления реестром есть три редактора, имеющих разные названия, и расположенные в разных местах. Если не удается найти требуемую информацию в одном редакторе, возможно, следует использовать другой редактор. Все редакторы расположены в каталоге Windows\system32, при этом в окно «Выполнить» вводится:

regedit.exe — для x64 редактора реестра 64;

SYSWOW64\regedit.exe — для x64 редактора реестра 32;

SYSWOW64\regedt32.exe — для x86 редактора реестра 32.

Первый редактор показывает информацию для приложений, использующих 64-битную архитектуру. Два следующих редактора показывают информацию 32-х битных приложений. Последний редактор является 32-х битным приложением. В его названии пропущена буква "i", с тем, чтобы название файла соответствовало формату "8+3".

4.4. Чтение информации из файла инициализации Windows

В файле win.ini информация хранилась в ранних версиях Windows. Сейчас в этом файле информации нет, он оставлен для совместимости, а чтение этого файла на самом деле извлекает информацию из реестра. Для чтения информации из системного файла инициализации используются функции вида GetProfileXXX. Особенностью этих функций является параметр, задающий значение по умолчанию, которое возвращается в случае, если запрашиваемая информация отсутствует в файле инициализации.

4.4.1. Чтение названия страны

Модуль rega.h, функция ShowCountry.

Откроем редактор реестра, в нем откроем следующий узел:

HKEY_CURRENT_USER\Control Panel\International.

В правой части окна редактора находится информация о национальных особенностях текущего пользователя. Заметим, что информация в этом узле соответствует той локали, на которую была локализована операционная система во время ее установки, информация о национальных особенностях других локализаций находится в другом месте.

Нас интересует название страны. Видим, что этот параметр сохранен в реестре как строковое значение с именем sCountry.

Вызываем функцию GetProfileString. Параметры:

- строковый литерал "intl",
- строковый литерал "sCountry",
- строковый литерал "?" (значение по умолчанию),
- буфер buf для приема значения,
- константа MAX_BUF, размер буфера приема значения.

Возвращаемое функцией значение принимаем в переменную n. Оно равно числу записанных в буфер знаков с учетом завершающего нуля.

Следующий оператор устанавливает текст из буфера в окно с дескриптором hWndCoun. Используем для этого функцию SetWindowText.

Точка остановки на функции GetProfileString. Запускаем программу, выбираем в списке какую-нибудь локаль, управление приходит в точку остановки, исполняем функцию GetProfileString, убеждаемся, что в буфере содержится название страны. Остановим или завершим программу.

Недостаток использования файла win.ini — в нем нет информации о других локализациях. Поэтому продублируем извлечение информации при помощи функции GetLocaleInfo. Вызываем эту функцию непосредственно за вызовом функции GetProfileString. Параметры:

- параметр locale функции ShowCountry,
- константа LOCALE_SENGCOUNTRY (запрашиваемый параметр),
- буфер buf для приема значения,
- константа MAX_BUF, размер буфера приема значения.

Запускаем программу, пробуем менять локаль в списке, убеждаемся, что название страны выводится. Заметим, что эта информация доступна только на английском языке.

4.4.2. Форматирование вещественного числа

Модуль rega.h, функция ShowSumm.

Несколько параметров участвуют в форматировании вещественного числа, которое обозначает у нас денежную сумму.

Во-первых, это разделитель целой и дробной частей, параметр обозначается sDecimal. Во-вторых, это количество знаков после запятой, параметр называется iCurrDigits. В-третьих, это разделитель групп цифр целой части числа, и знаки между группами, параметры называются sGrouping и sThousand. Эти два последних параметра использовать не будем. Наконец, для вывода денежной суммы используются также параметры sCurrency и iCurrency. Первый параметр задает название денежной единицы, второй параметр указывает ее положение — спереди, сзади, с пробелом или без пробела от числа. Есть еще параметры для форматирования отрицательной суммы, не учитываем их.

Получим параметр sDecimal при помощи функции GetProfileString.

Параметры вызова аналогичны тем, что были использованы для получения параметра sCountry, за исключением того, что буфер приема add.

Дублируем этот вызов при помощи функции GetLocaleInfo. Параметры также аналогичны предыдущему вызову этой функции, за исключением того, что используется константа LOCALE_SDECIMAL и буфер add.

Далее получаем параметр iCurrDigits. Опять сначала используем вызов функции GetProfileString, затем вызов функции GetLocaleInfo, используем в обоих случаях буфер приема buf. В первой функции параметр, задающий значение по умолчанию, равен литералу "?", во второй функции используем константу LOCALE_ICURRDIGITS.

Отлаживаем эту часть, убеждаемся, что в буфере add находится запятая, а в буфере buf находится двойка.

Предположим теперь, что в буфере add находится строка, содержащая запятую, а в буфере buf находится строка, содержащая двойку. Тогда число выводится при помощи следующего формата:

```
sprintf(buf, "%d,02%d", integral, fraction);
```

Иначе говоря, сейчас нужно сформировать строку "%d,02%d", в которую вместо запятой нужно вставить содержимое буфера add, а вместо двойки вставить содержимое буфера buf. Делаем это при помощи функции sprintf, буфер вывода sub. Формат формируем из строки "%d,02%d". Сначала каждый процент продублируем, иначе он будет использован как формат вывода. Затем вместо запятой и вместо двойки вставляем формат %s для вывода строк буферов add и buf.

Отлаживаем эту часть кода, убеждаемся, что в буфере sub формируется правильная строка "%d,02%d".

Используем полученный формат для вывода чисел в буфер buf:

```
sprintf(buf, sub, integral, fraction);
```

После этого остается вывести полученный текст из буфера buf в окно с дескриптором hWndSumm. Используем функцию SetWindowText.

Запускаем программу, убеждаемся, что при переключении локали выводится вещественное число в том формате, который определяет локаль.

Теперь в окно с дескриптором hWndCurr нужно вывести денежный знак. Получаем параметр sCurrency в буфер buf, сначала при помощи функции GetProfileString, затем при помощи функции GetLocaleInfo.

Запускаем программу, убеждаемся, что при переключении локали выводится денежный знак (новый денежный знак рубля отсутствует в ANSI).

Заметим, что все эти действия проще выполнить при помощи системной функции GetCurrencyFormat, и обычно так и делается. Но нам хочется знать все тонкости, поэтому мы так детально все расписали.

4.4.3. Форматирование даты

Модуль rega.h, функция ShowDate.

Различают короткий и длинный формат даты. В коротком формате дата выводится цифрами, между которыми вставляется национальный разделитель. Порядок элементов в формате даты также является национальной особенностью. В коротком формате каждый из элементов даты записывается жестко заданным числом цифр, поэтому строка даты в этом формате всегда имеет одну и ту же длину. В длинном формате месяц выводится словом, а день выводится наименьшим количеством цифр.

Нас интересует короткий формат. В реестре, в узле, отображающем файл win.ini, он задан как параметр sShortDate, и имеет примерно следующий вид: dd.MM.yyyy. В принципе, этот формат уже задает и национальный разделитель (точку), но отдельно этот элемент задан параметром sDate.

Будем формировать дату, разбирая строку формата по частям. Заметим, что обычно для вывода даты используется функция GetDateFormat, но это не наш путь.

Получим в буфер add параметр sDate, сначала при помощи функции GetProfileString, затем при помощи функции GetLocaleInfo. В первой функции значение по умолчанию равно литералу "?", во второй функции используем константу LOCALE_SDATE. Запомним первый символ буфера add в переменной sdate типа char.

Затем получим в буфер buf параметр sShortDate. При вызове функции GetProfileString используем значение по умолчанию "yyyy.MM.dd". При вызове функции GetLocaleInfo используем константу LOCALE_SSHORTDATE.

Затем формируем формат вывода в буфере sub, как и ранее.

В конечном итоге нам нужно использовать следующий код:

```
sprintf(buf, sub, v[0], v[1], v[2]);
```

Здесь массив v содержит элементы даты в требуемом порядке, а буфер sub содержит строку формата, например "%02d.%02d.%04d". В этой строке вместо точек нужно вывести значение переменной sdate, а вместо чисел вывести количество букв в формате соответствующего элемента даты.

Например, если задан формат MM/dd/yy, то нужно подсчитать сначала количество букв M, запомнив в v[0] значение параметра month, затем подсчитать количество букв d, запомнив в v[1] значение параметра day, затем подсчитать количество букв y, запомнив в v[2] значение параметра year.

Очевидно, что каждая из частей формата разбирается одинаковым образом, причем каждая из частей завершается знаком, хранящимся в sdate, а последняя часть завершается нулевым знаком конца строки.

Поэтому формируем цикл по параметру j из трех итераций, в которых параметр j принимает значения от нуля до двух:

```
for (; j < 3; j++) {
}
```

Сначала в этом цикле анализируем элемент buf[i]. Если он равен знаку 'd', записываем в v[j] значение переменной day. Если элемент buf[i] равен знаку 'M', записываем в v[j] значение переменной month. Если не первое и не второе, записываем в v[j] значение переменной year.

Затем нам нужно подсчитать количество одинаковых букв и записать в буфер sub часть формата. Считать будем количество итераций, в которых буква формата не изменяется. Для подсчета обнуляем переменную digits, и формируем внутренний цикл по переменной i:

```
digits = 0;
for (; i < n; i++) {
}
```

Часть формата завершается либо знаком sdate, либо нулем. Поэтому во внутреннем цикле анализируем текущий знак buf[i]. Если он равен знаку sdate, то формируем часть формата и выходим из цикла при помощи break. Если знак buf[i] равен нулю, то также формируем часть формата и выходим из цикла при помощи break. Если знак buf[i] не является ни знаком sdate, ни нулем, то подсчитываем его, инкрементируя digits.

Теперь о том, как получить часть формата. Часть формата — это формат вывода одного целого числа, то есть "%0Nd", где вместо N нужно записать значение переменной digits. Причем, если завершающий часть знак равен sdate, то сам знак sdate также нужно вывести, то есть использовать в этом случае формат "%0Nd%c", где по формату %c выводится sdate. По аналогии с предыдущим случаем, заменяем первый процент формата двумя процентами, N заменяем спецификацией %d, и выводим по этому формату переменные digits и sdate в буфер sub при помощи sprintf.

Но здесь есть одна тонкость. Если просто выводить в буфер sub, то новый формат перепишет записанный ранее формат. Поэтому нужно выводить не в буфер sub, а использовать положение в буфере, на которое указывает переменная subp. Результат функции sprintf при этом нужно прибавлять к значению переменной subp:

```
subp += sprintf(subp, "%%0%dd%c", digits, sdate);
```

В результате переменная subp всегда будет указывать на конец информации, уже записанной в буфере sub, и так можно прибавлять информацию к уже записанной информации.

После выхода из внутреннего цикла нужно инкрементировать значение переменной i, чтобы пропустить завершающий часть знак.

По завершении внешнего цикла выводим в буфер buf части формата v[0], v[1] и v[2], используя формат, полученный в sub, после чего выводим текст из буфера buf в окно с дескриптором hWndDate при помощи функции SetWindowText.

Внимательно отлаживаем эту часть кода, убеждаемся, что для локали 1049 формируется строка формата "%02d.%02d.%04d". После этого запускаем программу, и убеждаемся, что смена локали выводит дату в разных форматах. В американской локализации дата имеет формат "d/M/YYYY".

4.5. Запись информации в файл инициализации

В частный файл инициализации информация записывается при помощи функции WritePrivateProfileStringA или WritePrivateProfileStringW, обычно используется макрос WritePrivateProfileString. Параметры:

- строка — название приложения,
- строка — ключ записываемого элемента,
- строка — записываемая информация,
- строка — путь к ini-файлу.

Используя разные первые параметры, информацию можно группировать. Мы всю информацию будем записывать в одну группу, то есть использовать одно название приложения, заданное константой APPNAME.

Строки, задающие ключи элементов информации определены в модуле regina.h следующим образом:

```
// ключи данных
#define KEYLCID  "Locale"
#define KEYYEAR  "Year"
#define KEYMONTH "Month"
#define KEYMDAY  "Day"
#define KEYINTG  "Integral"
#define KEYFRAC  "Fraction"
#define KEYWLEFT "WLeft"
#define KEYWTOP  "WTop"
```

Путь к ini-файлу находится в переменной szINIPath.

Модуль rega.h, функция PutLongToIni.

Элемент данных нужно преобразовать в строку при помощи функции sprintf, используя буфер buf и спецификацию %d. Затем вызываем функцию WritePrivateProfileString, передаем ей необходимые параметры.

Модуль rega.h, функция PutDocumentDataToIni.

Эта функция восемь раз вызывает функцию PutLongToIni, передавая ей необходимый ключ и одно из полей структуры data.

После определения функции запускаем программу, выбираем в меню «Данные — Записать в ini-файл». Завершаем программу. Открываем FAR, переходим в каталог C:\regina\Debug, открываем файл regina.ini при помощи клавиши F4, изучаем вывод. Он имеет примерно следующий вид:

```
[Regina]
locale=1049
year=2018
month=12
day=31
integral=531
fraction=31
wleft=760
wtop=340
```

4.6. Чтение информации из файла инициализации

Для чтения информации из частного файла инициализации используется функция, определяемая макросом GetPrivateProfileInt для чтения целого числа, или макросом GetPrivateProfileString для чтения строки.

Модуль rega.h, функция GetLongFromIni. Функция возвращает значение, которое возвращает функция GetPrivateProfileInt, приведенное к типу DWORD. В этой функции, в отличие от функции WritePrivateProfileString, вместо третьего параметра используется значение по умолчанию, в качестве которого используем параметр defval.

Модуль rega.h, функция GetDocumentDataFromIni.

Эта функция сложнее, потому что требуется сформировать значения по умолчанию для всех параметров. Значения по умолчанию берутся в основном из системного времени, поэтому сначала объявляем переменную st типа SYSTEMTIME, затем получаем системное время при помощи функции GetSystemTime.

Далее при помощи функции GetLongFromIni получаем параметр за параметром, и записываем значения в поля структуры data. Значением по умолчанию для локали является значение переменной dwDefaultLCID. Значениями по умолчанию для полей wleft и wtop являются результаты вызова функций GetWLeft и GetWTop соответственно, они вычисляют такие координаты, которые устанавливают окно в центр экрана.

По завершении считывания всех элементов нужно вызывать функцию SetDocumentToWindow, и передать ей структуру data.

4.7. Формирование записей приложения в реестре

Информацию, относящуюся к приложению в целом, записывают в реестр в улей (корневой узел) HKEY_LOCAL_MACHINE. Информацию, относящуюся к использованию приложения пользователями, записывают в улей HKEY_CURRENT_USER. Информацию о документе и приложении, которое его обрабатывает, записывают в улей HKEY_CLASSES_ROOT. Корневые узлы в программе обозначены константами HKLM, HKCU и HKCR вместо длинных системных констант (модуль regina.h).

4.7.1. Вспомогательные функции

Определим несколько вспомогательных функций.

Модуль rega.h, функция CreateReginaKey.

При помощи этой функции создается ключ программы. Параметры:

- Key — ключ отсчета,
- path — путь от ключа отсчета,
- ASM — доступ к ключу,
- sa — атрибуты безопасности,
- key — возвращаемый дескриптор ключа.

Ключ создаем при помощи функции RegCreateKeyEx, подставляя параметры Key, path, 0, 0, константу REG_OPTION_NON_VOLATILE, ASM, sa, key по ссылке, и dwDisposition по ссылке. В dwDisposition возвращается признак создания или существования ключа.

Возвращаемое функцией RegCreateKeyEx значение принимаем в переменную lResult типа LONG. Переменную lResult проверяем. Если ее значение не равно нулю (константе ERROR_SUCCESS), то значение является кодом ошибки из файла winerror.h. В этом случае присваиваем NULL возвращаемому значению key. В завершении переменную lResult возвращаем для последующего анализа как результат функции CreateReginaKey.

Модуль rega.h, функция OpenReginaKey.

При помощи этой функции будем получать доступ к ключу.

Параметры функции OpenReginaKey:

- Key — ключ отсчета,
- path — путь от ключа отсчета,
- ASM — желаемый доступ к ключу.

Открываем ключ при помощи функции RegOpenKeyEx, подставляя в нее параметры Key, path, 0, ASM, key по ссылке. Результат вызова функции RegOpenKeyEx принимаем в переменную lResult, значение которой проверяем. Если оно не равно нулю, то присваиваем NULL возвращаемому значению key. В завершении переменную lResult возвращаем для последующего анализа как результат функции OpenReginaKey.

Модуль rega.h, функция PutStringToReg.

