1. Общие сведения об управлении ресурсами.

**Физическими ресурсами** считается центральный процессор, оперативная память, внешняя память, шины передачи данных, устройства ввода вывода.

**Логическими ресурсами (информационными)** считаются данные и программы для их обработки.

Все вместе принято называть **системными ресурсами.**

Для выполнения программы необходимо чтобы она имела доступ к системным ресурсам. Этот доступ обеспечивает ОС. Для этого она выстраивает следующую схему

Пользователи > прикладное ПО > системное ПО > аппаратное обеспечение.

Системное программирование тесно связано с ОС и их интерфейсами для разработки программ.

ОС обеспечивает работу прикладных программ, предоставляя им сервисные функции и организуя доступ к ресурсам

Наиболее важными подсистемами управления ресурсами являются подсистемы управления процессами, памятью, файлами и внешними устройствами. Подсистемы пользовательского интерфейса, защиты данных и администрирования являются общими для всех ресурсов.

2. Подсистема управление процессами.

Это подсистема непосредственно влияет на функционирование всей вычислительной системы. Для каждой выполняемой программы ОС организует необходимое количество процессов. Каждый процесс представляется информационной структурой (таблица, дескриптор, контекст процесса). Эта структура содержит данные о потребностях процесса в ресурсах и об уже выделенных ему ресурсах. Ещё в ней хранятся данные о текущем состоянии процесса, приоритетах, состояний регистров и т.п.

В мультипрограммных ОС одновременно существуют несколько процессов, порожденных пользователем, прикладными программами и системных процессов. Все они могут одновременно претендовать на одни и те же ресурсы, поэтому подсистема управления процессами планирует очередность их выполнения, обеспечение их ресурсами, гарантирует их взаимодействие и синхронизацию.

3. Подсистема управления памяти.

Это подсистема распределения физической памяти между процессами, загрузку и удаление программных кодов процессов и их данных в отведенные области памяти, настройку программных адресов на физические адреса выделенной области, защиту областей памяти каждого процесса.

Стратегия управления памятью складывается из стратегий выборки, размещения и замещения блока программы или данных в основной памяти соответственно используются различные алгоритмы, определяющие когда загрузить очередной блок в память, в какое место памяти его загрузить, какой блок удалить из основной памяти.

Одним из наиболее применяемых способов управления памятью является виртуальная память. Ее реализация позволяет программисту использовать однородную оперативную память, объем которой ограничивается только возможностями адресации в системе программирования.

Важная функция управления памятью это ее защита, нарушение защиты связаны с обращениями процессов к участкам памяти, выделенным для других процессов. Средство защиты памяти должны пресекать такие нарушения путем аварийного завершения программы-нарушителя.

4. Подсистемы управления файлами и внешними устройствами.

Функции управления файлами сосредоточено в файловой системе. ОС виртуализирует отдельный набор данных в виде файла, т.е. простой не структурированный последовательности байтов, имеющей символьное имя. Для удобства работы с данными файлы группируются в каталоги, которые так же могут группироваться в каталоги высокого уровня и т.д. Файловая система преобразует символьные имена файлов пользователя в физические адреса данных на внешнем носителе, организует совместный доступ к файлам и защищает их от несанкционированного доступа.

Подсистемой управления внешними устройствами чаще называют подсистемой ввода и вывода. Она является интерфейсом между ядром компьютера и периферийными устройствами. Программа, управляющая конкретной моделью внешнего устройства, называется драйвером. Созданием драйверов занимаются и разработчики ОС, и производители внешних устройств. ОС должна поддерживать строго определенный интерфейс между драйверами и остальными блоками ОС тогда производители внешних устройств могут поставлять вместе со своими изделиями и драйверы конкретных ОС.

5. Защита данных и администрирование.

Безопасность данных обеспечивается средствами отказоустойчивости ОС, направленными на защиту от сбоев и отказов аппаратуры и ошибок ПО, а также средствами защиты от несанкционированного доступа. Для каждого пользователя системы обязательна процедура логического входа, которая подтверждает, что этот пользователь разрешен системным администратором. Системный администратор устанавливает права пользователей по обращению и использованию системных ресурсов.

Важным средством защиты является функция аудита ОС, то есть в фиксации всех событий, от которых зависит безопасность системы. Поддержка отказоустойчивости реализуется на основе резервирования (дисковые RAID-массивы, резервные устройства, зеркалирование серверов…). Ответственным за обеспечение отказоустойчивости системы является системный администратор. Для этого у него имеются специальные средства и инструменты.

6. Интерфейс прикладного программирования.

Прикладные программисты используют в своих приложениях обращения к ОС тогда, когда для выполнения некоторых действий требуется особый доступ, которым обладает только ОС.

Возможности ОС доступны программисту в виде наборов функций, которые называются API. Приложения обращаются к функциям API с помощью системных вызовов. API предоставляет стандартные блоки, которые используются прикладными программами, написанными для Windows, при этом API позволяет абстрагироваться от того, как реализованы эти функции. Программные компоненты взаимодействуют друг с другом посредством API, при этом обычно компоненты образуют иерархию, т.е. высокоуровневые компоненты используют API низкоуровневых компонентов, а те в свою очередь используют API еще более низкоуровневых компонентов и т.д.

Работа через WinAPI – это наиболее близкий к ОС способ взаимодействия с ней прикладных программ, более низкий уровень требуется только для драйверов устройтсв.

Интерфейс прикладного программирования – это множество системных функций и системных вызовов, которые представляются прикладным программам через WinAPI.

Приложениям доступны различные службы:

1. Базовые – управление процессами и памятью, функции ввода/вывода и безопасности.

2. Компонентов для взаимодействия приложений.

3. Пользовательского интерфейса.

4. Графиков и мультимедиа.

5. Обмена сообщениями и программами.

6. Сетевые службы.

7. Веб службы.

Центральным понятием программирования в среде Windows является сообщение. Система посылает сообщение приложению. Приложение должно правильно на него отреагировать. Получателями сообщения являются функции окон приложения, на программирование которых уходит больше всего времени. Стандартные API гарантируют, что все программы, использующие общий API будут иметь схожий интерфейс.

WinAPI изначально спроектировали для использования в программах, написанных на C, C++ - т.е. это множество функций, структур данных и числовых констант, следующих соглашений этих языков. Поэтому все языки программирования, способные обрабатывать такие функции и данные могут пользоваться этим API. WinAPI может использоваться во всех базирующихся на Windows прикладных программах.

7. Подсистема ввода/вывода.

Это подсистема, которая выполняет обмен данными между пользователем, приложениями и периферийными устройствами. Основными ее компонентами являются драйверы и файловая система. В ее работе активно участвуют диспетчер прерываний. Она выполняет следующие основные функции:

1. Организация параллельной работы устройств ввода/вывода и процессора.

2. Согласование скоростей обмена и кеширование данных.

3. Разделение устройств и данных между процессами.

4. Обеспечение удобного логического интерфейса между устройствами и остальной частью системы.

5. Поддержка драйверов.

6. Динамическая загрузка и выгрузка драйверов (без дополнительных действий с ОС).

7. Поддержка нескольких различных файловых систем.

8. Поддержка синхронных и асинхронных операций ввода/вывода.

Синхронный режим означает, что программный модуль приостанавливает свою работу до тех пор, пока не завершиться операция ввода/вывода.

При асинхронном режиме программный модуль продолжает выполняться в мультипрограммном режиме одновременно с операцией ввода/вывода.

8. Управление файлами.

Логической записью называется упорядоченное множество данных разных типов. Порядок следования этих данных называется структурой записи. На уровне прикладной программы файл - это множество логических записей. На физическом уровне файл - это поименованное множество секторов или кластеров, хранящихся на диске. Часть ОС, которая обеспечивает доступ к файлам и выполняет связывание между логическими записями файлов и их физическим предоставлением называется системой управления файлами или файловой системой. Для выполнения операции доступа к логическим записям файлов с каждым файлом связывают указатель файла, который показывает на текущую логическую запись файла. После каждой операции записи или чтения файловая система передвигает указатель файла на следующую логическую запись.

Буфером ввода/вывода называется область оперативной памяти, предназначенная для временного хранения логических записей файла. Они предназначены для решения двух задач:

1. Устранение несоответствия между размером логической записью файла и размером кластера, который записывается на диск.

