ГЛАВА 1

Знакомство с Win32 и Win64

В этой главе вы познакомитесь с семейством операционных систем (ОС) Microsoft Windows и интерфейсом прикладного программирования (Application Programming Interface, API), который используется всеми членами этого семейства. Здесь также кратко описывается новейший 64-разрядный API Win64 и достаточно подробно обсуждается проблема переносимости программного обеспечения между Win32 и Win64. Для удобства изложения мы будем ссылаться, главным образом, просто на Windows и Windows API. Как правило, раздельные ссылки на Win32 и Win64 будут делаться лишь в тех случаях, когда различия между этими интерфейсами будут иметь существенное значение. Сориентироваться в том, что именно автор имеет в виду, когда говорит о Windows, — операционную систему или интерфейсе для разработки программ, — читателю поможет контекст.

Подобно API любой другой ОС, Windows API также располагает собственным набором соглашений и приемов программирования, укладывающихся в рамки философии Windows. Со стилем программирования Windows вы ознакомитесь на примере обычного копирования файлов, однако тот же стиль используется при управлении файлами, процессами, памятью, а также такими более развитыми средствами, как синхронизация потоков. Для упомянутого примера будет приведен также код, в котором используется стандартная библиотека C, что облегчит вам сравнение стиля программирования, принятого в Windows, с более распространенными стилями.

Мы начнем с общего обзора основных средств, предоставляемых любой современной операционной системой, чтобы понять, как эти средства используются в Windows.

Основные возможности операционных систем

Windows обеспечивает доступность базовых средств ОС в столь непохожих друг на друга системах, как мобильные телефоны, карманные устройства, переносные компьютеры и серверы масштаба предприятия. Возможности ОС можно охарактеризовать, рассмотрев наиболее важные ресурсы, которыми управляют современные операционные системы.

- **Память.** ОС управляет сплошным, или плоским (flat), виртуальным адресным пространством большого объема, перемещая данные между физической памятью и диском или иным накопительным устройством прозрачным для пользователя образом.
- **Файловые системы.** ОС управляет пространством именованных файлов, предоставляя возможности прямого и последовательного доступа к файлам, а также средства управления файлами и каталогами. Используемые в большинстве систем пространства имен являются иерархическими.
- **Именование и расположение ресурсов.** Файлы могут иметь длинные, описательные имена, причем принятая схема именования распространяется также на такие объекты, как устройства, а также объекты синхронизации или межпроцессного взаимодействия. Размещение именованных объектов и управление доступом к ним также являются прерогативой ОС.
- **Многозадачность.** ОС должна располагать средствами управления процессами, потоками и другими единицами, способными независимо выполняться в асинхронном режиме. Задачи могут планироваться и вытесняться в соответствии с динамически определяемыми приоритетами.
- **Взаимодействие и синхронизация.** ОС управляет обменом информацией между задачами и их синхронизацией в изолированных системах, а также взаимодействием сетевых систем между собой и сетью Internet.
- **Безопасность и защита.** ОС должна предоставлять гибкие механизмы защиты ресурсов от несанкционированного или непреднамеренного доступа и нанесения ущерба системе.

Microsoft Windows Win 32/Win64 API обеспечивает поддержку не только этих, но и множества других средств ОС, и делает их доступными в ряде версий Windows, некоторые из которых постепенно выходит из употребления, а некоторые поддерживает лишь то или иное подмножество полного API.

Эволюция Windows

Windows API поддерживается несколькими версиями Windows. Существование ряда различных версий Windows может вносить некоторую неразбериху, однако с точки зрения программиста все они аналогичны друг другу. В частности, все версии поддерживают подмножества *идентичных* Windows API. Программы, разработанные для одной системы, без особых трудностей будут выполняются и в любой другой, и при этом обеспечивается переносимость исходных, а в большинстве случаев — и бинарных кодов.

Каждая новая версия в определенной степени расширяла функциональные возможности API, хотя уже с самого начала API в целом отличался замечательной стабильностью. Ниже перечислены главные факторы, которые определяют эволюционное развитие Windows:

- **Масштабируемость.** Новые версии способны выполняться на более широком спектре систем, включая серверы масштаба предприятия, использующие оперативную память большого объема и запоминающие устройства повышенной емкости.
- **Интеграция.** Каждый новый выпуск системы интегрирует в себя элементы дополнительных технологий, такие, например, как использование мультимедийных данных или подключение к беспроводным сетям, Web-службы или самонастраивающиеся (plug-and-play) устройства. В целом, рассмотрение этих технологий выходит за рамки данной книги.
- **Простота использования.** Элементы улучшения внешнего вида графического рабочего стола или средства, упрощающие пользование системой, без труда обнаруживаются в каждом новом выпуске системы.
- **Совершенствование API.** За время своего существования API был дополнен новыми замечательными возможностями. Именно API является центральной темой данной книги.

Версии Windows

ОС Windows, в виде развивающейся последовательности версий, используется, начиная с 1993 года. Во время написания данной книги на Web-сайте компании Microsoft в качестве основных фигурировали следующие версии:

- **Windows XP**, включая выпуски Home, Professional и ряд других, которая ориентирована на индивидуальных пользователей. Большинство коммерческих PC, поступающих на рынок на сегодняшний день, включая лэптопы и ноутбуки, поставляются с уже установленной Windows XP соответствующего типа. В рамках данной книги различия между версиями, как правило, существенного значения не имеют.
- **Windows Server 2003**, которая выпускается в виде продуктов Small Business Server, Storage Server 2003, а также некоторых других, и ориентирована на управление приложениями для предприятий и серверными приложениями. В системах, работающих под управлением Windows Server 2003, часто применяется симметричная многопроцессорная обработка (Symmetric Multiprocessing, SMP), характеризующаяся использованием одновременно нескольких независимых процессоров. Новые 64-разрядные приложения, требующие Win64, появляются преимущественно на системах Windows Server 2003.
- **Windows 2000**, которая доступна в виде выпусков Professional и нескольких разновидностей выпусков Server и по-прежнему широко используется как в персональных, так и в серверных системах. Со временем Windows XP и будущие версии Windows вытеснят Windows 2000, продажа которой уже прекращена.

- **Windows Embedded, Windows CE и Windows Mobile**, которые представляют собой специализированные версии Windows, ориентированы на использование в малых системах, таких, например, как ручные (palmtop) и встроенные (embedded) устройства обработки данных, и предоставляют широкие подмножества возможностей Windows.

Устаревшие предыдущие версии Windows

В настоящее время продолжает использоваться ряд предыдущих версий Windows, которые считаются устаревшими и более не предлагаются или не поддерживаются компанией Microsoft, однако многие, хотя и не все, из обсуждаемых в данной книге примеров будут выполняться и под управлением этих систем. Перечень таких систем, постепенно выходящих из употребления, приводится ниже.