Функция записывает строковое значение в реестр. Параметры:

- Key — ключ отсчета,
- valname — имя записываемого параметра,
- value — записываемая строка.

Записываем строку при помощи функции RegSetValueEx, подставляя параметры Key, valname, 0 (зарезервировано), REG_SZ (тип данных), value (приводим к типу LPBYTE), и strlen(value) + 1. Возвращаемый результат принимаем в переменную lResult, значение которой возвращаем для последующего анализа.

Модуль rega.h, функция PutLongToReg.

Функция записывает в реестр целое число типа DWORD в указанный первым параметром ключ, с именем, заданным вторым параметром. Записываемое значение задается третьим параметром.

Записываем значение при помощи функции RegSetValueEx, подставляя параметры Key, valname, 0 (зарезервировано), REG_DWORD (тип данных), value по ссылке (приводим к типу LPBYTE), и sizeof(value). Возвращаемый результат принимаем в переменную lResult, значение которой возвращаем для последующего анализа.

Модуль rega.h, функция GetLongFromReg.

Функция считывает из реестра целое число типа DWORD из указанного первым параметром ключа, с именем, заданным вторым параметром. Считанное значение возвращается через третий параметр, по ссылке.

Читаем значение при помощи функции RegQueryValueEx, подставляя параметры Key, valname, 0 (зарезервировано), dwType по ссылке (в эту переменную возвращается тип данных), dwRead по ссылке (возвращаемое значение, приводим к типу LPBYTE), dwSize по ссылке (возвращаемый размер данных). Если возвращаемое RegQueryValueEx значение lResult равно нулю, присваиваем возвращаемому значению value значение dwRead. Переменную lResult возвращаем для последующего анализа.

4.7.2. Создание ключей программы

Модуль rega.h, функция CreateRegKeys.

Нужно создать четыре ключа, а также записать значение идентификатора локали по умолчанию. Если не удалось создать какой-либо ключ, функция должна завершиться и вернуть FALSE. Это вызовет завершение старта программы, поскольку функция CreateRegKeys является заключительной частью функции создания приложения InitInstance.

Эта функция должна выводить сообщения об ошибках, если они возникнут. Для вывода сообщений об ошибках в программе есть две функции, Critical и ShowReginaError. Первый параметр этих функций предназначен для указания на место возникновения ошибки. Второй параметр функции Critical — сообщение об ошибке. Некоторые сообщения описаны в модуле regina.h. Второй параметр функции ShowReginaError — это значение переменной lResult, возвращаемое из функций.

Сначала нужно сформировать завершение функции, поскольку в случае неудачи какой-либо из функций нужно выполнить очистку:

```
    return AssocDocument(NULL);
failure:
    ShowReginaError("CreateRegKeys", lResult);
    if (key) {
        RegCloseKey(key);
    }
    return FALSE;
```

Функция возвращает то, что возвращает функция AssocDocument, которая рассматривается позднее. В случае неудачного завершения какой-либо функции нужно выполнить очистку, переходя на метку failure при помощи оператора goto. Очистка заключается в закрытии временного ключа key, если он был открыт.

Создаваемые пути в ульях HKLM и HKCU одинаковы, и равны строке "Software\ОТИ-ПМ\Regina\1.0". Эта строка формируется в глобальной переменной szReginaPath, функция OnStartApp модуля regina.cpp.

Записи создаем в следующем порядке:

HKLM\Software\ОТИ-ПМ\Regina\1.0,

HKLM\Software\ОТИ-ПМ\Regina\1.0\Default,

HKLM\Software\ОТИ-ПМ\Regina\1.0\Default = значение-по-умолчанию,

HKCU\Software\ОТИ-ПМ\Regina\1.0,

HKCU\Software\ОТИ-ПМ\Regina\1.0\User Data.

Для создания первой записи вызываем функцию CreateReginaKey, подставляя параметры HKLM, szReginaPath, KEY_READ, NULL, HKLMReginaKey по ссылке. Здесь HKLMReginaKey — это глобальная переменная, которая будет держать ключ реестра до завершения программы. Результат вызова проверяем, в случае неудачи переходим на метку failure:

```
    // HKLM\Software\ОТИ-2000\Regina\1.0
    lResult = CreateReginaKey(. . .);
    if (lResult) goto failure;
```

В этом месте следует проверить, как создается ключ. Запускаем программу, останавливаемся на вызове CreateReginaKey, заходим в эту функцию, выполняем вызов функции RegCreateKeyEx. Проверяем значение переменной lResult, убеждаемся, что оно равно 5 (доступ запрещен), продолжаем выполнение до завершения программы, чтобы убедиться, что выводится правильное сообщение об ошибке.

Доступ запрещен, потому что создавать записи в реестре может только администратор системы. Поэтому закрываем проект, выходим из ограниченной учетной записи, и входим в систему под *учетной записью администратора*. Открываем проект *от имени администратора*, и повторяем описанные действия. После выполнения функции RegCreateKeyEx с результатом lResult, равным нулю, изучаем возвращаемое значение dwDisposition.

После того, как ключ HKLMReginaKey будет успешно создан, открываем редактор реестра, находим запись в улье HKLM. В узле Software находится узел ОТИ-ПМ, в нем узел Regina, в нем узел 1.0.

Продолжаем в случае, если созданы правильные записи. В противном случае их можно удалить, щелкнув на значок папки ключа и нажав клавишу Delete. При этом будем внимательны, чтобы не удалить другой ключ. Восстановить удаленный ключ чрезвычайно сложно, чаще невозможно.

Теперь нужно создать ключ Default.

Вызываем функцию CreateReginaKey, подставляя HKLMReginaKey, константу KEYDEFAULT, константу KEY_ALL_ACCESS, NULL, результат вызова key по ссылке. Если вызов функции CreateReginaKey неуспешен, переходим на метку failure.

Выполняем программу, убеждаемся в редакторе реестра, что в ключе с названием 1.0 появился ключ Default.

Записываем значение идентификатора локали по умолчанию. Вызываем функцию PutLongToReg, передавая key, константу KEYLOCALE, переменную dwDefaultLCID. Если вызов функции PutLongToReg неуспешен, переходим на метку failure. Затем временный ключ key нужно закрыть при помощи функции RegCloseKey.

Запускаем программу, убеждаемся в редакторе реестра, что в ключе Default появился параметр Locale со значением 1049.

Аналогичным образом создаем две оставшихся записи. Результат создания четвертой записи должен быть записан в глобальную переменную HKCUReginaKey, которая используется для записи и чтения данных. Она используется так же как ключ отсчета для создания ключа User Data. Временный ключ key закрываем при помощи функции RegCloseKey.

4.7.3. Запись и чтение данных

Модуль rega.h, функция PutDocumentDataToReg.

Сначала определим завершение функции:

```
    goto cleanup;
failure:
    Critical("PutDocumentDataToReg", MSGDATATOREG);
cleanup:
    RegCloseKey(key);
```

Код функции располагается перед этим кодом. На метку failure переходим в случае неудачи записи элемента данных.

В начале функции проверяем значение переменной HKCUReginaKey. Если оно равно NULL, выводим сообщение MSGCLOSEDHKCU, и завершаем функцию.

Если ключ HKCUReginaKey открыт, то открываем ключ данных User Data при помощи функции OpenReginaKey, подставляя HKCUReginaKey, константу KEYDATA, константу KEY_ALL_ACCESS, ключ key по ссылке.

Результат вызова получаем в переменную lResult. Если вызов не успешен (lResult не равно нулю), выводим сообщение при помощи функции ShowReginaError, и завершаем функцию.

Далее поочередно записываем данные. Один элемент данных записывает функция PutLongToReg, параметрами являются ключ key, константа имени данных, элемент структуры data. Результат проверяем, в случае неудачи переходим на метку failure, например:

```
    if (PutLongToReg(key, KEYLCID, data.locale)) goto failure;
```

После описания записи всех элементов данных запускаем программу, выбираем в меню «Данные — Записать в реестр», проверяем в редакторе реестра записи.

Функция чтения элементов данных GetDocumentDataFromReg похожа на функцию PutDocumentDataToReg. Отличия следующие:

- вместо функции PutLongToReg используем функцию GetLongFromReg;
- в конце функции вызываем функцию SetDocumentToWindow, точно так же, как это было сделано в функции GetDocumentDataFromIni.

4.7.4. Ассоциирование файла документа с приложением

Установим ассоциацию файла документа с приложением и значок документа. Функция AssocDocument, модуль rega.h.

Сначала в функции определим код завершения:

```
    return TRUE;
failure:
    ShowReginaError("AssocDocument", lResult);
    if (key) RegCloseKey(key);
    return FALSE;
```

Данные записываются в улей HKEY_CLASSES_ROOT, в котором нужно сформировать следующие ключи и записи по умолчанию:

HKCR\.regina
HKCR\.regina = Regina
HKCR\Regina
HKCR\Regina = ОТИ-ПМ Regina
HKCR\Regina\Shell\Open\Command
HKCR\Regina\Shell\Open\Command = c:\regina\Debug\regina.exe %1
HKCR\Regina\DefaultIcon
HKCR\Regina\DefaultIcon = c:\regina\Debug\regina.exe,2

Создаем эти записи в указанном порядке.

Запись HKCR\.regina.

В буфере regbuf формируем строку ".regina" при помощи функции sprintf, константы szDocDefExt и спецификации %s. Обратим внимание на точку перед словом regina, она существенная часть этой записи.

Создаем ключ при помощи функции CreateReginaKey, подставляя параметры HKCR, regbuf, KEY_ALL_ACCESS, sa, и key по ссылке. Если функция завершилась неуспешно, переходим на метку failure.

Запись HKCR\.regina = Regina.

Вызываем функцию PutStringRoReg, подставляя полученный ключ key, две двойных кавычки (пустая строка) "", константу APPNAME. Результат вызова принимаем в переменную lResult, анализируем, и если значение не нулевое, переходим на метку failure.

Закрываем временный ключ key при помощи функции RegCloseKey.

Запускаем программу, убеждаемся, что в реестре сформирован ключ и его значение по умолчанию равно Regina. Запись означает, что документ с расширением .regina открывает приложение Regina, запись о котором находится в этом же улье дальше, и она показывает, как открыть документ.

Запись HKCR\Regina.

Вызываем функцию CreateReginaKey, подставляя ключ отсчета HKCR, константу APPNAME, константу KEY_ALL_ACCESS, sa, и key по ссылке. Результат проверяем, при неудаче переходим на метку failure.

Теперь в этот ключ записываем значение по умолчанию, оно описывает тип документа приложения Regina строкой ОТИ-ПМ Regina. В буфере regbuf получаем эту строку при помощи констант COMPANY и APPNAME, устанавливаем значение по умолчанию при помощи функции PutStringRoReg, проверяем результат. В конце этой части закрываем ключ key.

Запись HKCR\Regina\Shell\Open\Command.

В буфере regbuf формируем строку Regina\Shell\Open\Command при помощи константы APPNAME. Создаем ключ функцией CreateReginaKey, ключ отсчета HKCR, результат проверяем.

Чтобы записать значение в созданный ключ, нужно сформировать команду, которая открывает документ приложением Regina. Эта команда состоит из пути к файлу программы, пробела и %1. Путь к файлу программы находится в переменной szEXEPath. Команду формирует следующий код:

```
sprintf(regbuf, "%s %%1", szEXEPath);
```

Записываем значение по умолчанию из буфера regbuf при помощи функции PutStringRoReg, результат проверяем. Закрываем ключ key.

Запись HKCR\Regina\DefaultIcon.

Формируем в буфере regbuf строку Regina\DefaultIcon, используя константу APPNAME. Формируем ключ при помощи функции CreateReginaKey, ключ отсчета HKCR. Результат также проверяем.

Запись HKCR\Regina\DefaultIcon = c:\regina\Debug\regina.exe,2.

Формируем в буфере regbuf строку c:\regina\Debug\regina.exe,2, используя переменную szEXEPath. Обратим внимание, что перед и после запятой не должно быть пробелов. Формируем запись при помощи функции PutStringRoReg, результат проверяем. Закрываем ключ key.

Проверка правильности записей выполняется следующим образом.

Запускаем программу, выбираем в меню «Файл — Сохранить». Сохраняем документ в файл с именем test. Закрываем программу. Открываем проводник, открываем папку с файлом программы.

Дважды щелкаем на значок файла test.regina. Если при этом открывается программа, ассоциация файла с программой установлена. Программа должна открываться также, если файл выбрать в FAR и нажать Enter. В проводнике у файла test.regina должны измениться значок и описание типа файла на «ОТИ-ПМ Regina».

4.7.5. Завершение работы программы

Модуль rega.h, функция OnCloseApp.

Если открыт ключ HKLMReginaKey, закрываем его.

Если открыт ключ HKCUReginaKey, закрываем его.

4.7.6. Обеспечение доступа к программе обычному пользователю

Модуль rega.h, функция CreateRegKeys.

В начале функции проверяем, есть ли запись в улье HKCU, чтобы обеспечить доступ к реестру обычному пользователю.

Проверяем наличие записи HKCU\Software\ОТИ-ПМ\Regina\1.0. Вызываем функцию OpenReginaKey, подставляя HKCU, szReginaPath, KEY_READ, переменную HKCUReginaKey по ссылке. Если вызов успешен, то возвращаем истину и функция завершается, ключ HKCUReginaKey открыт.

В результате обычный пользователь сможет записывать свои данные в реестр и считывать их оттуда. В этом следует убедиться, войдя в систему под ограниченной учетной записью.

В завершение работы выполните экспорт созданных ключей из редактора реестра в формате Win9x/NT4, папка сохранения C:\regina, названия файлов hkcr1 для ключа HKCR\.regina, hkcr2 для ключа HKCR\Regina, hkcu для ключа HKCU\ОТИ-ПМ, hklm для ключа HKLM\ОТИ-ПМ. Запишите содержимое файлов реестра в отчет.

4.8. Контрольные вопросы и упражнения

1. Опишите структуру файла инициализации.
2. Опишите порядок записи значения в файл инициализации.
3. Опишите национальные особенности локализации.
4. Опишите структуру реестра Windows.
5. Опишите записи, формируемые в улье HKCR.
6. Опишите назначение ульев HKLM и HKCR.
7. Опишите порядок записи значения в реестр.

5. Работа SY-105. Атрибуты безопасности

Цели:
- изучение атрибутов безопасности.
Задачи:
- формирование маркеров доступа;
- формирование списков доступа;
- формирование прав доступа;
- формирование дескриптора безопасности.
Опорные документы:
[1, «Атрибуты безопасности»]

5.1. Шаблон проекта

Войдите в систему под учетной записью администратора. Установите на диск C: приложение regina, полученное в предыдущей работе. Работа должна быть полностью выполнена. Откройте проект *от имени администратора*. Убедитесь, что целевой платформой является x86 или Win32. Откройте редактор реестра и удалите из реестра все записи, сформированные в ходе выполнения предыдущей работы. Список удаляемых узлов:

HKCR\.regina
HKCR\Regina
HKCU\ОТИ-ПМ
HKLM\ОТИ-ПМ

Рабочим модулем является модуль regb.h. Укажите в начале этого модуля сведения об организации, о себе, о проекте, дату начала работы:

```
// ОТИ НИЯУ МИФИ
// 1ПО-00Д
// Студент Имя Отчество
// Системное программирование
// SY-105. Атрибуты безопасности
// 01.01.2000
```

В программе должен быть определен символ SECURESY. В модуле regina.h есть его определение, которое нужно раскомментировать.

5.2. Записи в реестре

В реестре будем формировать такие же записи, что и в предыдущей работе, и некоторую дополнительную информацию. Каждой записи будем назначать определенные права доступа.

Мы рассматриваем три категории пользователей:
- системный администратор,
- текущий пользователь,
- всякий другой.

Системный администратор создает записи в реестре, он должен запускать программу первым, чтобы установить ее.

Будем называть записи в улье HKLM глобальной областью, а записи в улье HKCU областью пользователя. Во время процедуры установки в глобальную область записывается фамилия и организация, которые пользователь вводит в диалоге установки. В глобальной области записывается также глобальное значение идентификатора локали, выбранное пользователем в диалоге установки, и признак установки программы, позволяющий определить, что программа установлена, и записи в улье HKLM созданы.

Текущий пользователь может изменять свои данные, расположенные в области пользователя. Для каждого пользователя в улье HLCU должны создаваться свои собственные записи, не пересекающиеся с записями других пользователей. Заметим, что системный администратор одновременно является текущим пользователем, и для него точно также создаются свои собственные записи. Текущий пользователь может читать записи из глобальной области, и имеет полный доступ к своим записям. В области пользователя также записывается дополнительная информация. Во-первых, это значение идентификатора локали по умолчанию, которое пользователь может менять в диалоге «О программе». Во-вторых, это признак установки, позволяющий определить, что записи в улье HKCU созданы.