2. Снижение влияния внешних устройств на скорость работы процессора.

Для решения этих задач, при выводе данных файловая система сначала полностью заполняет буфер логическими записями, а затем дает команду для записи данных на диск.

Кеширование ввода данных подразумевает, что система выполняет упреждающее чтение данных с диска без ожидания следующей команды на чтение.

9. Объекты в операционной системы.

Объектом в ОС называется структура данных представляющая системный ресурс. ОС Windows представляет 3 категории объектов:

1. Это объекты, которые используются приложением для интерфейса с пользователем.

2. Graphics Device interfase – это объекты которые используются для вывода информации на графические устройства

3. Kaernale – это объекты ядра ОС

**Ядро** – это часть ОС, обеспечивающая приложениям контролируемый доступ к ОС, ещё оно предоставляет сервисы в памяти системы и сетевых протоколов.

Под доступом к объектам понимается возможность приложения выполнять над ними некоторую функцию. Для этого в Windows каждому объекту ставится в соответствие его **дескриптор** – это запись в таблице, которая поддерживается системой и содержит адрес объекта и средства для идентификации его типа. Дескриптор создаётся в системе с использованием функции WinAPI и обслуживаются ими же. Как правило, эти функции имеют вид create объект, где слово объект заменяется именем конкретного объекта, а не возвращает дескриптор созданного процесса. Если это значение не равно null или отрицательно значения, то объект создан успешно. После завершения работы с объектом его дескриптор нужно закрыть.

Bool CloseHandle(handle hObject).

При успешном завершении она вернет не нулевое значение, при не успешном - false.

10. Процессы.

**Процессом** называется исполняемый на компьютере код приложения вместе со всеми ресурсами, которые требуются для его выполнения. Ресурс, необходимый для исполнения процесса, называется **контекстом процесса**. Задачей ОС является организация рационального использования ресурсов процессов, наиболее эффективного выполнения процессов. Для этого ОС должна знать о текущем состоянии каждого процесса и каждого ресурса. Универсальный подход предоставления этой информации – создание и поддержка таблиц с данными по каждому процессу и каждому ресурсу. Наибольшие сложности при этом возникают в мульти программных ОС.

**Мультипрограммирование** - это способ организации вычислительного процесса при котором на одном процессоре попеременно выполняется несколько программ. Для поддержки мультипрограммирования ОС должна определить внутренние единицы работы, между которыми будут разделяться ресурсы компьютера.

В Windows существует такая единица работы, как **задание** – набор из одного или нескольких процессов. Процессы можно рассматривать как заявки для всех видов ресурсов, кроме процессорного времени. Оно распределяется между другими единицами работы – **потоками**, т.е. последовательностью команд. Каждый процесс владеет следующими ресурсами:

1. Виртуальным адресным пространством.

2. Рабочим множеством страниц в реальной памяти.

3. Маркером доступа, который содержит информацию для системы.

4. Таблицей для хранения дескрипторов объекта ядра.

5. Мультипроцессорная обработка (мультипроцессирование) - одновременное выполнение нескольких задач на нескольких процессорах, входящих в одну вычислительную систему.

11. Потоки.

Понятию поток соответствует последовательный переход процессора от одной команды к другой. Процессу назначают адресное пространство и набор ресурсов, которые совместно используются всеми его потоками. Это означает, что каждый поток может иметь доступ к любому виртуальному адресу и может задействовать стек другого потока в пределах одного процесса, т.е. между потоками одного процесса нет полной защиты, поэтому чтобы организовать взаимодействие и обмен данными между ними достаточно использовать общую память – один поток записывает в неё данные, а другой читает. С другой стороны потоки разных процессов хорошо изолированы друг от друга. Таким образом, мультипрограммирование более эффективно на уровне потоков, а не процессов. Если задача выполняется в многопроцессорной системе, то эффект многопоточной обработки увеличивается многократно.

Каждому потоку принадлежат следующие ресурсы: код исполняемой функции; набор регистров процессора; стек для работы приложений; стек для работы с ОС; маркер доступа, который содержит информацию для системы безопасности.

В ОС Windows различаются потоки 2х типов:

1. Системный – выполняет сервисы ОС и запускается её ядро.

2. Пользовательский – служит для решения задач пользователя и запускаются приложением.

В выполняемом приложении различаются рабочие потоки и потоки интерфейса пользователя. Рабочие потоки выполняют фоновые задачи в приложении. Потоки интерфейса пользователя связаны с окнами приложения и выполняют обработку приложений поступающих этим окнам.

Каждое приложение создаёт как минимум один поток, который называется первичным или главным. При корректной организации потоки имеют следующие преимущества перед процессами.

Первое преимущество - им требуется меньше времени для создания нового потока.

Второе преимущество - затрачивается меньше времени для изменения потока.

Третье преимущество - им требуется меньше времени для переключения между двумя потоками в пределах одного процесса.

Четвертое - им требуется меньше времени для коммуникационных затрат.

12. Параллельная обработка потоков.

Один из способов, обеспечивающих взаимодействие приложений и пользователей в процессе обработки, этим приложением, других событий является использование нескольких протоков программы. Это может значительно повысить производительность работы, но создаёт необходимость следить за несколькими потоками. Существование нескольких потоков позволяет процессу разделять работу для параллельного выполнения, основная трудность при этом синхронизация одновременно работающих потоков. Она используется в следующих случаях:

1. Для явного управления порядка управления кода.

2. Для предотвращения проблем использования одного ресурса одновременно двумя потоками.

Проблемы синхронизации являются основной причиной ошибок в многопоточных приложениях. Они приводят к состоянию, которое называется взаимоблокировкой. Переключение потоков занимает довольно много времени. Большие затраты процессорного времени требуются на планирование и диспетчеризацию потоков. Для предоставления сильно облегченного псевдо-параллелизма, начиная с Windows 200 и в последующих версиях используются волокна. Они подобны потокам, но планируются в пространстве пользователя, создавшей их программы. У каждого потока может быть несколько волокон. Когда одно волокно логически блокируется, оно помещается в область блокированных волокон, а для работы выбирается другое волокно того же потока.

Содержимое памяти, к которой поток имеет доступ, называется контекстом потока. Время работы процессора делится на кванты, которые выделяются потокам для работы. Распределением квантов времени между потоками занимается менеджер потоков. Для переключения процессора на другой поток он должен:

1. Сохранить контекст прерываемого потока.

2. Восстановить контекст запускаемого потока на момент его прерывания.

3. Передать управление запускаемому потоку.

Контекст потока полностью определяется содержимым регистров процессора в заданный момент времени.

13. Планирование выполнения.

Правила обслуживания потоков определяют порядок выделения потокам ресурсам компьютера. Для этого могут использоваться различные алгоритмы. Эти алгоритмы называются планированием. Существует много алгоритмов планирования. Их можно разделить на группы:

**1. Вытесняющие/не вытесняющие алгоритмы.** При использовании вытесняющего алгоритма ОС в любой момент времени может прервать выполнение текущего потока и переключить процессор на другой поток. В не вытесняющих алгоритмах только сам текущий поток определяет когда ему освобождать процессор.

**2. Алгоритмы с квантованием.**

**3. Алгоритмы с приоритетами.** Каждому потоку присваивается приоритет – целое число. Каждый раз, когда освобождается процессор ОС отдает его потоку с наивысшим приоритетом.

Алгоритм, устанавливающий порядок, в котором процесс обслуживает потоки, называется диспетчеризацией.

14. Состояния потока в системе.

