Интерфейсы операционных систем

DOI: 10.25586/RNU.V9187.21.01.P.172 УДК 004.451.9

А.Г. Скуратовский, Д.В. Пустынников

ИССЛЕДОВАНИЕ АРХИТЕКТУРЫ ОПЕРАЦИОННОЙ СИСТЕМЫ WINDOWS

Описаны уровни модульной архитектуры ОС Windows и режимы их работы. Охарактеризовано взаимодействие программных интерфейсов ОС Windows. Рассмотрена модель работы и программирования GUI-приложений в ОС Windows, ее развитие, начиная с первой версии ОС Windows, а также библиотека стандартных GUI-компонентов. Исследуется архитектура современной операционной системы Windows – наиболее распространенной операционной системы в мире.
Ключевые слова: современные операционные системы, операционная система Windows, архитектура операционной системы, программный интерфейс операционной системы, графический пользовательский интерфейс.

A.G. Skuratovskij, D.V. Pustynnikov

RESEARCH OF ARCHITECTURE OF WINDOWS OPERATING SYSTEM

The architecture of the modern Windows operating system - the most widespread operating system in the world – is investigated. The levels of the modular architecture of the Windows OS and their modes of operation are described. The interaction of the program interfaces of the Windows OS is characterized. The model of operation and programming of GUI-applications in Windows OS, its development starting from the first version of Windows OS, as well as a library of standard GUI-components are considered.
Keywords: modern operating systems, Windows operating system, operating system architecture, operating system application programming interface, graphical user interface.

Вводные замечания

Операционная система (ОС) является системным программным обеспечением, которое (в порядке важности) [5]:

1) управляет работой аппаратного обеспечения (железа) компьютера [6];

2) предоставляет унифицированный программный интерфейс (Application Programming Interface, API) для функционирования прикладного программного обеспечения (ПО);

3) реализует многозадачность, то есть одновременное выполнение нескольких процессов;

4) предоставляет пользовательский интерфейс (User Interface, UI) для взаимодействия с компьютером конечных пользователей.

Операционная система является своего рода промежуточным ПО (MiddleWare), которое помогает решать стоящие перед пользователем задачи с использованием аппаратных мощностей и возможностей компьютера. Если провести аналогию, то аппаратное обеспечение является «телом» компьютера, а операционная система – его «душой».

В мире существует множество операционных систем различного назначения и сфер применения. ОС Windows занимает доминирующее положение среди операционных систем. Это кроссплатформенная многозадачная операционная система для персональных компьютеров и рабочих станций, последняя версия которой (Windows 10) работает на платформах ARM, IA-32 и x86_64. В настоящее время современные версии ОС Windows – версии Windows из линейки Windows NT.

Целью данной статьи является исследование архитектуры операционной системы Windows.

Архитектура ОС Windows

ОС Windows имеет модульную архитектуру (рис. 1), которая состоит из двух уровней – режима ядра и режима пользователя, изолированных друг от друга, то есть нельзя повлиять на работу этих систем иначе, чем через определенные интерфейсы [3].