Всякий другой пользователь может только просматривать записи.

5.3. Процедура старта программы

Процедура старта программы достаточно сложна. В предыдущей работе старт программы завершался функцией CreateRegKeys. В этой работе старт программы завершается функцией CreateAppKeys модуля regb.h.

Цель функции CreateAppKeys заключается в проверке признака установки сначала в глобальной области, затем в области пользователя.

Если в глобальной области признак установки не обнаружен, программа считает, что она не установлена, при этом запускается процедура установки, функция InstallApp.

Если в области пользователя признак установки не обнаружен, программа считает, что ее использует новый пользователь, при этом запускается процедура создания области пользователя для него, функция InitUser.

Процедура установки программы сначала проверяет, что текущий пользователь является администратором. Если нет, программа выводит сообщение. Если да, то на экран выводится диалог установки, в котором нужно ввести фамилию и организацию, а также выбрать глобальное значение идентификатора локали по умолчанию. Если все эти данные введены, диалог завершается вызовом функции StoreAppConfig, которая формирует необходимые записи в улье HKLM. Если какие-то данные не введены, диалог просит ввести их. Если нажата кнопка отмены, или неудачно завершается функция StoreAppConfig, программа завершает работу.

Процедура создания области пользователя создает записи в улье HKCU для текущего пользователя. Эта процедура, если запускается, то только после того, как записи в улье HKLM созданы (или были созданы ранее). Она также копирует начальное значение идентификатора локали по умолчанию из глобальной области в область пользователя. Если эта процедура завершается неудачно, то программа завершает работу. Если процедура завершается успешно, то в переменную dwDefaultLCID из области пользователя считывается начальное значение идентификатора локали.

5.4. Функция старта

Модуль intsal.h, функция CreateAppKeys.

В функции определена переменная installed, признак наличия установки программы или признак инициализации области пользователя.

При помощи функции OpenReginaKey пытаемся получить ключ глобальной области в глобальную переменную HKLMReginaKey. Ключом отсчета является HKLM, путь задан переменной szReginaPath, требуемый доступ к ключу KEY_READ. Результат, возвращаемый функцией OpenReginaKey, принимаем в переменную lResult.

Если вызов этой функции завершается неудачно, то присваиваем признаку installed нулевое значение. Если вызов завершается успешно, то при помощи функции GetLongFromReg пытаемся прочитать признак установки. Ключ отсчета HKLMReginaKey, имя параметра KEYINSTALL, значение из реестра считываем в переменную dwInstalled, которая никак не используется. Результат, возвращаемый функцией GetLongFromReg, принимаем в переменную lResult. Далее анализируем lResult и dwType. Если lResult не равно нулю, или dwType не равно REG_DWORD, то присваиваем признаку installed нулевое значение.

После всех этих проверок признак installed примет либо единичное, либо нулевое значение. Если значение нулевое, программа не установлена должным образом, и мы вызываем функцию установки InstallApp. Если эта функция возвращает ложь, закрываем ключ HKLMReginaKey, возвращаем ложь, функция завершается, программа также завершает работу.

Вторая часть функции CreateAppKeys совершенно аналогична первой части, только используется ключ отсчета HKCU и ключ HKCUReginaKey, который должен быть открыт в конечном итоге, так как он используется для записи или чтения данных пользователя. Кроме того, если признак установки installed окажется равным нулю, то вместо функции InstallApp вызывается функция InitUser, которой передается ключ HKLMReginaKey и переменная szReginaPath.

Если обе проверки завершились удачно, то в конце функции получаем идентификатор локали по умолчанию в переменную dwDefaultLCID при помощи функции GetLongFromReg, используя ключ отсчета HKCUReginaKey и имя переменной KEYLOCALE. Результат не проверяем.

5.5. Функция установки приложения

Модуль regb.h, функция InstalApp.

Эта функция самая простая, вот она вся:

```
BOOL InstallApp() {
    if (!IsRunningAdmin()) {
        Critical("InstallApp", MSGNOTADMIN);
        return FALSE;
    }
    if (!DialogBox(hInst, MAKEINTRESOURCE(IDD_INST),
        hWndMain, InstalProc)) {
        return FALSE;
    }
    return TRUE;
}
```

Сначала функция проверяет, является ли текущий пользователь администратором, вызывая функцию IsRunningAdmin. Если не является, выводится сообщение, что установку может выполнять только администратор.

Если пользователь администратор, то вызывается диалог установки при помощи функции DialogBox. Диалог появляется на экране, пользователь вводит требуемую информацию, и нажимает кнопку «Установить».

Диалог вызывает функцию StoreAppConfig, передает ей путь в реестре, фамилию, организацию, значение идентификатора локали по умолчанию. Если функция StoreAppConfig завершается неудачно, функция диалога возвращает ложь, функция InstallApp также возвращает ложь, и программа завершает работу.

Следующей функцией должна была бы быть функция IsRunningAdmin или StoreAppConfig, однако перед этим требуется описать несколько вспомогательных функций.

5.6. Вспомогательные функции

Модуль regb.h, функция GetToken.

Эта функция получает маркер доступа (токен) текущего потока или процесса, в зависимости от того, что есть.

Сначала пытаемся получить маркер текущего потока при помощи функции OpenThreadToken, подставляя параметры GetCurrentThread(), константу TOKEN_QUERY, значение FALSE, и hToken по ссылке. Функция возвращает FALSE в случае неудачи. Если это так, то сравниваем код последней ошибки с константой ERROR_NO_TOKEN. Код последней ошибки возвращает функция GetLastError.

Если код ошибки совпадает с константой, значит, у потока нет маркера. В этом случае пытаемся получить маркер доступа процесса. Вызываем функцию OpenProcessToken, подставляя параметры GetCurrentProcess(), константу TOKEN_QUERY, и hToken по ссылке. Функция возвращает FALSE в случае неудачи. Если это так, то возвращаем NULL.

Если код последней ошибки не равен константе ERROR_NO_TOKEN, то также возвращаем NULL.

Если хотя бы одна из функций завершилась удачно, сбрасываем код последней ошибки при помощи функции SetLastError, и возвращаем из функции hToken.

Модуль regb.h, функция GetTokenInfo.

Эта функция получает информацию, связанную с токеном.

Сначала функция получает токен при помощи функции GetToken в переменную hToken. Если получен результат NULL, функция возвращает NULL.

Далее нужно выяснить размер информации, выполнив вызов функции GetTokenInformation с нулевым полем информации. Подставляем в вызов параметры hToken, tic, NULL, 0, dwSize по ссылке. Если функция возвращает FALSE, то мы возвращаем NULL из GetTokenInfo. Если функция завершается успешно, dwSize содержит требуемый размер информации.

Непосредственно за этим следует сравнение кода последней ошибки с константой ERROR_INSUFFICIENT_BUFFER. Если код последней ошибки не равен этой ошибке, возвращаем NULL. Если код равен этой константе, все нормально, мы продолжаем, и сбрасываем эту ошибку при помощи функции SetLastError.

Затем выделяем буфер для информации при помощи LocalAlloc:

```
if (!(lpInfo = LocalAlloc(LPTR, dwSize))) return NULL;
```

Переменная lpInfo — это буфер, в который мы получим информацию. Если буфер выделить не удалось, возвращаем NULL.

Наконец, получаем информацию, выполняя второй вызов функции GetTokenInformation, подставляя параметры hToken, tic, lpInfo, dwSize, и dwSize по ссылке. Переменная dwSize подставляется два раза. Первый раз она указывает размер буфера, второй раз в нее возвращается записанное в буфер количество байт.

Если второй вызов неуспешен, освобождаем память при помощи функции LocalFree, возвращаем NULL. Если вызов успешен, возвращаем переменную lpInfo.

Модуль regb.h, функция GetSID.

Эта функция формирует идентификатор безопасности, называемый также системным идентификатором. Функция является оберткой функции AllocateAndInitializeSid, которая имеет слишком много параметров, большая часть из которых нулевые. Функция позволяет прикрепить к идентификатору безопасности до восьми полномочий, в то время как нам требуется не более двух.

Вызываем функцию AllocateAndInitializeSid, передавая ей параметры Authority, Count, SA0, SA1, шесть нулей (не требуемые полномочия), и psid по ссылке. Если функция возвращает FALSE, то мы возвращаем NULL. Если функция завершается успешно, мы возвращаем переменную psid.

Модуль regb.h, функция IsRunningAdmin.

Эта функция проверяет, является ли текущий пользователь системным администратором. Она возвращает признак isAdmin, сначала равный нулю.

Первое действие — получить информацию о группе пользователей маркера доступа. Вызываем функцию GetTokenInfo, подставляя параметр TokenGroups, элемент перечисления классов информации токена. Возвращаемый результат приводим к типу PTOKEN_GROUPS и записываем в переменную ptg.

Если полученное значение равно NULL, то освобождаем память, выделенную для информации ptg при помощи LocalFree, формируем сообщение об ошибке MSGTOKENINFO при помощи функции Critical, возвращаем из функции IsRunningAdmin значение FALSE.

Если информация получена, формируем идентификатор безопасности для группы администраторов. Вызываем функцию GetSID, подставляя параметры SystemAuthority по ссылке, количество требуемых полномочий 2, константы SECURITY_BUILTIN_DOMAIN_RID и DOMAIN_ALIAS_RID_ADMINS, указывающие на два требуемых полномочия. Возвращаемое значение принимаем в переменную psidAdmin, проверяем ее значение. Если переменная равна NULL, формируем сообщение об ошибке MSGSIDERROR при помощи функции Critical, возвращаем из функции IsRunningAdmin значение FALSE.

Наконец, просматриваем список групп токена, и пытаемся найти в нем идентификатор безопасности, равный psidAdmin:

```
for (g = 0; g < ptg->GroupCount; g++) {
    if (EqualSid(ptg->Groups[g].Sid, psidAdmin))  {
        isAdmin = TRUE;
        break;
    }
}
```

Если идентификатор найден, переменная isAdmin получает истинное значение. В завершении функции освобождаем идентификатор psidAdmin при помощи функции FreeSid, освобождаем память, выделенную для информации ptg при помощи функции LocalFree, и возвращаем из функции IsRunningAdmin переменную isAdmin.

Эти вспомогательные функции нужно проверить.

Точка остановки в начале функции IsRunningAdmin. Запускаем программу и проверяем выполнение каждой строчки кода каждой функции. Функция IsRunningAdmin должна определить, что текущий пользователь является администратором.

5.7. Создание глобальной области

Переходим к функции StoreAppConfig, модуль regb.h. Эта функция создает записи в глобальной области, в улье HKLM. Для этого определяются полномочия для группы администраторов и всяких других пользователей.

Сначала сформируем завершение функции. Здесь требуется несколько вариантов, описываемых метками:

```
    return TRUE;
registry_failure:
    ShowReginaError("StoreAppConfig: registry failure", lResult);
assoc_failure:
    if (key) RegDeleteKeyA(HKLMReginaKey, KEYDEFAULT);
    if (HKLMReginaKey) RegDeleteKeyA(HKLM, szRegPath);
    goto clean_up;
security_failure:
    ShowReginaError("StoreAppConfig security fail", GetLastError());
clean_up:
    if (psidAdmins) FreeSid(psidAdmins);
    if (psidEveryone) FreeSid(psidEveryone);
    return FALSE;
```

В случае успеха функция возвращает значение TRUE.

Если возникнет ошибка при создании структур безопасности, будем переходить на метку security_failure, в которой выполняется очистка структур. Если возникнет ошибка при создании ключей реестра, будем переходить на метку registry_failure, в которой удаляются ключи глобальной области, чтобы программа не могла стартовать. Если возникнет ошибка при создании записей в улье HKCR в функции AssocDocument, перейдем на метку assoc_failure. При любой ошибке функция возвращает значение FALSE.

Переходим в начало функции.

Определяем поля структуры атрибутов безопасности:

```
// атрибуты безопасности глобального ключа
sa.nLength = sizeof(SECURITY_ATTRIBUTES);
sa.bInheritHandle = FALSE;
sa.lpSecurityDescriptor = &sdPermissions;
```

Далее нужно сформировать разрешения sdPermissions.

Создаем идентификатор безопасности psidAdmins для группы администраторов при помощи функции GetSid, точно так же, как это сделано в функции IsRunningAdmin для идентификатора psidAdmin. Если идентификатор psidAdmins равен NULL, переходим на метку security_failure.

Создаем идентификатор безопасности psidEveryone для всякого другого пользователя, вызывая функцию GetSid, подставляя по ссылке структуру WorldAuthority, количество полномочий 1, константу SECURITY_WORLD_RID, и 0 (нет другого полномочия). Если полученный идентификатор безопасности psidEveryone равен NULL, переходим на метку security_failure.

Инициализируем дескриптор безопасности sdPermissions при помощи функции InitializeSecurityDescriptor, подставляя sdPermissions по ссылке, и константу SECURITY_DESCRIPTOR_REVISION1. Если эта функция возвращает значение FALSE, то переходим на метку security_failure.

Прикрепляем разрешения к идентификатору безопасности psidAdmins при помощи функции SetSecurityDescriptorOwner.

Подставляем параметры sdPermissions по ссылке, psidAdmins и 0. Если функция возвращает значение FALSE, переходим на метку security_failure.

Выделяем память для списка дискреционного доступа при помощи функции _alloca:

```
if (!(paclKey = (PACL)_alloca(MAX_ACL))) {
    Critical("StoreAppConfig: pacl", MSGNOTENOUGHMEMORY);
    goto clean_up;
}
```

Функция _alloca выделяет память в стеке, поэтому нет необходимости ее освобождать, это происходит автоматически. В случае неудачи переходим к метке очистки.

Инициализируем список дискреционного доступа при помощи функции InitializeAcl, подставляя параметры paclKey, константу MAX_ACL, и константу ACL_REVISION2. Если функция возвращает значение FALSE, переходим на метку security_failure.

Формируем список дискреционного доступа. В списке будет две записи управления доступом (ACE). Порядок формирования имеет значение.

Первая запись предоставляет полный доступ группе администраторов. Вторая запись предоставляет доступ для чтения всякому другому пользователю. Два раза вызываем функцию AddAccessAllowedAce.

При первом вызове подставляем список доступа paclKey, константу ACL_REVISION2, константу доступа KEY_ALL_ACCESS, дескриптор безопасности psidAdmins.

При втором вызове также подставляем список доступа paclKey, константу ACL_REVISION2, но константу доступа KEY_READ, и дескриптор безопасности psidEveryone.

Заметим, что в качестве констант доступа используются константы доступа к реестру, коды которых занимают младшие 16 бит кода доступа.

В обоих случаях, если функция AddAccessAllowedAce возвращает значение FALSE, переходим на метку security_failure.

Последнее действие, связанное с определением разрешений — связываем список дискреционного доступа с дескриптором безопасности. Вызываем функцию SetSecurityDescriptorDacl, подставляя sdPermissions по ссылке, TRUE, paclKey, и FALSE. Если функция возвращает значение FALSE, переходим к метке security_failure.

Теперь создаем ключи реестра, используя полученные атрибуты безопасности. Код этой части примерно соответствует тому, что было написано в функции CreateRegKeys, однако создаются дополнительные значения.

В этой части используем функцию создания ключа CreateReginaKey, в нее подставляем атрибуты безопасности sa, функцию PutStringRoReg, и функцию PutLongToReg. Результаты функций принимаем в переменную lResult, значение проверяем. Если в lResult получено ненулевое значение, переходим к метке registry_failure.

1. Создаем запись HKLM\Software\ОТИ-ПМ\Regina. Константа доступа к ключу KEY_ALL_ACCESS. Возвращаемый ключ сохраняем в глобальной переменной HKLMReginaKey.

2. Записываем в ключ HKLMReginaKey значение фамилии szName, константа KEYNAME, и значение организации szOrg, константа KEYORG при помощи функции PutStringRoReg.

3. Создаем запись HKLM\Software\ОТИ-ПМ\Regina\Default. Константа доступа к ключу KEY_ALL_ACCESS. Возвращаемый ключ сохраняем в локальной переменной key.

4. Записываем в ключ key значение идентификатора локали по умолчанию dwDefLCID, константа KEYLOCALE, функция PutLongRoReg.

5. Сбрасываем данные на диск при помощи функции RegFlushKey, параметр HKLMReginaKey.

6. Записываем в ключ HKLMReginaKey значение TRUE, константа имени KEYINSTALL, функция PutLongRoReg.

7. Вызываем функцию AssocDocument, параметр sa по ссылке, результат принимаем в переменную lResult и проверяем.

