Лабораторная работа № 1

Основы программирования с использованием Windows API

Системное программирование

Операционная система — единственная программа с прямым доступом к аппаратным ресурсам компьютера. Пользовательские программы могут получить к ним доступ только через посредничество ОС. С этой целью ОС предоставляет им набор средств, называемый интерфейсом программирования приложений (application programming interface, API). Основную часть API ОС составляют функции в системных библиотеках, например, `ntdll.dll` в Windows. Примеры функций API ОС: запись в файл, выделение области памяти, запуск процесса. Каждая ОС, например, Windows, GNU/Linux или OS X, имеет собственный API; API ОС семейства *nix основаны на стандарте POSIX, и потому частично совместимы.

Изложение выше описывает API ОС в первом приближении, достаточном для выполнения лабораторной работы. Подробности сообщаются в лекционном курсе.

Системным программированием, помимо разработки самой ОС, могут называть несколько областей программирования, основу которых составляет вызов функций API ОС:

1) разработку драйверов устройств;
2) написание служебных программ для низкоуровневых операций;
3) использование системных вызовов в прикладных программах;

Драйверы — это библиотеки, обеспечивающие взаимодействие ОС с конкретным оборудованием. Они действуют как часть ОС и получают прямой доступ к аппаратным ресурсам. Их задача — принять от ОС стандартную команду, выполнить её специфичным для устройства образом и выдать ОС результат в стандартном виде.

Примером приложения, выполняющего низкоуровневые операции, может являться диспетчер задач, программа для форматирования диска, компьютерный вирус. Прямого доступа к аппаратным ресурсам они не имеют, но обращаются к ОС за совершением нужных операций. Низкоуровневыми условно называются действия, связанные с техническими особенностями работы компьютера. Например, нахождение сектора диска с данными файла — низкоуровневое действие (программа «знает» о диске и о секторах на нем), а запись текста в файл — высокоуровневое (не важно, куда и как это делается).

Обращения к API ОС, строго говоря, используются в любой реальной программе, иначе ей невозможно было бы выделить память, напечатать текст и т. п. (ОЗУ и экран —

аппаратные ресурсы). Имеется же в виду явные обращения к API ОС. Обычно это нужно для нетипичных действий: например, в Delphi или C/C++ есть средства работы с файлами, но они обеспечивают лишь базовые возможности; чтобы выполнить запись в фоновом режиме, приходится использовать системный вызов с применением API ОС. Заметим, что стандартные языковые функции реализованы на основе API ОС: например, код `fopen()` и `std::fstream` в C++ и `OpenFile()` в Delphi вызывает одну и ту же функцию `OpenFile()` из Windows API.

Разработка драйверов весьма сложна. Отчасти программирование на этом уровне изучалось в курса «Технические средства автоматизации и управления», но без системных вызовов (в DOS они не использовались так широко, как в современных ОС). Чисто служебные программы объемны и специфичны, а принцип их написания не слишком отличается от пользовательских приложений, меняется только API. Эта и последующие лабораторные работы посвящены использованию API ОС в пользовательских программах, что является чрезвычайно употребительным на практике случаем.

Использование Windows API

Получение справки

Microsoft предоставляет официальный и полный справочник функций Windows API в составе библиотеки MSDN на английском языке. Перевод на русский известен неполнотой и ошибками, его использования следует избегать. Библиотека MSDN содержит не только описание Windows API, но и примеры использования, полезные в учебе и на практике.

Описания функций в MSDN даны на языке C, причем в специфичном виде: используются переименованные типы данных, например, `LPSTR` вместо **`char`**`*`. Урок «Reading C code in Win32 API» *(англ.)* дает сжатое объяснение, как читать их и переводить на Delphi. Подробнее использование Windows API в Delphi освещает статья «Основы работы с Win API в VCL-приложениях».