За время своего существования поток может находиться в нескольких состояниях:

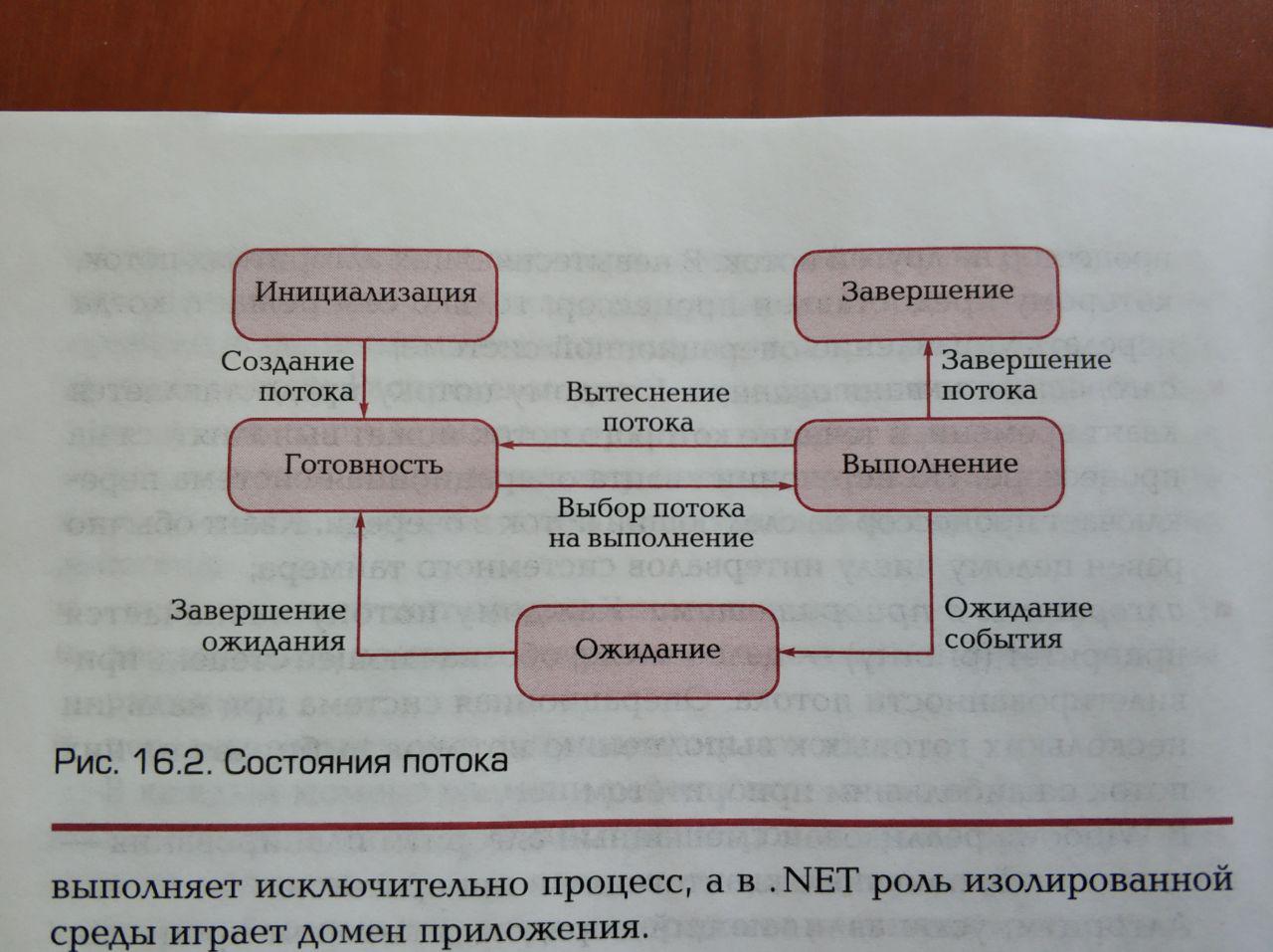
1. Готовность – готов к выполнению и ждет своей очереди в процессор.

2. Выполнение – выполняется в процессоре.

3. Ожидание – не может выполняться, т.к. ждет наступление некоторого события.

Кроме основных есть еще несколько состояний: инициализация, завершение, переходное состояние, состояние отложенной готовности.

Основные состояния потока, возможные переходы между состояниями и условия переходов показаны на следующей схеме:



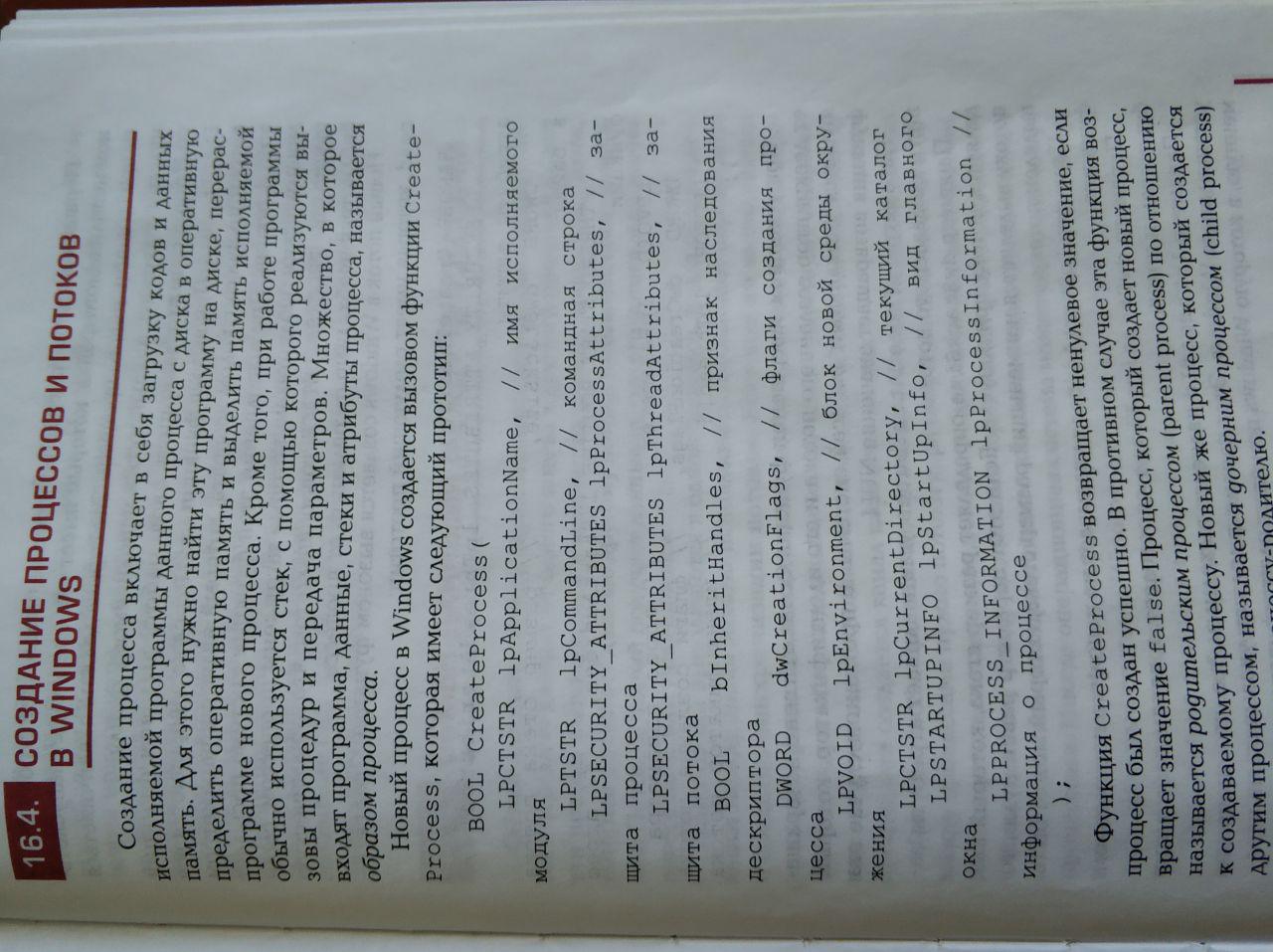
В программной платформе . NET Framework реализован механизм, называемый доменом приложения. Он позволяет запустить группу приложений в одном процессе. При этом обеспечивает их относительную изоляцию друг от друга, но позволяет им взаимодействовать друг с другом значительно быстрее, чем в случае отдельных процессов.

.NET создаёт для каждого приложения отдельный домен в рамках одного процесса. Этот домен изолирует свои ресурсы от других доменов того же процесса. Для обмена данными между доменами используется специальный протокол удаленного взаимодействия .NET. Что касается потоков, то каждый из них не привязан к одному домену приложения в течении своего жизненного цикла, они могут пересекать границы домена, подчиняясь правилам потоков и API, но при этом в любой конкретный момент времени может выполняться в рамках только одного домена. Цель всего этого - улучшить реакцию приложения на действия пользователя, не снижая производительности компьютера.