В конце функции закрываем ключ key, освобождаем идентификаторы безопасности psidAdmins и psidEveryone при помощи функции FreeSid, возвращаем значение TRUE.

В результате выполнения этой функции в реестре должны быть сформированы следующие записи и значения:

Ключ/подключ	Имя	Тип	Значение
HKEY_CLASSES_ROOT			
— .regina	(По умолчанию)	REG_SZ	Regina
— Regina	(По умолчанию)	REG_SZ	ОТИ-ПМ-Regina
— DegaultIcon	(По умолчанию)	REG_SZ	c:\regina\Debug\regina.exe,2
— Shell			
— Open			
— Command	(По умолчанию)	REG_SZ	c:\regina\Debug\regina.exe %1
HKEY_LOCAL_MACHINE			
— Software			
— ОТИ-ПМ			
— Regina			
— 1.0	(По умолчанию)	REG_SZ	(значение не присвоено)
	Installed	REG_DWORD	0x00000001 (1)
	Организация	REG_SZ	ОТИ
	Фамилия	REG_SZ	Студент
— Default	(По умолчанию)	REG_SZ	(значение не присвоено)
	Locale	REG_DWORD	0x00000419 (1049)

5.8. Создание области текущего пользователя

Модуль regb.h, функция InitUser. Функция создает ключи и записи в улье HKCU, а ее код во многом повторяет код функции StoreAppConfig.

Сначала определим код завершения функции:

```
    return TRUE;
registry_damage:
    ShowReginaError("InitUser: registry damage", lResult);
    RegDeleteValueA(keyHKLM, KEYINSTALL);
    goto clean_up_registry;
registry_failure:
    ShowReginaError("InitUser: registry failure", lResult);
clean_up_registry:
    if (HKCUReginaKey) RegDeleteKeyA(HKCU, szRegPath);
    if (key) RegCloseKey(key);
    goto clean_up;
security_failure:
    ShowReginaError("InitUser: security failure", GetLastError());
clean_up:
    if (psidAdmins) FreeSid(psidAdmins);
    if (ptu) LocalFree(ptu);
    return FALSE;
```

Здесь определяется больше меток, так как появляется возможность нарушения записей в реестре, созданных предыдущей функцией. Они могут быть, например, удалены.

Переходы на метку security_failure выполняются, если какая-либо функция создания структур безопасности завершается неудачно. Переходы на метку registry_failure выполняются, если неудачно завершается функция создания записи в реестре. Переходы на метку registry_damage выполняются, когда не удается прочитать данные из глобальной области.

Переходим в начало функции.

Определяем поля структуры атрибутов безопасности точно так же, как это сделано в предыдущей функции.

Создаем идентификатор безопасности для группы администраторов, точно так же, как это сделано в предыдущей функции.

Чтобы получить идентификатор безопасности текущего пользователя, сначала получаем информацию о нем при помощи функции GetTokenInfo, подставляя значение TokenUser, возвращаемое значение приводим к типу PTOKEN_USER и записываем в переменную ptu. Проверяем значение ptu. Если оно равно NULL, то переходим на метку security_failure.

Затем получаем идентификатор безопасности текущего пользователя psidUser, который записан в поле User.Sid идентификатора информации:

```
    psidUser = ptu->User.Sid;
    if (!psidUser) goto security_failure;
```

Результат проверяем, при неудаче переходим на метку security_failure.

Инициализируем дескриптор безопасности точно так же, как это сделано в предыдущей функции.

Прикрепляем разрешения sdPermissions к текущему пользователю при помощи функции SetSecurityDescriptorOwner, подставляя sdPermissions по ссылке, psidUser и 0.

Если функция возвращает значение FALSE, переходим на метку security_failure.

Выделяем память для списка дискреционного доступа точно так же, как это сделано в предыдущей функции. При ошибке в выводе сообщения вместо названия функции StoreAppConfig записываем InitUser.

Инициализируем список дискреционного доступа точно так же, как это сделано в предыдущей функции.

Формируем список дискреционного доступа. В списке также будет две записи управления доступом (ACE).

Первая запись предоставляет полный доступ текущему пользователю.

Вторая запись предоставляет полный доступ группе администраторов.

Два раза вызываем функцию AddAccessAllowedAce.

При первом вызове подставляем список доступа paclKey, константу ACL_REVISION2, константу доступа KEY_ALL_ACCESS, дескриптор безопасности psidUser.

При втором вызове подставляем список доступа paclKey, константу ACL_REVISION2, константу доступа KEY_ALL_ACCESS, дескриптор безопасности psidAdmins.

Связываем список дискреционного доступа (DACL) с дескриптором безопасности точно так же, как это сделано в предыдущей функции.

Далее создаем ключи с полученными атрибутами безопасности sa.

1. Создаем запись HKCU\Software\ОТИ-ПМ\Regina. Константа доступа к ключу KEY_ALL_ACCESS. Возвращаемый ключ сохраняем в глобальной переменной HKCUReginaKey. Если значение lResult не равно нулю, переходим к метке registry_failure.

2. Создаем запись HKCU\Software\ОТИ-ПМ\Regina\User Data. Ключ отсчета HKCUReginaKey, константа доступа KEY_ALL_ACCESS. Возвращаемый ключ сохраняем в локальной переменной key. Если значение lResult не равно нулю, переходим к метке registry_failure.

3. Закрываем ключ key.

4. Открываем ключ данных по умолчанию в глобальной области. Функция OpenReginaKey, параметры keyHKLM, имя параметра KEYDEFAULT, константа доступа KEY_READ, key по ссылке. Если значение lResult не равно нулю, переходим к метке registry_damage.

5. Считываем глобальное значение идентификатора локали по умолчанию из ключа key. Функция GetLongFromReg, параметры key, константа имени значения KEYLOCALE, переменная dwDefaultLCID, dwType по ссылке.

В переменную dwDefaultLCID принимается считанное значение, а в переменную dwType — тип данных этого значения.

Проверка результата в этом случае сложнее. Функция завершается неудачно при двух условиях: если значение lResult не равно нулю, или если значение переменной dwType не равно REG_DWORD. В случае неудачи переходим на метку registry_damage.

6. Закрываем ключ key.

7. Записываем считанное глобальное значение идентификатора локали по умолчанию dwDefaultLCID в ключ отсчета HKCUReginaKey, константа имени параметра KEYLOCALE. Если значение lResult не равно нулю, переходим к метке registry_failure.

8. Сбрасываем данные на диск при помощи функции RegFlushKey, параметр HKCUReginaKey.

9. Записываем в ключ HKCUReginaKey значение TRUE, константа имени KEYINSTALL, функция PutLongRoReg. Результат можно не проверять.

В завершение освобождаем идентификатор безопасности psidAdmins при помощи функции FreeSid, затем освобождаем память, выделенную под структуру данных ptu, при помощи функции LocalFree, возвращаем TRUE.

В результате выполнения этой функции в реестре должны быть сформированы следующие записи и значения:

Ключ/подключ	Имя	Тип	Значение
HKEY_CLASSES_ROOT			
HKEY_CURRENT_USER			
— Software			
— ОТИ-ПМ			
— Regina			
— 1.0	(По умолчанию)	REG_SZ	(значение не присвоено)
	Installed	REG_DWORD	0x00000001 (1)
	Locale	REG_DWORD	0x00000419 (1049)
HKEY_LOCAL_MACHINE			

В завершение работы выполните экспорт созданных ключей из редактора реестра в формате Win9x/NT4, папка сохранения C:\regina, названия файлов hkcr1, hkcr2, hkcu, hklm. Запишите содержимое файлов в отчет.

5.9. Контрольные вопросы и упражнения

1. Назовите корневые ключи реестра.
2. Назовите основные типы данных реестра.
3. Какая информация хранится в корневых ключах?
4. Поясните соотношение корневых узлов, ульев и файлов.
5. Что такое SID? Каково его строковое представление?
6. Опишите хорошо известные идентификаторы безопасности.
7. Поясните записи улья HKU. Каким субъектам они соответствуют? Используйте для поиска информации файл winnt.h.
8. Что содержит маркер доступа? Какие объекты имеют его?
9. Что содержит дескриптор безопасности? Какие объекты имеют его?
10. Как происходит сравнение маркера с дескриптором безопасности?
11. Что такое ACL, DACL, SACL?
12. В чем заключается различие между нулевым и пустым DACL?
13. Поясните процесс создания дескриптора безопасности.

6. Работа SY-106. Управление памятью

Цели:
- исследование механизмов управления памятью.
Задачи:
- исследование адресного пространства процесса;
- исследование динамически распределяемой памяти;
- изучение механизма проецирования файлов в память.
Опорные документы:
[1, «Управление памятью»]

6.1. Шаблон проекта

Войдите в систему под учетной записью администратора. Получите архив работы SY-106. Извлеките каталог memory в корневой каталог диска C:. Откройте проект. Убедитесь, что целевой платформой является x86 или Win32. В проекте два модуля. Модуль memory.h предназначен для вспомогательных функций. Модуль memory.cpp предназначен для алгоритмов программы. В начале этого модуля укажите сведения об организации, о себе, о проекте, дату начала работы:

```
// ОТИ НИЯУ МИФИ
// 1ПО-00Д
// Студент Имя Отчество
// Системное программирование
// SY-106. Управление памятью
// 01.01.2000
```

Заметим, что основная функция расположена в модуле memory.h и она содержит обработчик исключительных ситуаций. Алгоритмы программы размещаем в функции mainf модуля memory.cpp.

6.2. Системная информация

Сначала получим системную информацию при помощи системной функции GetSystemInfo. Четыре параметра, относящиеся к памяти, запишем в отчет. Определим объем виртуального адресного пространства процесса в гигабайтах. Размер страницы запомним в переменной page, которая определена в модуле memory.h. Информацию о процессоре рассмотрим, в отчет запишем количество процессоров.

Далее получим информацию о памяти, функция GlobalMemoryStatus. Перед тем, как передать структуру данных этой функции, установите размер структуры в поле dwLength. Данные, полученные в структуру данных, запишем в отчет. Сравним их с размером виртуального адресного пространства, полученным ранее. Вызов функции GlobalMemoryStatus больше не понадобится, вынесем его в комментарий в конце модуля memory.cpp.

6.3. Вспомогательные функции

В модуле memory.h необходимо определить несколько вспомогательных функций. В начале модуля определим предикат any:

```
// предикат любое значение
int any(DWORD value) { return 1; }
```

Он возвращает истину при любом значении параметра.

Предикат isa возвращает истину, когда битовая маска присутствует в значении:

```
// предикат маски
int isa(DWORD value, DWORD mask) { return (value & mask) == mask; }
```

Функция ShowProtect показывает атрибуты защиты, который передается как параметр value. Цель — сформировать в потоке строку типа

Prot: READWRITE GUARD.

Атрибут защиты складывается из собственно атрибута и флага (который может быть и не задан). Для отделения атрибута от флага нужна функция isa. Первый параметр указывает поток для вывода информации. Выводить информацию будем в файл, который мы впоследствии откроем.

Сначала выводим в поток строку "Prot: " при помощи fprintf. Перевод строки не требуется, после двоеточия пробел.

Затем проверяем значение value. Если оно нулевое, выводим в поток слово "NONE", строку не переводим. Если значение не нулевое, то сравниваем value с каждой из констант атрибута защиты следующим образом:

```
if (isa(value, PAGE_NOACCESS)) fprintf(stream, "NOACCESS ");
```

Обратим внимание на пробел после NOACCESS, он нужен для того, чтобы слова не сливались.

Строк столько, сколько всего есть констант:

1) PAGE_NOACCESS, 2) PAGE_READONLY, 3) PAGE_READWRITE,

4) PAGE_WRITECOPY, 5) PAGE_EXECUTE, 6) PAGE_EXECUTE_READ,

7) PAGE_EXECUTE_READWRITE, 8) PAGE_EXECUTE_WRITECOPY.

За атрибутами точно так же проверяем три флага:

1) PAGE_WRITECOMBINE, 2) PAGE_NOCACHE, 3) PAGE_GUARD.

Префикс констант PAGE_ в выводе опускаем.

В конце функция выводит в поток перевод строки.

Для тестирования функции пробуем вывести в консоль какое-нибудь значение, например, так:

```
void mainf(void) {
    ShowProtect(stdout, PAGE_READWRITE | PAGE_GUARD);
    return;
    . . .
}
```

Вывод должен быть следующим:

Prot: READWRITE GUARD

Далее аналогичным образом определяем функцию ShowState. Функция показывает состояние памяти, которое может быть одним из четырех:

NONE, MEM_FREE, MEM_RESERVE, MEM_COMMIT.

Сначала выводим в поток строку "Stat: ". Затем проверяем значение value. Если оно нулевое, выводим в поток слово "NONE", строку не переводим. Если значение value не нулевое, описываем проверку точно так же, как в функции ShowProtect. Префикс MEM_ выводить не нужно.

Следующая функция, ShowType, выводит тип памяти. Она полностью аналогична функции ShowState, только вместо слова Stat выводит в начале строки слово Type, а проверяемые константы следующие:

MEM_IMAGE, MEM_MAPPED, MEM_PRIVATE.

Префикс MEM_ выводить не нужно.

Если функции определены, переходим к функции, которая покажет сведения о регионе или блоке памяти.

6.4. Вывод сведений о регионе

Сведения о регионе выведет функция ShowMem. Второй параметр addr задает адрес, который должен попадать в исследуемый регион. Функция возвращает размер региона, который будет получен функцией VirtualQuery. Сначала объявим структуру MEMORY_BASIC_INFORMATION, переменная mbi. Затем вызываем функцию VirtualQuery:

```
DWORD result = VirtualQuery((LPCVOID)addr, &mbi, page / 8);
```

Функция возвращает число записанных в структуру mbi байт, или ноль в случае неудачи, это значение попадает в переменную result. Непосредственно за вызовом VirtualQuery проверяем значение result, и если оно равно нулю, возвращаем из ShowMem значение 0x80000000, равное размеру памяти в 4 Гбайт. Для чего это нужно, будет понятно далее.

Затем в переменную size типа DWORD получим размер региона из поля mbi.RegionSize. Это возвращаемое значение функции ShowMem, и его нужно возвращать в случае любого завершения. Чтобы дальнейшие действия были понятны, приведем пример вывода функции ShowMem:

```
31
Addr: 0x00400000
Base: 0x00400000
Size: 0x1000
Prot: EXECUTE_WRITECOPY
Stat: COMMIT
Type: IMAGE
```

Здесь выведены сведения о регионе, порядковый номер которого 31. Был задан адрес addr, равный 0x00400000. Базовый адрес региона оказался таким же. Размер региона 0x1000 (одна страница), атрибут защиты равен PAGE_EXECUTE_WRITECOPY, флагов нет, статус памяти равен MEM_COMMIT, тип памяти равен MEM_IMAGE (образ PE).

В соответствии с требуемым выводом сначала выводим порядковый номер, который хранит переменная block, определенная в memory.h. Перед выводом ее нужно инкрементировать, чтобы первый блок имел номер 1. Спецификация вывода "%d\n".

Затем выводим адрес, заданный параметром addr. Спецификация вывода "Addr: 0x%p\n". Значение addr нужно примести к типу int*. Далее аналогично выводим базовый адрес региона, взятый из поля mbi.AllocationBase. Спецификация вывода "Base: 0x%p\n". Затем выводим размер блока, который записан в переменную size. Спецификация вывода "Size: 0x%x\n". Значение size нужно привести к типу int.

Далее вызываются функции ShowProtect, ShowState, ShowType, им передается первый параметр функции ShowMem, и соответствующие поля структуры mbi, а именно AllocationProtect, State и Type. В завершение функция выводит в поток перевод строки, хотя это не обязательно, и возвращает значение переменной size.

Теперь функцию нужно отладить. Функция VirtualQuery капризна, она может не выдавать никаких сведений по соображениям безопасности, если параметры подобраны неподходящим образом, и, вообще говоря, после вызова VirtualQuery следовало бы вызвать GetLastError. Мы поступим проще, введя в окно Watch переменную "@err, hr", которая будет показывать текущее значение ошибки. Это вообще обязательно делать всегда. Экспериментально было установлено, что в некоторых операционных системах при неудачном сочетании адреса и размера региона функция генерирует ошибку номер 24 (когда размер мал) и 998 (в других случаях). Поэтому наша цель сейчас — убедиться, что функция VirtualQuery выдает сведения.