- **Windows NT** 3.5, 3.5.1 и 4.0, предшествовавшие современным версиям NT и восходящие своими корнями к 1993 году, из которых наибольшей популярностью пользовалась версия Windows NT 4.0, модернизированная с помощью пакета обновлений Service Pack 3 (SP3). Первоначально системы NT предназначались для профессиональных пользователей и серверов, в то время как версиями, распространяемыми для персональных и офисных нужд, служили версии Windows 9x (см. ниже). Первоначально Windows 2000 называлась Windows NT Version 5.0, в связи с чем многие пользователи, имея в виду системы Windows 2000, Windows Server 2003 или Windows XP, все еще употребляют названия Windows NT или Windows NT5. Несмотря на важные исключения, что особенно касается последних глав, под управлением NT Version 4.0 будут корректно, хотя и не всегда оптимально, работать почти все программы из числа тех, которые представлены в данной книге.
- **Windows 95, 98 и Windows ME** (или просто — **Windows 9x**, поскольку различия между указанными системами вряд ли могут считаться существенными), которые предназначались главным образом для настольных (desktop) и переносных (laptop) компьютеров и не включали в себя, помимо прочего, средств безопасности Windows NT. В настоящее время эти системы вытесняются системой Windows XP, которая объединяет предоставляемые ими средства со средствами Windows NT. Многие примеры программ, хотя и не все, особенно из числа тех, которые приводятся в начальных главах книги, будут корректно выполняться и под управлением Windows 9x.

Возвращаясь назад в прошлое, можно отметить, что до появления Windows 95 доминировала 16-разрядная ОС Windows 3.1, графический пользовательский интерфейс (Graphical User Interface, GUI) которой можно считать предшественником современных Windows GUI. Вместе с тем, многие важнейшие возможности ОС, такие как истинная многозадачность, управление памятью и средства защиты, она не поддерживала.

Если обратиться к еще более отдаленному прошлому, к концу восьмидесятых годов прошлого столетия, то в качестве исходной “IBM PC”-системы можно ука-

зать DOS. В DOS имелся всего-навсего простейший интерфейс командной строки, но DOS-команды используются и по сей день. В действительности, большинство из программ, рассматриваемых в данной книге в качестве примеров, написаны в виде консольных приложений и поэтому могут запускаться из командной строки, а для тестирования производительности даже специально предусмотрены пакетные файлы DOS.

Windows NT 5.x

По отношению к системам Windows 2000, Windows XP и Windows Server 2003 используют собирательное название Windows NT Version 5.x или просто NT5. В каждой из трех указанных систем эксплуатируется пятая версия ядра Windows NT, хотя младший номер версии ("x" в "5.x") может быть различным. Так, в Windows XP используется ядро версии NT 5.1.

Несмотря на то что многие программы будут выполняться и под управлением предыдущих версий Windows, мы, как правило, предполагаем применение версии NT5, некоторые возможности которой отсутствуют в предыдущих версиях. Поскольку любая современная Windows-система обладает возможностями NT5, такое предположение не чревато никакими осложнениями, но зато снимает любые ограничения на использование усовершенствованных возможностей ОС. Вместе с тем, на многих устаревших системах все еще могут оставаться установленными ранние версии Windows NT или Windows 9x, и поэтому программы примеров сразу же после запуска проверяют номер версии Windows и в необходимых случаях прекращают свою работу с выводом сообщения об ошибке.

В документации по Microsoft API приводятся требования к номеру версии, сформулированные в терминах NT, Windows (что в данном контексте относится к версиям 9x) и CE, и ряд других требований. В случае любого рода сомнений относительно возможности использования тех или иных функций API в конкретной версии Windows следует обратиться к документации.

Другие интерфейсы программирования для Windows

Windows (под этим термином, если отдельно не оговаривается иное, мы подразумеваем API Win32 и Win64, а также NT5) способна обеспечивать также поддержку среды других "подсистем", хотя этой возможностью пользуются редко и прямого отношения к тематике данной книги она не имеет. Ядро ОС NT имеет действительно надежную защиту от воздействия со стороны приложений. Windows является для него лишь одной из нескольких возможных сред. Так, предоставляемый компанией Microsoft набор инструментальных средств Windows Resource Kit включает подсистему POSIX, но кроме того существуют подсистемы POSIX, предлагаемые в рамках проекта разработки программного обеспечения с открытым исходным кодом.

Поддержка процессоров

Хотя это не и не входит в сферу непосредственных интересов разработчика приложений, вам следует знать, что Windows поддерживает самые различные базовые процессоры и архитектуры систем, для чего предусмотрен уровень аппаратных абстракций (Hardware Abstraction Layer, HAL), который обеспечивает переносимость программ на системы с другой архитектурой процессора.

Windows выполняется преимущественно на процессорах семейства Intel x86, новейшими представителями которого являются процессоры Pentium и Xeon, а одним из предыдущих — Intel 486. Широкое распространение получили совместимые с ними процессоры компании Advanced Micro Devices (AMD). Кроме того, Windows изначально проектировалась таким образом, чтобы обеспечивалась независимость от типа процессора. Что немаловажно, Windows Server 2003 поддерживается процессором Intel Itanium, новая 64-разрядная архитектура которого самым радикальным образом отличается от классической архитектуры процессоров семейства x86.

Другими примерами независимости Windows от архитектуры процессоров, относящимися как к прошлому, так и к настоящему, могут служить следующие:

- Windows CE способна выполняться на целом ряде процессоров, не принадлежащих семейству x86.
- Windows NT первоначально поддерживалась Alpha-процессорами компании Digital Equipment Corporation (которая была приобретена сначала компанией Compaq, а затем компанией Hewlett Packard).
- 64-разрядные процессоры компании AMD Athlon 64 и Opteron (AMD64) обеспечивают 64-разрядное расширение архитектуры x86, отражающее иной подход по сравнению с тем, который используется в архитектуре Itanium.
- Недавно объявленные компанией Intel 32/64-разрядные процессоры будут представлять собой 64-разрядные расширения процессоров x86.

Воздействие Windows на ситуацию на рынке

Тот факт, что система Windows способна обеспечивать важнейшие функциональные возможности на нескольких платформах, вряд ли можно считать уникальным. В конце концов, этим могут похвастаться многие коммерческие ОС, не говоря уже об ОС с открытым исходным кодом, а UNIX[1] и Linux уже в течение длительного времени эксплуатируются на самых разнообразных системах. Между тем, использование Windows и разработка приложений для Windows сулят значительные преимущества как в коммерческом, так и в техническом отношении.

[1] Замечания, сделанные в адрес UNIX, в равной степени относятся также к Linux и некоторым другим системам, поддерживающим POSIX API.