15. Создание процессов в Windows.

Создание процесса включает в себя загрузку кода и данных исполняемой программы данного процесса из внешней памяти в оперативную. Для этого нужно найти программу во внешней памяти, перераспределить оперативную память и выделить оперативную память исполняемой программе нового процесса. Кроме того, при вызове процедур и передаче параметров используется стек. Все это вместе (программа, данные, стеки и атрибуты процесса) называется образом процесса.

Новый процесс в Windows создаётся вызовом функции CreateProcess.



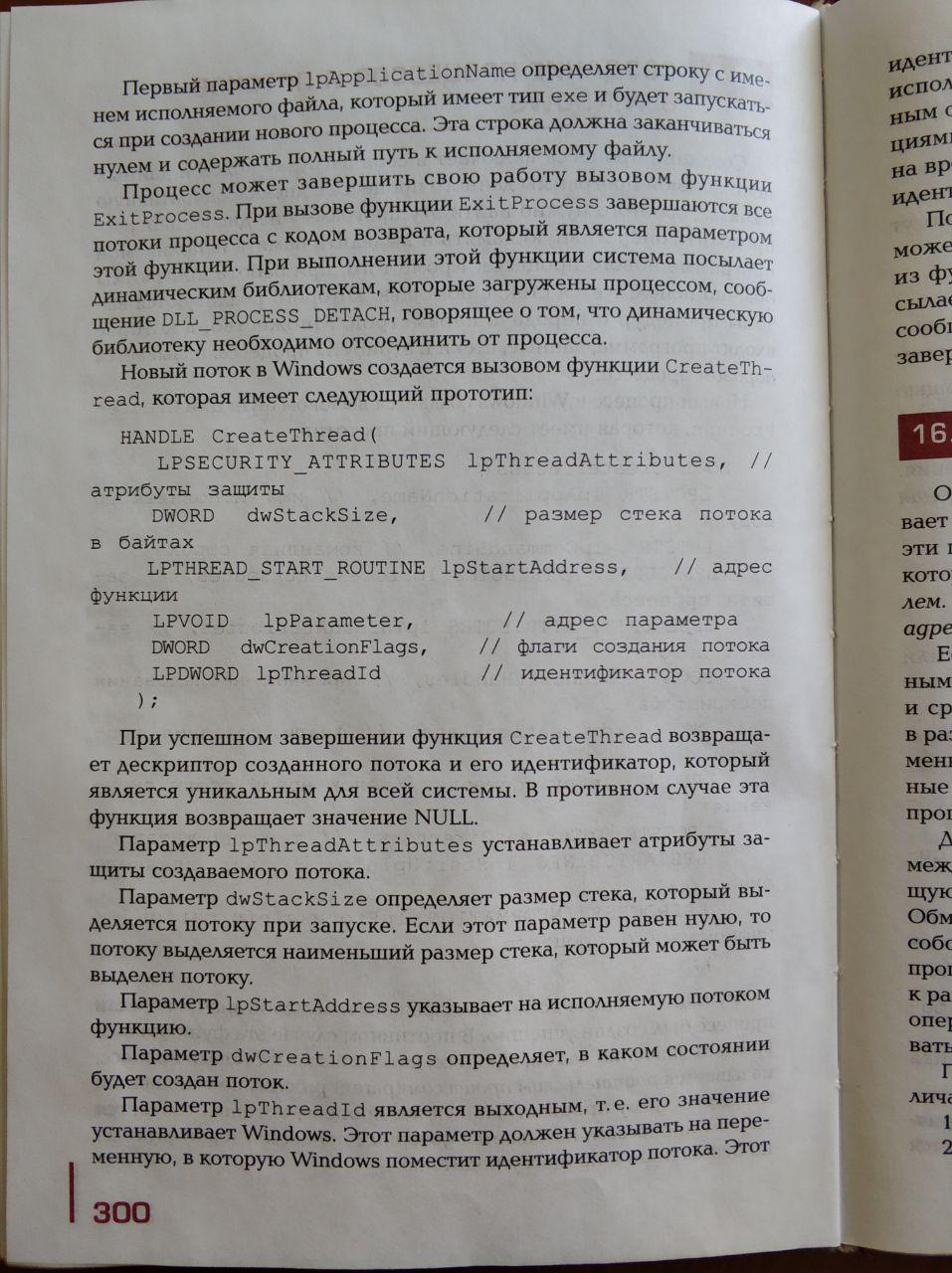
Эта функция возвращает ненулевое значение, если процесс создан успешно. В случае неудачи возвращаться значение False. Процесс, который создает новый процесс, называется родительским по отношению к нему. Новый же процесс называется дочерним. Первый параметр определяет полное имя .exe файла, который будет запускаться и эта строка должна заканчиваться нулем.

Процесс может завершить свою работу вызовом функции ExitProcess. При этом завершаются все потоки процесса. Система посылает динамическим библиотекам, которые загружены этим процессом, сообщение об отсоединении от него.

DLL\_PROCESS\_DETACH

16. Создание потоков в Windows.

Новый поток создаётся вызовом следующей функции:



Завершается поток функцией ExitThread, эта функция может быть вызвана явно или не явно - при возврате значения из функции потока. При этом система сообщает динамическим библиотекам, что поток завершён.

17. Обмен данными между процессами.

Такой обмен подразумевает передачу данных от одного потока к другому предполагая, что эти потоки выполняются в разных процессах. Поток, который посылает данные называется отправителем. Поток, который получает данные называется адресатом или получателем. Для такого обмена ОС создаёт канал передачи данных. Для этого нужно потоки ядра ОС и общую память, используемую для обмена данных заменить файлом. Такой общий файл является простейшим каналом передачи данных между процессами. Сама передача может осуществляться одним из двух способов:

1. Потоком - то есть непрерывной последовательностью байтов.

2. Сообщением - то есть группами байтов.

Прежде чем передавать данные между процессами нужно установить связь между ними. Связь может быть установлена как на аппаратном, так и на программном уровне. С точки зрения направления передачи связь может быть:

1. Полудуплексная - то есть в любой момент времени данные могут передаваться только в одном направлении.

2. Дуплексная – то есть данные могут одновременно передаваться в обоих направлениях.

18. Передача сообщений.

Обмен сообщениями между процессами выполняется с помощью двух функций:

1. Send – передать сообщение.

2. Receive – получить сообщение.

Сообщение состоит из двух частей –тела сообщения, содержащее само сообщение, и заголовка, содержащего:

1. Тип сообщения.

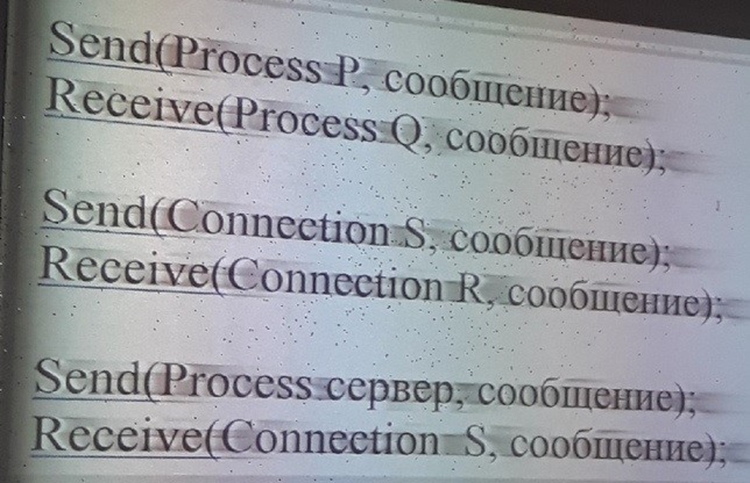
2. Имя адресата.

3. Имя отправителя.

4. Длина.

При передаче сообщений может использоваться прямая или косвенная адресация процесса. При прямой передачи, функция Send\Receive явно указывает процесс отправитель и адресат. При косвенной адресации вместо адресов указывается имя связи, по которой передаётся сообщение.

Адресация может быть симметричной и асимметричной. Если при обмене сообщениями между процессами используется только прямая или только косвенная адресация, то она называется симметричной. Если же при обмене используется и прямая и косвенная адресация, то она называется асимметричной.