Переходим в функцию mainf. Отладочный код из нее удалим. Функция должна определять размер страницы page. Затем мы выделим регион памяти, вызовем функцию ShowMem, и освободим регион памяти:

```
void mainf(void) {
    SYSTEM_INFO si;
    GetSystemInfo(&si);
    page = si.dwPageSize;
    PVOID b = VirtualAlloc(0, SIZE, MEM_COMMIT, PAGE_READWRITE);
    ShowMem(stdout, (DWORD)b);
    VirtualFree(b, SIZE, MEM_DECOMMIT);
    ShowMem(stdout, (DWORD)b);
}
```

Константа SIZE задана в модуле, она равна размеру памяти 64 Кбайт.

Установим точку остановки на вызове VirtualAlloc, запускаем, вводим в окно Watch переменную "@err, hr", исполняем программу шаг за шагом, контролируя переменную при вызовах VirtualAlloc, VirtualQuery, VirtualFree. Если вызов VirtualQuery генерирует ошибку, можно попытаться изменить размер блока, например, page / 4, page / 2, page. В отчет запишем адрес выделенного блока памяти и статус памяти после освобождения.

6.5. Исследование виртуальной памяти процесса

Наша следующая задача — исследовать распределение виртуальной памяти процесса в пределах первых 2 Гбайт. Для этого вызываем ShowMem с начальным адресом 0. Выводится первый блок, а ShowMem возвращает его размер. Прибавляем размер к начальному адресу, выводим следующий блок и так далее до того момента, когда адрес превысит значение 2 Гбайт. Фактически это просто бесконечный цикл.

Сначала нужно создать файл для записи блоков, например, так:

```
void mainf(void) {
    SYSTEM_INFO si;
    GetSystemInfo(&si);
    page = si.dwPageSize;
    stream = fopen(memmap, "wt");
    if (!stream) return;
    SetLastError(0);
    /* здесь цикл исследования */
    if (stream != stdout) fclose(stream);
}
```

Затем формируем бесконечный цикл (место указано выше).

Перед собственно циклом объявляем переменную addr типа DWORD с начальным значением 0. Затем описываем бесконечный цикл. В цикле:

- объявим переменную size типа DWORD, вызовем ShowMem, результат примем в size;
- прибавим size к addr;
- если addr больше 0x40000000, или count больше 1000, то break;
- инкрементируем count.

Значение 0x40000000 равно 2 Гбайт, поэтому, если ShowMem вернет значение в 4 Гбайт, цикл немедленно завершится.

После цикла выведем в консоль значение переменных block и count. Эти значения покажут разницу между всего блоков и выведенных блоков.

Запускаем программу, ждем окончания и изучаем файл вывода. При разных запусках распределение может быть разным, и число блоков может изменяться, в диапазоне примерно 40-60.

Нужно понять разницу между регионом и блоком. Блок — это часть региона, возможно, равная ему. Если не равная, то в регионе несколько блоков с одним и тем же базовым адресом, но разными статусами и типами памяти. Найдите в выводе регион с минимум двумя блоками, запишите в отчет сведения об этих блоках.

6.6. Поиск определенных блоков

Теперь модернизируем функцию ShowMem так, чтобы она выводила блоки с заданными параметрами. Для этого в определение функции нужно добавить три дополнительных параметра — указателя на предикаты:

```
DWORD ShowMem(FILE * stream, DWORD addr, int(*pcond)(DWORD),
        int(*scond)(DWORD), int(*tcond)(DWORD)) {
```

Теперь в самой функции используем предикаты для вывода только интересующих нас блоков:

```
    if (result == 0) return 0x80000000;
    DWORD size = mbi.RegionSize;
    if (!protcond(mbi.AllocationProtect)) return size;
    if (!statcond(mbi.State)) return size;
    if (!typecond(mbi.Type)) return size;
    fprintf(stream, "%d\n", ++block);
```

Соответственно, в вызове ShowMem теперь потребуются предикаты.

У нас есть только один предикат any, функция mainf:

```
    DWORD size = ShowMem(stream, addr, any, any, any);
```

Другие предикаты определим в модуле memory.h. Для примера предикат для поиска блоков с атрибутом защиты PAGE_EXECUTE_WRITECOPY определяется следующим образом:

```
int ew(DWORD value) { return isa(value, PAGE_EXECUTE_WRITECOPY); }
```

Для атрибутов защиты определим таким же образом предикат ro для блоков PAGE_READONLY, и предикат rw для блоков PAGE_READWRITE.

Для статуса памяти определим предикаты image, mapped, privat.

Для типа памяти определим предикаты free, reserv, commit.

Теперь определяем и записываем в отчет число блоков:

- PAGE_EXECUTE_WRITECOPY;
- PAGE_READONLY;
- PAGE_READWRITE;
- MEM_IMAGE;
- MEM_MAPPED;
- MEM_PRIVATE;
- MEM_FREE;
- MEM_RESERVE;
- MEM_COMMIT.

Естественно, можно одновременно использовать два или три предиката, это на усмотрение преподавателя.

6.7. Образы файлов

Интересно также узнать, образы каких файлов находятся в блоках типа MEM_IMAGE и MEM_MAPPED. Функция GetMappedFileName возвращает путь к файлу, который отображен на регион. Она требует адрес памяти, поэтому изменим определение функции ShowType, добавив параметр:

```
void ShowType(FILE * stream, DWORD value, DWORD addr) {
```

В функции ShowMem подставим параметр addr в вызов ShowType:

```
    ShowType(stream, mbi.Type, addr);
```

Функцию ShowType изменяем так, чтобы к слову IMAGE добавлялся путь к соответствующему файлу:

```
if (isa(value, MEM_IMAGE)) {
    fprintf(stream, "IMAGE ");
    if (GetMappedFileName(GetCurrentProcess(), (LPVOID)addr,
        buf, MAX_PATH)) fprintf(stream, buf);
}
```

То же самое нужно сделать для памяти типа MEM_MAPPED.

Если возникает ошибка компоновщика, откроем свойства проекта, раздел Linker | Command Line, впишем библиотеку psapi.lib в окно ввода.

Запускаем программу и анализируем файл вывода. Используем сначала предикат image, затем предикат mapped. Запишем в отчет разницу между количеством блоков MEM_MAPPED и количеством таких же блоков, с которыми сопоставлены файлы.

На этом исследование виртуальной памяти процесса закончим, хотя много вопросов осталось «за кадром». Например, как найти в блоки, соответствующие стеку потока, освобождение физической памяти при сбросе, сборка мусора и другие.

6.8. Кучи

Это совсем маленькое исследование. Сначала имеющийся код mainf вынесем в комментарий в конце модуля, оставив вызов GetSystemInfo и определение размера страницы, это нужно для функции ShowMem.

Сначала получим в переменную heap типа HANDLE дескриптор кучи процесса при помощи функции GetProcessHeap. Затем в переменную b типа PVOID выделим память размером 3 байта при помощи функции HeapAlloc. Затем определим фактический размер выделенной памяти при помощи функции HeapSize, возвращаемое значение примем в переменную size типа SIZE_T. Затем выведем в консоль блок памяти b при помощи ShowMem, b при этом приводим к типу DWORD. Затем вернем память в кучу при помощи HeapFree. Все полученные значения запишем в отчет, а код перенесем в комментарий в конце модуля, за исключением определения размера страницы.

Теперь создадим новую кучу при помощи HeapCreate, возвращаемое значение принимаем в переменную heap типа HANDLE, параметры: первый константа HEAP_GENERATE_EXCEPTIONS, второй и третий константа SIZE.

Проверяем heap, если NULL, то return. Далее разрушаем кучу при помощи HeapDestroy. Перед разрушением кучи пытаемся выделить блок памяти размером SIZE при помощи HeapAlloc, возвращаемый адрес принимаем в переменную b типа PVOID. Затем вызываем ShowMem, передаем адрес b, приведенный к типу DWORD. Если генерируется исключение, последовательно вычитаем из SIZE числа, кратные 1024, пока исключение не будет возникать. Запишем в отчет вычтенное число и вывод ShowMem.

6.9. Проецирование файлов в память

Проецирование файлов в память несложно, поэтому мы разнообразим задачу следующим образом. Первая копия программы создает файл и отображает его в память. Вторая копия программы отображает файл и изменяет его. Первая копия ожидает изменения и только после этого завершается.

Прежде чем все это получится, нужно отладить две функции. Первая функция создает файл, вторая — изменяет его. Код функции mainf сейчас нужно удалить в комментарий в конце модуля.

В любом случае первое действие копии программы — открыть файл при помощи CreateFile. Открывать нужно в режиме OPEN_ALWAYS. Если файл существует, GetLastError вернет ERROR_ALREADY_EXISTS, и это будет служить признаком того, какая функция должна быть вызвана.

Сначала перед функцией mainf опишем две похожие функции. Первая называется map_write, вторая — map_read. Обе функции имеют тип void, единственный параметр hFile имеет тип HANDLE, то есть дескриптор файла.

Теперь в функции mainf вызываем функцию CreateFile.

Возвращаемое значение присваиваем переменной hFile типа HANDLE.

Имя файла задано переменной mapfile в начале модуля.

Тип доступа GENERIC_READ и GENERIC_WRITE.

Режим разделения FILE_SHARE_READ и FILE_SHARE_WRITE.

Атрибуты безопасности нас не интересуют, значит, NULL.

Режим создания, как сказано выше, OPEN_ALWAYS.

Атрибуты файла FILE_ATTRIBUTE_NORMAL. Последний параметр 0.

Далее проверяем hFile. Если hFile равно INVALID_HANDLE_VALUE, возникла ошибка создания файла, при помощи printf выводим в консоль сообщение "failed to create file.\n", возвращаемся из функции.

Далее сравниваем значение, возвращаемое функцией GetLastError, с константой ERROR_ALREADY_EXISTS. Если совпадает, файл существует.

Тогда сбрасываем код последней ошибки при помощи SetLastError, спим при помощи функции Sleep, например, 5 секунд, после чего вызываем функцию map_read, передаем ей hFile.

Если GetLastError возвращает иное значение, файла не существует, в этом случае просто вызываем map_write, передаем ей hFile.

Для отладки открываем FAR, переходим в каталог c:\memory. Исполняем программу, убеждаемся, что файл успешно создается и имеет нулевую длину. По завершении файл нужно удалить (клавишей F8).

6.9.1. Запись файла

Переходим к функции map_write. Создаем проекцию при помощи функции CreateFileMapping. Возвращаемое значение принимаем в переменную hMap типа HANDLE. Атрибуты безопасности NULL. Атрибут защиты страниц PAGE_READWRITE. Размер файла 0 и 16. Имя проекции mapname.

Проверяем hMap. Если NULL, то закрываем дескриптор hFile, выводим в консоль сообщение "1: failed CreateFileMapping.\n", завершаем функцию.

Если проекция создана, проецируем файл на память при помощи функции MapViewOfFile. Возвращаемое значение принимаем в переменную addr типа PVOID. Тип доступа FILE_MAP_ALL_ACCESS. Размер файла 0 и 0, проецируем 16 байт.

Проверяем addr. Если NULL, то закрываем дескриптор hMap, закрываем дескриптор hFile, выводим в консоль сообщение "1: failed MapViewOfFile.\n", завершаем функцию.

Для работы с памятью объявляем указатель p типа char*, присваиваем ей значение addr, приведенное к типу char*. Копируем в указатель p строку "1234567890123456", состоящую из 16 цифр, при помощи функции strcpy. При этом нулевой байт запишется в страницу как семнадцатый символ, но это не имеет значения. Далее выводим в консоль записанное содержимое при помощи функции printf, строка формата "1: write: %s\n", переменная p.

В завершении функции отменяем проецирование при помощи функции UnmapViewOfFile, передавая ей addr, закрываем дескриптор hMap, закрываем дескриптор hFile.

Отлаживаем функцию, исполняя ее шаг за шагом, контролируя значение переменной "@err, hr". Если все верно написано, будет создан файл из 16 байт. Этот файл удаляем только в случае, если во время отладки возникли какие-то ошибки. По завершении файл не удаляем, он требуется для отладки второй функции.

6.9.2. Чтение файла

Переходим к функции map_read. Она практически полностью аналогична предыдущей, поэтому скопируем код. Различия следующие.

1) Функция начинается с определения размера файла. Используем функцию GetFileSize, принимая возвращаемое значение в переменную size типа DWORD. Второй параметр этой функции равен NULL.

2) Вместо литерала 16 (длина файла) используем переменную size (в двух местах, функция CreateFileMapping и функция MapViewOfFile).

3) В сообщениях заменим "1:" на "2:". Единица и двойка обозначают первый и второй процесс, получающие доступ к проекции файла.

Действия функции map_read после определения указателя p другие.

Сначала функция выводит содержимое файла при помощи функции printf, строка формата "2: read: %s\n", переменная p. Таким образом мы показываем, что находится в файле до его изменения. Затем функция записывает в файл строку "abcdefghijklmnop" при помощи функции strcpy, используя указатель p. Затем повторно выводит содержимое файла при помощи printf, строка формата "2: write: %s\n".

Отлаживаем функцию, убеждаемся, что файл изменяется, но размер остается равным 16 байт. При необходимости удалим файл map.txt.

6.9.3. Взаимодействие процессов через проекцию файла

В конце функции mainf вызовем функцию _getch, чтобы программа процесса приостановилась до нажатия какой-нибудь клавиши, и мы могли видеть, что пишут процессы в консоль.

Внесем изменения в функцию map_write (первую). После того, как функция выведет содержимое файла, выведем сообщение "1: waiting...\n". Далее описываем бесконечный цикл. В цикле сначала спим 0 секунд, затем сравниваем нулевой байт указателя p со значением '1'. Если не совпадает, завершаем цикл при помощи break. После цикла повторно выводим содержимое файла, строка формата при этом "1: read: %s\n". Далее следует отмена проекции и закрытие дескрипторов.

Компилируем программу, переходим в FAR. Каталог c:\memory.

Создаем пакетный файл start.bat (Shift+F4):

```
start Debug\memory.exe
start Debug\memory.exe
```

Удаляем файл map.txt.

Запускаем пакет и наблюдаем взаимодействие процессов.

6.10. Контрольные вопросы и упражнения

1. Каков размер виртуальной памяти процесса, как делится адресное пространство процесса между приложением и операционной системой?
2. Какие разделы есть в адресном пространстве процесса?
3. Что называется резервированием памяти и передачей памяти?
4. Что такое грануляция памяти?
5. В чем разница между регионом памяти и блоком памяти?
6. Назовите атрибуты защиты блока памяти, поясните их.
7. Назовите флаги атрибутов защиты блока, поясните их.
8. Назовите статусы памяти блоков, поясните их.
9. Назовите типы памяти блоков, поясните их.
10. Какова последовательность работы с проекцией файла?
11. Как задается размер файла при проецировании?
12. Что необходимо для взаимодействия процессов через проекцию?
13. Назовите 4 причины создания дополнительных куч процесса.
14. Исследуйте адресное пространство приложения Блокнот. Для этого модернизируйте функции ShowType и ShowMem так, чтобы они принимали дескриптор процесса. Вместо функции VirtualQuery в функции ShowMem используйте VirtualQueryEx. Далее создайте процесс приложения Блокнот и выведите распределение его памяти.

7. Работа SY-107. Библиотеки

Цели:
- исследование статических и динамически подключаемых библиотек.

Задачи:
- построение и компоновка статической библиотеки;
- построение DLL;
- неявное связывание с DLL;
- явное связывание с DLL.

Опорные документы:
[1, «Библиотеки»]

7.1. Шаблон проекта

Получите шаблон работы SY-107. Извлеките из архива каталог library в корневой каталог диска C:. Откройте проект. Убедитесь, что целевой платформой является x86 или Win32. В решении четыре проекта: для статической библиотеки onelib, для ее тестирования testlib, для динамической библиотеки onedll, для ее тестирования testdll.

7.2. Статическая библиотека

Проект onelib, модуль onelib.h.

Описываем экспортируемую функцию:

```
// возвращает сумму
int Sum(int, int);
```

Модуль onelib.cpp. В начале модуля указываем информацию об организации, о себе, о проекте, дату начала работы:

```
// onelib.cpp
// ОТИ НИЯУ МИФИ
// 1ПО-00Д
// Фамилия Имя Отчество
// Системное программирование
// SY-107. Статическая библиотека
// 01.01.2000
```

Ниже описываем собственно функцию:

```
// возвращает сумму
int Sum(int x, int y) {
    return (x + y);
}
```

Компилируем проект, убеждаемся, что в каталоге c:\onelib\Debug появился файл onelib.lib. Это и есть статическая библиотека.

Переходим в тестовый проект testlib.