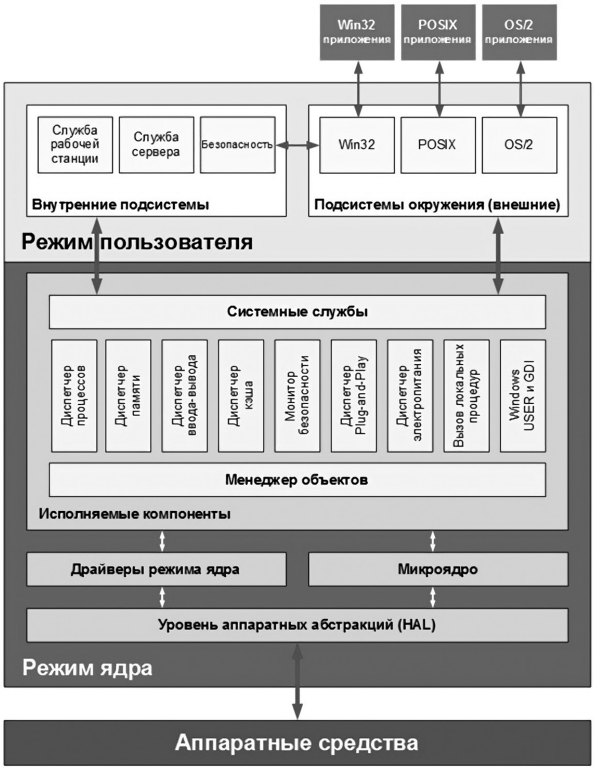

Рис. 1. Архитектура ОС Windows

Режим ядра является привилегированным уровнем, имеющим прямой и полный доступ к аппаратным ресурсам компьютера; при этом компоненты, работающие в режиме ядра, также могут быть разделены по уровням.

Самым нижним уровнем режима ядра, интерфейсом которого пользуются все располагающиеся выше компоненты уровня, является уровень аппаратных абстракций (Hardware Abstraction Layer, HAL). HAL сосредоточивает в себе весь платформозависимый код

и реализует интерфейс для взаимодействия с аппаратным обеспечением. Предоставляемый им интерфейс абстрактен и не зависит от конкретных реализаций. Кроссплатформенность ОС реализуется путем создания для каждой из поддерживаемых платформ своего варианта HAL, который должен быть как можно менее объемным.

Поверх HAL работает ядро ОС и драйверы отдельных типов устройств. Ядро является центральной и основополагающей частью ОС, предоставляющей единую точку доступа к ресурсам компьютера, включая процессорное время, память, внешние устройства ввода/вывода. Ядро выполняет ключевые функции ОС по реализации многозадачности: управление памятью, процессами, прерывания и исключения, наборы базовых объектов.

ОС Windows имеет ядро гибридного типа, где переплетены элементы микроядерной и монолитной архитектур:

1) микроядро Windows слишком большое и реализует слишком много функций для того, чтобы называться микроядром; однако многие компоненты ядра располагаются в вытесняемой памяти и взаимодействуют между собой при помощи посылки сообщений, как и предусмотрено в микроядерных архитектурах;

2) одновременно с этим все компоненты ядра Windows работают в едином адресном пространстве и активно используют общие структуры данных, что характерно для монолитной архитектуры.

Причина такого «распухания» ядра кроется в том, что производительность микроядерной архитектуры недостаточна за счет накладных расходов на передачу сообщений. Внутриядерные вызовы соответствующих функций более эффективны, поэтому, начиная с Windows NT 4.0, все ядра Windows имеют именно такую архитектуру.

Известным фактом является то, что корпорация Microsoft для разработки Windows NT наняла группу специалистов из компании DEC во главе с Дэвидом Катлером, которые обладали опытом разработки операционных систем реального времени и многозадачных ОС, таких как VAX/VMS и RSX-11. Это очень сильно повлияло на архитектуру операционной системы, где переносимость является одной из первоочередных задач при разработке.

Если ядро ОС реализует низкоуровневые функции ОС, то исполняемые компоненты реализуют функции ОС на высоком уровне, например: разделение на пользователей с помощью учетных записей, графический пользовательский интерфейс и его элементы и др.

Режим пользователя является непривилегированным уровнем, изолированным от уровня ядра, на котором функционируют:

- реализация API для взаимодействия с ОС на программном уровне (Windows API и POSIXAPI);
- различные службы, выполняющие сервисные задачи;
- графическая оболочка ОС;
- прикладное ПО.

Программные интерфейсы ОС Windows

В ОС Windows для прикладных приложений реализованы два API (табл. 1) – Windows API и POSIX API [4].

Таблица1

API в ОС Windows для прикладных приложений

API	Описание
Windows API	Собственный программный интерфейс ОС Windows
POSIX API	Реализует поддержку набора стандартов POSIX (Portable Operating System Interface – переносимый интерфейс операционных систем). Для приложений, работающих на POSIX API, реализована подсистема POSIX

У ОС Windows есть внутренний программный интерфейс, называемый Native API. Фактически Windows API и POSIX API реализованы поверх Native API и являются его «обертками».