Асимметричная используется в системах клиент сервер. Клиент знает адрес сервера и посылает ему сообщение с прямой адресацией. Сервер проверяет канал связи и принимает сообщение от всех клиентов используя косвенную адресацию.

Протокол - набор правил, по которым устанавливаются связи и передаются данные между процессами.

Буфер – это вместимость связи между процессами, то есть количество сообщений, которые могут одновременно пересылаться по этой связи.

19. Анонимные каналы.

Анонимным каналом – это объект ядра ОС, который обеспечивает передачу данных между процессами на одном компьютере. Процесс, который создаёт анонимный канал называется его сервером. Процессы, которые связываются с анонимным каналом, называются его клиентами. Доступ к такому каналу имеют только родительский процесс-сервер и дочерние процессы-клиенты этого канала.

По анонимному каналу можно передавать данные только в одном направлении, но каждый его конец имеет свой дескриптор. Этот дескриптор можно передать любому дочернему процессу, поэтому каждый из этих процессов может как записывать данные в канал, так и читать.

Порядок работы с анонимными каналами, следующий:

1. Создание канала сервером.

2. Соединение клиентов с каналом.

3. Обмен данными по каналу.

4. Закрытие канала.

Так как анонимные каналы не имеют имени, то для соединения процесса-клиента с ним необходимо передать клиенту один из дескрипторов канала. При этом процесс-клиент должен быть дочерним процесса-сервера канала, а дескриптор должен быть наследуемым.

Анонимные каналы часто используются для перенаправления стандартного ввода\вывода.

20. Именованные каналы.

**Именованный канал** — это объект ядра ОС, который обеспечивает передачу данных между процессами, выполняющихся в одной локальной сети. Процесс, который создаёт Именованный канал называется его сервером. Процессы, которые связываются с именованным каналом, называются его клиентами.

**Характеристики именованных каналов:**

1. Они могут быть полудуплексными и дуплексными.

2. Передача данных может быть как потоком, так и сообщением.

3. Обмен данных может быть синхронным и асинхронным.

4. Имеется возможность моделирования любой топологии связи.

Порядок работы с именованными каналами:

1. Создание канала сервером.

2. Соединение сервера с экземпляром канала.

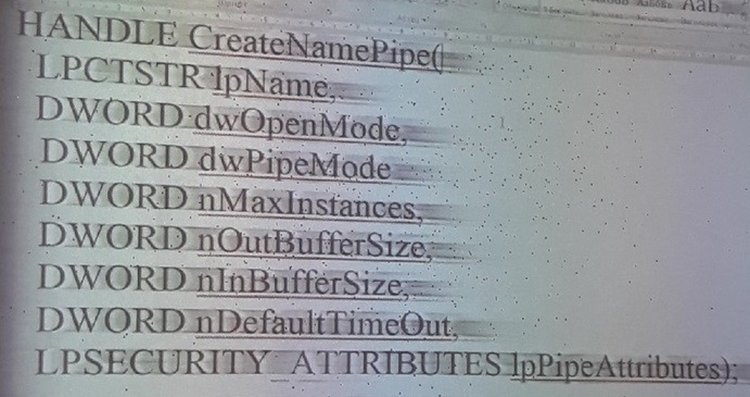
3. Соединение клиента с экземпляром канала.

4. Обмен данными по каналу.

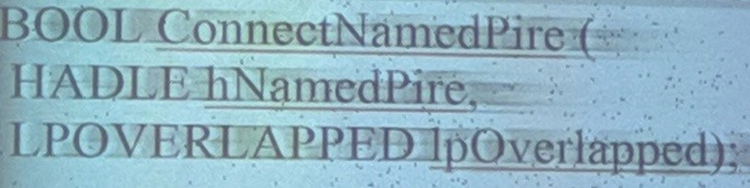
5. Отсоединение сервера от экземпляра канала.

6. Закрытие канала клиентом и сервером.

Именованные каналы создаются процессом-сервером с помощью функции:

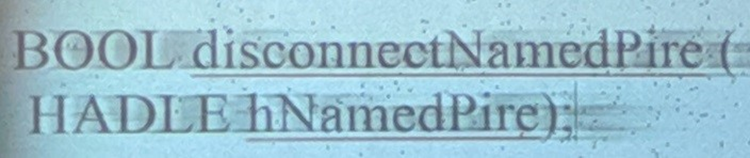


При удачном завершении функция возвращает дескриптор именованного канала. После того как сервер создал канал, он должен дождаться соединения клиента с этим каналом, для этого сервер вызывает функцию:



При успешном завершении эта функция возвращает ненулевое значение, а при неудаче False. Сервер использует эту функцию для связи по каждому свободному экземпляру канала.

После окончании обмена с клиентом сервер разрывает связь с ним использую функцию:



После этого сервер снова может вызвать функцию ConnectNamedPipe, чтобы установить связь по этому же каналу с другим клиентом. Для обмена данными по каналу, используется функция чтения и записи файла.

21. Сетевое программирование сокетов.

Сокеты - программный интерфейс для обеспечения обмена данными между процессами. Процессы могут исполнятся как на одной, так и на различных ЭВМ, связанных между собой сетью. Каждый сокет имеет тип и ассоциированный с ним процесс.

В сети TCPip каждый узел имеет уникальный IP адрес для обмена данными с другими узлами. Каждый посылаемый пакет имеет адрес отправителя и адрес получателя. Это позволяет его однозначно идентифицировать, но, если на компьютере выполняется несколько приложений одновременно использующих сеть, то только адресов не достаточно. Для решения этой неоднозначности каждое соединение с каждой стороны имеет идентификатор под названием порт, являющийся целым числом. Образуется пара адрес и порт, которая представляет собой сокет-канал. Только одно приложение на одном компьютере в одно и то же время может использовать конкретный порт исключения составляют серверные части, для которых возможно создание нескольких сокетов на одном порту для работы с несколькими клиентами.

Механизмы работы с сокетами:

На серверной стороне запускается серверный сокет, который сразу переходит в режим прослушивания. Обычно клиент явно подсоединяется к сокету, на стороне клиента создаётся сокет, для которого указывается IP адрес и порт сервера и даётся команда на соединение, когда сервер получает запрос на соединение ос создаёт новый экземпляр сокета, с помощью которого сервер обменивается данными с этим клиентом, при этом сокет, который создан для прослушивания продолжает находится в режиме приема соединений.

По существу, работа с сокетами - это операции ввода/вывода, которые бывают синхронными и асинхронными, при этом работа сокетов в синхронном режиме называется **не блокирующими сокетами**, а в асинхронном режиме называются **блокирующими сокетами**. Попытка соединения или приема данных в блокирующем режиме означает, что пока программа не соединится или не примет данные передачи управления, следующий оператор не произойдет. Созданы для реализации приложений в сети на основе стека TCPip называется WinSockAPI.

22. Динамически подключаемые библиотеки DLL.

Это модули, содержащие код, данные или ресурсы, которые могут совместно использоваться несколькими приложениями Windows. Прикладной программный интерфейс WinAPI осуществляется, как набор динамически подключаемых библиотек DLL. К DLL относится так же элементы управления, активы и драйвера. Программный модуль, оформленный в виде dll, хранится во внешней памяти в виде файла с расширением .dll и может содержать как функции, так и данные.

Dll можно загрузить и во время создания исполняемого модуля процесса (статически) и во время исполнения процесса (динамически). Для загрузки DLL в память используется механизм отображения файлов в память, т.е. всему файлу или некоторой непрерывной его части ставится в соответствии определенный участок памяти, при этом чтение данных из этого участка приводит к чтению из файла, а запись данных соответственно к записи данных. Таким образом динамически подключаемые библиотеки позволяют уменьшить объем используемой физической памяти при одновременной работе нескольких приложений, которые используют одну и ту же библиотеку. Библиотеки DLL не присоединены непосредственно к выполняемым файлам, в выполняемые файлы занесены только их адреса.

23. Сервисы

Сервис - это процесс, который выполняет служебные функции. Обычно это функции необходимые для работы приложений. К другому типу сервисов относятся драйверы, т.е. программы, обеспечивающие доступ к внешним устройствам. Третий вид сервиса - процессы, называемые мониторами. Каждый из них отслеживает работу какого-либо приложения. Управляет работой сервиса программа ОС - менеджер сервисов (service control manager). Он выполняет следующие функции:

1. Поддержка базы данных, установленной сервисом.