В начале модуля testlib.cpp указываем информацию об организации, о себе, о проекте, дату начала работы:

```
// testlib.cpp
// ОТИ НИЯУ МИФИ
// 1ПО-00Д
// Фамилия Имя Отчество
// Системное программирование
// SY-107. Тестирование статической библиотеки
// 01.01.2000
```

Добавляем в проект модуль onelib.lib, в каталог Source Files. Если возникнут какие-то диалоги, закрываем их. В конечном итоге в списке файлов проекта testlib должен появиться файл onelib.lib.

Вызываем Sum в функции main, и убеждаемся, что она суммирует.

Добавим в проект onelib какой-нибудь класс, например, класс point.

В модуле onelib.h описываем интерфейс класса, состоящий из конструктора по умолчанию, метода void move(int, int) и метода int getX(void). Элементами данных являются x и y в закрытой секции.

В модуле onelib.cpp описываем реализацию конструктора и методов.

Перекомпилируем проект onelib.

Переходим в проект testlib, функция main. Описываем точку и вызовы методов move и getX. Компилируем проект. Запускаем программу, убеждаемся, что класс работает.

7.3. Динамически подключаемая библиотека

Проект onedll, модуль onedll.h.

Описываем модификатор экспорта и импорта функций:

```
// модификатор экспорта или импорта
#ifdef ONEEXPORTS
#define ONEDLLAPI __declspec(dllexport)
#else
#define ONEDLLAPI __declspec(dllimport)
#endif
```

Если перед включением этого модуля в другой модуль определен символ ONEEXPORTS, константа ONEDLLAPI получит значение модификатора экспорта __seclspec(dllexport), иначе импорта __seclspec(dllimport).

Ниже описываем собственно импортируемую функцию:

```
// экспортируемая функция
extern "C" ONEDLLAPI int Sum(int x, int y);
```

Модуль onedll.cpp.

В начале модуля указываем информацию об организации, о себе, о проекте, дату начала работы:

```
// ОТИ НИЯУ МИФИ
// 1ПО-00Д
// Фамилия Имя Отчество
// Системное программирование
// SY-107. DLL
// 01.01.2000
```

Ниже описываем константу ONEEXPORTS, так как библиотека экспортирует, после чего включение заголовочного модуля:

```
#define ONEEXPORTS
#include "onedll.h"
```

Ниже описываем собственно экспортируемую функцию:

```
// возвращает сумму
int Sum(int x, int y) {
    return (x + y);
}
```

Компилируем проект, убеждаемся, что появились файлы onedll.dll и onedll.lib.

Тестовый проект testdll. В начале модуля testdll.cpp указываем информацию об организации, о себе, о проекте, дату начала работы:

```
// testlib/cpp
// ОТИ НИЯУ МИФИ
// 1ПО-00Д
// Фамилия Имя Отчество
// Системное программирование
// SY-107. Тестирование DLL
// 01.01.2000
```

Ниже включаем модуль onedll.h.

Указываем расположение файла onedll.lib (именно lib-файла, а не dll-файла). Для этого в открываем свойства проекта, раздел Linker, подраздел Input, поле Additional Dependencies, нажимаем кнопочку Edit (если нет, то кнопочка с троеточием), вводим строку ..\Debug\onedll.lib.

В принципе, все. После этого в функции main вызываем экспортируемую функцию Sum, убеждаемся, что она суммирует.

Копируем из предыдущего решения класс point и вставляем в это решение, в модуль onedll.h интерфейс, в модуль onedll.cpp реализацию. Для экспорта класса перед именем класса нужно вставить ONELIBAPI. Убеждаемся, что класс экспортируется. Создаем объект и вызываем только move, то есть только конструктор и move связываются.

Открываем FAR, каталог c:\onedll. Вводим команды:

```
dumpbin -exports onedll.dll > dll.txt
dumpbin -imports testdll.exe > testdll.txt
```

Записываем в отчет таблицу экспорта из файла dll.txt, таблицу импорта из файла test.txt (только для onedll). Убеждаемся, что экспортируются и импортируются декорированные имена методов. Вероятно, эту DLL нельзя будет использовать с другим компилятором для экспорта класса point. И без декорации методов тоже нельзя, как сказано в msdn.

В таблице экспорта ordinal предназначено для обратной совместимости с Win16, hint используется операционной системой, RVA показывает смещение функции (relative virtual address).

Теперь обратимся к функции DllMain.

Нужно исследовать, какие события возникают. Для этого в каждое событие вписываем вызов MessageBox, например:

```
case DLL_PROCESS_ATTACH:
    MessageBox(0, "PROCESS_ATTACH", 0, MB_OK);
    break;
```

Компилируем, запускаем, записываем в отчет возникшие сообщения.

Удаляем вызовы MessageBox.

7.4. Явное связывание

Модуль testdll.cpp. Описываем тип EXPFUN:

```
typedef int (*EXPFUN)(int, int);
```

Функция main. У нас есть неявное связывание, поэтому получим дескриптор DLL в переменную hModule типа HMODULE при помощи функции GetModuleHandle. Затем получим адрес функции Sum в переменную addr типа EXPFUN при помощи функции GetProcAddress. Затем вызываем функцию Sum, используя имя addr. Проверки всех значений обязательны!

7.5. Контрольные вопросы и упражнения

1. Поясните различие между статической библиотекой и DLL.
2. Назовите преимущества и недостатки тех и других библиотек.
3. Как к проекту подключается статическая библиотека?
4. Как к проекту подключается DLL?
5. Назовите 4 вида соглашений о вызове функций (методов). Какими модификаторами они задаются?
6. Что такое декорация имен? Как ее избежать для экспорта?
7. Как указывается экспортируемая (импортируемая) функция, класс?
8. В чем различие между явным и неявным связыванием?
9. Назовите порядок поиска DLL.
10. Для чего нужна функция DllMain?
11. Что называется переадресацией вызова?
12. Какие DLL называются известными?
13. Для чего нужна команда dumpbin?
14. Для чего нужна команда rebase?
15. Создайте статическую библиотеку regina.lib, скомпонуйте ее с законченным приложением regina. Единственная функция GetReginaRecordLib должна возвращать новую запись ReginaRecord. В приложении regina есть пункт меню для вызова этой функции.
16. Создайте DLL regina.dll с одной функцией GetReginaRecordDll, которая возвращает новую запись ReginaRecord. В приложении regina есть пункт меню для вызова этой функции. Используйте неявное связывание.

8. Работа SY-108. Передача информации между процессами

Цели:
- изучение основных механизмов связи между процессами.

Задачи:
- передача текста через буфер обмена;
- передача картинки через буфер обмена;
- передача текста сообщением WM_COPYDATA;
- исследование механизма DDE;
- передача текста через почтовый ящик;
- передача текста через канал (pipe).

Опорные документы:
[1, «Взаимодействие между процессами»]

8.1. Рабочее пространство

Войдите в систему под учетной записью администратора. Получите архив работы SY-108. Извлеките каталог interproc в корневой каталог диска C:. Откройте решение. Убедитесь, что целевой платформой является x86 или Win32. В решении несколько проектов, по одному на каждую задачу.

Все проекты, за исключением clipboard, являются оконными, они построены примерно одинаково, и используют один общий модуль ccreate.h.

В этом модуле определены:

- все необходимые для работ переменные, за исключением разве что итераторов циклов; некоторые переменные определены в заголовочных модулях соответствующих проектов;
- дескриптор приложения hInst; дескрипторы основных окон hWnd1 и hWnd2, так как в обмене обычно участвуют два приложения; дескрипторы полей выводимой и принимаемой информации hWndOut и hWndIn; а также дескриптор шрифта приложения tahoma;
- названия окон, массив name; он инициализируется каждым приложением самостоятельно в функции WinMain;
- переменная myname, номер типа окна текущего приложения; она индексирует массив name для вывода названия окна (в его заголовке);
- функция SetName, устанавливающая название приложения;
- три буфера для разных операций; размер буфера задается константой MAX_TEXT, которая определяется в заголовочном модуле проекта;
- функция clearbuf для очистки конкретного буфера;
- функция ShowMessage для вывода сообщений;
- функция ShowError для вывода ошибки по ее номеру;
- разные константы.

При выполнении работ следует ориентироваться на примеры кода, приведенные в [1] и на сайте docs.microsoft.com.

8.2. Передача текста через буфер обмена

Сделайте стартовым проект clipboard. Для это щелкните на него правой кнопкой мыши в Project Explorer, и в контекстном меню выберите «Set as StartUp Project». Откройте модуль clipboard.cpp. В начале этого модуля укажите сведения об организации, о себе, о проекте, дату начала работы. В этом же модуле определены четыре функции, которые требуется описать.

8.2.1. Функция SetText

Цель функции — записать текст в буфер обмена.

Порядок действий функции следующий.

1. Присвоить переменой result значение 0.
2. Открыть буфер обмена при помощи функции OpenClipboard с параметром NULL. Если неудачно, то return.
3. Очистить буфер обмена при помощи функции EmptyClipboard. Если неудачно, перейти на метку closec.
4. Выделить 13 байт памяти при помощи функции GlobalAlloc. Если неудачно, перейти на метку closec.
5. Заблокировать память hGlobal в переменную lpData. Если неудачно, перейти на метку closec.
6. Скопировать в разделяемую память lpData строку "1234567890\r\n" при помощи функции lstrcpy. В ней 12 символов, плюс завершающий нулевой байт, всего 13 символов. Символы CR и LF часть формата CF_TEXT.
7. Разблокировать память lpData при помощи функции GlobalUnlock.
8. При помощи функции SetClipboardData записать данные в буфер обмена. Формат данных CF_TEXT, разделяемая память hGlobal.
9. Освободить память hGlobal при помощи функции GlobalFree.
10. Присвоить переменой result значение 1.
11. Метка closec.
12. Закрыть буфер обмена при помощи функции CloseClipboard.
13. Вернуть переменную result.

В функции main вызываем SetText. Запускаем программу, после ее завершения при помощи Ctrl+V вставляем текст куда-нибудь, да хоть в текст модуля. Если не вставляется, значит, что-то неверно.

8.2.2. Функция GetText

Цель функции — вывести текст из буфера обмена в консоль.

Порядок действий функции следующий.

1. Присвоить переменой result значение 0.
2. При помощи функции IsClipboardFormatAvailable убедиться в наличии в буфере обмена информации типа CF_TEXT, если неудачно, то return 0.
3. Открыть буфер обмена при помощи функции OpenClipboard с параметром NULL. Если неудачно, то return 0.

4. При помощи функции GetClipboardData принять указатель на данные в переменную hGlobal. Если hGlobal равно нулю (NULL), то перейти на метку closec.

5. Вывести текст в консоль при помощи функции printf, для чего нужно привести hGlobal к типу char*.

6. Освободить память hGlobal при помощи функции GlobalFree.

7. Присвоить переменой result значение 1.

8. Метка closec.

9. Закрыть буфер обмена при помощи функции CloseClipboard.

10. Вернуть переменную result.

В функции main вызываем GetText. Выделяем текст модуля при помощи Ctrl+A, Ctrl+C, запускаем программу, наблюдаем вывод в консоль.

8.2.3. Функции SetStruct и GetStruct

Цель этих функций — передать через буфер обмена пользовательскую структуру данных mydata. Код этих функций отличается тем, что:

1) в функции SetStruct нужно задать какое-то значение полю x.

2) в функции SetStruct требуется зарегистрировать пользовательский формат данных при помощи функции RegisterClipboardFormat. Результат функции запоминается в переменной myformat.

3) в функции GetStruct используется переменная myformat для проверки наличия в буфере обмена информации данного типа.

4) в функции GetStruct в консоль выводится значение поля x.

В функции main нужно вызвать функции SetStruct, затем GetStruct. После запуска программы в консоль должно выводится заданное значение.

8.3. Передача картинки через буфер обмена

Другим распространенным стандартным форматом буфера обмена является формат CF_BITMAP, растровая картинка (bmp). Это проще, чем передача текста.

Сделайте стартовым проект clipwapp. Откройте модуль clipwapp.cpp.

В начале этого модуля укажите сведения об организации, о себе, о проекте, дату начала работы. В этом же модуле определены две функции, которые требуется описать при выполнении работы.

Функция CopyToClipboard копирует картинку в буфер обмена.

Порядок действий функции следующий.

1. Если не удалось открыть буфер обмена с параметром hWnd1, то вывести в окно hWndErr текст "!OpenClipboard", завершить.

2. Если не удалось очистить буфер обмена, то вывести в окно hWndErr текст "!EmptyClipboard", завершить.

3. Если переменная hBitmap равна нулю, то вывести в окно hWndErr текст "!hBitmap", завершить.

4. При помощи функции SetClipboardData записать в буфер обмена переменную hBitmap, указав формат CF_BITMAP.

5. Закрыть буфер обмена.

Откройте Microsoft Word.

Запустите программу, нажмите кнопку BMP1, при этом в окно программы выведется первая двух из картинок, имеющихся в ресурсах приложения. Нажмите кнопку Copy to, перейдите в Word, нажмите Ctrl+V. Если картинка появилась в документе, считаем, что первая функция работает.

Функция PasteFromClipboard извлекает из буфера обмена картинку.

Порядок ее действий следующий.

1. Если в буфере обмена нет формата CF_BITMAP, то вывести в окно hWndErr текст "!IsClipboardFormatAvailable", завершить.

2. Если не удалось открыть буфер обмена с параметром hWnd1, то вывести в окно hWndErr текст "!OpenClipboard", завершить.

3. В переменную hBitmap принять картинку из буфера обмена, указав формат CF_BITMAP. Если переменная оказалась равной нулю, то вывести в окно hWndErr текст "!hBitamp", завершить.

4. Закрыть буфер обмена.

5. Вывести картинку при помощи функции DisplayBitmap, передавая ей переменную hBitamp.

Перейдем в Word. В нем должна быть картинка. Скопируем ее, запустим программу и нажмем кнопку Paste from. Картинка должна появиться в окне приложения. Если это произошло, считаем, что функция работает.

Далее открываем FAR, переходим в каталог c:\interproc\Debug, запускаем две копии программы clipwapp, убеждаемся, что картинки передаются из одного приложения в другое. Закройте Word, не сохраняя документ.

8.4. Передача текста механизмом Data Copy

Сделайте стартовым проект copydata. Откройте модуль copydata.cpp. В начале этого модуля укажите сведения об организации, о себе, о проекте, дату начала работы.

Перейдем в модуль copydata.h.

В модуле copydata.h определена структура данных copydata, которая будет пересылаться. Она состоит только из массива символов, поэтому указатель на эту структуру указывает на строку символов.

Перейдем в модуль copydata.cpp.

Функция SendData посылает данные.

Порядок действий функции следующий.

1. Очистить структуру data функцией cleardata.

2. При помощи функции GetWindowText получить текст из окна, на которое указывает дескриптор hWndOut, в буфер структуры data.buffer.

Функция возвращает число скопированных байт без учета завершающего нуля. Возвращаемое значение нужно принять в переменную dwSize.

3. Если dwSize равно нулю, то при помощи функции ShowMessage вывести сообщение «Nothing to send.» и завершить функцию SendData.

4. Установить поля переменной cd типа COPYDATASTRUCT.

В поле dwData установить в значение CDCMD, в поле cbData установить в значение dwSize, в поле lpData установить значение адреса переменной data, приведя его к типу PVOID.

5. Далее нужно определить, есть ли еще одно окно этого приложения.

Первое запущенное окно этого приложения будет иметь заголовок First CopyData, последующие окна — Second CopyData.

Функция FindWindow находит окно и возвращает его дескриптор. Если второго окна нет, то посылаем сообщение WM_COPYDATA сами себе, иначе посылаем его второму окну:

```
hWnd2 = FindWindow("COPYDATA", name[(myname + 1) % 2]);
if (hWnd2) {
    SendMessage(hWnd2, WM_COPYDATA, (WPARAM)hWnd1, (LPARAM)&cd);
} else {
    SendMessage(hWnd1, WM_COPYDATA, (WPARAM)hWnd1, (LPARAM)&cd);
}
```

Перейдем к функции GetData. Эта функция вызывается, когда приложению приходит сообщение WM_COPYDATA. При этом ей передается указатель pcd на структуру COPYDATASTRUCT.

Функция выполняет следующие действия.

1. Копирует текст из структуры данных COPYDATASTRUCT, на которую указывает параметр pcd, в буфер bufc при помощи функции lpstrcpy. При этом указатель lpData нужно сначала привести к типу copydata*, а затем все вместе привести к типу LPCSTR.

2. Записать в элемент pcd->cbData буфера bufc нулевой байт.

3. При помощи функции SetWindowText вывести буфер bufc в поле, на которое указывает дескриптор hWndIn.

Заметим, что необходимые действия можно выполнить одной строкой кода, минуя буфер bufc и не записывая нулевой байт.