Подключение и выбор версии Windows API

Можно обращаться к Windows API из любых других языков программирования. В Delphi для этого требуется подключить модуль `Windows`, в C/C++ следует использовать заголовочный файл `<windows.h>`. Иногда требуются и иные модули или заголовочные файлы; в руководстве по каждой функции это указывается.

По мере развития ОС Windows в API добавлялись новые функции, недоступные в более старых версиях. Если в программе использовать возможности новых версий, она не сможет работать на старых системах. Макрос (константа) `WINVER` регулирует, какой версией Windows ограничен набор доступных программисту функций:

```
#define WINVER 0x0502
#include <windows.h>
```

Значение `0x0502` соответствует Windows XP SP2. Константы для других версий и прочие возможности по ограничению Windows API см. в справке. По умолчанию ограничения жесткие, поэтому рекомендуется устанавливать их, как предложено выше.

Типовые приемы и распространенные ошибки

Обычно функции, результатом работы которых является строка, принимают два параметра для этого: указатель буфер символов, который будет заполнен функцией, и размер этого буфера. Использованы подобные функции могут быть примерно так:

C/C++

```
char buffer[256];

GetUserName(
  buffer, sizeof(buffer));
```

Delphi

```
var
  Buffer: array [0..255] of Char;
begin
  GetUserName(@Buffer, SizeOf(Buffer));
```

Распространенной ошибкой является попытка использовать тип-указатель `LPSTR` (и подобные) как буфер без выделения памяти:

C/C++

```
LPSTR buffer;

GetUserName(
  buffer, sizeof(buffer));
```

Delphi

```
var
  Buffer: PAnsiChar;
begin
  GetUserName(@Buffer, SizeOf(Buffer));
```

Указатель `buffer` содержит неизвестный адрес, по которому `GetUserName()` попытается записать данные, что приведет к ошибке. Размер переменной `buffer` равен размеру адреса, обычно 4 или 8 байт, а не размеру буфера, как в правильном примере выше.

Нулевой указатель, `NULL` в C или **`nullptr`** в C++, в Delphi обозначается **`nil`**.

В Windows API широко применяются битовые флаги и их комбинации. Например, для комбинации `C` флагов `A` и `B` (с. 3):

C/C++	Delphi	Смысл
`c = A | B;`	`C := A or B;`	Комбинация `C` включает и флаг `A`, и флаг `B`.
`if (c & A)` `  …`	`if (C and A) <> 0 then` `  …`	Если комбинация `C` включает флаг `A`, то…

Обработка ошибок

Как правило, по возвращаемому функцией Windows API значению можно определить, завершился ли вызов успешно. Об этом сообщается в разделе «Return value» в описании каждой функции. Установить причину ошибки позволяет функция `GetLastError()`. Она возвращает один из стандартных кодов, смысл которых описан в справочных таблицах. Разумный первый шаг в решении проблем — распечатать и проверить возвращаемое значение функции и результат `GetLastError()`.

Аннотации параметров функций

При описании функций в библиотеке MSDN перед типами параметров используются аннотации `_In_`, `_Out_` и другие, например:

```
BOOL WINAPI GetComputerName(
    _Out_   LPTSTR  lpBuffer,
    _Inout_ LPDWORD lpnSize);
```

Они ничего не имеют смысла с точки зрения языка программирования, а предназначены для того, чтобы указать назначение параметров в описании функции:

`_In_`	Обязательный входной параметр, значение которого используется функцией. Если это указатель, значение по хранимому адресу будет считано (поэтому недопустима передача `NULL`), но не будет изменено.
`_Inout_`	Обязательный входной и выходной параметр, указатель. Значение по находящемуся в нем адресу будет считано функцией и изменено ею.
`_Out_`	Обязательный выходной параметр, указатель. Должен содержать адрес области памяти, куда функция запишет результат своей работы.
`_In_opt`, `_Inout_opt_`, `_Out_opt_`	Необязательные параметры в том смысле, что существует некое значение (обычно `NULL`), которое можно передать в качестве данного параметра, если он не используется. Например, для многих функций можно задать особые атрибуты безопасности, но обычно это не нужно.

В примере выше `lpBuffer` является обязательным выходным параметром, и нужно передать указатель на выделенную область памяти, а значение по адресу в `lpnSize` не только используется функцией, но и будет изменено в результате её работы.

Описатели объектов (object handles)

Устройство внутренних структур ОС весьма сложно. Прикладному программисту же подробности не нужны и не должны быть доступны, а требуется простой способ указать

ОС на конкретный её внутренний объект. С этой целью широко применяются описатели объектов (object handles), называемые также дескрипторами (descriptor). С точки зрения программирования это простые переменные, обычно типа `HANDLE` (в Windows), хотя используются и другие типы. Например, `FindFirstVolume()` возвращает `HANDLE` первого найденного тома диска, который затем можно передать `FindNextVolume()` для поиска следующего. Прочие действия над описателями изучаются на следующих ЛР.

Задание на лабораторную работу

1. Написать программу, которая при помощи функций Windows API определяет параметры системы и компьютера, а именно:
    1) версию операционной системы (функция `GetVersionEx()`);
    2) системный каталог (функция `GetSystemDirectory()`);
    3) название компьютера и псевдоним текущего пользователя (функции: `GetComputerName()`, `GetUserName()`);
    4) для каждого тома (функции: `FindFirstVolume()`, `FindNextVolume()`, `FindVolumeClose()`) вывести следующие характеристики:
        — служебное имя тома (получаемое при переборе);
        — первый путь в файловой системе (`GetVolumePathNameForVolumeName()`);
        — объем тома и количество свободного места, доступного текущему пользователю (функция `GetDiskFreeSpaceEx()`).
    5) список программ, запускаемых при старте системы, из реестра Windows (функции: `RegOpenKeyEx()`, `RegEnumValue()`).

    Указание 1. Из структуры (`U`)`LARGE_INTEGER` здесь и далее нужно поле `QuadPart`.

    Указание 2. Реестр Windows — древовидное хранилище настроек ОС и программ. Реестр состоит из разделов (keys), в которых есть набор значений (value) с именами (name), а также, возможно, дочерние ключи. Работать с реестром позволяет штатная программа `regedit`. Искомый список хранится в следующем разделе реестра: `HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run`.
2. Добавить в программу функциональность измерения производительности ЦП:
    1) замер рабочей частоты f ЦП функцией `QueryPerformanceFrequency()`;
    2) подсчет количества тактов Δt ЦП, которое занимает выполнение программой пункта 1), функцией `QueryPerformanceCounter()` и выдать ответ в мкс.

    Указание. $\Delta t\ [\text{мкс}] = 10^6\ \left[\frac{\text{мкс}}{\text{с}}\right] \cdot \frac{t_{\text{конца}}\ [\text{тактов}] - t_{\text{начала}}\ [\text{тактов}]}{f\left[\frac{\text{тактов}}{\text{с}}\right]}$, где f — частота ЦП.

Контрольные вопросы

1. Что такое интерфейс программирования приложений (API) операционной системы?
2. В каких случаях прикладные (пользовательские) программы обращаются к API ОС?
3. Где доступна официальная справка по Windows API и какие типовые сведения доступны в ней для каждой функции?
4. Как и почему нужно учитывать наличие разных версий Windows при программировании с использованием Windows API?
5. Как диагностировать ошибки, возникающие при вызовах функций Windows API?
6. Что в Windows API понимается под необязательными параметрами функций, как они используются при вызове? Привести пример из лабораторной работы.
7. Для чего предназначен тип (U)LARGE_INTEGER в Windows API, и как пользоваться им в собственных программах? Привести пример из лабораторной работы.
8. Что такое реестр Windows, для чего он предназначен и из каких элементов состоит?
9. Каким образом при программном открытии ключа реестра указывается желаемые права доступа (возможность чтения, записи и т. п.)?
10. Как функциями QueryPerformanceFrequency() и QueryPerformanceCounter() производить замеры времени выполнения участков программы?