2. Запуск сервиса при загрузке ОС.

3. Поддержка информации о состоянии работающих сервисов.

4. Передача управляющих запросов работающим сервисом

5. Блокировка и разблокировка баз данных сервисов.

Для того чтоб менеджер сервиса знал о существование определенного сервиса его нужно установить. Информация обо всех установленных сервисах хранится в реестре. Сервис может запускаться с ОС как при загрузке, так и из приложения. Для работы с сервисами в Windows предоставлены специальные функции.

Поскольку сервисы работают под управлением менеджера, то они должны удовлетворять определенными соглашениям, которые определяют их интерфейс. Обычно сервисы оформляются как консольные приложения, хотя могут быть приложениями с графическим интерфейсом.

24. Виртуальная память. Выделение памяти процессам.

Всем процессам операционная система представляет ресурс виртуальной памяти, именно в ней хранятся данные для выполнения процесса. Самим процессом реальное расположения виртуальной памяти неизвестно. Размер виртуальной памяти теоретически ограничен разрядностью операционной системы. На практике в конкретных операционных системах устанавливается ограничение, ниже её пределов. Использование виртуальной памяти двери 2 основных преимущества:

1. Прикладным программистами не нужно заниматься вопросами реального размещения данных в памяти.

2. Дает возможность ОС запускать параллельные процессы, т.к. виртуальная память всегда больше, чем оперативная.

Виртуальная память делится на блоки одинакового размера - виртуальные страницы.

В Windows страницы бывают большими 4 или 2 мегабайта или маленькими 4 килобайта. Оперативная память делится на страницы такого же размера.

Обычно процессы используют не весь объем виртуальной памяти, а только небольшую её часть. Поэтому в ОЗУ находится ограниченное количество страниц, которое непосредственно необходимы процессу – рабочий набор процесса. Те виртуальные страницы, которые пока не требуются процессу операционная система выгружает в файл подкачки во внешней памяти. Для определения того, где в данный момент находится требуемая страница ОС создаёт таблицы страниц, т.е. массив указателя на страницу.

25. Структура консольного приложения

Консолью называется интерфейс, который используется приложением для ввода вывода информации. Соответственно приложение, которые использует консоль, для обмена данными с пользователем, называется консолью. В Windows консольные приложения внешне напоминают приложение dos, но это приложение в Windowsи и под dos работать не будет.

Чаще всего консольное приложение применяется в системном программировании, для разработки сервиса и обработки не устранил ошибок, возникающих при работе графического приложения. Консоль состоит из одного входного буфера и одного или нескольких буферов экрана. Входной буфер содержит информацию о событиях ввода. Каждая из них описывается записью. Все записи упорядочены в очередь. Буфер экрана содержит информацию для выводов окна приложения и является двумерным массивом, который содержит символ и данные о цвете.

На программном уровне для ввода и вывода информации консольные приложения используют стандартные устройства ввода и вывода. Хотя могут открывать другие файлы, сетевые соединения и т.д.

Консоль обеспечивает 2 уровня ввода и вывода текстовой информации. Функции высокого уровня обеспечивает ввод и вывод символов с консоли игнорируя остальные события.

Функции низкого уровня обеспечивают обработку всех событий.

Консольное приложение может не заботиться о реализации о реализации пользовательского интерфейса. Его обычно реализует ОС. События ввода подразделяются на следующие категории: Ввод с клавиатуры, мышью, изменение размеров окна.

26. Работа с консолями.

Процесс может быть связан только с одной консолью. Новая консоль создаётся следующими способами:

1. При создании консольного процесса команда CreateProcess нужно установить флаг CreateNewConsolе. В этом случае, если консольный процесс создаётся из консольного приложению, а флаг не установлен, новый процесс присоединяется к консоли родительского.

2. Следующая функция BOOL AllocConsole (VOID) возвращает ненулевое значение, если консоль создана успешно, если же нет, то False.

Приложение освобождает консоль вызовом функции BOOL FreeConsole, при успешном возвращает ненулевое, иначе False.

При создании новой консоли система создаёт три стандартных дескриптора:

STDIN - ввод.

STDOUT - вывод.

STDERR - ошибка.

Значение дескриптора ввода вывода можно получить значением функции GetStdHandle.

Дескриптор окна консоли вызывают функцию HWND GetConsoleWindow.

Если у приложения консоль есть, то функция возвращает дескриптор окна консоли, если нет, то Null.

27. Работа с буфером экрана.

Буфер экрана может быть создан посредством вызова функции CreateConsoleScreenBuffer.

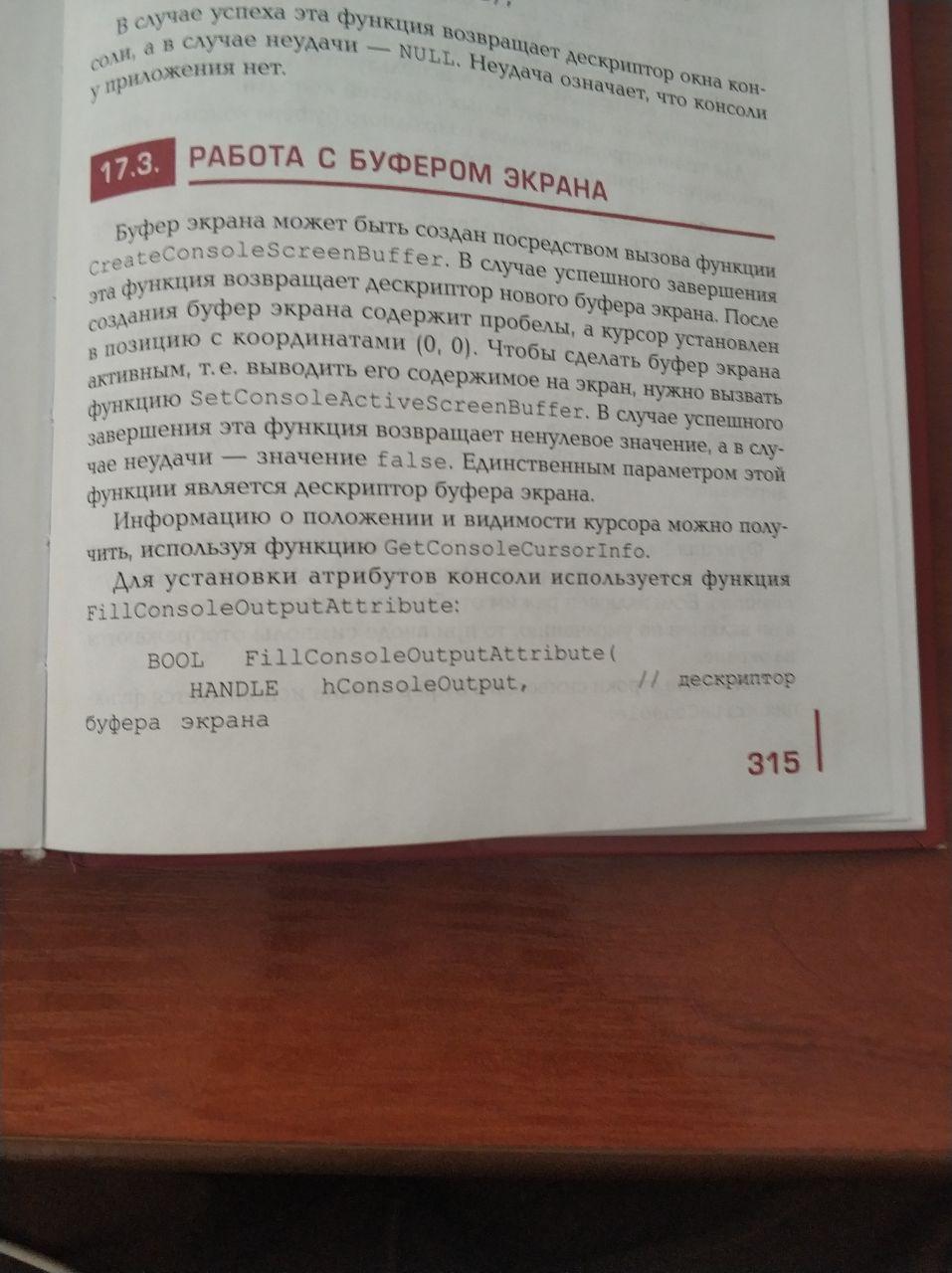
В случае успеха она возвращает дескриптор нового буфера экрана. После создания, буфер содержит пробелы, а курсор установлен в нулевых координатах.