Запускаем программу, нажимаем кнопку Send, убеждаемся, что текст из верхнего поля окна программы копируется в нижнее поле. Если это так, открываем FAR, переходим в каталог c:\interproc\Debug, запускаем две копии программы copydata, убеждаемся, что они передают текст друг другу.

8.5. Исследование механизма DDE

Сделайте стартовым проект ddeapp. Откройте модуль ddeapp.h.

Рассмотрим специфичные для работы структуры данных.

В начале модуля определения констант DEFAPP, DEFTOPIC, DEFITEM и DEFSEND, которые задают названия приложения, темы, элемента данных и посылаемого текста, которые появляются в полях окна приложения при его старте.

Далее определяются атомы. Атомы atomA и atomB используются при приеме сообщений от других приложений. Атомы atomAdvise и atomPoke используются при выполнении соответствующих операций. Остальные атомы используются дли названий приложения, темы и элемента данных.

Далее описываются структуры данных, указатели на эти структуры, флаги, которые приходят в сообщениях DDE, а также несколько повсеместно используемых переменных.

Далее описываются три состояния приложения и их названия.

Функции onStart и onStop вызываются при старте и завершении работы приложения. Функция onStop удаляет атомы.

Далее определяются константы, задающие заголовки окна и тип приложения: неизвестно, клиент, сервер, клиент и сервер одновременно.

Далее описываются структуры для организации приложения.

Перейдем в модуль ddeapp.cpp.

Ввиду сложности работы в целом, код приложения написан практически полностью, за исключением функций сервера. Иначе говоря, это готовое приложение, которое может взять на себя роль клиента DDE. Наша задача заключается в исследовании протокола, изучая последовательность действий на примере связи с тем сервером, который будет обнаружен.

Функции, которые вызываются приходящими в приложение сообщениями DDE, начинаются с On. Функции, которые вызываются нажатиями кнопок, начинаются либо с Send, если действие подразумевает посылку сообщения DDE, либо глаголом, описывающим действие. Некоторые функции вызываются другими функциями, они вспомогательные.

Запустим программу ddeapp, рассмотрим, из чего состоит интерфейс.

Окно делится на три части, расположенных рядом.

В первой части элементы, необходимые для установления связи, завершения связи и завершения работы. Начинается она с названия приложения, с которым устанавливается связь, и темы разговора. Далее идет кнопка Initiate, которая посылает соответствующее сообщение. Серверы, получив наше сообщение, посылают нам свои подтверждения, которые попадают в список Received Ack's (Ack — сокращение слова acknowledge, подтверждение). Выбирая в этом списке элемент, и нажимая кнопку Accept или Decline, мы либо подтверждаем связь, либо отвергаем ее (посылая сообщение о ее завершении). Ниже для справки выводятся дескриптор нашего окна (это необходимо, чтобы понимать, какие сообщение от нас же и поступили), дескриптор окна партнера и текущее состояние программы.

Кнопка Terminate завершает разговор, посылая партнеру сообщение WM_DDE_TERMINATE. Кнопка Close закрывает программу, при этом при необходимости партнеру посылается сообщение о завершении.

В средней части окна элементы, необходимые для ведения разговора.

Часть начинается с поля для названия элемента данных. Затем следует поле, в которое записывается посылаемая информация. Ниже расположены

кнопки действий: Request (запрос данных), Poke (запись данных в сервер), Advise warm (теплая) и Advise hot (горячая) постоянная связь, Unadvise (отмена постоянной связи) и Execute (посылка команд серверу).

Ниже находится поле Received data, в которое записывается приходящие данные, ниже поле Received Ack, Request or data fields, в котором отображаются подтверждения, запросы, флаги, пришедшие с данными.

Справа находятся списки входящих сообщений INITIATE и TERMINATE. Сообщения INITIATE можно отвергать или принимать. В последнем случае мы становимся сервером (если удастся). Сообщения TERMINATE можно только удалять все или по одному.

Все выполняемые действия должны быть запротоколированы. Всего возможны 10 команд, которым сопоставлены один или два параметра, как показано в следующем списке:

Кнопка	→Параметр 1	→Параметр 2
Initiate	→Application	→Topic
Accept	→Выбираемый элемент списка Received Ack's	
Terminate	→hWnd2 записываем в отчет	
Request	→Item	
Poke	→Item	→To send
Advise warm	→Item	
Advise hot	→Item	
Unadvise	→Item	
Execute	→To send	
Excel	→ячейка	→вводимое значение

Последняя команда вводит значение в ячейку Excel, это не кнопка, а действие, которое нужно выполнить в Excel.

Каждый акт действий пронумерован. В отчет прежде всего записывается номер акта, затем команда, затем наблюдаемые явления. Выполняемая команда указывается примерно так:

Poke→Selection→[dde.xls]dde!R9C1

То есть нужно нажать кнопку Poke, перед этим в поле Item ввести слово Selection, в поле To send строку [dde.xls]dde!R9C1. Команда записывается в отчет как есть, затем вводятся данные и нажимается кнопка.

В результате что-то произойдет в программе и, возможно, в сервере. Все наблюдаемые явления кратко записываются в отчет. Подтверждения и запросы в левом и правых списках записываются в отчет так, как они выглядят в списках, с указанием списка. В поле Received data приходит текст в формате CF_TEXT, элементы данных которого разделены табуляторами. Эти табуляторы нужно подсчитать, чтобы понять, сколько элементов пришло. Элементов на один больше числа табуляторов. В конце всегда записывается пара символов CR и LF, они часть формата CF_TEXT. При необходимости текст можно скопировать, вставить в файл c:\interproc\data.txt, затем изучить его при помощи FAR, используя клавиши F3 и F4.

В поле Received Ack, Request or data fields приходят подтверждения сервера, запросы клиента, а также информация, полученная с данными. В отчет записывается все, что есть в поле. Обращаем внимание на флаги, они заставляют программу автоматически очищать память и посылать подтверждения. Обращаем внимание на количество полученных байт, оно указывает, сколько информации пришло в поле Received data.

Если в поле To send требуется вставить табуляторы, то нужно остановить программу, записать необходимую строку в константу DEFSEND, после чего запустить программу и заново установить связь.

Закроем программу ddeapp.

8.5.1. Обнаружение клиентов и серверов

Откройте FAR. В каталоге c:\interproc откройте файл dde.xls. Если этого файла нет, откройте Microsoft Excel, создайте пустую книгу. В книге должен быть один лист с названием dde. В ячейку A1 введите слово Hello, в ячейку B1 введите слово World. Выделите ячейку A5. Сохраните книгу с именем dde.xls в каталог c:\interproc.

Порядок запуска приложений имеет значение!

Программа ddeapp запускается перед программой Excel. Когда Excel стартует, он рассылает сообщения DDE открытым окнам. Разместите окна так, чтобы одновременно было видно приложение, Excel и методичка.

1. Запишите в отчет «Поступили запросы Initiate при запуске ddeapp когда открыта Excel» и спишите их из правого верхнего списка. Отклоните эти запросы. Наблюдаемые явления запишите в отчет.

2. Initiate→пусто→пусто. Запишите в отчет полученные сообщения из левого и правого списков. В правом списке должен прийти запрос от нас самих, на это указывает дескриптор запроса. Отклоните его. Поочередно отклоните запросы левого списка, всегда выбирая последний элемент. Запишите в отчет все записи из правого нижнего списка. Очистите его.

Остановим программу ddeapp.

8.5.2. Разговор на системные темы Excel

Анализируем записи подтверждений левого списка. Нас интересует Excel и темы, которые он предлагает. Должна быть тема System. Описываем константы: DEFAPP "Excel", DEFTOPIC "System", DEFITEM "Topics".

Программа Excel должна быть открыта, запускаем программу ddeapp.

3. Initiate→Excel→System.
4. Accept→Excel→System. (hWnd2 записался, состояние Conversation).
5. Request→Topics.
6. Request→SysItems.
7. Request→Formats.
8. Request→Selection.

Заметим, что в поле данных пришло что-то вроде [dde.xls]dde!R5C1.

Это хорошая подсказка, она показывает, как задаются ячейки листа Excel, мы же этого не знали. Заметим, что R5C1 — это пятая строка, первая колонка, та ячейка, которая выделена сейчас.

9. Execute→[select("R9C1:R9C3")].

Аккуратно переключимся в Excel, чтобы не сбить выделение, и убедимся, что выделено три ячейки в девятой строке.

10. Request→Protocols.

11. Request→EditEnvItems.

12. Request→Help.

Остановим программу ddeapp.

8.5.3. Разговор с листом Excel

Меняем тему в константе DEFTOPIC на "[dde.xls]dde". Она должна быть записана в отчете ранее. Если нет, то выберем похожую. Меняем название данных в константе DEFITEM на "R1C1:R1C2". Это диапазон ячеек A1:B1, в которых записаны слова Hello и World. Запускаем программу ddeapp.

13. Initiate→Excel→[dde.xls]dde.

14. Accept→Excel→[dde.xls]dde.

15. Request→R1C1:R1C2.

16. Advise warm→R2C1:R2C2. Обращаем внимание на флаг Ack. Если он равен 1, постоянная связь подтверждается, иначе нет.

17. Excel→R2C1→1.

18. Request→R2C1:R2C2. Убеждаемся, что получено два значения.

19. Unadvise→R2C1:R2C2.

20. Excel→R2C1→0.

21. Request→R2C1:R2C2. Убеждаемся, что получено два значения.

22. Advise hot→R2C1:R2C2.

23. Excel→R2C2→2.

24. Unadvise→R2C1:R2C2.

25. Terminate.

Остановим приложение ddeapp.

Изменим значение константы DEFITEM на "R2C1:R2C3", значение константы DEFSEND на "2\t3\t=A2*B2". То есть в ячейки A2 и B2 записываем числа 2 и 3, а в ячейку C2 — формулу их перемножения.

Запускаем программу ddeapp, инициируем и устанавливаем разговор.

26. Poke→R2C1:R2C3→2\t3\t=A2*B2.

27. Request→R2C1:R2C3.

28. Excel→A3→undo.

29. Execute→[undo].

30. Execute→[open("dde2.xls")].

31. Execute→[save].

32. Execute→[close].

33. Initiate→Excel→пусто.
34. Accept→Excel→System.
35. Execute→[new].
36. Execute→[quit].

Остановим программу ddeapp.

Вы можете выполнять любые другие действия, чтобы лучше понять протокол. Эти действия записывать не требуется, если только для себя.

Кроме того, можно запустить две копии программы ddeapp и вести разговор между ними, но сервер может только отвечать на запрос. Разговор можно вести и с одной копией, если подтвердить собственный запрос.

Application, Topic и Item принимаем при этом равными строке Self.

8.6. Передача текста через почтовый ящик

Сделайте стартовым проект mailslot. Откройте модуль mailslot.cpp. В начале этого модуля укажите сведения об организации, о себе, о проекте, дату начала работы.

В этом же модуле определены функции, которые требуется описать при выполнении работы. Мы должны в первой копии программы создать почтовый ящик и ждать подключения. Вторая копия программы подключается к ящику и записывает сообщение. Первая копия считывает его.

Первая копия, создающая ящик, — это сервер. Вторая копия, соответственно, клиент. Клиент может только записывать. Сервер может читать сообщения и просматривать ящик. Он может записывать, если подключится как клиент.

8.6.1. Определение клиента и сервера

Когда мы нажимаем кнопку Open, управление приходит в функцию OpenMailslot. Далее мы должны либо создать ящик и стать сервером, либо подключиться к нему и стать клиентом. Сначала пытаемся подключиться, чтобы понять, есть ящик или нет. В программе есть два дескриптора для почтового ящика. Дескриптор hServer предназначен для сервера. Дескриптор hClient предназначен для клиента, однако, поскольку клиент и сервер являются разными программами, сервер может использовать hClient для подключения и записи в ящик. Клиент не может использовать hServer.

Функция OpenMailslot. Сначала убеждаемся, что ни один из дескрипторов не задан. Если какой-то дескриптор не равен нулю, возвращаем единицу (уже подключен).

Далее пытаемся подключиться, вызывая функцию OpenSlot с параметром TestSlot (имя ящика). Возвращаемое значение принимаем в hClient, поскольку подключаемся. Если hClient получает значение, то устанавливаем своё имя при помощи функции SetName с параметром MSClient, и возвращаем единицу.

Если подключиться не удалось и hClient остался равным нулю, то пытаемся создать ящик, вызывая функцию CreateSlot с параметром TestSlot. Возвращаемое значение принимаем в hServer. Если создать удалось и дескриптор hServer получил значение, то устанавливаем имя при помощи функции SetName с параметром MSServer, и возвращаем единицу. В противном случае вызываем функцию ShowError, передавая ей название функции и значение, возвращаемое GetLastError, и возвращаем ноль.

8.6.2. Создание почтового ящика

Функция CreateSlot описывает дескриптор h типа HANDLE и получает в него значение, которое возвращает функция CreateMailslot. Первый параметр функции CreateMailslot равен параметру функции CreateSlot, другие параметры равны нулю. Если значение h равно INVALID_HANDLE_VALUE, возвращаем NULL, иначе возвращаем h.

8.6.3. Подключение к почтовому ящику

Функция OpenSlot описывает дескриптор h типа HANDLE и получает в него значение, которое возвращает функция CreateFile. Первый параметр этой функции равен параметру функции OpenSlot. Для доступа используем константу GENERIC_WRITE, режим разделения файла задаем константами FILE_SHARE_READ и FILE_SHARE_WRITE, атрибуты безопасности 0, режим открытия OPEN_EXISTING, атрибуты FILE_ATTRIBUTE_NORMAL, последний параметр ноль. Если значение h равно INVALID_HANDLE_VALUE, возвращаем NULL, иначе возвращаем h.

8.6.4. Получение сведений о почтовом ящике

Функция CheckData проверяет наличие сообщений при помощи функции GetMailslotInfo. Параметры hServer, 0, dwNextSize по ссылке, dwCount по ссылке, и 0. Если функция завершается успешно, то при помощи функции sprintf формируем сообщение в буфере, например, bufc, используя строку формата "Count: %d\r\nNextSize: %d\r\n", выводимые переменные dwCount и dwNextSize. Далее выводим текст из буфера в поле, на которое указывает дескриптор hWndIn, функция SetWindowText.

8.6.5. Запись сообщения в почтовый ящик

Функция PutData получает текст из поля, на которое указывает дескриптор hWndOut, в буфер bufc, при помощи функции GetWindowText, максимальный размер текста равен константе MAX_TEXT. Возвращаемое значение принимаем в переменную dwSize. Если dwSize равно нулю, выводим сообщение «Nothing to send» при помощи функции ShowMessage, выходим из функции. Если текст есть, то при помощи функции WriteFile записываем сообщение, параметры hClient, bufc, dwSize + 1, dwWritten по ссылке, 0.

Если функция WriteFile завершается успешно, выходим из функции. В противном случае формируем сообщение при помощи ShowError.

8.6.6. Чтение сообщения из почтового ящика

Функция GetData сначала проверяет наличие сообщений в ящике так же, как это делает функция CheckData. Если dwCount равно нулю, выводим сообщение «Nothing to read», выходим из функции.

Если сообщения есть, читаем первое при помощи функции ReadFile, параметры hServer, bufc, dwNextSize, dwRead по ссылке, 0.

Если успешно, выводим буфер bufc в поле с дескриптором hWndIn, функция SetWindowText. Если неуспешно, формируем сообщение об ошибке, функция ShowError.

8.6.7. Передача сообщений

Компилируем проект. Открываем FAR, каталог c:\interproc\Debug. Запускаем три копии программы, создаем сервер, создаем клиентов (заголовки окон показывают, кто есть кто), записываем и читаем сообщения.

8.7. Передача текста через канал

Сделайте стартовым проект pipes. Откройте модуль pipes.cpp. В начале этого модуля укажите сведения об организации, о себе, о проекте, дату начала работы. В этом же модуле определены три функции, которые требуется описать. Мы должны в первой копии программы создать канал и ждать подключения. Вторая копия программы подключается к каналу, записывает сообщение в канал и читает ответное сообщение из канала. Первая копия получает сообщение, в ответ отправляет свое.

Первая копия — это сервер, вторая копия — клиент. После обмена сообщениями канал закрывается. Сервер использует дескриптор hPipe, а клиент использует дескриптор hFile.

Кнопка Wait вызывает функцию CreatePipe, которая создает канал и ждет подключения (функция сервера), кнопка Write вызывает функцию WritePipe, которая подключается к каналу, записывает в него, затем читает из канала (функция клиента).