Такой подход обеспечивает гибкость и обратную совместимость, так как изменение разработчиками Windows реализации компонентов операционной системы не требует переписывания API для прикладного ПО, а ограничивается лишь незначительным изменением его реализации. Разработчикам прикладных приложений не понадобится переписывать свои приложения под новый API, что сохраняет обратную совместимость с приложениями, написанными для устаревших версий ОС Windows.

Native API состоит из двух частей – клиентской и серверной. Взаимодействие между ними реализовано по клиент-серверной архитектуре (рис. 2, табл. 2) [7].

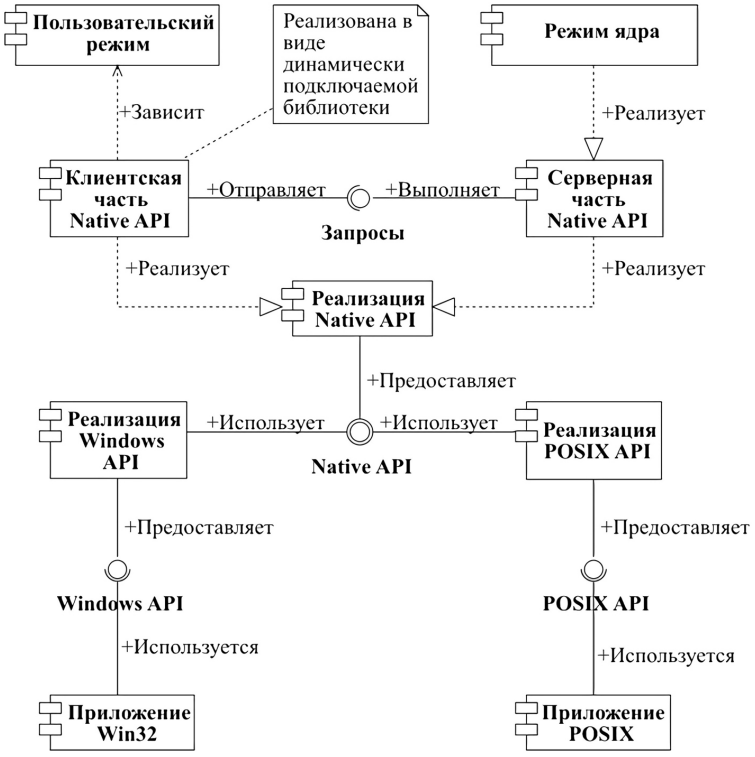

Рис. 2. Native API

Интерфейсы операционных систем

Таблица 2

Части Native API

Часть Native API	Описание
Клиентская	Реализована в виде отдельной динамически подключаемой библиотеки, работающей в режиме пользователя и отправляющей запросы к серверной части
Серверная	Реализована в режиме ядра. Принимает и выполняет запросы от клиентской части Native API

Прикладные приложения для взаимодействия с ОС Windows выполняют вызовы функций из Windows API, а те, в свою очередь, выполняют системные вызовы внутри посредством функций из внутреннего API ОС Windows, что приводит к посылке запросов к ядру Windows. Таким образом, любой системный вызов фактически сводится к взаимодействию по клиент-серверной архитектуре.

Модель работы и программирования GUI-приложений

Модель работы и программирования приложений с GUI (Graphical User Interface) является отличительной чертой всех версий и разновидностей Windows по сравнению с другими ОС [1].

Первоначально модель была реализована в самой первой ОС Windows – Windows 1.0, выпущенной в 1985 г. С этого времени ее реализация присутствовала во всех последующих ОС Windows, где претерпевала серьезные изменения в деталях реализации, но вне зависимости от вносимых изменений не меняла своей сути. Благодаря этому сохраняется обратная совместимость с GUI-приложениями, написанными для предыдущих версий Windows, как на уровне исходного кода с внесением минимальных изменений с последующей перекомпиляцией приложения, так и на уровне бинарных файлов в отдельных случаях.

Первые системы Windows версий 1, 2 и 3, а также Windows 9x (Windows 95, Windows 98, Windows ME) работали «поверх» системы MS-DOS и являлись для нее оболочкой. Они имели оригинальную, сложную и не полностью документированную внутреннюю структуру. До Windows 95 требовалось, чтобы была предварительно установлена система MS-DOS, загружавшаяся при включении компьютера. Затем система запускалась как приложение, работа которого могла быть прервана в любое время, в результате чего пользователь возвращался к командной строке в MS-DOS. MS-DOS представляла драйверы для некоторых типов устройств, в то время как Windows требовала специальным образом написанных приложений и работала с отличным от приложений для MS-DOS форматом исполняемых файлов, значительно более сложным, чем аналогичный формат в MS-DOS. Кроме того, у Windows было большое количество собственных драйверов и имелась собственная система управления памятью.

MS-DOS не была многозадачной ОС, в ней не была реализована концепция процессов и потоков, каждая запущенная программа выполнялась в одном адресном пространстве. Появление Windows в качестве оболочки над MS-DOS привнесло в однозадачную операционную систему невытесняемую многозадачность, которая была реализована по схеме, когда все задачи представляют собой потоки, работающие в одном адресном пространстве. В Windows при запуске каждый исполнимый файл приложения становился модулем, у которого был свой указатель, исполнимый код, а также свои ресурсы. Система позволя-

ла запускать одно и то же приложение в двух и более разных копиях. Технически эти копии принадлежали одному и тому же модулю, а значит, разделяли общие ресурсы и исполнимый код; но каждая копия имела свой набор переменных, расположенных в отдельных сегментах данных, и каждый такой отдельный сегмент данных тоже имел свой указатель. Таким образом, если запустить сначала одну копию приложения, а затем вторую, то вторая сможет прочитать данные из сегмента первой запущенной копии.

Начиная с Windows 95, MS-DOS и Windows уже разрабатывались и поставлялись в качестве единой системы, в которой ко времени выхода Windows ME код MS-DOS был полностью вытеснен кодом Windows, и MS-DOS нужен был только для запуска ядра Windows. Затем Windows, работавшая «поверх» MS-DOS, была полностью заменена на системы из семейства Windows NT, имеющие свое собственное ядро, полностью отделенное от MS-DOS, а также вытесняемую многозадачность вместо невытесняемой, как у прежних Windows.

Следы этой невытесняемой многозадачности остались в организации GUI-приложений с их обработчиками сообщений. Теперь каждый экземпляр приложений запускается в отдельном процессе с отдельной копией исполняемого кода, ресурсов и набором переменных. Данные хранятся в сегменте, который напрямую проецируется в адресное пространство процесса, и вторая запущенная копия одного и того же приложения не может получить доступ к данным первой, однако для сохранения обратной совместимости указатель на набор переменных остался, но стал указателем на модуль приложения.

Графическое приложение Windows (рис. 3), чтобы показать на экране свое главное окно, выполняет следующие стандартные действия [8]:

- регистрирует в системе класс своего окна вместе с оконной процедурой;
- создает главное окно, внутри которого размещает компоненты графического пользовательского интерфейса (Graphical User Interface, GUI);
- запускает цикл сообщений, через который получает сообщения от операционной системы.

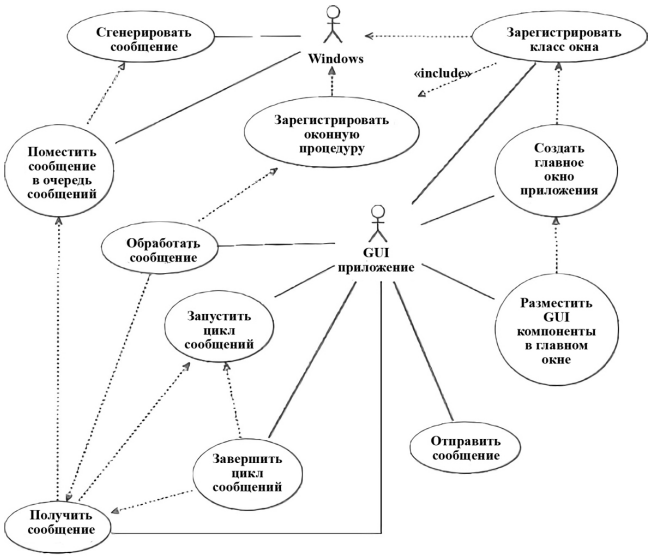

Рис. 3. Модель графических приложений Windows

Интерфейсы операционных систем

В Windows, основанных на MS-DOS, приложение регистрировало в системе класс своего окна только при запуске первой копии приложения, все последующие запускаемые копии, делая проверку, просто пропускали регистрацию класса; чтобы старые приложения продолжали работать в Windows NT, операционная система заставляет их «думать», что они всегда запускаются первой копией.

Оконная процедура – это специальная процедура, реализацию которой определяют сами разработчики приложения и которая вызывается в цикле сообщений для того, чтобы приложение могло обрабатывать поступающие сообщения. Помимо определяемой оконной процедуры существует стандартная оконная процедура, в которой реализовано стандартное поведение в ответ на те или иные сообщения от операционной системы. Смысл этой стандартной оконной процедуры состоит в том, чтобы приложение могло задействовать стандартное поведение для сообщений, которые пользовательской оконной процедурой не обрабатываются.

Цикл сообщений – это цикл в исходном коде приложения, с помощью которого зацикливается выполнение приложения. Без цикла сообщений приложение завершало бы свою работу, не успев показать пользователю свое главное окно. Выход из цикла сообщений осуществляется в том случае, если приложение завершает свою работу, например, когда пользователь нажал на кнопку закрытия окна. В этом случае от ОС поступает специальное сообщение, которое инициирует выход из цикла сообщений.

Операционная система при отправке сообщений окну помещает их в очередь. Сообщения, получаемые от операционной системы, касаются различных событий, которые происходят в процессе работы приложения, например, создания главного окна, его сворачивания или разворачивания с панели задач, перемещения по экрану, взаимодействия пользователя с его элементами, закрытия главного окна и др.

Важный нюанс – ОС Windows создает очереди сообщений для потоков. Из этого следует, что у процесса могут быть потоки со своими отдельными очередями сообщений. Однако не у каждого потока имеется очередь сообщений, так как она создается менеджером окон для потока по мере необходимости и прозрачным для него образом, то есть с точки зрения отдельного потока все выглядит так, будто очередь сообщений существовала всегда. Здесь создание по мере необходимости означает, что попытка отправить сообщение или проверить их наличие инициирует создание очереди сообщений для потока. Это обусловлено тем, что создание очереди сообщений для любых потоков было бы расточительством ресурсов системы, так как потоки, которые не используют GUI, не должны платить за то, что они не использовали. В большинстве случаев в GUI-приложениях очередь сообщений создается только для одного потока, обычно главного, ответственного за запуск цикла сообщений и создание главного окна приложения.

Библиотека стандартных GUI-компонентов

В ОС Windows есть библиотека стандартных GUI-компонентов, в которой реализованы стандартные компоненты графического пользовательского интерфейса: кнопки, текстовые поля, комбинированные списки и др. Физически эта библиотека воплощена в виде динамически подключаемой библиотеки, которая может быть использована прикладными приложениями для создания своего GUI [2].

Каждый GUI-элемент внутри главного окна приложения тоже является окном, только дочерним по отношению к главному окну. Каждый компонент GUI имеет свой собственный зарегистрированный в системе класс окна, в котором реализована логика компонента. Взаимодействие компонента с главным окном приложения осуществляется посредством посылки сообщений оконной процедуре главного окна. Например, пользователь нажал на кнопку, это нажатие сначала обрабатывается в оконной процедуре кнопки, затем сообщение о нажатии кнопки посылается оконной процедуре главного окна, являющегося родительским для кнопки. Помимо этого, сообщения также могут посылаться в оконные процедуры компонентов, например, чтобы в ответ получить введенный пользователем текст в текстовом поле.

Существует множество библиотек, в том числе кроссплатформенных, позволяющих создавать графические приложения. Их особенность в ОС Windows заключается в том, что они за слоями своих абстракций скрывают регистрацию класса окна, создание экземпляра окна, запуск цикла сообщений и реализацию оконной процедуры.

Заключение

ОС Windows продолжает активно развиваться: с выходом каждого крупного обновления появляются новые функции и улучшается пользовательский интерфейс.

Однако изменяются и ключевые свойства ОС, такие как способы межпроцессного взаимодействия. Например, в версии 10 добавлена поддержка сокетов домена Unix, что свело на нет разрыв между возможностями систем POSIX и Windows в способах организации межпроцессного взаимодействия.

Литература

1. *Рихтер Д.* Windows. Создание эффективных Win32 приложений с учетом специфики 64-разрядной версии Windows: пер. с англ. 4-е изд. СПб.: Питер, 2008. 720 с.
2. *Рихтер Д., Назар К.* Windows via C/C++. Программирование на языке Visual C++: пер. с англ. СПб.: Питер, 2009. 896 с.
3. *Руссинович М., Соломон Д.* Внутреннее устройство Microsoft Windows: пер. с англ. 6-е изд. СПб.: Питер, 2013. 800 с.
4. *Руссинович М., Соломон Д., Ионеску А.* Внутреннее устройство Microsoft Windows. Основные подсистемы ОС: пер. с англ. 6-е изд. СПб.: Питер, 2014. 672 с.
5. *Таненбаум Э., Бос Х.* Современные операционные системы: пер. с англ. 4-е изд. СПб.: Питер, 2015. 1120 с.
6. *Таненбаум Э., Остин Т.* Архитектура компьютера: пер. с англ. 6-е изд. СПб.: Питер, 2013. 816 с.
7. *Таненбаум Э., Уэзеролл Д.* Компьютерные сети: пер. с англ. 5-е изд. СПб.: Питер, 2012. 960 с.
8. *Щупак Ю.* Win32 API. Эффективная разработка приложений. СПб.: Питер, 2007. 572 с.

Literatura

1. *Rikhter D.* Windows. Sozdanie effektivnykh Win32 prilozhenij s uchetom spetsifiki 64-razryadnoj versii Windows: per. s angl. 4-e izd. SPb.: Piter, 2008. 720 s.

Интерфейсы операционных систем

2. *Rikhter D., Nazar K.* Windows via C/C++. Programmirovanie na yazyke Visual C++: per. s angl. SPb.: Piter, 2009. 896 s.
3. *Russinovich M., Solomon D.* Vnutrennee ustrojstvo Microsoft Windows: per. s angl. 6-e izd. SPb.: Piter, 2013. 800 s.
4. *Russinovich M., Solomon D., Ionesku A.* Vnutrennee ustrojstvo Microsoft Windows. Osnovnye podsistemy OS: per. s angl. 6-e izd. SPb.: Piter, 2014. 672 s.
5. *Tanenbaum E., Bos Kh.* Sovremennye operatsionnye sistemy: per. s angl. 4-e izd. SPb.: Piter, 2015. 1120 s.
6. *Tanenbaum E., Ostin T.* Arkhitektura komp'yutera: per. s angl. 6-e izd. SPb.: Piter, 2013. 816 s.
7. *Tanenbaum E., Uezeroll D.* Komp'yuternye seti: per. s angl. 5-e izd. SPb.: Piter, 2012. 960 s.
8. *Shchupak Yu.* Win32 API. Effektivnaya razrabotka prilozhenij. SPb.: Piter, 2007. 572 s.