- Windows занимает на рынке, особенно на рынке настольных систем, господствующее положение, которое прочно удерживается ею в течение многих лет.[2] Поэтому перед приложениями Windows открыт огромный целевой рынок, емкость которого исчисляется десятками миллионов систем, вследствие чего остальные настольные системы, включая UNIX, Linux и Macintosh, играют на рынке значительно меньшую роль.
- Приложения Windows могут использовать GUI, знакомый десяткам миллионов пользователей, а для многих приложений предусмотрена возможность их адаптации, или "локализации", в соответствии с региональными требованиями, предъявляемыми к языку и внешнему виду интерфейса.
- Windows поддерживает SMP-системы. Сфера применения Windows не ограничивается настольными системами и охватывает как серверы масштаба подразделения и предприятия, так и высокопроизводительные рабочие станции.[3]
- Windows (правда, это не относится к Windows 9x и Windows CE) сертифицирована Управлением национальной безопасности (National Security Agency, NSA) как система, обеспечивающая уровень безопасности C2.
- Применение большинства других ОС, отличных от UNIX, Linux и Windows, ограничивается только системами, предоставляемыми единственным поставщиком.
- Операционные системы семейства Windows предлагают ряд возможностей, которые в стандартной системе UNIX отсутствуют, хотя и могут быть доступными в некоторых реализациях. В качестве примера можно привести систему безопасности уровня C2, а также службы NT Services.

Windows предоставляет функциональность современных ОС, а диапазон систем, которые могут работать под ее управлением, простирается от текстовых процессоров и почтовых программ до интегрированных систем предприятия и серверов крупных баз данных. На принятие решений относительно способа разработки Windows-приложений оказывают влияние как соображения технического порядка, так и корпоративные требования.

---

[2] Иногда, имея в виду в основном серверы, но не исключая и персональные приложения, говорят о возможной угрозе преобладанию Windows со стороны Linux. Хотя сама по себе эта тема является чрезвычайно интересной, размышления о путях будущего развития систем, не имеющие непосредственного отношения к рассмотрению сравнительных достоинств и недостатков Windows и Linux, выходят за рамки данной книги.

[3] О том, насколько разнообразен круг систем, на которых может быть развернута Windows, говорит хотя бы тот факт, что диапазон компьютеров, использованных для тестирования приведенных в этой книге примеров программ, простирается от давно забытой 486-й модели с 16 Мбайт ОЗУ до четырехпроцессорного (процессоры Xeon с рабочей частотой 2 ГГц) сервера масштаба предприятия, оборудованного ОЗУ емкостью 8 Гбайт.

Windows, стандарты и открытые системы

Эта книга посвящена разработке приложений с использованием Windows API. Вполне естественно, что у программистов, воспитанных на UNIX и открытых системах, могут возникнуть следующие вопросы: “Является ли Windows открытой системой?”, “Представляет ли собой Windows промышленный стандарт?”, “Не является ли Windows всего лишь очередным патентованным API?” Ответы на эти вопросы во многом зависят от того, что именно понимается под определениями *открытая* (open), *промышленный стандарт* (industry standard) или *патентованный* (proprietary), а также от того, какие преимущества ожидаются от использования открытых систем.

Windows API полностью отличается от API стандарта POSIX, поддерживаемого системами Linux и UNIX. Windows не подчиняется стандарту X/Open, как не подчиняется и никакому другому открытому промышленному стандарту из тех, которые были предложены соответствующими органами стандартизации или промышленными консорциумами.

Windows контролируется единственным поставщиком. Хотя Microsoft и заявляет о своей готовности приспосабливаться к требованиям отрасли и учитывать их, в этих вопросах сама же она является арбитром и исполнителем в одном лице. Отсюда следует, что, помимо других преимуществ, пользователи Windows получают многие из выгод, которые обычно предлагают открытые стандарты.

- Унифицированные реализации быстрее достигают рынка.
- Отсутствуют какие-либо неожиданные фирменные “улучшения” или “расширения”, с которыми потом приходится бороться программисту, хотя небольшие различия, существующие между различными платформами Windows, все же приходится учитывать.
- Вся совокупность полноценных ОС-продуктов, предлагающих все необходимые возможности, определена и реализована одним и тем же поставщиком. Разработчикам приложений остается решать только высокоуровневые задачи.
- Базовая аппаратная платформа является открытой. Разработчики могут выбирать любого из многочисленных поставщиков платформ по своему усмотрению.

Жаркие споры относительно того, к добру ли такая ситуация для пользователей и компьютерной индустрии в целом, или она только вредит общему делу, еще не закончились. Мы не будем пытаться участвовать в этом споре; задача данной книги состоит лишь в том, чтобы помочь разработчикам приложений как можно скорее приступить к работе в Windows.

В действительности системы Windows поддерживают многие важные стандарты. Так, Windows поддерживает стандартные библиотеки C и C+ и целый ряд открытых стандартов межплатформенного взаимодействия. В качестве примера можно привести сокеты Windows (Windows Sockets), предоставляющие стандарт-

ный интерфейс сетевого программирования, который обеспечивает возможность использования TCP/IP и других сетевых протоколов и тем самым открывает возможности доступа в Internet и взаимодействия с системами, не принадлежащими семейству Windows. То же самое остается справедливым и по отношению к протоколу удаленного вызова процедур (Remote Procedure Calls, RPC).[4] Системы самой различной природы могут связываться с высокоуровневыми системами управления базами данных (СУБД) при помощи языка структурированных запросов (SQL). Наконец, в общий круг предложений Windows входит поддержка Internet, обеспечиваемая Web-серверами и серверам иного рода. Windows поддерживает такие ключевые стандарты, как TCP/IP, а на активно действующем рынке поставщиков решений Windows вам предлагают приобрести за разумную плату множество других ценных дополнительных продуктов, в том числе клиенты и серверы X Window.

Резюмируя, можно утверждать, что Windows поддерживает наиболее важные из стандартов межплатформенного взаимодействия, и хотя основной API является собственностью компании, он доступен по умеренной цене для широкого ряда систем.

Библиотеки совместимости

Несмотря на наличие библиотек совместимости (compatibility libraries), ими пользуются очень редко. Существуют две возможности.

- В системах на основе UNIX, Linux, Macintosh и некоторых других может быть развернута одна из библиотек совместимости Windows, например, эмулятор Windows с открытым исходным кодом Wine, что обеспечивает переносимость исходного кода из Windows.
- За счет использования программного обеспечения с открытым исходным кодом и набора инструментальных средств Windows Resource Kit компании Microsoft поверх подсистемы Windows может быть развернута библиотека совместимости POSIX. Весьма ограниченная по своим возможностям библиотека совместимости входит в состав среды визуальной разработки приложений Microsoft Visual C++.

Таким образом, имеется, пусть даже и редко используемая, возможность выбора одного API и развертывания разработанных с его помощью переносимых приложений на системах Windows, POSIX и даже Macintosh.

[4] Протоколы Windows Sockets и RPC не являются частью самой Windows, что не воспрепятствовало описанию сокетов в данной книге, поскольку они самым непосредственным образом укладываются в рамки интересующей нас общей темы и используемого подхода.

Принципы, лежащие в основе Windows

Полезно никогда не забывать о некоторых базовых принципах Windows. В Windows API имеется множество как самых незаметных, так и значительных отличий от других API, таких как POSIX API, с которым знакомы программисты, работающие в UNIX и Linux. И хотя с применением Windows не связаны какие-либо специфические трудности в работе, она потребует от вас внесения некоторых изменений в привычные стиль и методику программирования.

Ниже описаны некоторые из важнейших характеристик Windows, с которыми вы ближе познакомитесь по мере дальнейшего изложения материала.

Многие системные ресурсы Windows представляются в виде *объектов ядра* (kernel objects), для идентификации и обращения к которым используются *дескрипторы* (handles). По смыслу эти дескрипторы аналогичны дескрипторам (descriptors) файлов и идентификаторам (ID) процессов в UNIX.[5]

- Любые манипуляции с объектами ядра осуществляются только с использованием Windows API. "Лазеек" для обхода этого правила нет. Подобная организация работы согласуется с принципами абстрагирования данных, используемыми в объектно-ориентированном программировании, хотя сама система Windows объектно-ориентированной не является.
- К объектам относятся файлы, процессы, потоки, каналы межпроцессного взаимодействия, объекты отображения файлов, события и многое другое. Объекты имеют атрибуты защиты.
- Windows — богатый возможностями и гибкий интерфейс. Во-первых, одни и те же или аналогичные задачи могут решаться с помощью сразу нескольких функций; так, имеются вспомогательные функции (convenience functions), полученные объединением часто встречающихся последовательностей функциональных вызовов в одну функцию (к числу подобных функций принадлежит и функция `CopyFile`, используемая в одном из примеров далее в этой главе). Во-вторых, функции часто имеют многочисленные параметры и флаги, многие из которых обычно игнорируются. Данная книга не претендует на роль энциклопедического справочника, и основное внимание в ней концентрируется лишь на наиболее важных функциях и параметрах.
- Windows предлагает многочисленные механизмы синхронизации и взаимодействия, обеспечивающие удовлетворение самых разнообразных запросов.
- Базовой единицей выполнения в Windows является поток (thread). В одном процессе (process) могут выполняться один или несколько потоков.
- Для функций Windows используются длинные описательные имена. Приведенные ниже в качестве примера имена функций иллюстрируют не только соглашения об использовании имен, но и многоликость функций Windows:

[5] Несмотря на аналогию между упомянутыми дескрипторами и дескрипторами HWND и HDC, используемыми при написании программ для Windows GUI, между ними существует ряд отличий.

```
WaitForSingleObject
WaitForSingleObjectEx
WaitForMultipleObjects
WaitNamedPipe
```

Существует также несколько соглашений, регулирующих порядок использования имен типов:

- Имена предопределенных типов данных, необходимых API, также являются описательными, и в них должны использоваться прописные буквы. К числу наиболее распространенных относятся следующие типы данных:

  `BOOL` (определен как 32-битовый объект, предназначенный для хранения одного логического значения)

  `HANDLE`

  `DWORD` (вездесущее 32-битовое целое без знака)

  `LPTSTR` (указатель на строку, состоящую из 8- или 16-битовых символов)

  `LPSECURITY_ATTRIBUTES`

  С другими многочисленными типами данных вы будете знакомиться по мере изложения материала.

- В именах предопределенных типов указателей операция `*` не используется, и они отражают дополнительные отличия между указателями различного типа, как, например, в случае типов `LPTSTR` (определен как `TCHAR *`) и `LPCTSTR` (определен как `const TCHAR *`). *Примечание.* Тип `TCHAR` может обозначать как обычный символьный тип `char`, так и двухбайтовый тип `wchar_t`.

- В отношении использования имен переменных, — по крайней мере, в прототипах функций, — также имеются определенные соглашения. Так, имя `lpszFileName` соответствует "длинному указателю на строку, завершающуюся нулевым символом", которая содержит имя файла. Этот пример иллюстрирует применение так называемой "венгерской нотации", которой мы в данной книге, как правило, не стремимся придерживаться. Точно так же, `dwAccess` — двойное слово (32 бита), содержащее флаги прав доступа к файлу, где "dw" означает "double word" — "двойное слово".

Примечание

Будет очень полезно, если вы просмотрите системные заголовочные (включаемые) файлы, в которых содержатся определения функций, констант, флагов, кодов ошибок и тому подобное. Многие из представляющих для нас интерес файлов, аналогичных тем, которые предложены ниже в качестве примера, являются частью среды Micrrosoft Visual C++ и обычно устанавливаются в каталоге `Program Files\Microsoft Visual Studio.NET\Vc7\PlatformSDK\Include` (или `Program Files\Microsoft Visual Studio\VC98\Include` в случае VC++ 6.0):

`WINDOWS.H` (файл, обеспечивающий включение всех остальных заголовочных файлов)

`WINNT.H`

`WINBASE.H`

Наконец, несмотря на то что оригинальный API Win32 с самого начала разрабатывался как совершенно независимый интерфейс, он проектировался с учетом обеспечения обратной совместимости с API Win16, входившим в состав Windows 3.1. Это привело к некоторым досадным с точки зрения программиста последствиям:

- В названиях типов встречаются элементы анахронизма, как, например, в случае типов `LPTSTR` и `LPDWORD`, ссылающихся на "длинный указатель", который является простым 32- или 64-битовым указателем. Необходимость в указателях какого-либо иного типа отсутствует. Иногда составляющая "длинный" опускается, и тогда, например, типы `LPVOID` и `PVOID` являются эквивалентными.[6]
- В имена некоторых символических констант, например `WIN32_FIND_DATA`, входит компонент "`WIN32`", хотя те же константы используются и в Win64.
- Несмотря на то что упомянутая проблема обратной совместимости в настоящее время потеряла свою актуальность, она оставила после себя множество 16-разрядных функций, ни одна из которых в этой книге не используется, хотя и могло бы показаться, что эти функции играют весьма важную роль. В качестве примера можно привести функцию `OpenFile`, которая, судя по ее названию, нужна для открытия файлов, тогда как в действительности для открытия существующих файлов всегда следует пользоваться только функцией `CreateFile`.

Подготовка к работе с Win64

Интерфейс Win64, который во время написания данной книги поддерживался Windows XP и Windows Server 2003 на процессорах семейства AMD64 (Opteron и Athlon 64) компании AMD и процессорах семейства Itanium (ранее известных под кодовыми названиями Merced, McKinley, Madison и IA-64) компании Intel, будет играть все более важную роль при создании крупных приложений. Существенные отличия между Win32 и Win64 обусловлены различиями в размере указателей (64 бита в Win64) и объеме доступного виртуального адресного пространства.

Проблемы переноса приложений на платформу Win64 обсуждаются по мере изложения материала на протяжении всей книги, а программы организованы таким образом, чтобы создание их в виде приложений Win64 обеспечивалось простым указанием соответствующих параметров на стадии компиляции. В находящихся на Web-хосте книги проектах с программами примеров в необходимых случаях предусмотрен вывод сообщений, предупреждающих о возникновении проблем при переходе к 64 разрядам, но большинство ситуаций (хотя и не пол-

[6] Такие типы, как `PVOID`, входят в `include`-файлы без префикса, но в примерах мы будем придерживаться правил их употребления, принятых во многих книгах и документации Microsoft.

ностью все), которые могли бы приводить к генерации таких сообщений, из программного кода исключены.

С точки зрения программиста основные отличия при переходе к Win64 обусловлены размерами указателей и необходимостью помнить о том, что длины указателя и целочисленной переменной (`LONG`, `DWORD` и так далее) не обязательно должны совпадать. С этой целью определены, например, типы `DWORD32` и `DWORD64`, позволяющие явно управлять размером переменных. Два других типа, `POINTER_32` и `POINTER_64`, позволяют управлять размером указателей.

Как вы сами убедитесь, приложив лишь самые незначительные усилия, можно добиться того, чтобы программы работали как в Win32, так и в Win64, и поэтому мы будем часто ссылаться на API просто как на Windows или, иногда, Win32. Дополнительная информация относительно Win64 содержится в главе 16, где, в частности, обсуждаются вопросы совместимости исходных и двоичных кодов.

Программисты, работающие с UNIX и Linux, столкнутся в Windows с рядом интересных особенностей. Так, в Windows дескрипторы `HANDLE` являются "непрозрачными". Они не представляют собой ряд последовательно возрастающих целых чисел. В то же время, например, в UNIX дескрипторы файлов `0`, `1` и `2` имеют специальное назначение, что должно обязательно учитываться при написании программ. Ничего подобного в Windows вы не обнаружите.

Многие из различий, например грань между идентификаторами процессов и дескрипторами файлов, в Windows оказываются стертыми. В Windows объекты обеих типов описываются дескрипторами типа `HANDLE`. Во многих важных функциях могут наравне использоваться дескрипторы файлов, процессов, событий, каналов и других объектов.

Программистам, которые, работая в UNIX, привыкли к коротким именам функций и параметров и использовали преимущественно строчные буквы, придется приспосабливаться к более пространному стилю Windows. Стиль Windows близок к стилю интерфейса компании Hewlett Packard (ранее — DEC и Compaq); программистам, работающим с OpenVMS, многое покажется знакомым. Указанное сходство между OpenVMS и Windows частично объясняется тем, что Дэвид Катлер (David Cutler), создатель первоначальной архитектуры VMS, предполагал, что она должна играть ту же роль, что и NT или Windows.

Радикальные отличия касаются такого хорошо знакомого нам всем понятия, как процессы. В Windows процессы не обладают свойствами наследования, хотя и могут быть организованы в виде объектов заданий.

В завершение следует отметить, что в текстовых файлах Windows конец строки отмечается последовательностью управляющих символов CR-LF, а не LF, как в это принято в UNIX.

О целесообразности привлечения функций стандартной библиотеки C для обработки файлов

Несмотря на всю уникальность возможностей Windows, старый добрый язык C и его стандартная библиотека ANSI C по-прежнему могут с успехом использоваться при решении большинства задач, связанных с обработкой файлов. Кроме того, библиотека C (указание на ее соответствие стандарту ANSI C мы будем часто опускать) содержит большое число очень нужных функций, аналогов которых среди системных вызовов нет. К их числу относятся, например, функции, описанные в заголовочных файлах `<string.h>`, `<stdlib.h>` и `<signal.h>`, а также функции форматированного и символьного ввода/вывода. В то же время, имеются и такие функции, как `fopen` и `fread`, описанные в заголовочном файле `<stdio.h>`, для которых находятся близко соответствующие им системные вызовы.

В каких же случаях при обработке файлов можно обойтись библиотекой C, а в каких необходимо использовать системные вызовы Windows? Тот же вопрос можно задать и в отношении использования потоков (streams) ввода/вывода C++ или системы ввода/вывода, которая предоставляется платформой .NET. Простых ответов на эти вопросы не существует, но если во главу угла поставить переносимость программ на платформы, отличные от Windows, то в тех случаях, когда приложению требуется только обработка файлов, а не, например, управление процессами или другие специфические возможности Windows, предпочтение следует отдавать библиотеке C и потокам ввода/вывода C++. Вместе с тем, многими программистами ранее уже делались попытки выработать рекомендации относительно адекватности использования библиотеки C в тех или иных случаях, и эти же рекомендации должны быть применимы и в отношении Windows. Кроме того, с учетом возможностей расширения функциональности, а также повышения производительности и гибкости программ, обеспечиваемые Windows, нередко оказывается более удобным или даже необходимым не ограничиваться библиотекой C, в чем вы постепенно станете убеждаться уже начиная с главы 3. К числу возможностей Windows, не поддерживаемых библиотекой C, относятся блокирование и отображение файлов (необходимое для разделения общих областей памяти), асинхронный ввод/вывод, произвольный доступ к файлам чрезвычайно крупных размеров (4 Гбайт и выше) и взаимодействие между процессами.

В случае простых программ вам будет вполне достаточно использовать функции библиотеки C, предназначенные для работы с файлами. Воспользовавшись библиотекой C, можно написать переносимое приложение даже без изучения Windows, однако возможности выбора при этом будут ограниченными. Так, в главе 5 для повышения производительности программы и упрощения программирования применено отображение файлов, однако библиотека C такие возможности не предоставляет.

Что требуется для работы с данной книгой

Ниже перечислено все то, что необходимо вам для создания и выполнения примеров, приведенных в этой и последующих главах книги.

Разумеется, прежде всего, вам потребуется весь ваш опыт в области разработки приложений; предполагается также, что язык C вам знаком. Однако прежде, чем браться за решение упражнений и разбор примеров, вы должны убедиться в том, что располагаете всем необходимым аппаратным и программным обеспечением, перечень которого приводится ниже.

- Система с установленной ОС Windows.
- Компилятор C и любая подходящая среда разработки приложений, например, Microsoft Visual Studio .NET или Microsoft Visual C++ версии 6.0. Имеются также системы разработки приложений от других поставщиков, и хотя примеры из книги нами на них не тестировались, из поступивших от нескольких читателей писем нам стало известно, что примеры, пусть даже после внесения в них незначительных изменений, в некоторых случаях успешно выполнялись даже при использовании других систем. Кроме того, в приложении A содержится информация, касающаяся использования инструментальных средств с открытым исходным кодом. *Примечание.* Наше внимание будет сосредоточено на разработке консольных приложений Windows, и поэтому возможности Microsoft Visual Studio .NET будут задействованы далеко не в полной мере.
- Достаточный для разработки программ объем ОЗУ и наличие свободного места на жестком диске. Практически любая коммерчески доступная система предоставит вам достаточный объем памяти, место на диске и процессорную мощность, которых хватит для запуска примеров и среды разработки приложений, однако предварительно необходимо проверить, какие именно требования к ресурсам предъявляет эта среда.[7]
- Привод компакт-диска, системного или сетевого, для установки среды разработки приложений.
- Оперативная документация наподобие той, которая поставляется вместе с Microsoft Visual C++. Желательно, чтобы вы установили эту документацию

[7] О том, какими быстрыми темпами улучшаются показатели стоимости и производительности, вы можете судить хотя бы по тому факту, что еще в 1997 году в первом издании этой книги автор, без тени смущения или неловкости, в качестве необходимых требований указывал 16 Мбайт ОЗУ и 256 Мбайт свободного места на жестком диске. Для написания настоящего, третьего издания книги используется лэптоп стоимостью менее $1000, с объемом ОЗУ в более чем 10 раз превышающим прежний (что больше ранее требуемого объема дискового пространства), 100-кратной емкостью жесткого диска и 50-кратным превышением быстродействия процессора по сравнению с аналогичными характеристиками компьютера стоимостью $2500, который использовался при подготовке первого издания.

на своем жестком диске, поскольку к ней будет требоваться частый доступ. Дополнительную информацию вы всегда сможете получить на Web-сайте компании Microsoft.

Пример: простое последовательное копирование файла

В следующих разделах приведены примеры коротких программ, реализующих простое последовательное копирование содержимого файла тремя различными способами:

1. С использованием библиотеки C.
2. С использованием Windows.
3. С использованием вспомогательной функции Windows — `CopyFile`.

Кроме того, что эти примеры дают возможность сопоставить между собой различные модели программирования, они также демонстрируют возможности и ограничения, присущие библиотеке C и Windows. Альтернативные варианты реализации усилят программу, увеличивая ее производительность и повышая гибкость.

Последовательная обработка файлов является простейшей, наиболее распространенной и самой важной из возможностей, обеспечиваемых любой операционной системой, и почти в каждой большой программе хотя бы несколько файлов обязательно подвергаются этому виду обработки. Поэтому простая программа обработки файлов предоставляет прекрасную возможность ознакомиться с Windows и принятыми в ней соглашениями.

Копирование файлов, нередко осуществляемое совместно с обновлением их содержимого, и слияние отсортированных файлов являются распространенными формами последовательной обработки файлов. Примерами других приложений, осуществляющих последовательный доступ к файлам, могут служить компиляторы и инструментальные средства, предназначенные для обработки текста.

Несмотря на концептуальную простоту последовательной обработки файлов, эффективная реализация этого процесса, обеспечивающая оптимальную скорость его выполнения, может оказаться нелегкой задачей. Для этого может потребоваться использование перекрывающегося ввода/вывода, отображения файлов, потоков и других дополнительных методов.

Само по себе копирование файлов не представляет особого интереса, однако сравнение программ не только позволит вам быстро оценить, чем отличаются друг от друга различные системы, но и послужит хорошим предлогом для знакомства с Windows. В последующих примерах реализуется ограниченный вариант одной из команд UNIX — `cp`, осуществляющей копирование одного файла в другой и требующей задания имен файлов в командной строке. В приведенных программах организована лишь простейшая проверка ошибок, которые могут возникать на стадии выполнения, а существующие файлы просто перезаписыва-

ются. Эти и другие недостатки будут учтены в последующих Windows-реализациях этой и других программ. *Примечание.* Реализация программы для UNIX находится на Web-сайте книги.

Копирование файлов с использованием стандартной библиотеки C

Как видно из текста программы 1.1, стандартная библиотека C поддерживает объекты потоков ввода/вывода `FILE`, которые напоминают, несмотря на меньшую общность, объекты Windows `HANDLE`, представленные в программе 1.2.

Программа 1.1. `cpC`: копирование файлов с использованием библиотеки C

```
/* Глава 1. Базовая программа копирования файлов cp.
   Реализация, использующая библиотеку C. */
/* cp файл1 файл2: Копировать файл1 в файл2. */
#include <stdio.h>
#include <errno.h>
#define BUF_SIZE 256
int main (int argc, char *argv [])
{
   FILE *in_file, *out_file;
   char rec [BUF_SIZE];
   size_t bytes_in, bytes_out;
   if (argc != 3) {
      printf ("Использование: cpC файл1 файл2\n");
      return 1;
   }
   in_file = fopen (argv [1], "rb");
   if (in_file == NULL) {
      perror (argv [1]);
      return 2;
   }
   out_file = fopen (argv [2], "wb");
   if (out_file == NULL) {
      perror (argv [2]);
      return 3;
   }
   /* Обработать входной файл по одной записи за один раз. */
   while ((bytes_in = fread (rec, 1, BUF_SIZE, in_file)) > 0) {
      bytes_out = fwrite (rec, 1, bytes_in, out_file);
      if (bytes_out != bytes_in) {
         perror ("Неустранимая ошибка записи.");
         return 4;
      }
   }
   fclose (in_file);
   fclose (out_file);
   return 0;
}
```

Этот простой пример может служить наглядной иллюстрацией ряда общепринятых допущений и соглашений программирования, которые не всегда применяются в Windows.

1. Объекты открытых файлов идентифицируются указателями на структуры `FILE` (в UNIX используются целочисленные дескрипторы файлов). Указателю `NULL` соответствует несуществующий объект. По сути, указатели являются разновидностью дескрипторов объектов открытых файлов.
2. В вызове функции `fopen` указывается, каким образом должен обрабатываться файл — как текстовый или как двоичный. В текстовых файлах содержатся специфические для каждой системы последовательности символов, используемых, например, для обозначения конца строки. Во многих системах, включая Windows, в процессе выполнения операций ввода/вывода каждая из таких последовательностей автоматически преобразуется в нулевой символ, который интерпретируется в языке C как метка конца строки, и наоборот. В нашем примере оба файла открываются как двоичные.
3. Диагностика ошибок реализуется с помощью функции `perror`, которая, в свою очередь, получает информацию относительно природы сбоя, возникающего при вызове функции `fopen`, из глобальной переменной `errno`. Вместо этого можно было бы воспользоваться функцией `ferror`, возвращающей код ошибки, ассоциированный не с системой, а с объектом `FILE`.
4. Функции `fread` и `fwrite` возвращают количество обработанных байтов непосредственно, а не через аргумент, что оказывает существенное влияние на логику организации программы. Неотрицательное возвращаемое значение говорит об успешном выполнении операции чтения, тогда как нулевое — о попытке чтения метки конца файла.
5. Функция `fclose` может применяться лишь к объектам типа `FILE` (аналогичное утверждение справедливо и в отношении дескрипторов файлов UNIX).
6. Операции ввода/вывода осуществляются в синхронном режиме, то есть прежде чем программа сможет выполняться дальше, она должна дождаться завершения операции ввода/вывода.
7. Для вывода сообщений об ошибках удобно использовать входящую в библиотеку C функцию ввода/вывода `printf`, которая даже будет использована в первом примере Windows-программы.

Преимуществом реализации, использующей библиотеку C, является ее переносимость на платформы UNIX, Windows, а также другие системы, которые поддерживают стандарт ANSI C. Кроме того, как показано в приложении B, в том, что касается производительности, вариант, использующий функции ввода/вывода библиотеки C, ничуть не уступает другим вариантам реализации. Тем не менее, в этом случае программы вынуждены ограничиваться синхронными операциями ввода/вывода, хотя влияние этого ограничения будет несколько ослаблено использованием потоков Windows (начиная с главы 7).

Как и их эквиваленты в UNIX, программы, основанные на функциях для работы с файлами, входящих в библиотеку C, способны выполнять операции произвольного доступа к файлам (с использованием функции `fseek` или, в случае текстовых файлов, функций `fsetpos` и `fgetpos`), но это является уже потолком сложности для функций ввода/вывода стандартной библиотеки C, выше которого они подняться не могут. Вместе с тем, Visual C++ предоставляет нестандартные расширения, способные, например, поддерживать блокирование файлов. Наконец, библиотека C не позволяет управлять средствами защиты файлов.

Резюмируя, можно сделать вывод, что если простой синхронный файловый или консольный ввод/вывод — это все, что вам надо, то для написания переносимых программ, которые будут выполняться под управлением Windows, следует использовать библиотеку C.

Копирование файлов с использованием Windows

В программе 1.2 решается та же задача копирования файлов, но делается это с помощью Windows API, а базовые приемы, стиль и соглашения, иллюстрируемые этой программой, будут использоваться на протяжении всей этой книги.

Программа 1.2. `cpW`: копирование файлов с использованием Windows, первая реализация

```
/* Глава 1. Базовая программа копирования файлов cp.
   Реализация, использующая Windows. */
/* cpW файл1 файл2: Копировать файл1 в файл2. */
#include <windows.h>
#include <stdio.h>
#define BUF_SIZE 256
int main (int argc, LPTSTR argv [])
{
   HANDLE hIn, hOut;
   DWORD nIn, nOut;
   CHAR Buffer [BUF_SIZE];
   if (argc != 3) {
      printf ("Использование: cpW файл1  файл2\n");
      return 1;
   }
   hIn = CreateFile (argv [1], GENERIC_READ, 0, NULL,
         OPEN_EXISTING, 0, NULL);
   if (hIn == INVALID_HANDLE_VALUE) {
      printf ("Невозможно открыть входной файл. Ошибка: %x\n",
            GetLastError ());
      return 2;
   }
   hOut = CreateFile (argv [2], GENERIC_WRITE, 0, NULL,
         CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
   if (hOut == INVALID_HANDLE_VALUE) {
      printf ("Невозможно открыть выходной файл. Ошибка: %x\n",
            GetLastError ());
```

```
      return 3;
   }
   while (ReadFile (hIn, Buffer, BUF_SIZE, &nIn, NULL) && nIn > 0) {
      WriteFile (hOut, Buffer, nIn, &nOut, NULL);
      if (nIn != nOut) {
         printf ("Неустранимая ошибка записи: %x\n", GetLastError ());
         return 4;
      }
   }
   CloseHandle (hIn);
   CloseHandle (hOut);
   return 0;
}
```

Этот простой пример иллюстрирует некоторые особенности программирования в среде Windows, к подробному рассмотрению которых мы приступим в главе 2.

1. В программу всегда включается файл `<windows.h>`, в котором содержатся все необходимые определения функций и типов данных Windows.[8]
2. Все объекты Windows идентифицируются переменными типа `Handle`, причем для большинства объектов можно использовать одну и ту же общую функцию `CloseHandle`.
3. Рекомендуется закрывать все ранее открытые дескрипторы, если в необходимость в них отпала, чтобы освободить ресурсы. В то же время, при завершении процессов относящиеся к ним дескрипторы автоматически закрываются ОС, и если не остается ни одного дескриптора, ссылающегося на какой-либо объект, то ОС уничтожает этот объект и освобождает соответствующие ресурсы. (*Примечание.* Как правило, файлы подобным способом не уничтожаются.)
4. Windows определяет многочисленные символические константы и флаги. Обычно они имеют длинные имена, нередко поясняющие назначение данного объекта. В качестве типичного примера можно привести имена `INVALID_HANDLE_VALUE` и `GENERIC_READ`.
5. Функции `ReadFile` и `WriteFile` возвращают булевские значения, а не количества обработанных байтов, для передачи которых используются аргументы функций. Это определенным образом изменяет логику организации работы циклов.[9] Нулевое значение счетчика байтов указывает на попытку чтения метки конца файла и не считается ошибкой.

[8] В приложении А показано, как исключить ненужные определения для ускорения компиляции и экономии дискового пространства.

[9] Обратите внимание на то, что логика цикла зависит от принятого в стандарте ANSI C порядка вычисления логических операций “и” (&&) и “или” (||) в направлении слева направо.

6. Функция `GetLastError` позволяет получать в любой точке программы коды системных ошибок, представляемые значениями типа `DWORD`. В программе 1.2 показано, как организовать вывод генерируемых Windows текстовых сообщений об ошибках.
7. Windows NT обладает более мощной системой защиты файлов, описанной в главе 15. В данном примере защита выходного файла не обеспечивается.
8. Такие функции, как `CreateFile`, обладают богатым набором дополнительных параметров, но в данном примере использованы значения по умолчанию.

Копирование файлов с использованием вспомогательной функции Windows

Для повышения удобства работы в Windows предусмотрено множество вспомогательных функций (convenience functions), которые, объединяя в себе несколько других функций, обеспечивают выполнение часто встречающихся задач программирования. В некоторых случаях использование этих функций может приводить к повышению производительности (см. приложение В). Например, благодаря применению функции `CopyFile` значительно упрощается программа копирования файлов (программа 1.3). Помимо всего прочего, это избавляет нас от необходимости заботиться о буфере, размер которого в двух предыдущих программах произвольно устанавливался равным 256.

Программа 1.3. `cpCF`: копирование файлов с использованием вспомогательной функции Windows

```
/* Глава 1. Базовая программа копирования файлов cp.
   Реализация, в которой для повышения удобства использования и
   производительности программы используется функция Windows CopyFile. */
/* cpCF файл1 файл2: Копировать файл1 в файл2. */

#include <windows.h>
#include <stdio.h>

int main (int argc, LPTSTR argv [])
{
  if (argc != 3) {
    printf ("Использование: cpCF файл1 файл2\n");
    return 1;
  }
  if (!CopyFile (argv [1], argv [2], FALSE)) {
    printf ("Ошибка при выполнении функции CopyFile: %x\n", GetLastError ());
    return 2;
  }
  return 0;
}
```

Резюме

Ознакомительные примеры, в качестве которых были использованы три простые программы копирования файлов, демонстрируют многие из отличий, существующих между программами, в которых применяется с одной стороны библиотека C, а с другой — Windows. Отличия в производительности различных вариантов реализации анализируются в приложении B. Примеры, в которых используется Windows, наглядно демонстрируют стиль программирования и некоторые соглашения, принятые в Windows, но дают лишь отдаленное представление о тех функциональных возможностях, которые Windows предлагает программистам.

Целевыми платформами для данной книги и содержащихся в ней примеров являются системы NT5 (Windows XP, 2000 и Server 2003). Тем не менее, большая часть материала книги применима также к ранним версиям NT и системам Windows 9x (95, 98 и Me).

В следующих главах

Главы 2 и 3 посвящены гораздо более пристальному рассмотрению функций ввода/вывода и файловой системы. Они включают в себя такие темы, как консольный ввод/вывод, обработка символов ASCII и Unicode, работа с файлами и каталогами, а также программирование реестра. В указанных главах разрабатываются базовые методики и закладывается фундамент для остальной части книги.

Дополнительная литература

Полная информация о рекомендуемых ниже книгах приведена в библиографическом списке в конце книги.

Win32

Двумя доступными в настоящее время книгами, в которых вопросы программирования для Windows рассматриваются с всех возможных точек зрения, являются [5] и [31]. В то же время, существует множество других книг, которые не обновлялись и не отражают прогресс, достигнутый с момента выхода Windows 95 или Windows NT.

По каждой функции Microsoft Visual C++ имеется оперативная гипертекстовая справочная документация, но ту же информацию можно получить, посетив домашнюю страницу компании Microsoft — `http://www.microsoft.com`, где вы найдете целый ряд ссылок на технические статьи, посвященные различным аспектам Windows. Начните с раздела MSDN (Microsoft Developer's Network) и произведите поиск по любой интересующей вас теме. Вы обнаружите огромное разнообразие официальной документации, описаний продуктов, примеров программного кода, а также другую полезную информацию.

Win 64

Win64 обсуждается в нескольких книгах, но обширный материал по этой теме можно найти на домашней странице компании Microsoft.

Архитектура Windows NT и история ее развития

Читателям, которые хотят больше узнать о целях проектирования Windows NT или понять основные принципы, лежащие в основе ее архитектуры, будет полезна книга [38]. В этой книге рассматриваются объекты, процессы, потоки, виртуальная память, ядро и подсистемы ввода/вывода. Вместе с тем, собственно функции API, а также Windows 9x и CE в ней не обсуждаются. Рекомендуем время от времени заглядывать в упомянутую книгу для получения дополнительной информации. Кроме того, обратитесь к ранее вышедшим книгам [9] и [37], в которых содержится важный ретроспективный анализ эволюции NT.

UNIX

В книге [40], написанной ныне покойным Уильямом Ричардом Стивенсом (W. Richard Stevens), UNIX обсуждается во многом в тех же терминах, которые в настоящей книге используются для обсуждения Windows. Книга Стивенса попрежнему остается стандартным справочником по средствам UNIX, но в ней не рассматриваются потоки. Стандартизация UNIX претерпела изменения, однако в книге Стивенса содержатся удобные рабочие определения всего того, что предлагается в UNIX, а также в Linux. В этой книге сопоставлены возможности функций файлового ввода/вывода библиотеки C и функций ввода/вывода системы UNIX, что имеет отношение и к Windows.

Если вас интересуют сравнительные характеристики ОС и более глубокое обсуждение UNIX, обратитесь к книге [29] и ее русскоязычному изданию [49], которая помимо того, что является весьма полезной, еще и увлекательно написана, хотя некоторым читателям позиция автора может показаться несколько предвзятой.

Программирование с использованием Windows GUI

Пользовательский интерфейс в настоящей книге не рассматривается. В случае необходимости можете обратиться к [30] или [25].

Теория операционных систем

Существует масса хороших учебников по общей теории ОС. Одной из наиболее популярных является книга [35].

Стандартная библиотека ANSI C

Исчерпывающим руководством по этой теме служит книга [27]. Для получения беглого обзора можно обратиться к книге [20] или к ее русскоязычному изданию [48], которая содержит полное описание библиотеки и остается классическим учебником по языку программирования C. Эти книги помогут вам принять решение относительно того, достаточно ли возможностей библиотеки C для решения стоящих перед вами задач обработки файлов.

Windows CE

Тем, кто хочет применить материал настоящей книги к Windows CE, можно порекомендовать книгу [23].

Эмуляция Windows в UNIX

Для получения необходимой информации по этому вопросу и загрузки пакета с открытым исходным кодом Wine, позволяющего эмулировать Windows API поверх UNIX и X, посетите сайт `http://www.winehq.com`.

Упражнения

1.1. Скомпилируйте, скомпонуйте и выполните каждую из трех программ, предназначенных для копирования файлов. К числу других возможных вариантов реализации относится использование библиотек совместимости с UNIX, включая библиотеку Microsoft Visual C++ (программа, использующая эту библиотеку, доступна на Web-сайте книги). *Примечание.* На Web-сайте книги находятся исходные коды всех программ. Краткие рекомендации относительно порядка использования этих кодов в средах Microsoft Visual Studio .NET и Microsoft Visual C++ 6.0 вы найдете в приложении A.

1.2. Ознакомьтесь с одной из сред разработки приложений, например, Microsoft Visual Studio .NET или Microsoft Visual C++. В частности, научитесь создавать в выбранной среде консольные приложения. Для проведения самостоятельных экспериментов с использованием рассмотренных в данной главе программ пользуйтесь отладчиком. Инструкции относительно того, как следует приступать к работе, содержатся в приложении A, а обширную дополнительную информацию вы найдете на Web-сайте компании Microsoft и в документации к используемой вами среде разработки приложений.

1.3. В Windows в качестве метки конца строки используется последовательность символов "возврат каретки–перевод строки" (CR–LF). Определите, как изменится поведение программы 1.1, если входной файл открывать в двоичном режиме, а выходной — в текстовом, или наоборот. Как это будет проявляться в системе UNIX и в других системах?

1.4. Выполните для каждой из программ хронометраж при копировании файлов большого размера. Получите соответствующие данные для как можно большего числа различных вариантов и сравните полученные результаты между собой. Вряд ли следует подчеркивать, что быстродействие программ зависит от множества факторов, однако, в предположении, что все остальные параметры системы остаются неизменными, сопоставление результатов, полученных с использованием различных вариантов реализации программы, может представлять определенную ценность. *Совет.* Для облегчения анализа результатов расположите их в виде таблицы. Программа, обеспечивающая количественный контроль длительности временных промежутков, приведена в главе 6, а некоторые экспериментальные результаты представлены в приложении B.