8.7.1. Функция сервера

Функция CreatePipe сначала убеждается, что ни один из дескрипторов не открыт. Если один из дескрипторов не равен нулю, функция завершается. Затем вызывается функция CreateNamedPipe, результат которой принимается в дескриптор hPipe. Параметры TestPipe (имя канала), доступ канала PIPE_ACCESS_DUPLEX, тип канала задаем константами PIPE_TYPE_MESSAGE и PIPE_READMODE_MESSAGE, число копий 1, размеры буферов оба 0, таймаут PIPE_WAIT, атрибуты безопасности 0.

Если значение hPipe равно INVALID_HANDLE_VALUE, то выводи сообщение об ошибке при помощи функции ShowError, принимаем значение hPipe равным нулю, выходим из функции. Если hPipe не равно нулю, то устанавливаем имя при помощи функции SetName с параметром PipeServer, и запускаем поток ServerThread при помощи функции CreateThread. Все параметры этой функции могут быть нулевыми, за исключением третьего.

Поток совершенно необходим. Если ожидать подключения клиента в первичном потоке, он зависнет, и приложение перестанет отвечать на сообщения, его трудно будет закрыть.

Далее описываем функцию потока ServerThread. Она ждет подключения, вызывая функцию ConnectNamedPipe с параметрами hPipe и 0. Если функция завершается неуспешно, выводим сообщение об ошибке, завершаем поток со значением 0. Функция ConnectNamedPipe не возвратится до тех пор, пока клиент не подключится. Как только это произойдет, управление придет в поток к следующей строке кода, в которой мы описываем чтение канала при помощи функции ReadFile, параметры hPipe, bufc, MAX_TEXT, dwBytes по ссылке, 0.

Если ReadFile завершается неуспешно, выводим сообщение об ошибке, завершаем поток со значением 0. Если успешно, то выводим текст из буфера bufc в поле с дескриптором hWndIn, функция SetWindowText.

Далее в потоке получаем текст поля с дескриптором hWndOut в буфер bufa, функция GetWindowText, максимальный размер буфера MAX_TEXT. Возвращаемое функцией GetWindowText значение принимаем в переменную dwSize. Если dwSize равно нулю, это плохо, клиент зависнет. Поэтому в этом случае при помощи функции strcpy запиваем в буфер bufa строку "Server has no data.", после чего вычисляем ее размер в переменную dwSize при помощи функции strlen.

Далее записываем в канал строку из буфера bufa при помощи функции WriteFile с параметрами hPipe, bufa, dwSize, dwBytes по ссылке, 0).

В конце закрываем дескриптор hPipe, принимаем его равным нулю и возвращаем 0.

8.7.2. Функция клиента

Функция WritePipe сначала убеждается, что ни один из дескрипторов не открыт. Если хотя бы один не равен нулю, функция завершается.

Далее создаем файл при помощи функции CreateFile с параметрами TestPipe, GENERIC_READ и GENERIC_WRITE, 0, 0, OPEN_EXISTING, 0, 0. Возвращаемое значение принимаем в переменную hFile. Если оно равно значению INVALID_HANDLE_VALUE, то выводим сообщение об ошибке, выходим из функции. Если hFile не равно нулю, то устанавливаем имя при помощи функции SetName с параметром PipeClient.

Далее получаем текст окна с дескриптором hWndOut в буфер bufa, функция GetWindowText, возвращаемое значение принимаем в dwSize.

Если dwSize равно нулю, то при помощи функции strcpy запиваем в буфер bufa строку "Client has no data.", после чего вычисляем ее размер в переменную dwSize при помощи функции strlen.

Далее записываем в канал строку из буфера bufa при помощи функции WriteFile с параметрами hPipe, bufa, dwSize, dwBytes по ссылке, 0). В случае неудачи выводим сообщение об ошибке.

Далее читаем канал при помощи функции ReadFile с параметрами hFile, bufb, MAX_TEXT, dwBytes по ссылке, 0. В случае неудачи выводим сообщение об ошибке. В случае удачи текст из буфера bufb выводим в поле с дескриптором hWndIn. В конце закрываем дескриптор hFile и принимаем его равным нулю.

8.7.3. Передача сообщений

Запускаем две копии программы pipes, в одной копии нажимаем кнопку Wait, в другой — кнопку Write. Наблюдаемые явления записываем в отчет.

8.8. Контрольные вопросы и упражнения

1. Что такое буфер обмена, какие данные он передает?
2. Поясните порядок работы с буфером обмена.
3. Поясните порядок работы с разделяемой памятью.
4. Поясните, в чем заключается механизм Data Copy.
5. Поясните, что такое атом, для чего он нужен.
6. Для чего нужны название приложения, темы и элемента данных?
7. Что такое системная тема, какие системные темы бывают.
8. Какое приложение называют сервером DDE, клиентом DDE?
9. Сколько разговоров может вести сервер, клиент?
10. Какой объект принимает и посылает сообщения DDE?
11. Назовите все сообщения DDE. Поясните их цель и направление.
12. Поясните протокол установления связи DDE, кто и что посылает.
13. Как клиент и сервер узнают дескрипторы окон партнеров?
14. Опишите флаги структуры DDEDATA.
15. Что называется постоянной связью (permanent data link)?
16. Почему в ваших записях дескриптор Excel все время разный?
17. Что такое почтовый ящик? Как задается его имя?
18. Кто сервер почтового ящика? Какие действия ему разрешены?
19. Кто клиент почтового ящика? Какие действия ему разрешены?
20. Что такое канал? Как задается его имя?
21. Поясните режимы работы канала.
22. Поясните действия сервера канала.
23. Поясните действия клиента канала.

9. Работа SY-109. Сервисы NT

Цели:
- изучение структуры сервиса;
- управление сервисами.

Задачи:
- построение основной функции сервиса;
- создание, запуск, остановка и удаление сервиса;
- описание клиентского приложения;
- создание ресурсов с сообщениями сервиса.

Опорные документы:
[1, «Сервисы NT»]

9.1. Шаблон проекта

Войдите в систему под учетной записью администратора. Получите архив работы SY-109. Извлеките каталог service в корневой каталог диска C:, и только в корневой каталог диска C:. Откройте решение. Убедитесь, что целевой платформой является x86 или Win32. В решении содержится проект сервиса и проект клиентского приложения. Стартовым проектом должно быть клиентское.

9.1.1. Проект сервиса

Мы разрабатываем сервис с названием Reverser. Он принимает строку через именованный канал, инвертирует ее, и отсылает обратно.

Откроем модуль pipename.h. Этот модуль разделяемый, он включен также в проект клиента, и содержит имя именованного канала.

В модуле service.h описываются необходимые константы.

Откроем модуль service.cpp. Он содержит структуру сервиса, то есть точку входа в консольное приложение (main), основную функцию сервиса с именем service_main, обратно-вызываемую функцию управления сервисом с именем service_ctrl, а также вспомогательные функции:

- SendStatus для отправки текущего состояния сервиса SCM;
- LogEvent для записи сообщений в журнал «Приложения»;
- InstallService для установки сервиса в базу данных SCM;
- RemoveService для удаления сервиса из базы данных SCM;
- ErrorText для формирования сообщения об ошибке.

Этот шаблон сервиса может устанавливать сервис и удалять его, если запускать программу с ключом -i для создания, с ключом -r для удаления.

Основная часть сервиса вынесена в функцию ReverserService.

Откроете модуль reverser.cpp. В начале модуля укажите сведения об организации, о себе, о проекте, дату начала работы.

В модуле объявлены необходимые структуры данных.

9.2. Код сервиса

Отлаживать сервис в режиме пошагового исполнения возможно, но в этом шаблоне эта возможность исключена, поэтому будем контролировать выполнение, посылая сообщения при помощи функции MessageBox. Сначала напишем весь код сервиса и клиента целиком, затем запустим сервис, и посмотрим, какие сообщения получим.

1. Сначала сервис посылает SCM состояние начала инициализации и входа в режим зависания (pending):

```
// посылаем SCM состояние SERVICE_START_PENDING
if (!SendStatus(SERVICE_START_PENDING, WHINT)) goto cleanup;
```

Второй параметр указывает, что следующее уведомление поступит не позднее, чем через WHINT миллисекунд.

2. Создаем именованный канал функцией CreateNamedPipe, параметры SZ_PIPE_NAME, флаги FILE_FLAG_OVERLAPPED и PIPE_ACCESS_DUPLEX, флаги PIPE_TYPE_MESSAGE и PIPE_READMODE_MESSAGE и PIPE_WAIT, число копий 1, остальные параметры 0.

3. Проверяем результат, в случае ошибки выводим сообщение:

```
if (hPipe == INVALID_HANDLE_VALUE) {
    MessageBox(0, "Failed to CreateNamedPipe.", "Reverser",
        MB_OK | MB_SERVICE_NOTIFICATION);
    goto cleanup;
}
```

4. Посылаем SCM состояние SERVICE_CONTINUE_PENDING.

5. Создаем два события с ручным сбросом, дескрипторы записываем в массив hEvents:

```
// событие для остановки сервиса
hEvents[0] = CreateEvent(NULL, TRUE, FALSE, NULL);
if (!hEvents[0]) goto cleanup;
// событие завершения асинхронного ввода-вывода
hEvents[1] = CreateEvent(NULL, TRUE, FALSE, NULL);
if (!hEvents[1]) goto cleanup;
```

6. Посылаем SCM состояние SERVICE_RUNNING.

7. Описываем бесконечный цикл.

8. В цикле инициализируем структуру OVERLAPPED:

```
// инициализируем структуру OVERLAPPED
memset(&ov, 0, sizeof(OVERLAPPED));
ov.hEvent = hEvents[1];
ResetEvent(hEvents[1]);
```

9. Вызываем функцию ConnectNamedPipe, ожидая подключения:

```
// ждем подключения клиента
result = ConnectNamedPipe(hPipe, &ov);
```

10. Результат в result истинный, если клиент успел подключился до завершения ConnectNamedPipe.

Результат ложный, если операция зависла (pending), то есть вошла в асинхронный режим, или же функция завершилась неудачно. Чтобы понять, что произошло, проверяем код последней ошибки:

```
// результат подключения
dwError = GetLastError();
if (result == 0 && dwError == ERROR_IO_PENDING) {
    dwWait = WaitForMultipleObjects(2, hEvents, FALSE, INFINITE);
    if (dwWait != WAIT_OBJECT_0 + 1) {
        // сигнал остановки или ошибка overlapped ввода-вывода
        break;
    }
} else if (result == 0 && dwError != ERROR_IO_PENDING) {
    MessageBox(0, "Failed to ConnectNamedPipe.", "Reverser",
        MB_OK | MB_SERVICE_NOTIFICATION);
    break;
}
```

Если ConnectNamedPipe вошла в асинхронный режим, то ожидание выхода из него выполняет функция WaitForMultipleObjects. Когда она завершается, dwWait содержит причину завершения:

- сработало событие завершения, это остановка сервиса;
- сработало событие OVERLAPPED, это то, что мы ждем;
- истекло время ожидания, но мы задали ждать вечно;
- функция завершилась неудачно (и вышла немедленно).

Во всех случаях, кроме второго, сервис выходит из цикла.

11. Снова инициализируем структуру OVERLAPPED.

12. Читаем канал, функция ReadFile:

```
// читаем сообщение клиента
result = ReadFile(hPipe, inbuf, MAXBUF, &dwBytesIn, &ov);
```

13. Результат обрабатывается так же, как в предыдущем случае. При этом сообщение MessageBox "Failed to ReadFile.".

14. Обрабатываем сообщение:

```
// обрабатываем сообщение
p = _strrev(inbuf);
```

15. Снова инициализируем структуру OVERLAPPED.

16. Записываем в канал, функция WriteFile:

```
// посылаем ответное сообщение
result = WriteFile(hPipe, p, dwBytesIn, &dwBytesOut, &ov);
```

17. Проверяем результат так же, как и в предыдущих случаях. При этом сообщение MessageBox "Failed to WriteFile.".

18. Закрываем подключение для повторного использования:

```
// закрываем подключение
DisconnectNamedPipe(hPipe);
```

19. Конец бесконечного цикла.

9.3. Код клиента

Проект client, модуль client.cpp.

1. Подключаемся к каналу, функция CreateFile:

```
hFile = CreateFile(SZ_PIPE_NAME, GENERIC_READ | GENERIC_WRITE, 0,...
```

2. Проверяем результат hFile, выводим сообщение, если ошибка:

```
// результат подключения
if (hFile == INVALID_HANDLE_VALUE) {
    printf("Failed to CreateFile: %s.\n", ErrorText(errbuf, MAXERR));
    return 1;
}
```

3. Определяем размер dwSize буфера output, функция strlen.

4. Записываем в канал буфер output.

5. Проверяем результат записи так же, как в предыдущем случае.

6. Читаем канал в буфер intput.

7. Проверяем результат чтения так же, как в предыдущем случае.

8. Выводим оба буфера в консоль для сравнения.

9. Закрываем канал, функция CloseHandle.

Компилируем проект.

9.4. Управление сервисом

1. Открываем FAR от имени администратора, каталог c:\service\Debug.

2. Создаем сервис командой service -i.

3. Если сервис создан, открываем Панель управления, Администрирование, Службы, находим сервис Reverser, дважды щелкаем на него. Нажимаем кнопку Пуск, убеждаемся, что сервис запустился.

4. Запускаем программу клиента (прямо из FAR), убеждаемся, что клиент успешно произвел обмен строками с сервисом.

5. Удаляем сервис командой service -r. Сначала он будет остановлен.

Следующие команды и успешный результат записываем в отчет.

6. sc create Reverser binpath= c:\service\Debug\service.exe

7. sc start Reverser

8. sc stop Reverser

9. sc delete Reverser

9.5. Сообщения сервиса

Проект service, модуль reverser.cpp.

1. Перед первым действием функции ReverserService вписываем:

```
LogEvent(255, EVENTLOG_INFORMATION_TYPE);
MessageBox(0, "ReverserService.", "Reverser", MB_OK | MB_SERVICE_N
```

2. Открываем консоль Администрирование, Просмотр событий.

3. Очищаем журналы System и Application, не сохраняя.

4. Создаем и запускаем сервис.

5. В журнале System ищем Application Popup, результат MessageBox, в журнале Application сообщение сервиса, результат LogEvent.

6. Останавливаем и удаляем сервис.

7. Закрываем в Visual Studio решение service.

8. Каталог c:\service\service.

9. Компилируем файл сообщений командой mc reverser.mc. Если команда mc не найдена, ищем файл mc.exe в каталоге C:\Program Files (86) и копируем его в каталог c:\service\service.

10. Изучаем полученные файлы reverser.bin, reverser.h, reverser.rc.

11. Открываем в Visual Studio решение service.

12. В папку Resource Files проекта service включаем файл reverser.rc.

13. Заменяем строку в начале функции ReverserService:

```
LogEvent(MSG_SERVICE_ENTER, EVENTLOG_INFORMATION_TYPE);
```

14. Компилируем сервис.

15. Каталог c:\service\Debug.

16. Запускаем файл reverser.reg, чтобы внести информацию в реестр.

Если неудачно, открываем реестр, записываем информацию вручную.

В ключ HKLM\System\CurrentControlSet\Services\Eventlog\Application добавляем подключ Reverser, в нем создаем параметр EventMessageFile типа REG_SZ, записываем в него значение c:\service\Debug\service.exe.

17. Очищаем журналы System и Application.

18. Создаем и запускаем сервис.

19. Наблюдаем сообщение в журнале Application.

20. Останавливаем и удаляем сервис.

9.6. Контрольные вопросы и упражнения

1. Опишите структуру сервиса.
2. Поясните назначение функций main, service_main, service_ctrl.
3. Опишите состояния сервиса.
4. Опишите процесс остановки сервиса.
5. Как сервис передает сообщения?
6. Как описываются сообщения в файле сообщений?
7. Куда и как записывается информация об источнике сообщений?

10. Литература

1. Вл. Пономарев. Системное программирование. Учебно-методическое пособие. Озерск: ОТИ НИЯУ МИФИ, 2019. — 88 с.

2. Рихтер Дж. Windows для профессионалов: создание эффективных Win32-приложений с учетом специфики 64-разрядной версии Windows / Пер. с англ. — 4-е изд. — СПб.: Питер; Издательско-торговый дом «Русская Редакция», 2001. — 753 с.: ил.

3. MSDN, October, 2001.

Пономарев Владимир Вадимович
Практикум по системному программированию
Учебно-методическое пособие

Наиболее актуальную версию пособия см. revol.ponocom.ru

Отпечатано с готового оригинал-макета
Издательство ОТИ НИЯУ МИФИ, 2019
Тираж 11 экз.