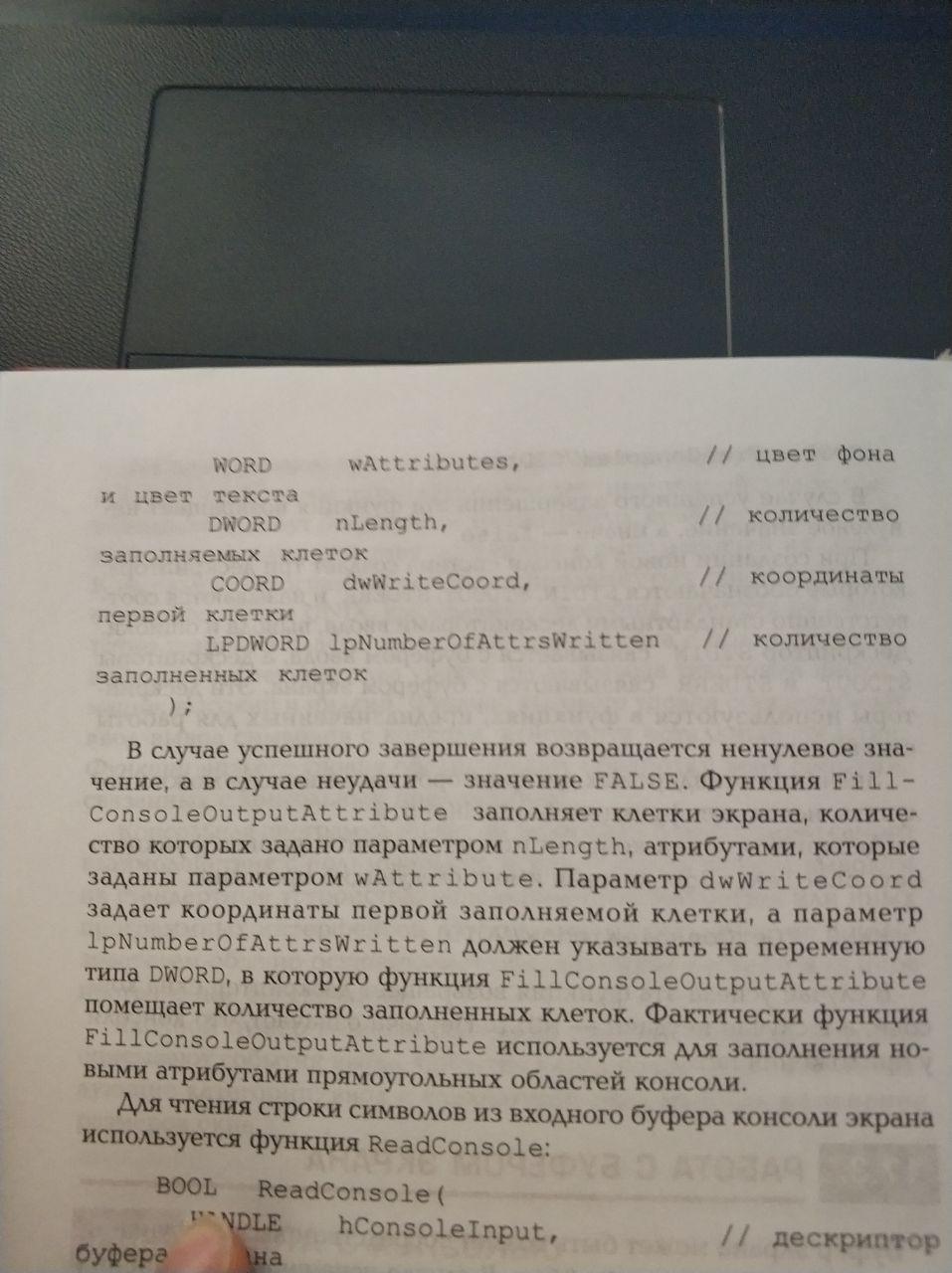
Что бы сделать буфер активным используют функцию: SetConsoleActiveScreenBuffer.

Единственным параметром этой функции является дескриптор буфера экрана. Информацию о положении и видимости курсора можно получить, используя функцию:

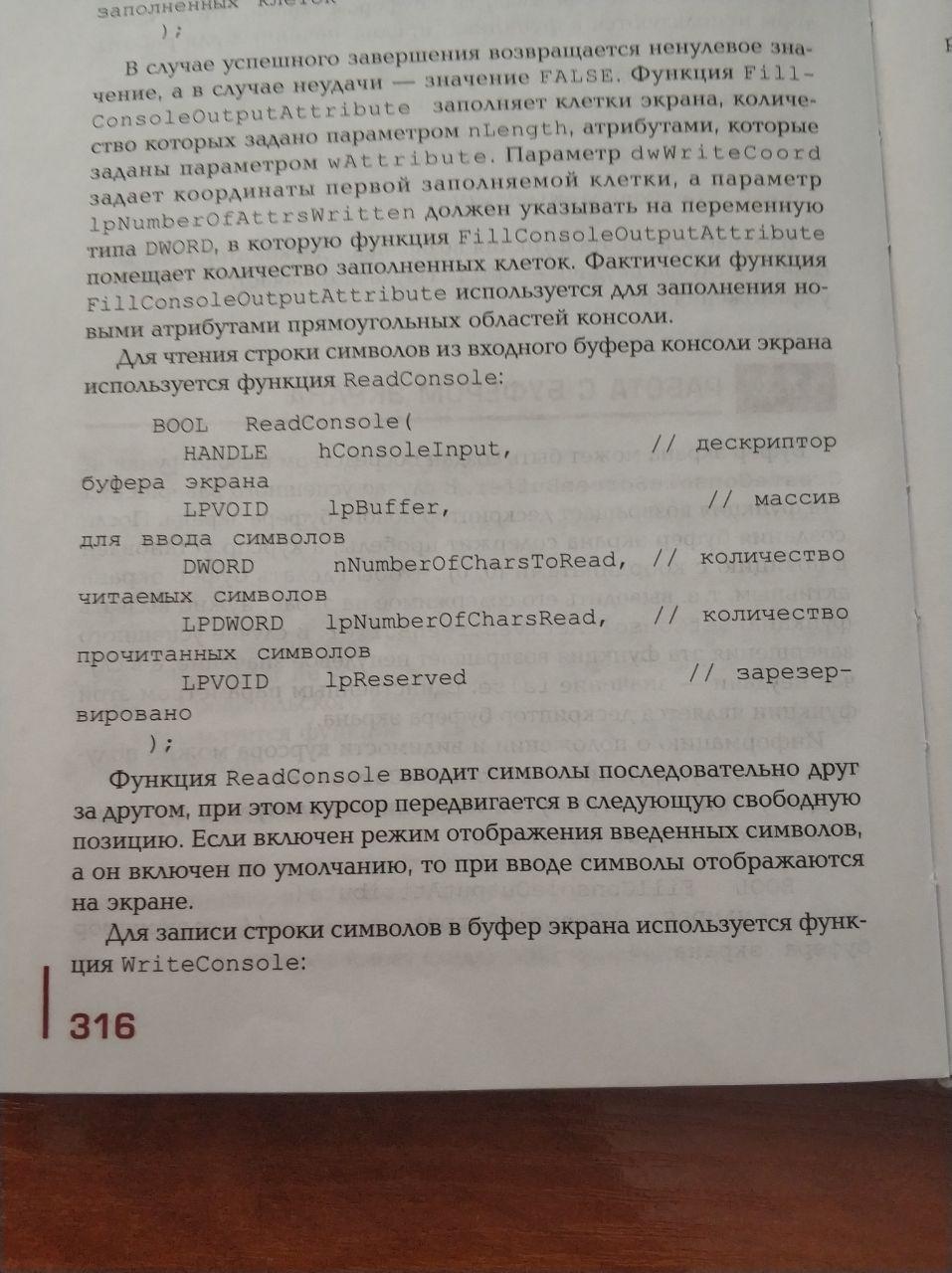
GetConsoleCursorInfo.

Для установки атрибутов консоли используется функция:

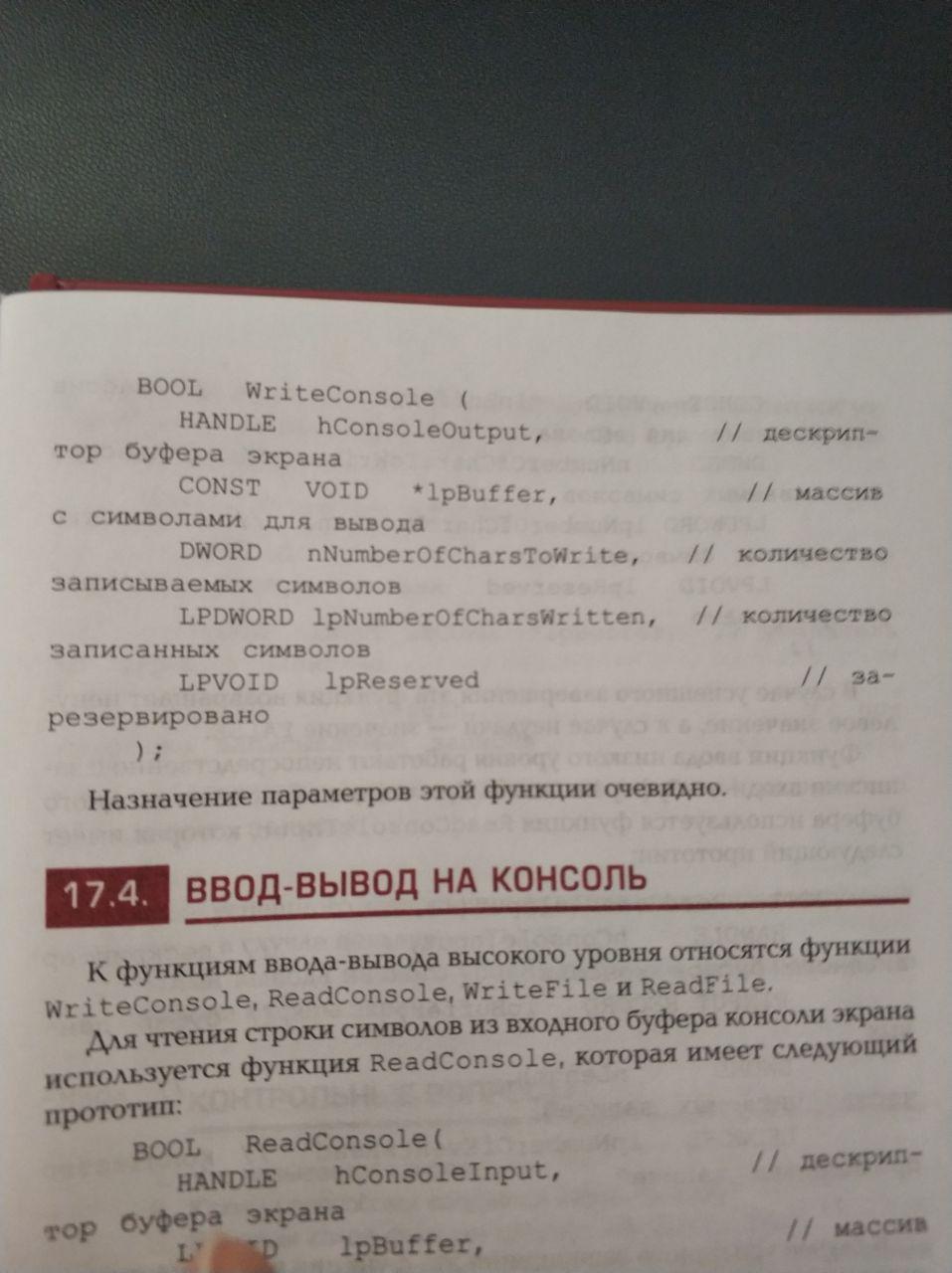




Для чтения строки символов из входного буфера консоли экрана используется функция:

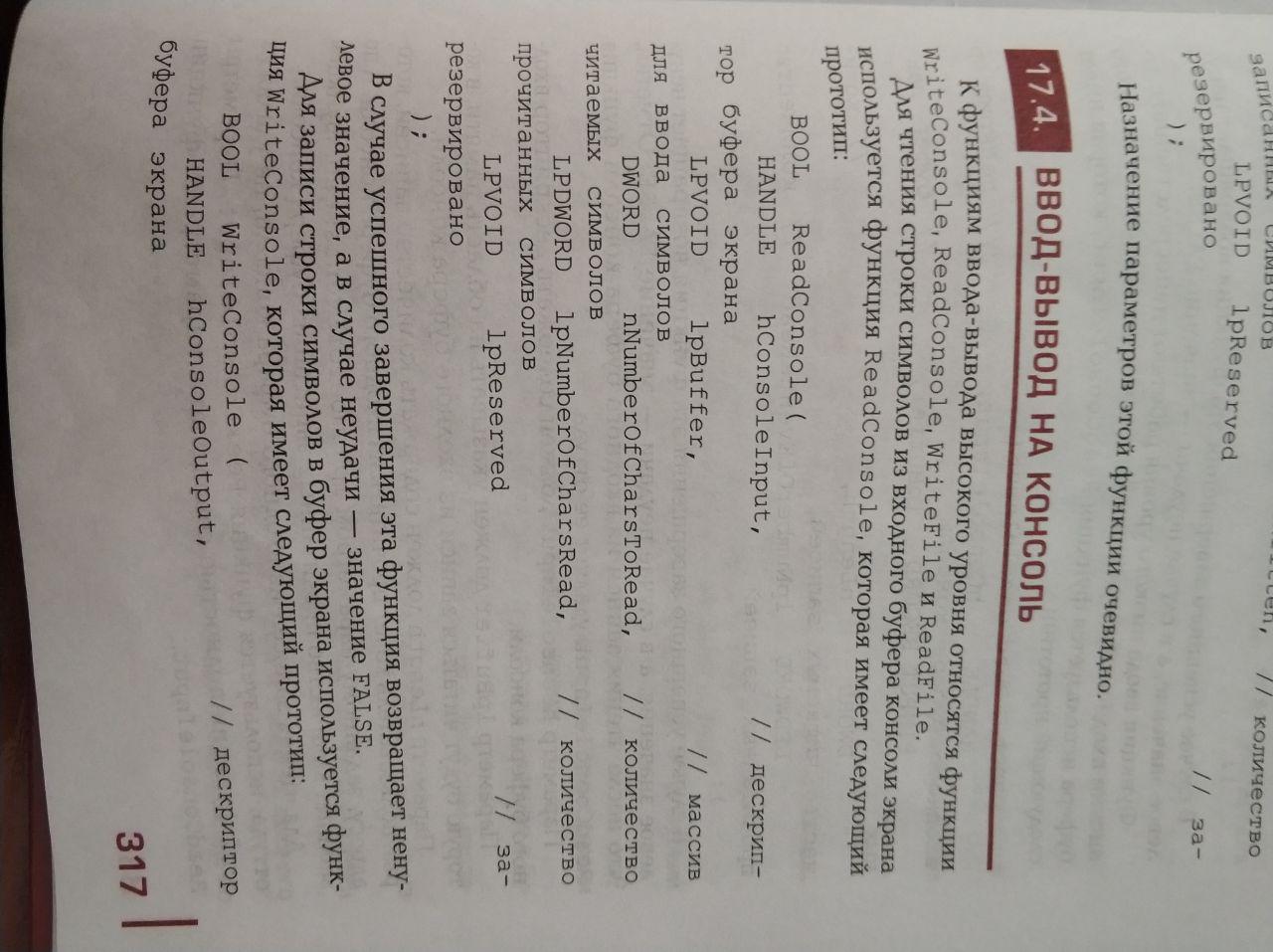


Для записи строки символов в буфер экрана используется функция:



28. Ввод/вывод на консоль.

Функции WriteConsole, ReadConsole, WriteFile, ReadFile являются функциями высокого уровня. Для чтения строки символов из входного буфера консоли экрана используется функция ReadConsole:



Для записи строки символов в буфер экрана используется функция WriteConsole:

