Глава 4. Загрузка и средства сохранения информации о параметрах операционных систем персональных компьютеров.

4.1 Загрузка операционных систем

Для простого пользователя после включения компьютера выделяются, в лучшем случае, два шага: проверка работоспособности оборудования и загрузка операционной системы. Но в действительности от момента, когда нажата кнопка питания системного блока, до возможности запускать программы пользователем происходит множество разнообразных действий. Это зависит от того, какая (или какие) операционные системы установлены на компьютере. Рассмотрим это вопрос более подробно. Для придания конкретности рассматриваются этапы загрузки операционных систем семейств Linux и Windows.

Сначала рассмотрим загрузку Linux. Этот процесс описан во многих источниках, например, в материале Костромина [28]. В нем выделено 10 этапов, а некоторые из них разбиты на свои подэтапы.

Этап 1: BIOS.

Этап 2: Master Boot Record и Boot Record.

Этап 3: Вторичный загрузчик.

3.1. Задачи, решаемые загрузчиком.

3.2. Файл /boot/initrd и его назначение.

3.3. Файл /boot/system.map.

3.4. LILO.

3.5. GRUB.

3.6. Параметры начальной загрузки, обрабатываемые загрузчиком.

Этап 4: Запуск ядра.

4.1. Загрузка и инициализация ядра.

4.2. Параметр "init=".

4.3. Параметры начальной загрузки, обрабатываемые ядром.

Этап 5: Процесс init и файл inittab.

5.1. Уровни выполнения.

5.2. Файл inittab.

5.3. Стили начальной загрузки.

Этап 6: Инициализационный скрипт rc.sysinit.

6.1. Обзор действий, выполняемых скриптом rc.sysinit.

6.2. Конфигурационные файлы, вызываемые из rc.sysini.t

Этап 7: Скрипт rc и запуск системных сервисов.

7.1. Структура каталога /etc/rc.d/.

7.2. Скрипт /etc/rc.d/rc.

7.3. Системные сервисы (демоны).

Этап 8: Запуск процессов getty и login.

Этап 9: Старт оболочки bash.

Этап 10: Выполнение команд пользователя.

10.1. Загрузка в однопользовательском режиме.

10.2 Загрузка в графическом режиме.

Опишем их подробнее.

Из за ограничений объема представленной работы не будем останавливаться подробно на этапе использования BIOS, а отошлем к упомянутому материалу Костромина. В последнем имеются свои ссылки на источники по этой теме. Отметим только, что выполняемые действия осуществляются на аппаратном уровне.

Посредством BIOS (Basic Input Output System, базовой системы ввода/вывода) после проверки уровня напряжения подается сигнал на материнскую плату и реализуется [28]: «. . . функция самотестирования (POST - Power-On Self Test). При этом тестируются процессор, память и системные средства ввода/вывода, а также производится конфигурирование программно-управляемых аппаратных средств компьютера. Кроме того, производится поиск и обнаружение периферийных устройств. При этом производится сравнение установок, записанных в CMOS (Complementary Metal Oxide Semiconductor) с тем, что реально обнаружено в системе.».

Далее на этом этапе осуществляется поиск загрузочного устройства. Оно должно содержать «Магическое число» 0x55AA, которое должно быть расположено в двух последних байтах блока (сектора) диска с минимальным адресом (первый сектор нулевой дорожки на нулевом цилиндре) [29].

Задача второго этапа - отработать информацию, расположенную в первом секторе жесткого диска называемую MBR (Master Boot Record, главным загрузочным сектором). В нем записана соответствующая программа (внесистемный загрузчик). Он должна поместить в оперативную память загрузчик ядра операционной системы и передать ему управление. В случае неудачной попытки активизировать загрузчик выдается сообщение

об ошибках. Заметим, что программа, которая находилась в MBR и управляла ходом загрузки операционной системы со времен MS-DOS принято называть Boot Record, и соответственно этот первый сектор диска - Boot Sector (загрузочным сектором).

Для современных типов процессоров (Itanium с 64-битной архитектурой) появились диски, поддерживающие разделы диска GPT (GUID Partition Table), вместо MBR. Диски с GPT могут обеспечить поддержку 128 основных разделов (против 4 в MBR) и рабочее пространство 18 экзабайт (против 2 Тбайт в MBR) [30].

Когда на компьютере имеется более одной операционной системы, выбором какая из них будет активизирована, управляется программой, называемой загрузчик (вторичный загрузчик).

Программа, называемая вторичный загрузчик, должна организовать выбор операционной системы. Для многих из них это означает размещение в оперативной памяти и активизацию ядра, причем необходимо не только выбрать одну операционную систему из списка, а возможно и одну из версий ядра одной и то же операционной системы.

Стандартным загрузчиком Liux является LIILO (LInux LOader). Работа программы управляется информацией из файла lilo.conf. Он представляет собой простой текстовый файл, строки которого записываются в формате: ИМЯ_ПАРАМЕТРА ЗНАЧЕНИЕ. В этом файле указаны раздел диска с загрузочной записью (boot), время задержки для процесса выбора (delay), имя файла ядра (image), метка (имя) которое позволяет выбрать систему (label) и другая информация. Подробно об этом файле узнаем после выполнения команды

man –S 5 lilo.conf

или, например, в Интернете, по адресу, где размещены страницы справочного руководства man.

Приведем фрагменты примера содержимого файла lilo.conf с сайта opennet.ru [31]

```
boot = /dev/hda
delay = 40
root = /dev/hda1
read-only
image = /zImage-2.5.99
    label = try
other = /dev/hda3
    label = dos
```

Дадим некоторые комментарии. В параметре delay указываются десятые доли секунд, то есть delay = 40 означает задержку 4 секунды. Строка read-only определяет

монтирование файловой системы в режим только чтения. С помощью переменной default можно задать систему, которая будет грузиться по умолчанию.

Другой часто используемый загрузчик, поставляемый с дистрибутивами Linux, называется GRUB (GRand Unified Boot Loader). Он так же имеет конфигурационный файл grub.conf, расположенный, чаще всего, в /boot/grub/, для которого в каталоге /etc может существовать символическая ссылка /etc/grub/grub.conf. Принцип взаимодействия ″программа-конфигурационный файл″ для LILO и GRUB похожи, но такой файл для второй системы имеет отличия в синтаксисе. Так же вторая система имеет расширенные возможности получать и обрабатывать команды пользователя при загрузке. Это не только выбор загружаемой ОС, но и выполнение действий, не предписанных первоначальной конфигурацией. Например, можно посмотреть содержимое каталогов файловой системы, настроить загрузчик по-другому и записать результаты такой настройки.

Некоторые фирмы, выпускают собственные загрузчики. Таким примером является ASP Loader фирмы ASP Linux. Такая программа управляется содержимым файла aspldr.conf, который как и многие конфигурационные файлы расположен в каталоге /etc. Приведем пример содержимого этого файла [32], строки которого так же имеют структуру ИМЯ_ПАРАМЕТРА ЗНАЧЕНИЕ, но разделенного на секции.

```
[linux@ASPLinux]
    icon linux
    kernel /boot/vmlinuz mem=188M root=/dev/hda6 ro
    initrd /boot/initrd.img
[linux1@ASPLinux Test Kernel]
    icon linux
    kernel /boot/bzImage root=/dev/hda6 ro
    initrd /boot/initrd.img
[SEPARATOR]
    [nt1@Windows 2001]
    icon windows
    sysboot 1-1
[SEPARATOR]
    [floppy@Boot from floppy]
```

icon floppy

sysboot a:

[BOOTMGR]

video graphics

default linux

timeout 30

clock 12

[ACTIVATOR]

writeboot off

writembr on

biosnum 1

mbrdev /dev/hda

language ru

Дополнительную информацию о ASP Loader можно, к примеру, посмотреть на странице aspldr.conf интерактивного справочного руководства man.

Все три описанных загрузчика операционных систем управляются программой, скорее всего расположенной в каталоге /sbin, а их конфигурационные файлы или ссылки на них в/etc. Но это может быть изменено разработчиками инсталлятора.

Продолжим рассмотрение шагов загрузки ОС. До загрузки Linux монтируется временный виртуальный диск, с помощью которого запускается на выполнение само ядро. Образ этой ФС хранится в файле /boot/initrd. Это имя дополняется символами, идентифицирующими конкретную версию ядра. Этим обеспечивается этап загрузки связанный с монтированием всех устройств, после чего временная файловая система заменяется на постоянную.

На следующем шаге загрузки используется файл /boot/System.map, который содержит символьную таблицу функций и процедур, используемых ядром операционной системы Linux. Такая таблица используется, прежде всего, в случае возникновения нестандартных ситуаций в работе ядра (например, Kernel panic или Linux oops).

Параметров, передаваемых ядру операционной системы много. В основном формат задания параметров ядра такой [33]: ИМЯ=значение1[,значение2, значение3,…, значениеN]. Подробную информацию об их составе можно получить в man-странице

bootparam установленной системы.

Один из способов классификации параметров выделяет три их типа.

1. Воспринимаемые загрузчиком. Конкретные действии для задания значений таких параметров зависят от конкретного загрузчика.

2. Передаваемые непосредственно ядру. Ядро представляет собой обычный исполнимый файл, поэтому этот тип параметров являются аргументами командной строки. Чаще всего это - информации о характеристиках аппаратного обеспечения, которые оно не может определить.

3. Направляемые процессу init. Аргументы этого типа передаются такой программе при ее запуске ядром.

Особенностью загрузки ядра Linux является то, что оно на диске хранится в сжатом виде. Первым действием этого этапа будет его распаковка. Затем выполняется множество операции, полный отчет о которых хранится в файле. Последний можно посмотреть, выполнив команду dmesg. Эта команда выводит содержимое одноименного файла, расположенного, скорее всего, в /var/log.

Один из параметров начальной загрузки операционной системы (″init=″) определяет, как программа будет запущена первой после ядра. Обычно это - init, расположенная в /sbin. Но можно предложить любую другую программу (или такую же, но расположенную в другом каталоге). Если ядру не удалось запустить не одну из программ, то наступает ситуация Kernel panic (режим "паника").

Как уже было сказано ранее, ядро представляет собой обычный исполнимый файл. До начала его выполнения ему могут быть переданы параметры, влияющие на режимы его работы. Это оформляется как параметры вызова функции в смысле языка программирования С.

Узнать какие параметры были переданы ядру при выполнении можно в командном режиме, например, так [34]:

cat /proc/cmdline

Что происходит на компьютере, когда работает такая операционная система как Linux – выполняются разнообразные процессы. Хотя целью работы компьютера является исполнение задач пользователей, кроме процессов последних в системе существуют и системные. Первым и прародителем всех остальных является процесс с именем init. Он использует специальный файл /etc/inittab. Последний представляет собой текстовый файл, в котором отдельными строками определены некоторые действия - типичная структура для конфигурационных файлов UNIX. В файле inittab строки состоят из 4 полей, разделенных двоеточиями [28, 33]:

id:runlevels:action:process.

Такие элементы строк определяют следующее.

1. id — идентификатор строки. Это произвольная комбинация, содержащая от 1 до 4 символов. В файле inittab не может быть двух строк с одинаковыми идентификаторами.
2. runlevels — уровни выполнения, на которых действия описанные этой строкой будет выполнены. Уровни задаются цифрами или буквами без разделителей.
3. process — процесс, который должен запускаться на указанных уровнях. Другими словами в этом поле указывается имя программы, вызываемой при переходе на указанные уровни выполнения.
4. action — действие.

Сделаем некоторое отступление. В системах UNIX/Linux реализованы механизм ограничения возможных действий системы. Такие ограничения определяются этапом и режимом работы системы. Всего существует восемь комбинаций этап+режим работы, и они называются уровням выполнения системы. В файле inittab они идентифицируется одним символом [28]:

0 — остановка системы;

1 — однопользовательский режим (для специальных случаев администрирования);

2 — многопользовательский режим без NFS (то же, что и 3, если компьютер не работает с сетью);

3 — полный многопользовательский режим;

4 — использование не регламентировано;

5 — обычно используется для запуска системы в графическом режиме;

6 — перезагрузка системы;

S (или s) — примерно то же, что и однопользовательский режим, но S и s используются в основном в скриптах.

Примерами задания уровней выполнения могут быть такие символы, идентифицирующие уровень (например, 5) или комбинации таких символов (к примеру, 2345).

В поле action стоит ключевое слово, которое определяет дополнительные условия выполнения команды, заданной полем process. Допустимые значения поля action такие [28]:

- respawn — перезапустить процесс в случае завершения его работы;
- once — выполнить процесс только один раз при переходе на указанный уровень;
- wait — процесс будет запущен один раз при переходе на указанный уровень и init

будет ожидать завершения работы этого процесса, прежде, чем продолжать работу;

- sysinit — это ключевое слово обозначает действия, выполняемые в процессе загрузки системы независимо от уровня выполнения (поле runlevels игнорируется). Процессы, помеченные этим словом, запускаются до процессов, помеченных словами boot и bootwait;
- boot — процесс будет запущен на этапе загрузки системы независимо от уровня выполнения;
- bootwait — процесс будет запущен на этапе загрузки системы независимо от уровня выполнения, и init будет дожидаться его завершения;
- initdefault — строка, в которой это слово стоит в поле action, определяет уровень выполнения, на который система переходит по умолчанию. Поле process в этой строке игнорируется. Если уровень выполнения, используемый по умолчанию, не задан, то процесс init будет ждать, пока пользователь, запускающий систему, не введет его с консоли;
- off — игнорировать данный элемент;
- powerwait — позволяет процессу init остановить систему, когда пропало питание. Использование этого слова предполагает, что имеется источник бесперебойного питания (UPS) и программное обеспечение, которое отслеживает состояние UPS и информирует init о том, что питание отключилось;
- ctrlaltdel — разрешает init перезагрузить систему, когда пользователь нажимает комбинацию клавиш <Ctrl>+<Alt>+<Del> на клавиатуре. Обратите внимание на то, что системный администратор может определить действия по комбинации клавиш <Ctrl>+<Alt>+<Del>, например игнорировать нажатие этой комбинации (что вполне разумно в системе, где много пользователей).

Теперь приведем примеры строк файла inittab [28].

```
id:5:initdefault
```

Так определяется, что по умолчанию устанавливается уровень выполнения 5.

```
1:2345:respawn:/sbin/mingetty tty1
2:2345:respawn:/sbin/mingetty tty2
3:2345:respawn:/sbin/mingetty tty3
4:2345:respawn:/sbin/mingetty tty4
5:2345:respawn:/sbin/mingetty tty5
6:2345:respawn:/sbin/mingetty tty6
```

Таким образом, задается то, что на 6-ти виртуальных терминалах выполняется программа mingetty. Последняя выдает на виртуальную консоль приглашение ввести имя пользователя и его пароль.

x:5:respawn:/etc/X11/prefdm –nodeamon

Этой строкой задается выполнение сценария для запуска менеджер дисплея с выбором графической среды.

Процесс начальной загрузки init расположен в оперативной памяти постоянно. После завершения загрузки он переводится в фоновый режим. При получении сигналов перехода системы на другой уровень он повторно выполняет действия из файла inittab. Это происходит и при выполнении команды shutdown (останов системы или перезагрузка).

В стандартной конфигурации после выполнения init наступает этапе работы скрипта rc.sysinit из каталога /etc/rc.d, Он выполняет много самых разнообразных действий, используя файлы из каталога /etc и его подкаталогов. В уже упомянутом материале Костромина [28] этому этапу определено 30 операций. Приведем некоторые из них. Задаются значения некоторых системных переменных окружения (PATH, HOSTNAME, NETWORKING и др.). Определяются и активизируются параметры ядра. Монтируются файловые системы по умолчанию (они перечислены в fstab), одна из них /proc. Устанавливается системные время, раскладка клавиатуры и шрифты. Задается сетевое имя компьютера. Проверяется возможность использования разнообразных устройств (USB, RAID и др). Загружаются модули, перечисленные в файлах каталога /etc/modules.

Прежде чем рассматривать следующий этап, на котором выполняется скрипт rc, надо сказать несколько слов о каталоге /etc/rc.d**.** Этот каталог играет важную роль в процессе загрузки, поскольку он содержит основные скрипты (программы на языке командного процессора shell), служащие для организации процесса загрузки. Рассматриваемый каталог rc.d содержит множество системных сценариев выполняющих действия на этапе загрузки. Но такие действия могут быть разными для отдельных уровней выполнения (см. выше) и поэтому в директории rc.d есть много подкаталогов с именами rcX.d, где X принимает значения от 0 до 6. Так же в /etc/rc.d имеется каталог init.d**.**

Если просмотреть (например, с помощью команды ls -al) содержимое подкаталогов rcX.d, то можно увидеть, что в этих подкаталогах содержатся не файлы, а только ссылки на файлы скриптов. Последние находятся (за редким исключением) в каталоге

/etc/rc.d/init.d. Он содержит по одному скрипту для каждой системной службы (NFS, sendmail, httpd и т. п.). Названия этих ссылок имеют имена, начинающиеся либо с буквы K, либо с буквы S.

После скрипта rc.sysinit процесс init запускает скрипт rc с опцией, равной заданному уровню выполнения. При этом реализуется переход системы с текущего уровня выполнения на заданный. Для этого реализуется такой алгоритм. При переходе на уровень X сначала просматривается каталог rcX.d и для всех ссылок, которые начинаются на K, выполняются скрипты с опцией stop. Таким образом, соответствующие службы останавливаются. Затем запускаются службы, которые на данном уровне выполнения должны быть запущены. Для этого выполняются скрипты с опцией start, начинающиеся на символ S. Символы S и K, с которых начинаются имена ссылок в подкаталогах rcX.d**,** происходят от слов start и kill. После символов S и K в именах ссылок стоят двузначные номера, которые определяют порядка запуска скриптов.

Кроме обеспечения непосредственных действий пользователей операционная система Linux поддерживает многие действия. Ими могут быть анализ сообщений ядра, монтирование файловых систем, например, при инициализации Flash-накопителя, обслуживание службы печати, запуск программ по расписанию и многие другие. Утилиты и программы, предназначенные для предоставления подобных услуг, называют системными службами. Их работа организована так. При начальной загрузке такие процессы запускается в фоновом режиме, и всё время работы системы большей частью бездействуют, ожидая, пока не возникнет запрос на их использование. Они составляют группу системных программ, называемых демоны. Таким программам посвящено много источников. Приведем ссылку на книгу Таненбаума и Вудхала [25], в которой приводится описание нескольких из них.

Одна из последних ссылок вида SXXname, используемых скриптом rc на уровнях 2–5, является ссылка на скрипт /etc/rc.d/rc.local [28]. Этот скрипт выполняется после всех других в процессе инициализации системы, поэтому если есть команды, задающие персональные настройки системы то они могут быть расположены тут.

Было рассмотрено несколько файлов (/etc/lilo.conf, /etc/inittab, /etc/rc.d/rc.sysinit, /etc/rc.d/rc, /etc/rc.d/rc.local) определяющих процесс загрузки и задающих некоторые параметры конфигурации системы. В действительности их гораздо больше. Среди них есть два типа файлов: конфигурационный текстовые файлы и сценарии (скрипты). Заметим, что в этой части монографии сценарии описываются без рассмотрения их содержимого, так как для этого надо знать язык программирования сценариев оболочки. Такое рассмотрение будет сделано позже.

Теперь перечислим (в алфавитном порядке) еще несколько файлов расположенных в каталоге /etc, а так же его подкаталогах, и задающих параметры конфигурации системы. Узнать о них подробнее можно изучая man-страницы справочного руководства.

Сценарии.

- /etc/bashrc — глобальный файл конфигурации bash, устанавливает синонимы (алиасы) и функции, и т.п.;
- /etc/profile — глобальный файл профилей — устанавливает значения важнейших переменных среды;
- файлы из каталога /etc/rc.d/init.d, запускающие демоны системных служб.

Текстовые конфигурационные файлы.

- /etc/modules.conf (или /etc/conf.modules) - содержит конфигурацию загружаемых модулей ядра;
- /etc/fstab - задает информацию, используемую для автоматического монтирования файловых систем;
- /etc/passwd содержит различную регистрационную информацию пользователей;
- /etc/issue — содержит сообщение, появляющееся на виртуальных терминалах перед запросом имени и пароля;
- /etc/motd — задает сообщение, выдаваемое пользователю после регистрации в системе;
- /etc/redhat-release — содержит название и номер версии дистрибутива.

Одним из действий выполняемых init – обеспечения начала сеансов пользователей. Это реализуется на восьмом этапе загрузки процессами getty и login. Первый из них образован от "get tty" — получить телетайп (элемент консоли вычислительных машин, для которых разрабатывались первые версии UNIX) Он обеспечивает вывод на экран виртуального терминала слова login, получение имени пользователя и передачу последнего программе идентификации пользователя – login. Современные версии Linux используют для «получения терминала» программу с именем mingetty. После регистрации пользователя запускается программа, называемая оболочка (shell). Она реализует много функций, но, прежде всего, - запуск всех прикладных процессов зарегистрированного пользователя. Взаимодействие названных процессов иллюстрирует укрупненная схема на рис. 4.1 из книги [35].

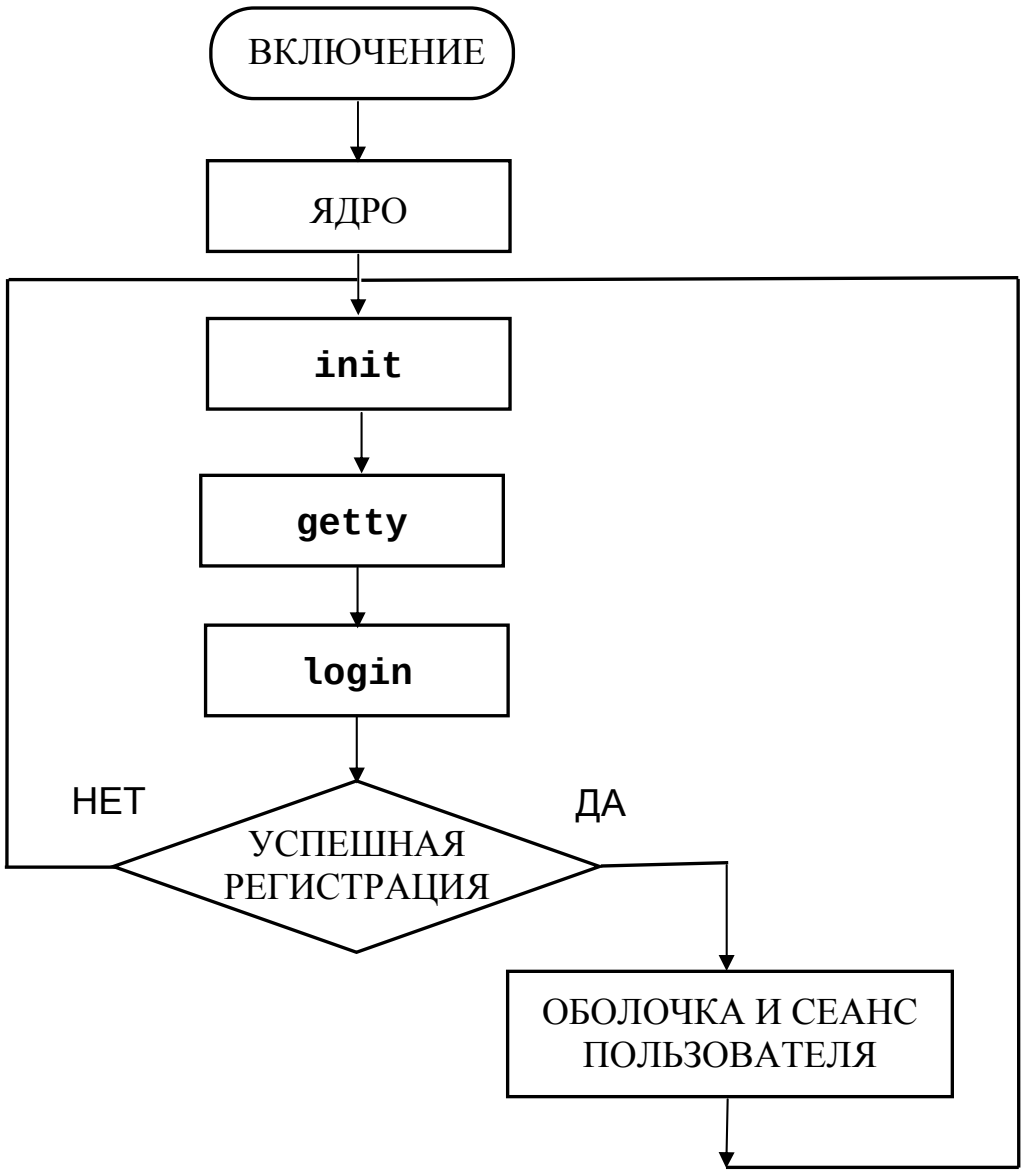


Рис.4.1. Порядок исполнения системных процессов при загрузке UNIX/Linux.

Программа login использует в качестве аргумента введенное имя. Она сверяет имя и пароль с записанными в файле /etc/passwd. Если результаты проверки положительны, то выполняются такие действия [28]:

- на экран выводится «сообщение дня» из файла /etc/motd;
- запускается интерпретатор команд shell (оболочка), указанный в /etc/passwd;
- устанавливается переменная среды TERM;
- выполняются инициализационные файлы оболочки shell;
- на экран выводится приглашение.

В случае неудачной регистрации для выбранного виртуального терминала вновь выполняется getty.

Следующий этап загрузки - старт оболочки. В операционных системах семейства UNIX/Linux имеется много программ, называемых оболочка. Они выполняют разные функции. Интерпретация команд, которые задают пользователи, – одна из них. Другими функциями оболочки являются поддержка языка программирования и специальных переменных, перенаправление ввод-вывода, генерация имен переменных. Все имеющиеся в системе оболочки можно узнать, набрав в командной строке

chsh --list-shell

после чего будет выведено, например, следующее.

/bin/sh

/bin/bash

/bin/nologin

/bin/ksh

/bin/csh

Имена 13 оболочек приведены в статье по адресу [36], в том числе и перечисленные выше:

- sh (оболочка, разработанная Борном и используемая в UNIX версии 7);
- bash (наиболее часто используемая в Linux программа, Bourne again - снова Борн, создана Брайаном Фоксом в рамках проекта GNU);
- csh (C-shell, разработана Биллом Джоем (Bill Joy) для BSD Unix);
- ksh (Korn-shell, создана Дэвидом Корном).

Оболочка с именем /bin/nologin используется при инициализации процессов, запускаемых программ пользователя. Каждая из оболочек имеет свои инициализационные скрипты. Если в качестве оболочки используется Bourne-shell, выполняется файл .profile, если C-shell — .login и .cshrc, если Korn-shell — .profile и .kshrc [28].

В домашнем каталоге пользователя могут быть расположены скрипты, выполняющие индивидуальные настройки при регистрации пользователя. Их имена могут быть: ~/.profile, ~/.bashrc и ~/.bash_profile. Здесь символ «~» задает имя домашнего каталога пользователя, а «.» - определяет, что такие файлы имеют свойство скрытый.

После запуска оболочки загрузка операционной системы закончена. Полный протокол, как было сказано ранее командой dmesg. Удобный способ просмотра такого протокола может быть получен при выполнении таких двух команд. Выход первой из них, передается второй, обеспечивающей постраничный режим просмотра текста.

dmesg | more

После успешной регистрации пользователя и старте оболочки он может выполнять различные программы или системные утилиты. Есть несколько уровней выполнения

системы для такой работы (2 – командный режим без поддержки сети, 3 – то же, но с поддержкой сети и 5 – графический режим). Но есть один особый случай состояния системы, при котором она нарушена, и главный администратор системы должен ее восстанавливать. Это выполняется на уровне равным 1. Для перехода на него стандартным способом является выполнение команды telinit с соответствующим параметром.

Операционные системы семейства UNIX/Linux обеспечивают только командный режим работы пользователя. Но сегодня последние предпочитают работать в режиме многопользовательского графического интерфейса. В семействе операционных систем UNIX (напомним, работающих на разных аппаратных платформах) графический интерфейс пользователя поддерживается системой X Window System. Инсталляторы операционных систем Linux для реализации графического режим чаще всего предлагают интегрированные графические среды: ГНОМ или KDE.

Linux – операционная система, которая обеспечивает и многопользовательский, и многозадачный режимы работы. Рассмотрев этапы загрузки, коротко приведем, как завершается работа системы. Для этого используется программа shutdown, которая обеспечивает как полный останов системы, так и ее перезагрузку. При выполнении этих операций необходимо выполнить множество операций: завершить все запущенные демоны, закрыть буферы жестких дисков и многое другое. Внезапное выключение компьютера может нарушить ее целостность.

Программа shutdown имеет параметры, задаваемые ей при вызове. Один из них задает время, через которое надо начать выключение компьютера. Есть еще опции, из которых мы приведем две: r (перезагрузка) и h (полный останов).

Приведем два примера использования программы. Для того чтобы дать такую команду надо завершить все программы и зарегистрироваться под именем пользователя root.

shutdown – h –now

Этот пример использования команды приводит к немедленному завершению работы системы. Еще пример

shutdown –r +10

демонстрирует, как можно задать перезагрузку системы через 10 минут.

В этой части кратко опишем кратко этапы загрузки Windows версии XP (учитывая, что его распространенность в мире составляет чуть менее 50%). Будем следовать следующим источникам Интернет [37, 38].

Опустив этапы, предшествующие загрузке собственно операционной системе, начнем с задач решаемых Ntldr. Имя этой программы образовано от NT loader и она является основным загрузчиком версий Windows NT до номера Windows NT 5.1, являющегося внутренним именем версии более известной как Windows XP.

Этот загрузчик чаше всего располагается в корневой директории загрузочного диска, что демонстрирует следующий рисунок.

```
C:\WINDOWS\system32\cmd.exe
C:\>dir /A:H
 Том в устройстве C имеет метку System
 Серийный номер тома: CC62-E7B8

 Содержимое папки C:\

04.10.2010  11:40               214 boot.ini
15.04.2008  16:00             4 952 Bootfont.bin
03.07.2008  21:16                 0 IO.SYS
03.07.2008  21:16                 0 MSDOS.SYS
04.07.2008  16:46    <DIR>          MSOCache
15.04.2008  16:00            47 564 NTDETECT.COM
15.04.2008  16:00           251 152 ntldr
29.02.2012  14:02     2 145 386 496 pagefile.sys
26.10.2010  18:08    <DIR>          RECYCLER
04.10.2010  11:48    <DIR>          System Volume Information
               7 файлов   2 145 690 378 байт
               3 папок    6 353 866 752 байт свободно

C:\>_
```

Рис.4.2. Расположение программы Ntldr в корневой директории диска C:.

Работа этой программы обеспечивает решение следующих задач.

1) Загружается минифайловая система.

2) Инициируется загрузочные файлы (одним из них является boot.ini – см. рис.4.2).

3) Выводится меню выбора операционной системы. Этим управляет содержимое файла boot.ini, в котором, в разделе [operating systems] расположены данные об установленных операционных системах.

4) Выполняется программа ntdetect.com, собирающая информацию об установленном оборудовании.

5) Запускается ntoskrnl.exe, пожалуй, являющуюся основным файлом ядра операционной системы. (обнаружен в скрытых папках наряду с ntkrnl* с окончанием mp, pa)

6) Стартует HAL (hardware abstraction layer – Слой аппаратных абстракций.

На этом функции ntldr заканчиваются, и в дело вступает ntoskrnl.exe, которое на этапе загрузки решает такие задачи.

1) Запускаются драйверы и сервисы, перечисленные как значения ключа реестра с адресом HKEY_LOCAL_MACHINE\SYSTEM.

2) Запускает smss.exe (Session Manager) - подсистема управления сессиями. Эта программа выполняет несколько задач, в том числе и инициализацию переменных окружения. Она может рассматриваться как слабый аналог init в UNIX.

3) Инициирует winlogon.exe (Logon Manager - менеджер входа в систему).

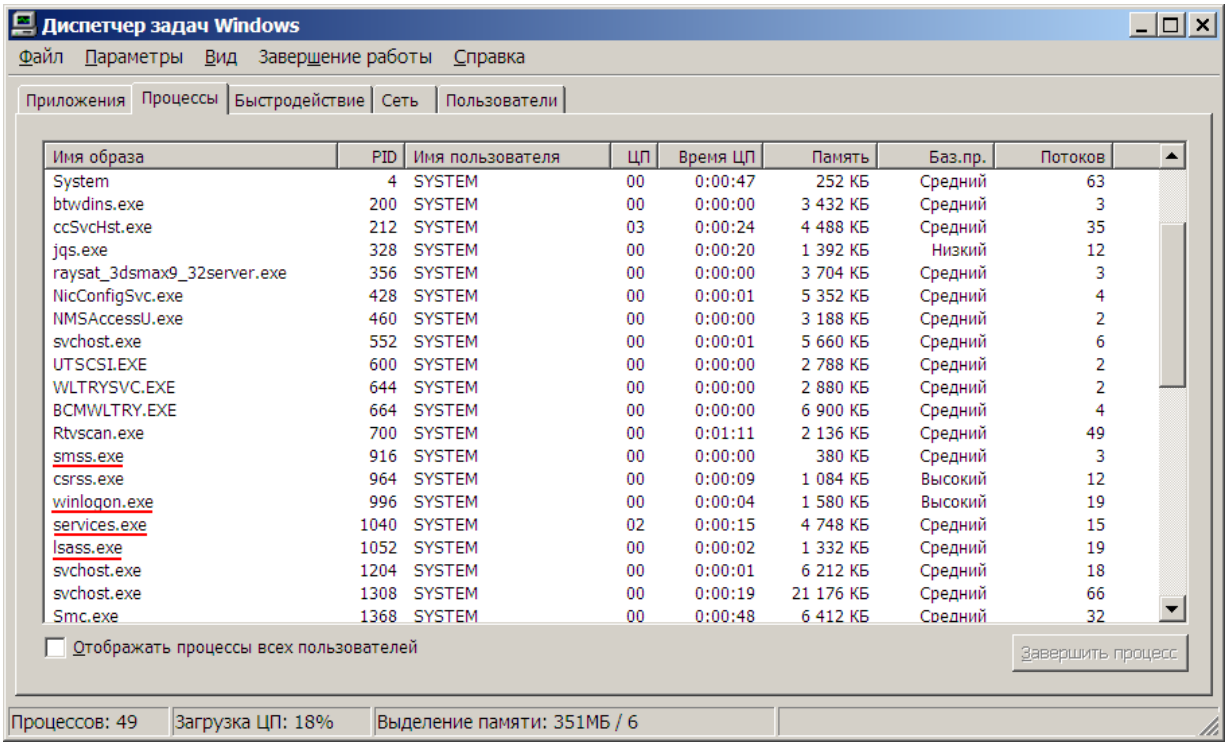
Последняя программа, в свою очередь, запускает графический режим и реализует три таких действия.

1) Запускает services.exe (подсистему сервисов).

2) Запускает lsass.exe (Local Security Authority - локальную систему безопасности).

3) Выводит экран с приглашением нажать комбинацию клавиш CTRL+ALT+DEL для входа в систему.

На представленном далее рисунке можно увидеть названные системные процессы.



Имя образа	PID	Имя пользователя	ЦП	Время ЦП	Память	Баз.пр.	Потоков
System	4	SYSTEM	00	0:00:47	252 КБ	Средний	63
btwdins.exe	200	SYSTEM	00	0:00:00	3 432 КБ	Средний	3
ccSvcHst.exe	212	SYSTEM	03	0:00:24	4 488 КБ	Средний	35
jqs.exe	328	SYSTEM	00	0:00:20	1 392 КБ	Низкий	12
raysat_3dsmax9_32server.exe	356	SYSTEM	00	0:00:00	3 704 КБ	Средний	3
NicConfigSvc.exe	428	SYSTEM	00	0:00:01	5 352 КБ	Средний	4
NMSAccessU.exe	460	SYSTEM	00	0:00:00	3 188 КБ	Средний	2
svchost.exe	552	SYSTEM	00	0:00:01	5 660 КБ	Средний	6
UTSCSI.EXE	600	SYSTEM	00	0:00:00	2 788 КБ	Средний	2
WLTRYSVC.EXE	644	SYSTEM	00	0:00:00	2 880 КБ	Средний	2
BCMWLTRY.EXE	664	SYSTEM	00	0:00:00	6 900 КБ	Средний	4
Rtvscan.exe	700	SYSTEM	00	0:01:11	2 136 КБ	Средний	49
smss.exe	916	SYSTEM	00	0:00:00	380 КБ	Средний	3
csrss.exe	964	SYSTEM	00	0:00:09	1 084 КБ	Высокий	12
winlogon.exe	996	SYSTEM	00	0:00:04	1 580 КБ	Высокий	19
services.exe	1040	SYSTEM	02	0:00:15	4 748 КБ	Средний	15
lsass.exe	1052	SYSTEM	00	0:00:02	1 332 КБ	Средний	19
svchost.exe	1204	SYSTEM	00	0:00:01	6 212 КБ	Средний	18
svchost.exe	1308	SYSTEM	00	0:00:19	21 176 КБ	Средний	66
Smc.exe	1368	SYSTEM	00	0:00:48	6 412 КБ	Средний	32

Рис.4.3. Системные процессы в памяти показывает Менеджер задач.

Что касается этапов загрузки Windows 7, то отсылаем к следующим двум источникам [39, 40].

4.2. Программирование в оболочке Linux.

Мощным инструментом настройки в семействе UNIX/Linux являются скрипты (script, называемые так же сценарии). В семействе операционных систем Microsoft их аналогом являются командные файлы. Сценарии обрабатываются оболочкой операционных систем так, как будто строки, составляющие его содержание, последовательно вводятся с клавиатуры. Таким образом, содержимое сценария представляет собой текстовой файл, содержащий строки-команды. Записывая последние,

следует соблюдать некоторые правила, поэтому говорят о синтаксисе языка сценариев. В Википедии их причисляют к языкам программирования скриптового типа. В этой группе есть несколько подгрупп, а рассматриваемые нами относят к языкам командной оболочки.

Представленный материал не исчерпывает всех вопросов программирования оболочек. Более подробную информацию можно найти в многочисленных источниках. Приведем только три [41, 42, 43].

Созданный текстовый файл (скрипт) можно запустить на выполнение из текущей директории, задав точку, пробел и имя файла. Для файла сценария можно установить свойство, обеспечивающее ему право на выполнение. В командном режиме оно устанавливается командой

chmod +x <имя файла сценария>.

В графическом режиме для придания сценарию права исполняться, надо использовать диалоговое окно свойств файла.

В операционных системах Linux существуют много оболочек. Создавая сценарий, важно ориентироваться на конкретную, которая будет его интерпретировать. Поэтому традиционно первой строкой в сценарии указывается то, на какую оболочку он рассчитан. Это оформляется как комментарий, например, так.

#!/bin/bash

Здесь первый символ – знак комментария, а затем полный путь к файлу программы оболочки.

Изучая язык сценариев, будем придерживаться следующей последовательности.

1. Символы.
2. Константы.
3. Переменные.
4. Операторы.
5. Функции.

В сценариях используют все символы клавиатуры, но некоторые из них имеют специальное назначение. Изучение правил использования символов оболочки относится к общему вопросу ее изучения и здесь не рассматривается. Назначение всех таких символов описано, например, в интерактивном справочном руководстве man на странице bash.

Константами являются произвольные наборы символов. Как обычно в языках программирования, прежде всего, различают символьные и числовые константы. В основном оболочка ориентируется на использование символьных констант. При их создании часто применяют кавычки. Это дает возможность использовать специальные символы, изменяя их особые функции. Например, задавая значение символьной

переменной из нескольких элементов, разделенных пробелами, используют двойные кавычки. Если необходимо использовать символьную переменную, как команду, выполняемую при интерпретации, используют обратные кавычки.

Некоторые сценарии предполагают использование численных констант. Наборы цифр имеют числовые значение в том случае, когда используются в специальных операторах присваивания - let.

Теперь кратко опишем, как в сценариях использовать переменные. Они делятся на две группы: специальные и введенные пользователем.

Существует много специальных переменных. Первую группу называют позиционные параметры или аргументы. Они имеют имена $1, $2,... Значения таких переменных задается в командной строке после имени сценария. Если создан файл с именем script1 и в командной строке набрано

script1 1 two

то специальная переменная $1 получит значение 1, а $2 – two.

Если значений, заданных в позиционных параметрах больше 9, то используется shift. Например, считать значение 10-ого параметра и напечатать его, можно записав shift, и далее echo $9 **(**для вывода на экран).

Приведем имена и значения еще нескольких специальных переменных:

$0 имя сценария (после знака доллара используем цифру 0);

$# число аргументов сценария;

$? код возврата последнего процесса;

$* и $@ все параметры сценария.

Для обозначения переменных, вводимых при создании скрипта, используйте английские буквы, цифры и знак «_». Следует помнить, что прописные и заглавные буквы задают разные переменные. Например, shell, SHELL и Shell будут разными именами.

Пользователь (или программист, использующий язык оболочки), может ввести переменную, задав ее значение. Примеры этого будут рассмотрены далее.

Теперь перейдем к описанию операторов языка программирования оболочки Linux.

Первым рассмотрим оператор вывода на экран echo. Он начинается с такого ключевого слова и выводит на экран, то, что записано далее в строке. Если в списке вывода надо использовать имя переменной, для вывода ее значения, то используется символ $.

Пример. Допустим, переменная v имеет значение 100, то строка сценария

echo v

выведет

v

а такая,

echo $v

напечатает на экране

100.

Опишем, как присваиваются значения переменным. Первый способ – использование оператора, начинающегося ключевым словом read, который позволяет ввести значение переменной. После ключевого слова просто задается список имен переменных, значения которых надо задать с клавиатуры. Список переменных записывается через символ пробела, в котором его элементы разделяются запятой. Если такой оператор встретился в сценарии, оболочка прекращает его интерпретацию, пока не будет нажаты клавиши на клавиатуре и в завершении <Enter>. После этого будет выполняться оператор, следующий за read. Те, кто программировал на Бейсике, вспомнит оператор input.

Значения переменных, вводимых пользователем, задается еще и присваиванием. Формат записи этого такой

переменная=константа.

Оператор из одного знака «=» задает значения символьных переменных.

Вариант оператора присваивания с использованием ключевого слова

let переменная=константа

позволяет присвоить значение для числовых переменных.

Создавая сценарии, для формирования арифметических выражений, нужно предпринимать специальные меры. Еще раз повторим, что в самих выражениях надо использовать не имя переменной, а его значение, указав вначале знак «$». Для записи операндов, соединенных знаками арифметических действий используйте expr.

Приведем примеры использования оператора присваивания. В первом из них придадим переменной символьное значение

my_office="в комнате 25 на втором этаже"

Теперь это значение можем использовать. Укажем, что обратный слеш (\) отменяет действие следующего за ним символа. В следующем примере двойные кавычки указанны для подавления действия пробелов. Записав в сценарии такие строки

echo "Мой офис расположен $my_office"

echo "Мой офис расположен \$my_office"

получим следующий результат

Мой офис расположен в комнате 25 на втором этаже.

Мой офис расположен $my_office”

Следующий пример демонстрирует присваивание переменным целочисленных значений и запись арифметических выражение. В левом столбце записан текст сценария, а его вывод с пояснениями - в правом.

Отметим, что форма записи оператора присвоения значений q11 выполняется правильно для операции сложения, а для умножения надо использовать символ (\), то есть записать q11=`expr $p \* 6`. Обращаем внимание на использование обратных кавычек.

Теперь еще один пример на присваивание числовых и символьных значений, в котором, переменная zyx без оператора let получит значение 1+2+3, а не 6, если z задать равным 3. В приведенном далее примере последним записан оператор read, для останова выполнения сценария до нажатия клавиши <Enter>.

Текст сценария	Комментарий и результат, выводимый на экран
#! /bin/bash	
clear	
p=1	ПРАВИЛЬНО, но не работает для умножения (*)
q11=`expr $p + 1`	ПРАВИЛЬНО
let q12=`expr $p + 1`	
let q13=expr $p+1	
echo " p=> " $p	p=> 1
echo " q11 => " $q11	q11 => 2
echo " q12 => " $q12	q12 => 2
echo " q13 => " $q13	q13 => 0
let q2=p+1	ПРАВИЛЬНО
let q3=$p+1	ПРАВИЛЬНО
q4=$p+1	
q5=($p)+1	
echo " p => " $p	p => 1
echo " q2 => " $q2	q2 => 2
echo " q3 => " $q3	q3 => 2
echo " q4 => " $q4	q4 => 1+1
echo " q5 => " $q5	q5 => 1
echo " Имя скрипта - " $0	Имя скрипта - ./r55_zero

echo " аргументы 2 и 4 -> " $2 $4 аргументы 2 и 4-> (ЗНАЧЕНИЯ)

echo " press " press

read qq останов исполнения сценария

Теперь рассмотрим команду test ([]), которая позволяет выполнять некоторые действия избирательно, при соблюдении некоторых условий. Эта встроенная команда, с использованием которой формируются операторы выбора и цикла. В нем используется выражение, которое записывается с ключевым словом

test выражение

или с квадратными скобками

[выражение].

Выражение может содержать операторы сравнения целых чисел, которые приведены в следующей таблице.

Табл.4.1. Операции сравнения целых чисел.

Обозначение	Операция
-eq	=
-ne	≠
-lt	<
–le	<=
-gt	>
-ge	>=

Так же выражением задаются следующие специальные комбинации из двух символов, являющиеся файловыми операторами (приводим только часть из них).

Табл.4.2. Файловые операции.

Обозначение	Проверка
-f	файл существует
-d	каталог
-x	выполняемый файл
-w	файл с разрешением записи
-s	пустой файл

Можно задать действия по проверке набора символов, например, -z позволяет определить является ли строка символов пустой. Символ «=» позволяет сравнивать на совпадение два набора символов.

Простые выражения можно соединять в сложные, задавая логические союзы: НЕ (символ «!»), ИЛИ (символы «-o»), а так же И (символ «-a»).

Приведем примеры использования команды test.

```
[ -f Dokument.txt ] ; echo $?
```

Будет выведено значение 0, если файл с именем Dokument.txt содержится в текущее директории, и другое число (чаще 1) – если нет.

```
[ -d New_folder ] ; echo $?
```

Будет выведено значение 1, если каталог существует в текущей директории, и другое число (чаще 1) – если нет.

```
p="This is March"; [ "$p" = "This is March"  ]; echo $? " 0 - is well"
q=March ;  [ "$q" = March  ] ; echo $? "if 0 - is well"
r=March ; [ ! "$r" ] ; echo $? "  0 - well, 1 - NO well"
```

В данном примере проверяется совпадение значений для символьных строк. Для первых двух (с именами p и q) выведено значение 0, так как значение переменной и символьная строка одинаковы. Для третьей строки проверяется на неравенство значения переменной и строки символов. Будет выдано значение отличное от 0 (чаще 1).

Теперь переходим к рассмотрению условного оператора, который позволяет выбирать для выполнения те или иные операторы в программе в зависимости от некоторых условий. Этот оператор имеет несколько форм записи. Первая соответствует выбору одной из двух групп команд, в зависимости от того выполнено или нет условие. Он записывается следующим образом.

```
if условие
then
команды1
else
команды 2
fi
```

Рассмотрим пример сценария, реализующего нахождение максимального из двух чисел. Они являются исходными данными (переменные с именами a и b). Далее сравниваются их значения, и, в зависимости от результата, третьей переменной (ее имя c) присваивается значение одной их исходных. Далее приведен текст скрипта.

```
#!/bin/bash
clear
echo " Вводим a"
read a
```

```
echo " Вводим b"
read b
if [ $b -gt $a ]
then
  let c=$b
else
  let c=$a
fi

echo " a=> " $a
echo " b=> " $b
echo " max{a,b} "$c

echo " Нажмите любую клавишу "
read q
```

Второй способ записи условного оператора предполагает выполнение не двух, а только одной группы команд. Она выполняется при истинности условия.

```
if условие
then
команды
fi
```

Приведем сценарий решения задачи нахождения максимального из двух чисел, но со второй формой записи условного оператора. В нем сначала результирующей переменной присваивается значение одной из входных переменных (с именем a), а уж потом выполняется сравнение. В отличие от предыдущего примера, в выражении используется операция не «строго больше», а «больше или равно». При равенстве, все равно, из какой переменной брать максимальное значение.

```
#!/bin/bash
clear
echo " Вводим a"
read a
echo "" Вводим b"
read b
c=$a
if [ $b -ge $c ]
```

```
then
  let c=$b
fi
echo " a=> " $a
echo " b=> " $b
echo " max{a,b} "$c

echo " Нажмите любую клавишу "
read q
```

Есть и третья форма записи условного оператора.

```
if условие
then
команды 1
elif
команды 2
else
 команды 3
fi
```

Приведем еще примеры на использование рассматриваемого оператора. Первым их них является сценарий, которому при вызове должно быть задано ровно два позиционных параметра.

```
clear
if [ $# -eq 2 ]
then
  echo “Имя скрипта $O”
  echo “Хорошо, задано ровно два позиционных параметра”
  echo “их значения $*”
else
  echo “Нет, вы задали не ровно два позиционных параметра”
fi
```

Следующий пример в литературе приводится довольно часто и обеспечивает проверку пользователя с задаваемым именем (оператор read Uname) в файле /etc/passwd. Если такое имя найдется, то он существует в системе. Это не означает, что пользователь открыл сеанс. Он может это сделать. В сценарии используется переменная ($?), принимающая значение 0, если предыдущая операция выполнена успешно. Но в условии

проверяется противоположная ситуация, то есть когда предыдущая операция закончилась неудачно. В этом случае $? не равна 0.

```
# /bin/bash
echo Введите имя пользователя
read Uname
grep -l $Uname /etc/passwd
if [ $? –ne 0 ]
then
  echo “Нет такого пользователя $Uname”
fi
```

Следующий пример показывает, как использовать третью форму записи условного оператора. В нем сначала вводится имя файла. И далее происходит следующее:

1) если введенное имя /bin, тогда будет выводиться Файл /bin - директория.

2) если введенное имя /bin/ls, тогда будет выводиться Файл /bin/ls – исполнимый.

3) если введенная строка что то иное, тогда будет выводиться NO.

```
read filename
if [ -d $filename ] ; then
 echo “Файл $filename - директория”
elif [ -x $filename ] ; then
  echo ”Файл $filename – исполнимый”
  ./$filename
else
  echo NO
fi
```

Обратим внимание, что ключевые слова if и then записаны в одну строку, но при этом требуется строгое соблюдение приведенного состава и порядка расположения символов. К примеру, после открывающей квадратной скобки обязателен пробел, закрывающаяся скобка обязательно выделяется пробелами.

А теперь рассмотрим, как в сценариях используются операторы цикла. Известно, что циклические алгоритмы реализуют многократные повторения группы команд. В языке сценариев есть три формы записи оператора цикла. Два первых из них мы рассмотрим на примере вычисления факториала. Известно, что для его вычисления надо последовательно перемножать целые числа. В циклическом алгоритме для нахождения факториала надо иметь две переменные. Одна будет последовательно принимать значения 1, 2, 3 и 4.

Вторую же надо сначала сделать равной 1, затем умножать ее старое значение на 2, 3, и, наконец, 4. В нашем алгоритме первая переменная имеет имя p, а вторая - q.

Первая форма записи оператора цикла начинается ключевым словом while и записывается так.

```
while условие
do
команды
done
```

Программа вычисления факториал с использование оператора цикла в форме while приводится ниж.

```
#! /bin/bash
clear
let p=1
let q=1
while [ $p -le 6 ]
do
  let p=$p+1
  let q=$q*$p        #  NO q=$q *$p
  echo "" > file$q   #  оператор не имеет отношение к вычислению
                     # факториала, а создает файлы file2 ... file5040
  echo -n  " p => " $p
  echo     " q => " $q
done
```

Она выведет следующие результаты

```
p =>  2 q => 2
p =>  3 q =>  6
p =>  4 q =>  24
p =>  5 q =>  120
p =>  6 q =>  720
p =>  7 q =>  5040
```

Обратим внимание на две строки, содержащие комментарии в тексте сценария. Одна из них предупреждает, что, записывая арифметические выражения, нельзя использовать пробелы. Вторая не имеет отношение к циклу подсчета факториала. Такой оператор, поставленный в цикле, создаст шесть пустых файлов с именами file2, …, file5040. Это демонстрирует возможность генерировать имена файлов в сценарии.

Вторая форма оператора цикла имеет такую форму записи.

```
until условие
do
команды
done
```

Для вычисления факториала может служить такой текст сценария.

```
 #! /bin/bash
clear
let p=1
let q=1
until [ $p -ge 6 ]       # $p - lt 6 PUSTO
do
   let p=$p+1
   let q=$q*$p
   echo -n  " p => " $p
   echo     " q => " $q
done
```

Он выдаст следующие результаты

```
p =>  2 q => 2
p =>  3 q =>  6
p =>  4 q =>  24
p =>  5 q =>  120
p =>  6 q =>  720
```

Так же обратим внимание на то, что условие окончания цикла формулируется иначе. В нем записывается, когда окончить цикл, и поэтому будет вычислен факториал только 6. При использовании оператора цикла в форме while записывалось условия продолжения цикла. Его окончательным результатом будет 7! равный 5040.

В разных языках программирования для операторов цикла используется конструкция for. В языке сценариев она имеет приведенную ниже форму, но при исполнении реализует необычные действия.

```
for переменная in список
do
команды
done
```

Она не перебирает значения своей основной переменой, изменяемой по фиксированному алгоритму. Команды, помещенные между ключевыми словами do и done, выполняются столько раз, сколько констант перечислено в списке.

К примеру, такой фрагмент сценария выполнит команду вывода на экран три раза. В отдельных строках будет записано то, что расположено после in.

```
# Печатать One 2 three
for int in One 2 three
do
  echo $int
done
```

Следующий пример распечатает названия всех файлов текущей директории.

```
# Просмотреть файлы в директории и печать их имен.
clear
for var in *
do
  echo Найден файл с именем $var
done
```

Переходим к последнему из описываемых нами операторов. Он реализует то, что называют множественный выбор. Общий формат его записи таков

```
case слово in
шаблон 1)
команды1
. . .
; ;
шаблон 2)
команды 2
. . .
; ;
*)
команды
. . .
; ;
esac
```

Его применение рассмотрим на конкретных примерах. В первом из них проверяется введенный символ. По результатам проверки выделяются случаи

1) Цифры от 1 до 9

2) Буква а (заглавная или строчная)

3) Буква б (заглавная или строчная)

Во всех остальных случаях выводится No….

Prim9. Пример case

```
case $1 in
[0-9]) echo Number ;;
[a,A]) echo “a A“ ;;
[b,B]) echo “d B“ ;;
*)      echo No…
```

Приведенный ниже сценарий реализует выбор одной из шести операций. Три их них монтируют следующие устройства: флеш диск с файловой системой FAT, компакт диск и раздел жесткого диска с файловой системой NTFS. Три другие операции размонтируют эти устройства. Каждое исполнение сценария реализует только одну операцию.

```
#/bin/bash
clear
echo "  Input regim"
echo "1. mount flash disk"
echo "2. umount flash disk "
echo "3. mount CD ROM"
echo "4. umount CD ROM"
echo "5. mount hdisk with NTFS"
echo "6. umount hdisk with NTFS"
read i
case $i in
'1')  mount -t vfat /dev/fd0 /mnt/sda1 -o rw,codepage=1251,iocharset=koi8-r ;;
'2')  umount -t vfat  /dev/fd0 /mnt/sda1 ;;
"3") mount -t iso9660 /dev/cdrom /mnt/cdrom  ;;
"4") umount -t iso9660 /dev/cdorm /mnt/cdrom ;;
"5") mount -t ntfs  /dev/hdb1 /mnt/hdisk -o rw  ;;
"6") umount -t ntfs    /dev/hdb1 /mnt/hdisk ;;
*)     echo " NO "
esac
```

В заключение этого раздела опишем, как в сценариях определять и использовать функции – группу операторов, наделенных именем, по которому такие операторы можно вызвать неоднократно. Вызываемая для исполнения группа операторов должна быть записана до ее первого появления. Функции можно передавать параметры и получать результат выполнения – значение, заданное в return.

В следующем примере файл скрипта определяет две функции. Одна из них выбирает максимум из двух чисел, а вторая - вычисляет факториал 6. Первая функция вызывается два раза. В первом - исходные данные для функции (значения переменных a и b) передаются как параметры вызова функции, а во втором задаются оператором присваивания.

```
fn_max() {
 #     let a=2  ; let b=3
#     echo "first parametr " $1 "two parametr: " $2
#      let a=$1 ; let b=$2
      if [ $b -gt $a ]
      then
        let c=$b
      else
        let c=$a
      fi

      echo " a=> " $a
      echo " b=> " $b
      echo " max{a,b} "$c
      echo "  "
      echo  "  "
}

fn_factorial()
    {
      let p=1
      let q=1
      until [ $p -ge 6 ]
      do
        let p=$p+1
```

```
        let q=$q*$p
        echo -n  " p => " $p
        echo     " q => " $q
      done
     return $q
}

echo " max"
  fn_max 1 2

   let a=2  ; let b=3
  fn_max

echo " factorial "
  fn_factorial

echo press
read qq
```

В заключение сделаем два замечания.

1) Если при выполнении сценария нажать комбинацию клавиш <Ctrl+D>, то он заканчивается досрочно.

2) Кратко опишем, как использовать редактора vi для подготовки сценария. Это может быть полезным в ситуации, когда надо исправить текстовый файл в командном режиме, а других средств нет. Вызвать редактор можно командой vi. Современная его версия имеет имя Vim, но вызывается приведенной выше командой. Этот редактор работает в двух режимах: исполнения собственных внутренних команд и набора текста. После вызова редактора активизируется первый из них, выполняющий общие действия над создаваемым файлом, а для перехода в режим набора или исправления текста надо нажать клавиши <a> или <i>. Чтобы возвратиться в командный режим, надо нажать сначала клавишу <Esc>, а затем символ «:». Например, напечатав необходимый текст надо его сохранить. Для этого, инициировав командный режим, надо набрать <w> и <имя файла>. Таким образом, выполнена команда сохранения файла. Для завершения работы с vi используют команду <q>. Вновь запустить программу для исправления текстовый файл можно командой «vi имя файла».

4.3. Системные сценарии Linux.

Раньше говорилось о том, что системные сценарии располагаются в разных каталогах и их выполнение привязано к разным событиям. Вначале разберем несколько скриптов, расположенных в каталоге /etc. Напомним, что такие сценарии исполняются после загрузки системы. Действия, составляющие эти файлы, влияют на настройки всей системы и режимов работы всех пользователей.

Первым рассмотрим, какие команды могут быть записаны в файле /etc/profile. Это не относится к какой-то конкретной версии Linux.

```
USER="`id -un`"
```

Эта строка присваивает значение переменной USER. Выполняется команда id, возвращающая значение текущей группы и пользователя. С опциями –un будет фиксировано буквенное значение только имени пользователя.

```
LOGNAME=$USER
```

Так переменной окружения LOGNAME присваивается имя текущего пользователя.

```
MAIL="/var/spool/mail/$USER"
```

Этой командой указывается каталог, где размещается почта пользователя. Обращаем внимание, что для каждого пользователя создается свой каталог.

```
HOSTNAME=`/bin/hostname`
```

Еще одну переменную инициализируют как результат выполнения команды (/bin/hostname). Она получает имя текущего хоста.

```
HISTSIZE=1000
```

Этим устанавливается количество команд, сохраняемых в системе. Затем их можно просматривать и вызывать для исполнения командой history.

```
if [ -z "$INPUTRC" -a ! -f "$HOME/.inputrc" ]; then
INPUTRC=/etc/inputrc
fi
```

Здесь используем оператор со сложным условием. Смысл его такой. Есть два простых условия:

1) переменная INPUTRC имеет в качестве значения пустую строку символов;

2) в домашнем каталоге пользователя нет файла inputrc. Для этого используется комбинация символов ! -f (напомним, что символ «!» задает отрицание условия). Два таких условия соединяются логическим оператором И (-a). Оно будет истинным только в одном случае, если истины оба простых условия.

Следующие группы операторов задают значения переменной PATH.

```
if [ `id -u` = 0 ] && ! echo $PATH | /bin/grep -q "/sbin" ; then
  PATH=/sbin:$PATH
```

```
fi
if [ `id -u` = 0 ] && ! echo $PATH | /bin/grep -q "/usr/sbin" ; then
   PATH=/usr/sbin:$PATH
fi
if [ `id -u` = 0 ] && ! echo $PATH | /bin/grep -q "/usr/local/sbin"
   then
   PATH=/usr/local/sbin:$PATH
fi
if ! echo $PATH | /bin/grep -q "/usr/X11R6/bin" ; then
   PATH="$PATH:/usr/X11R6/bin"
fi
```

Для задания значения этой переменной пользователю root (ID=0) добавляются пути к трем подкаталогам /sbin, а для других пользователей - /usr/X11R6/bin.

Установив значение этих переменных нужно сделать их эффективными. Для этого служит следующий оператор

```
export USER LOGNAME MAIL HOSTNAME HISTSIZE INPUTRC PATH
```

Использование оператора цикла иллюстрирую следующие строки, которые могут быть записаны в системном скрипте.

```
for i in /etc/profile.d/*.sh ; do
   if [ -x "$i" ]; then
         . $i
   fi
done
```

Этот цикл перебирает все файлы в каталоге /etc/profile.d. Если файл имеет свойство (-x), то он исполняется.

Теперь рассмотрим, что содержится в файле /etc/rc.d/sysinit. Операторов в нем много и мы приведем только несколько фрагментов.

```
if [ -f /etc/sysconfig/network ]; then
   . /etc/sysconfig/network
else
   NETWORKING=no
fi
```

Эти действия связаны с настройкой сети и устанавливают наличие файла /etc/sysconfig/network и его исполнение.

```
if [ -z "$HOSTNAME" -o "$HOSTNAME" = "(none)" ]; then
   HOSTNAME=localhost
fi
```

Так устанавливается значение переменной HOSTNAME. Как видно, ее значение задается не одним системным сценарием. Мы уже встречали команду в /etc/profile. Здесь же HOSTNAME будет присвоено значение localhost, если условный оператор, обнаружит пустое содержание этой переменной.

```
mount -n -t proc /proc /proc
```

Файловая система proc активизируется командой монтирования. Она используется для такой операции и для других устройств.

```
if [ -f /etc/rc.modules ]; then
        /etc/rc.modules
fi

if [ -f /etc/rc.d/rc.modules ]; then
        /etc/rc.d/rc.modules
fi
```

Выполняется активизация модулей ядра. Для этого просматривается два каталога.

Инициализация протоколов и служб TCP/IP выполняется несколькими сценариями. Мы приведем фрагмент одного из них - inet. Условным оператором проверяет значение переменной NETWORKING и выполняет exit 0 (выход с результатом выполнения команды равное 0), если ее значение no. Далее приводится пример запуска двух демонов с использованием команды daemon. Это выполняется, если первый позиционный параметр при вызове сценария получил значение start. В случае, когда такой параметр stop, эти демоны выгружаются (killproc).

```
if [ $(NETWORKING) = “no” ] ; then
        exit 0
fi

case “$1” in
start)
     echo –n “Starting INET services:”
     daemon rpc.portmap
     daemon inetd
     echo
;;
```

```
stop)
      echo –n “Stopping INET services:”
      killproc inetd
      killproc rpc.portmap
     echo
     rm -f /var/look/sybset/inet
;;
*)
  echo  “Usage: (start|stop)”
;;
esac
```

Работа сценария управляется первым позиционным параметром в строке при его вызове. При значении этого параметра stop останавливаются два процесса, и уничтожается файл inet, расположенные по адресу /var/look/sybset в файловой системе. Отметим, что если при вызове первый позиционный параметр задан неправильно (не start или stop, но * - все остальные случаи) выводится сообщение об ошибке.

Приведем фрагмент сценария, связанного с запуском демона cron - выполнение заданий по расписанию.

```
prog="crond"
start() {
          echo -n $"Starting $prog: "
          daemon $prog $CRONDARGS && success || failure
          RETVAL=$?
          [ "$RETVAL" = 0 ] && touch $LOCK_FILE
          echo
}
stop() {
         echo -n $"Stopping $prog: "
         if [ -n "`pidfileofproc $CROND`" ]; then
                 killproc $CROND
                 RETVAL=3
         else
         failure $"Stopping $prog"
         fi
         RETVAL=$?
```

```
        [ "$RETVAL" = 0 ] && rm -f $LOCK_FILE
        echo
}
reload() {
        echo -n $"Reloading $prog: "
        if [ -n "`pidfileofproc $CROND`" ]; then
                killproc $CROND -HUP
        else
                failure $"Reloading $prog"
        fi
        RETVAL=$?
        echo
}
case "$1" in
  start)
      start
        ;;
  stop)
      stop
        ;;
  restart)
        stop
        start
        ;;
  reload)
        reload
        ;;
  status)
      status $CROND
        ;;
  condrestart)
    if [ -f  $LOCK_FILE ]; then
      if [ "$RETVAL" = 0 ]; then
        stop
        sleep 3
```

```
        start
      fi
    fi
    ;;
  *)
      echo $"Usage: $0 {start|stop|status|reload|restart|condrestart}"
      RETVAL=3
esac
```

В этом сценарии определены три внутренние функции скрипта, которые выполняется в зависимости от того, какое значение задано первому позиционному параметру в строке при его вызове

Теперь перейдем к изучению системных сценариев, которые располагаются в домашних каталогах пользователей и содержащих такие настройки, которые активизируются при их регистрации.

Первым рассмотрим ~/.bash_profile. Напомним, что символ «~» означает домашний каталог пользователя. Таким образом, вызывается именно тот экземпляр файла, который относится к конкретному пользователю.

```
if [ -f ~/.bashrc ]; then
        . ~/.bashrc
fi
```

Первым в домашнем каталоге проверяется существование .bashrc. Если он существует, то исполняется. Таким образом, значению переменной PATH добавляется каталог, в котором можно поместить программы, исполняемые в сеансе конкретного пользователя.

```
PATH=$PATH:$HOME/bin
export PATH
```

Еще приведем команды, которые задают псевдонимы (aliases).

```
alias rm = ″rm –i″
alias cp = ″cp –i″
alias mv = ″mv –i″
alias ls = ″ls --color″
```

Такие три первых команды задают псевдоним командам удаления, копирования и перемещения файлов. После выполнения таких действий, названные команды будут исполняться с опцией обязательного запроса на выполнение операции. Четвертая команда

предписывает выполнение команды ls с цветовым выделением элементов файловой системы.

Сценарий ~/.bash_logout в нашем случае содержит только оператор clear

```
# ~/.bash_logout
   clear
```

Но такой файл всегда присутствует в системе и может выполнять разнообразные действия при завершении сеанса пользователя. Последний может добавить в него свои собственные команды.

В заключении приведем операторы, которые могут располагаться в разных сценариях, но, скорее всего, в /etc/profile или ~/.profile.

```
PATH = "PATH:/usr/X11R6/bin"
export PATH
```

К значению переменной окружения PATH добавляется подкаталог, где содержатся исполнимые файлы графической системы.

```
umask 022                # 777-022 = 755 ⇔ rwx  r-x  r-x
```

Такой командой устанавливается права доступа к вновь создаваемым файлам. Установленное значение 022 команды umask определяет то, что вновь созданному файлу будут присвоены права 755 (777-022 = 755) в цифровой записи, а в символьной rwx r-x r-x. При этом владелец файла может осуществлять с ним любые действия, а остальные пользователи - только чтение и выполнение.

```
if ["$TERM" = " " -o  "$TERM" = "unnown"]; then       #   -o (ИЛИ)
                    TERM = "linux"
fi
```

Этим оператором устанавливается значение linux переменной окружения TERM. В выражении применена логическая связка ИЛИ. Поэтому команда выполняется, если переменная TERM, имела либо пустое значение, либо unnown.

```
if [ "$SHELL" = "/bin/pdksh/"] ; then
                    PS1="! $"
elif ["$SHELL" = "/bin/ash"] ; then
                    PS1="$ "
else
                    PS1="\h:\w \$"   # w - рабочая директория
                                     # h – имя хоста
fi
```

Такой группой операторов присваивается значения переменной окружения PS1. Она определяет, какие символы будут в командной строке (строке приглашения), до той позиции, где пользователь будут набирать тест команды.

PS2 = ">"

Такая переменная определяет, что будет записано в строке приглашения команд, обеспечивающих ввод текста. К таким относится, например, оператор cat.

4.4. Реестр Windows XP

4.4.1. Назначение и структура реестра

Реестр Windows XP (Windows registry) представляет собой реляционную базу данных, в которой аккумулируются вся необходимая для нормального функционирования компьютера информация о настройках операционной системы, а также об используемом совместно с Windows программном обеспечении и оборудовании [5]. В процессе развития операционных систем Windows реестр пришел на смену конфигурационным файлам (INI-файлам) и был призван снять неудобства и ограничения, связанные с их использованием. В отличие от операционных систем прежних поколений информация в реестре Windows XP хранится в бинарном (двоичном) представлении, что позволяет не только помещать в реестр значительно больший объем информации, но и увеличить скорость работы с ним.

С реестром взаимодействуют следующие компоненты операционной системы [5]:

1. Программы установки (Windows Setup). Каждый раз при запуске программы Windows Setup или установочных программ для аппаратных и программных средств происходит добавление в реестр новых конфигурационных данных. Правильно разработанные программы установки считывают информацию реестра, чтобы определить. Присутствуют ли в системе компоненты, обязательные для успешного завершения установки. Реестр позволяет приложениям совместно использовать конфигурационную информацию и предоставляет им больше возможностей взаимодействия между собой.

2. Распознаватель (Recognizer). При запуске компьютера распознаватель аппаратных средств помещает в реестр список обнаруженных устройств. На компьютерах с процессорами Intel распознавание аппаратных средств осуществляется программой Ntdetect.com и ядром операционной системы Ntoskrnl.exe.

3. Ядро системы (Ntoskrnl.exe). При старте системы ядро извлекает из реестра сведения о загружаемых драйверах устройств и порядке их загрузки. Кроме того, ядро передает в реестр информацию о себе (номер версии и др.).

4. Драйверы устройств. Драйверы обмениваются с реестром параметрами загрузки и конфигурационными данными. Каждый драйвер устройства должен сообщить об используемых им системных ресурсах, включая аппаратные прерывания и каналы DMA.

Приложения и драйверы устройств могут считывать эту информацию из реестра, предоставляя пользователям интеллектуальные программы инсталляции и конфигурирования.

5. Административные средства. Эти средства, в том числе утилиты панели управления и оснастки, собранные в меню Администрирование, представляют собой удобные и безопасные (в части внесения ошибок) средства модификации реестра. Редактор реестра также полезен для просмотра реестра и для внесения изменений в конфигурацию системы.

6. Пользовательские профили (user profiles). Windows NT/2000/XP обеспечивают возможность создания множества пользовательских профилей. Вся информация, относящаяся к конкретному пользовательскому имени и ассоциированным с ним правам, хранится в реестре.

7. Аппаратные профили (hardware profiles), или профили оборудования. Реестр, в отличие от INI-файлов позволяет хранить множественные аппаратные конфигурации. Аппаратный профиль представляет собой набор инструкций, с помощью которого можно указать операционной системе, драйверы каких устройств должны загружаться при запуске компьютера.

Работа с реестром (его редактирование) позволяют пользователю [5]:

- решать проблемы, возникающие в процессе эксплуатации прикладного программного обеспечения, гибко настраивать режимы работы приложений;
- устранять неполадки в работе оборудования, вызванные некорректным использованием ресурсов операционной системы или драйверов различных устройств;
- настраивать параметры и ограничения пользовательской среды Windows, изменять заданные по умолчанию характеристики операционной системы;
- управлять быстродействием компьютера;
- перераспределять ресурсы операционной системы по усмотрению администратора компьютера;
- управлять конфигурацией компонентов Windows и системных сервисов, что позволяет оптимизировать работу операционной системы в зависимости от назначения компьютера и решаемых пользователем задач.

При редактировании системного реестра Windows XP в специальных программах он представляется в виде единой базы данных, имеющей жесткую иерархическую структуру. Однако на физическом уровне реестр неоднороден и состоит из множества файлов, каждый из которых отвечает за собственный объем представленной в этой базе

информации[1]. Следует учесть и тот факт, что некоторые из отображаемых в реестре сведений вообще не хранятся в виде файлов на диске, а помещаются в оперативную память компьютера в процессе загрузки операционной системы и утрачиваются в момент отключения питания. Такие разделы реестра называются энергозависимыми.

В частности к энергозависимым разделам реестра относится ветвь HKEY_LOCAL_MACHINE\HARDWARE[2], в котором находятся сведения об оборудовании компьютера и системных ресурсах, назначенных устройствам: запросах на прерывание (IRQ), каналах прямого доступа к памяти (DMA) и диапазонах памяти ввода-вывода (I/O Range).

Другие части реестра, хранящие данные о базовой конфигурации операционной системы, ее настройках и параметрах, содержатся в системной папке *%SystemRoot%*\System32\Config. Файлы, хранящие данные о профилях пользователей, находятся в папке % *SystemRoot%*\Profiles. Данные, относящиеся к конкретным настройкам системы для каждого пользователя и данные об их персональной конфигурации рабочей среды, представлены в папках *%Drive%*\Documents and Settings\%UserName%, где *%Drive%* – имя раздела диска, на котором установлена операционная система, а %UserName% – папка, имя которой соответствует имени одного из зарегистрированных в системе пользователей.

Дополнительные сведения о локальных пользователях системы по умолчанию содержатся в папке *%Drive%*\Documents and Settings\LocalService, а данные о настройках системы для удаленных пользователей – в папке *%Drive%*\Documents and Settings\NetworkService.

Нужно отметить, что для нормальной работы с реестром Windows знания его устройства на физическом уровне не требуется, поскольку программы, предназначенные для взаимодействия с реестром, позволяют его редактировать как единый файл. На логическом уровне реестр представляется иерархической структурой из четырех ступеней. Верхний уровень образуют так называемые *ветви* (Hive Keys), которые принято обозначать аббревиатурой HKEY_, где за символом подчеркивания следует название ветви. Всего в реестре пять ветвей: HKEY_CLASSES_ROOT, HKEY_CURRENT_USER, HKEY_LOCAL_MACHINE, HKEY_USERS и HKEY_CURRENT_CONFIG.

Второй ступенью являются *разделы*, или *ключи* (Keys). Они отображаются в программе Редактора реестра в виде подпапок ветвей HKEY_. Функционально ключи можно разделить на две условные категории: определяемые системой (их менять нельзя) и

---

[1] Постоянные компоненты реестра называются ульями (hives) или кустами.
[2] Не является ульем.

определяемые пользователем – эти имена могут быть изменены администратором, и такие изменения не приведут к каким-либо фатальным последствиям.

Ступенью ниже следуют *подразделы* (Subkeys). Их имена также могут быть определены системой или пользователем. Последней ступенью в иерархической структуре системного реестра являются параметры (Values) – элементы реестра, содержащие саму информацию, определяющую работу операционной системы и компьютера в целом. Параметры представляют собой цепочку *имя параметра – тип данных – значение*.

Типы данных, определенные для параметров реестра приведены ниже в табл. 4.3. Для значений параметров реестра вне зависимости от того, к какому типу данных они относятся, в программе Редактор реестра имеется набор встроенных мастеров, позволяющих легко изменять данные любого типа.

Табл.4.3. Список типов данных.

Наименование	Тип данных	Описание
REG_BINARY	Двоичный	Аппаратные компоненты используют информацию в виде двоичных данных. Редакторы реестра отображают ее в шестнадцатеричном формате
REG_DWORD	Числовой	Числа (4 байта), параметры драйверов устройств и сервисов. Редакторы реестра отображают ее в двоичном, шестнадцатеричном и десятичном формате
REG_DWORD_LITTLE_ ENDIAN	Числовой	Эквивалент REG_DWORD с младшим байтом в начале числа
REG_DWORD_BIG_ ENDIAN	Числовой	Эквивалент REG_DWORD со старшим байтом в начале числа
REG_SZ	Строковый	Описания компонентов
REG_EXPAND_SZ	Строковый	Расширяемая строка данных. Текст, содержащий переменную, которая может быть заменена при вызове со стороны приложения
REG_MULTI_SZ	Многостроковый	Списки текстовых строк в формате, удобном для восприятия
REG_LINK	Строковый	Символическая ссылка Unicode.

		Предназначен для внутреннего использования.
REG_NONE	Нет типа	Параметр не имеет определенного типа данных
REG_RESOURCE_LIST	Строковый	Список аппаратных ресурсов
REG_RESOURCE_ REQUIREMENTS_LIST	Строковый	Список необходимых аппаратных ресурсов
REG_FULL_RESOURCE_ DESCRIPTOR	Строковый	Дескриптор (описатель) аппаратного ресурса

4.2.2. Средства управления реестром

Основным инструментом администрирования реестра Windows является программа Редактор реестра. Кроме Редактора реестра в Windows XP имеются средства консоли управления – специальные средства, позволяющие управлять многими аппаратными, программными и сетевыми компонентами Windows. Часть программных средств, позволяющих изменять настройки реестра, расположена в Панели управления Windows XP. Практически все параметры операционной системы, связанные с окружением пользовательской среды, ее возможностями и ограничениями, можно изменить при помощи Редактора системных политик. Возможно и управление реестром Windows из командной строки посредством консольного интерпретатора команд CMD. Можно также создать командный (пакетный) файл, включающий набор команд среды CMD. Сам Редактор реестра поддерживает собственный набор команд, последовательность которых может быть записана в файл. Путем использования таких файлов та или иная информация может быть автоматически добавлена в реестр.

По умолчанию Редактор реестра (утилита regedit.exe) в процессе установки операционной системы копируется в каталог *%SystemRoot%*. Для его запуска нужно выполнить команды Пуск – Выполнить, а затем в поле Открыть набрать команду regedit и щелкнуть на кнопке ОК. Рабочее окно Редактора реестра разделено на две части (рис. 4.4). В левой (Панель разделов) отображается дерево ветвей, разделов и подразделов, из которых состоит реестр Windows XP. В правой части отображаются параметры, назначенные для выбранного элемента реестра (в данном случае – Select).

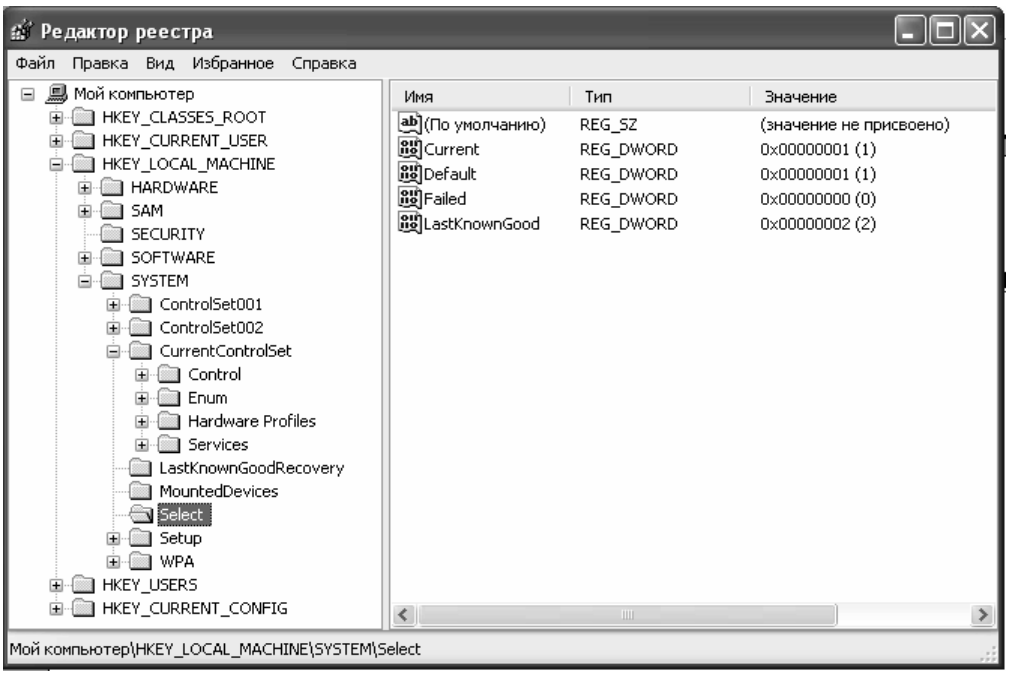


Рис. 4.4. Пример отображения вервей Реестра.

Перемещаясь при помощи мыши по иерархической структуре реестра в левой части окна, в правой части можно просматривать параметры каждого из разделов. Чтобы изменить значение какого-либо параметра, нужно дважды щелкнуть мышью на его значке. В открывшемся окне (рис. 4.5) можно установить требуемое значение параметра.

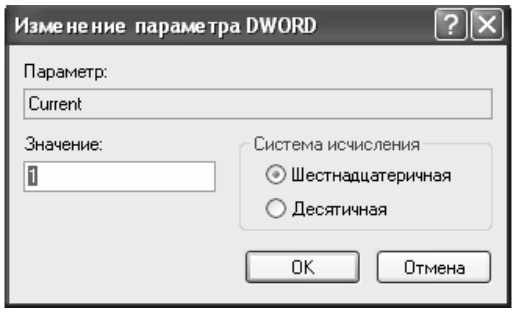


Рис. 4.5. Изменение параметра в Реестре.

В нижней части окна расположена строка состояния, которая отображает размещение выделенного элемента реестра. Включение и отключение строки состояния осуществляется установкой или снятием флажка Строка состояния в меню Вид.

Строка меню содержит пять подменю: Файл, Правка, Вид, Избранное и Справка.

Меню Файл позволяет осуществлять операции экспорта-импорта файла реестра и его отдельных компонентов, выводить содержимое какого-либо фрагмента реестра на печать или открывать для редактирования файлы реестра других компьютеров, расположенных в локальной сети.

Меню Правка позволяет выполнять операции создания, удаления, редактирования, переименования логических элементов реестра, выполнять процедуры поиска, настраивать режимы безопасности.

Меню Вид управляет настройками интерфейса Редактора реестра.

Меню Избранное позволяет устанавливать закладки на различные разделы и подразделы реестра для последующего быстрого перехода к ним.

Меню Справка вызывает встроенную справочную систему Редактора реестра.

Большое удобство для работы пользователя предоставляют встроенные мастера редактора, позволяющие удобно редактировать параметры, хранящие значения с различными типами данных. Таких мастеров – четыре по количеству типов данных, характерных для значений параметров реестра. Соответствующий мастер вызывается автоматически двойным щелчком по параметру реестра. Вызванный мастер позволяет изменить значение параметра, причем значение может быть изменено только с сохранением текущего типа данных. Пример окна мастера для настройки параметра DWORD показан на рис. 4.5.

Работа с реестром при помощи программы Редактор реестра состоит из следующих действий:

- просмотр разделов и подразделов (в данном случае механизм работы сходен с порядком работы пользователя в программе Проводник);
- поиск информации в реестре (меню Правка – Найти – Найти далее);
- быстрый переход к выбранному разделу реестра (используется механизм закладок, меню Избранное – Добавить в избранное);
- добавление новых разделов и параметров реестра[3] (меню Правка – Создать – Раздел) или (Правка – Создать, далее из меню выбирается тип создаваемого параметра, например, Строковый параметр);
- переименование, удаление и копирование раздела или параметра реестра (щелчок правой клавишей мыши по объекту переименования и использование контекстного меню).

Для того чтобы назначить пользователям или группам пользователей разрешения на доступ к какому-либо разделу реестра, нужно выполнить следующие действия (обладая правами администратора):

1. Открыть редактор реестра.
2. Выделить ветвь, раздел или подраздел, для которого требуется настроить разрешения.
3. В меню Правка выбрать команду Разрешения. На экране появится диалоговое окно настройки параметров безопасности для выбранного раздела реестра (рис. 4.6).

[3] Следует соблюдать осторожность, так как изменения могут повлиять на работу приложений, оборудования компьютера и операционную систему.

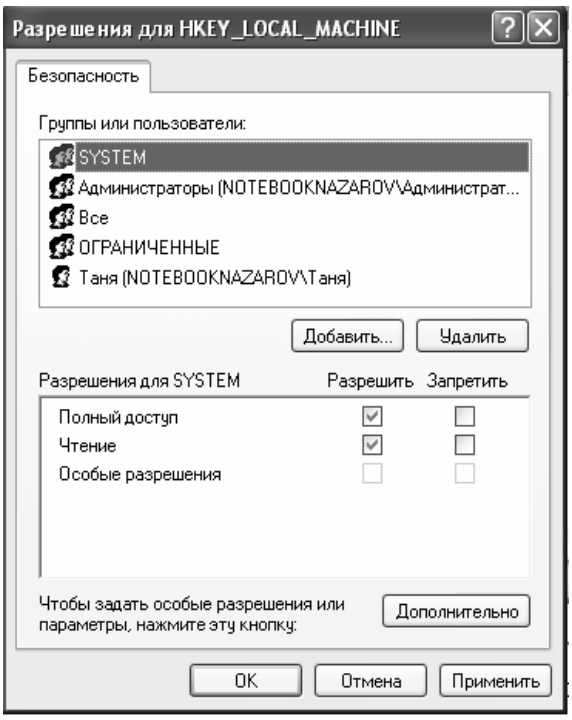


Рис. 4.6. Окно настройки безопасности.

4. Щелкнуть мышью на кнопке Дополнительно. Откроется окно Дополнительные параметры безопасности (рис. 4.7).

5. На вкладке Разрешения этого окна в списке Элементы разрешений выбрать компьютер, пользователя или группу. Чтобы добавить новый элемент в список нужно воспользоваться кнопкой Добавить.

6. Если необходимо настроить все разрешения для выбранного в списке пользователя или группы вручную, запретив автоматическое наследование разрешений от родительского объекта, следует сбросить флажок Наследовать от родительского объекта применимые к дочерним объектам разрешения, добавляя их к явно заданным в этом окне.

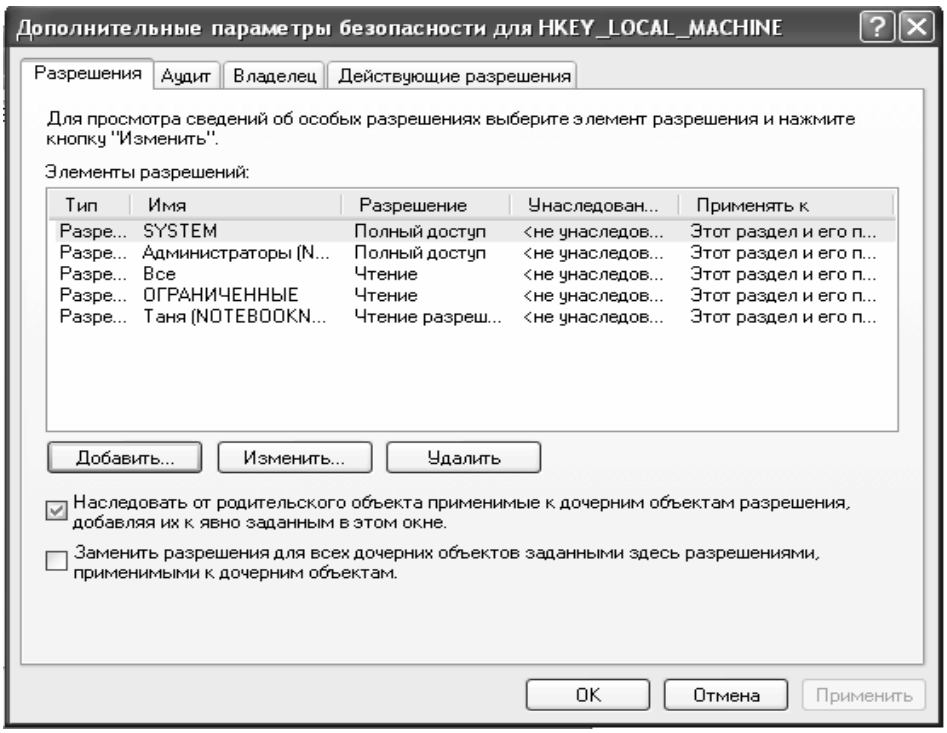


Рис. 4.7. Дополнительные параметры безопасности.

7. Для изменения списка разрешений для выбранного пользователя или группы щелкнуть на кнопке Изменить. В списке Разрешения открывшегося окна установить (сбросить) флажки, соответствующие разрешениям для данного пользователя или группы (рис. 4.8). Если флажок затенен, это означает, что разрешение наследуется. Чтобы изменить настройки такого разрешения, нужно выполнить действия, указанные в п. 6.

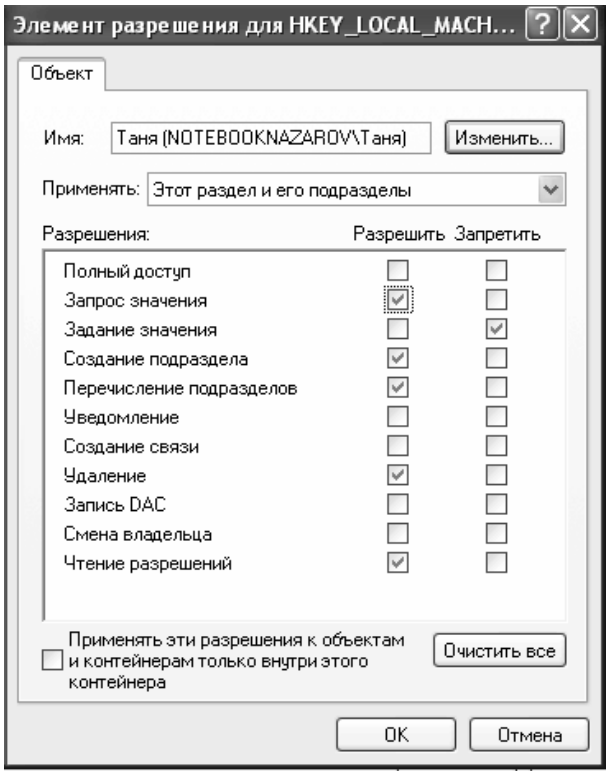


Рис. 4.8. Элемент разрешения для HKEY_LOCAL_MACHINE.

Для того чтобы дочерние объекты реестра автоматически наследовали разрешения от родительских объектов, в диалоговом окне Дополнительные параметры безопасности следует установить флажок Заменить разрешения для всех дочерних объектов заданными здесь разрешениями, применимыми к дочерним объектам. Изменения в настройках разрешений вступят в силу после нажатия на кнопку Применить, а затем ОК.

Контроль над действиями пользователей с реестром администратор может осуществлять посредством аудита. Для этого администратор выполнить следующие действия;

1. Открыть Редактор реестра.
2. Выделить ветвь, ключ или подраздел, для которого нужно настроить параметры аудита.
3. В меню Правка выбрать команду Разрешения. На экране появится окно настройки параметров безопасности для выбранного раздела реестра.
4. Щелкнуть по кнопке Дополнительно. Откроется окно Дополнительные параметры безопасности.
5. Перейти на вкладку Аудит.
6. Двойным щелчком выбрать в списке Элементы аудита пользователя или группу. Если список пуст, следует выполнить следующие действия:

- щелкнуть мышью на кнопке Добавить. На экране появится окно Выбор: Пользователи или Группа;
- для выбора типа объекта щелкнуть на кнопке Типы объектов и выбрать требуемый вариант (в нашем случае – Пользователи) и щелкнуть на кнопке ОК;
- произойдет возврат в окно Выбор: Пользователи или Группа, в котором можно ввести имена пользователей, после чего щелкнуть на кнопке Проверить имена ОК;
- на экране появится окно (рис. 4.9) Элементы аудита для выбранной ветви реестра (в нашем случае HKEY_LOCAL_MACHINE).

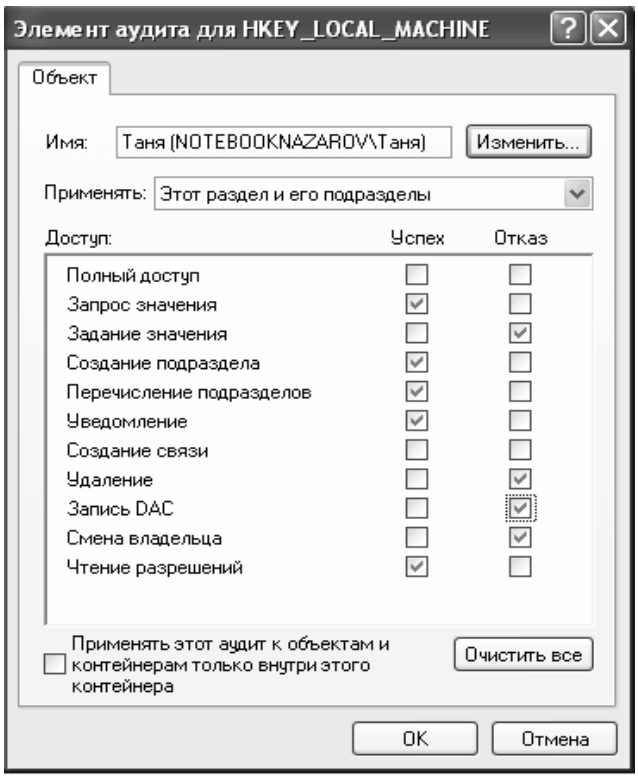


Рис. 4.9. Элемент аудита HKEY_LOCAL_MACHINE.

7. Выбрав пользователя, следует установить флажки Успех или Отказ в группе Доступ.

Значения параметров для различных событий аудита реестра приведены в табл. 4.4. Результаты аудита можно просматривать в системном журнале Безопасность с помощью оснастки Просмотр событий.

Табл.4.4. Параметры для различных событий аудита.

Событие	Тип аудита
Запрос значения	Аудит всех попыток чтения параметров из раздела реестра
Задание значения	Аудит всех попыток задания значений параметров в разделе реестра
Создание подраздела	Аудит всех попыток создания подразделов в данном разделе реестра
Перечисление подразделов	Аудит всех попыток определения подразделов данного раздела реестра
Уведомление	Аудит событий уведомления из подраздела реестра
Создание ссылки	Аудит попыток создания символьной ссылки в указанном реестре
Удаление	Аудит попыток удаления объектов реестра
Запись DAC	Аудит всех попыток записи в избирательную таблицу управления доступом для данного раздела

Смена владельца	Аудит всех попыток смены владельца выбранного раздела
Чтение разрешений	Аудит всех попыток открытия необязательной таблицы управления доступом для данного раздела

4.2.3. Резервное копирование и восстановление реестра

Реестр является жизненно важным компонентом системы, и единственная ошибка в его редактировании может привести к проблемам в работе с операционной системой, включая и проблемы с загрузкой. Если реестр окажется поврежденным, а в распоряжении пользователя не окажется работоспособной и пригодной к использованию резервной копии, то во многих случаях не остается ничего другого как переустановка операционной системы. Поэтому прежде чем начать редактирование реестра, необходимо выполнить его резервное копирование.

Windows XP предоставляет множество методов резервного копирования, часть из которых рассматривается ниже.

Экспорт файла реестра или его компонентов

Чтобы экспортировать в файл системный реестр целиком или какой-либо из его компонентов, потребуется выполнить следующую последовательность действий:

1. Открыть программу Редактор реестра.
2. Выбрать в панели Редактора реестра ветвь или раздел (подраздел), который необходимо экспортировать. Если нужно записать в файл реестр целиком, нужно выделить щелчком мыши корневой объект дерева реестра – Мой компьютер.
3. В меню Файл выбрать команду Экспорт и в открывшемся диалоговом окне сохранения файла указать место его сохранения и имя (рис. 4.10).
4. Щелкнуть на кнопке Сохранить.

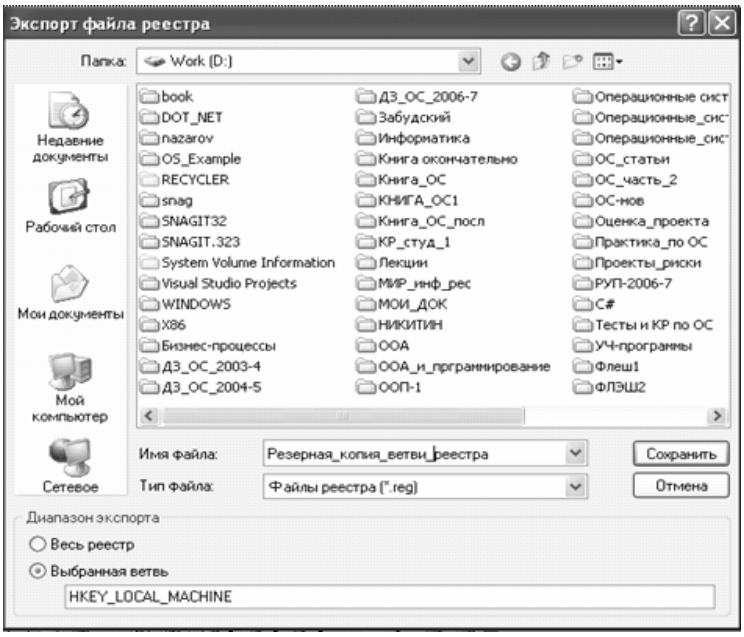


Рис. 4.10. Импорт файла реестра или его компонентов.

Альтернативой предложенному способу экспорта реестра может стать использование консольной команды REG EXPORT. Синтаксис команды приведен на рис. 4.11.

```
C:\WINDOWS\system32\cmd.exe

C:\>REG EXPORT /?

Программа редактирования системного реестра из командной строки, версия 3.0
(C) Корпорация Майкрософт, 1981-2001. Все права защищены


REG EXPORT <раздел> <имя файла>

  <раздел>        Полный путь к разделу реестра в виде: КОРЕНЬ\Подраздел
                  (только для локального компьютера).
    <КОРЕНЬ>      Корневой раздел. Значения: [ HKLM | HKCU | HKCR | HKU | HKCC ].

    <подраздел>   Полный путь к разделу реестра в выбранном корневом разделе.
  <имя файла>     Имя файла на диске для экспорта.

Примеры:

  REG EXPORT HKLM\Software\MyCo\MyApp AppBkUp.reg
    Экспортирует все подразделы и значения параметров раздела MyApp
    в файл AppBkUp.reg

C:\>_
```

Рис. 4.11. Описание команда REG EXPORT.

Обладая полномочиями администратора, можно импортировать данные реестра или его компоненты из ранее сохраненного REG-файла. Для этого достаточно дважды щелкнуть на значке соответствующего REG-файла или использовать консольную команду REG IMPORT. Но можно воспользоваться и Редактором реестра. Для этого нужно выполнить следующие действия:

1. Запустить программу Редактор реестра.
2. В меню Файл выбрать команду Импорт и в открывшемся диалоговом окне Импорт файла реестра указать место его расположения и имя (рис. 4.12).
3. Щелкнуть мышью на кнопке Открыть.

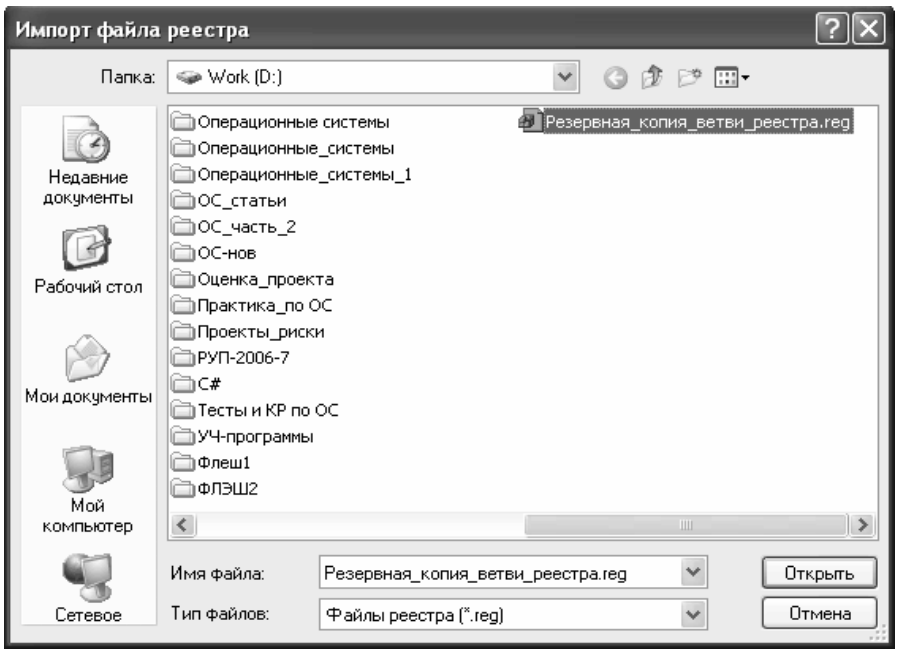


Рис. 4.12. Диалоговое окно импорта файла Реестра.

4.2.4. Альтернативные методы резервного копирования реестра Windows XP

В Windows существуют программы восстановление системы. Они создают точки восстановления системы. При создании таких точек программа автоматически размещает в резервной копии ключи реестра, отвечающие за пользовательские настройки системы, а также список установленного на компьютере программного обеспечения и оборудования. В процессе аварийного восстановления системы эти данные загружаются в реестр из резервной копии.

Для создания резервной копии реестра на каком-либо внешнем носителе информации (стример, устройство для записи компакт-дисков, DVD и др.) можно воспользоваться программой архивации.

Имеется также возможность копирования и восстановления реестра с помощью Консоли восстановления. Для создания резервной копии системного реестра нужно выполнить в ней следующую последовательность команд:

```
md tmp
copy c:\windows\system32\config\system  c:\windows\tmp\system.bak
copy c:\windows\system32\config\software  c:\windows\tmp\software.bak
copy c:\windows\system32\config\sam  c:\windows\tmp\sam.bak
copy c:\windows\system32\config\security  c:\windows\tmp\security.bak
copy c:\windows\system32\config\default  c:\windows\tmp\default.bak
```

Не рекомендуется создавать файлы резервной копии реестра с помощью Консоли восстановления за пределами папки установки Windows, поскольку впоследствии доступ к

ним может быть затруднен. Восстановить ранее сохраненный таким образом реестр можно обратным перемещением файлов в папку C:\windows\system32\config.

4.2.5. Очистка реестра

Записи приложений содержатся в разделах HKEY_CURRET_USER\Software (персональные параметры пользователя) и HKEY_LOCAL_MACHINE\SOFTWARE (параметры, общие для всех пользователей компьютера). После удаления многие приложения оставляют в реестре свои параметры – как правило, системные или пользовательские. Это может привести к следующим проблемам [5].

1. Неэффективное использование системных ресурсов. Windows “верит” всему, что записано в реестре. Если реестр содержит запись о том, что для обработки файла или решения задачи в системе имеется определенное приложение, то Windows будет искать его, потребляя системные ресурсы, даже если приложения на самом деле нет.

2. Нарушение надежности системы. Проблемы с устойчивостью системы начинают проявляться, когда компьютер пытается использовать несуществующий ресурс. Вообще система успешно восстанавливается после одиночных и даже двойных ошибок. Но по мере накопления в реестре некорректных записей, правильно реагировать на ошибки становится труднее, и начинают наблюдаться проблемы (компьютер ведет себя странным образом вплоть до краха системы).

3. Нарушение надежности приложений. Приложения получают данные из целого ряда источников – от других приложений, драйверов устройств, операционной системы. Что произойдет, если одно приложение посредством Windows посылает запрос другому приложению, но оказывается, что оно удалено из системы? Если запрашивающее приложение не обладает хорошим механизмом обработки ошибок, оно потерпит крах по причине некорректной записи в реестре.

4. Нарушение безопасности. Безопасность компьютера может пострадать из-за “мусора” в реестре. Вирусам проще скрыть себя среди устаревших записей, а антивирусным приложениям труднее их обнаружить. Более того, устаревшие записи способствуют проникновению в систему вредоносных программ. Например, загрузка из-за некорректной записи в реестре в память ненужной библиотеки DLL делает возможными атаки на систему через эту библиотеку.

Специальные средства очистки реестра помогают в поиске устаревших данных установки приложений, однако не гарантируют обнаружения всех этих данных. Большинство средств очистки ищут некорректные параметры, но с этими параметрами

часто связано множество записей. Однако для самостоятельного поиска таких записей можно воспользоваться рядом критериев:

- имя разработчика;
- имя приложения;
- имя исполняемого файла приложения;
- имена DDL;
- GUID компонентов (Globally Unique Identifier – глобально уникальный идентификатор);
- URL приложения;
- уникальные данные настройки приложения.

Поскольку реестр столь важен для работы Windows, следует не допускать ошибок при его редактировании. Поэтому прежде чем начать работу, нужно сохранить все ветви реестра и заархивировать их.

Хорошие утилиты для очистки реестра способны значительно упростить обнаружение некорректных записей и снизить вероятность удаления нужных. Утилита RegClean (http://www.majorgeeks.com/download458.html) является продуктом компании Microsoft. При первом запуске она долго сканирует реестр и выдает сообщение о готовности к устранению ошибок (рис. 4.13). Недостатком данной утилиты является отсутствие списка сделанных исправлений. Полезной особенностью утилиты является создание файла UNDO.REG, с помощью которого можно вернуть реестр к предыдущему состоянию. Для этого достаточно дважды щелкнуть на этом файле.

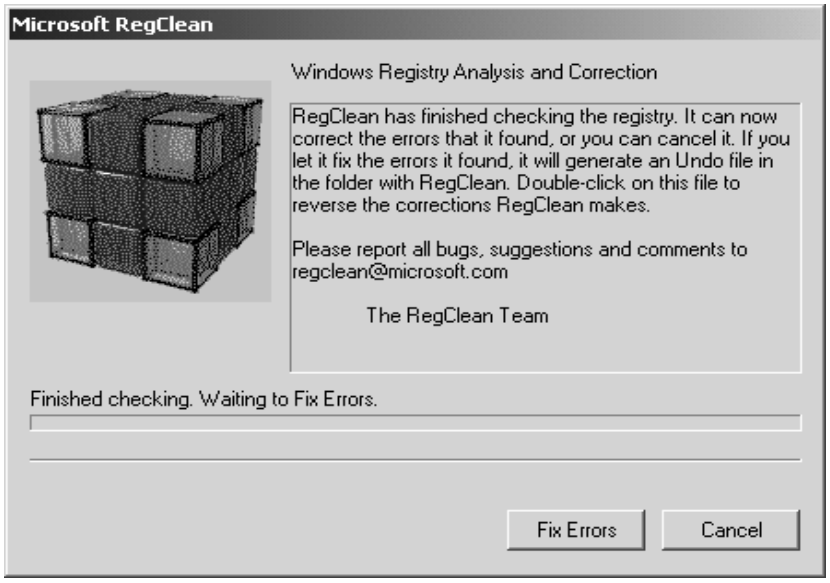


Рис. 4.13. Начальный экран диалога утилиты RegClean.

Утилита Registry Mechanic (http://www.winguides.com/regmech/) позволяет создать резервную копию реестра до начала сканирования. После запуска утилиты (рис. 4.14) можно увидеть большой список сканируемых объектов.

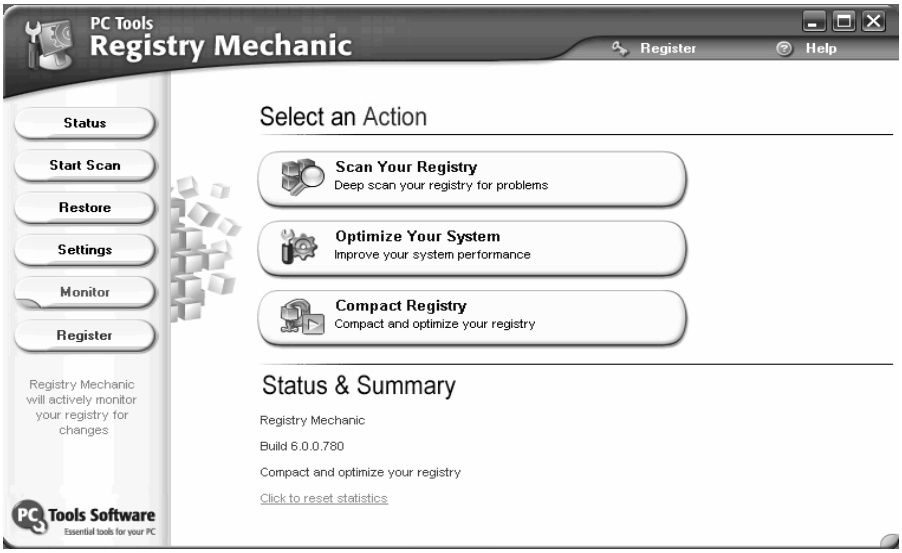


Рис. 4.14. Начало работы программы Registry Mechanic.

Особенность утилиты – возможность задания такого режима работы, при котором реестр сканируется при каждом запуске Windows. Сканирование, выполняемое утилитой Registry Mechanic, отличается чрезвычайной полнотой, но, по мнению автора книги [44], вряд ли является лучшей утилитой. Для восстановления реестра достаточно щелкнуть на кнопке Backup.

Утилита TweakNow RegCleaner (http://www.tweaknow.com/RegCleaner.html) обнаруживает в реестре наиболее распространенные ошибки (рис. 4.15). Ее возможностей вполне достаточно для домашних и офисных компьютеров. Хотя эта утилита находит в реестре меньше ошибок по сравнению с Registry Mechanic, но ее быстродействие значительно выше. Одна из функций TweakNow RegCleaner – индикация безопасности или небезопасности удаления записей реестра. Другие утилиты подобной функции не предоставляют. TweakNow RegCleaner автоматически создает резервную копию реестра перед удалением любых некорректных записей. При этом поддерживается лишь одна резервная копия, что при наличии нескольких вариантов очистки вынуждает пользователя тестировать их поочередно.

Рис. 4.15. Программа RegCleaner.

4.2.6. Редактирование реестра

Утилита Установка и удаление программ на панели управления предназначена для установки, удаления или модификации приложений, установленных в Windows. Несмотря удобство и простоту ее использования, в работе этой программы могут возникать неполадки, вызванные некорректно или не полностью удаленными приложениями. В частности, если приложение удалено некорректно, то ссылка на него по-прежнему будет фигурировать в списке Установленные программы.

При этом любая попытка воспользоваться мастером установки удаления приложений будет приводить к появлению сообщений об ошибке, информирующих пользователя об отсутствии файлов, требующихся для корректного удаления приложения и невозможности завершить процедуру удаления. Несуществующие приложения по-прежнему останутся в списке.

Чтобы устранить эту проблему и удалить несуществующие приложения из списка установленных программ, нужно выполнить следующие действия:

1. Запустить Редактор реестра и раскрыть ключ HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall.

2. В составе этого ключа найти вложенный ключ, соответствующий приложению, которое требуется удалить из списка установленных программ. Если наименование программы не следует с очевидностью из имени вложенного ключа, нужно просмотреть содержимое всех вложенных ключей списка. В составе каждого из вложенных ключей имеется параметр DisplayName, значение которого представляет собой строки, отображаемые мастером установки и удаления программ.

3. Обнаружив ключ, в составе которого имеется параметр DisplayName, задающий имя приложения, которое должно быть удалено из списка установленных программ, нужно удалить этот ключ вместе со всем его содержимым.

4. Закрыть редактор реестра и убедиться в том, что мастер установки и удаления программ больше не отображает несуществующего приложения.

Следует заметить, что существует вероятность того, что некоторые из файлов некорректно удаленного приложения все же останутся в системе. Чтобы полностью удалить такое приложение, необходимо удалить из системы все его файлы и все параметры реестра, относящиеся к этому приложению.

Ускорение работы системы с памятью

Если на компьютере установлена оперативная память достаточно большого объема (не менее 256 Мбайт) и не предполагается запускать одновременно множество приложений, то можно значительно повысить производительность компьютера, ликвидировав файл подкачки. Однако нужно понимать, что это может привести к снижению надежности функционирования компьютера. Чтобы решить, стоит ли отказываться от файла подкачки, нужно определить, сколько памяти потребляет система в каждый момент времени. Если свободная память составляет хотя бы 40% от ее общего объема, то отключение подкачки возможно [44]. Если же после этого система будет давать сообщения о недостатке оперативной памяти, нужно снова включить подкачку или установить оперативную память большего объема.

Для отключения файла подкачки нужно выполнить следующие действия:

1. Запустить Редактор реестра и найти ключ HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\Memory Management (рис. 4.16).

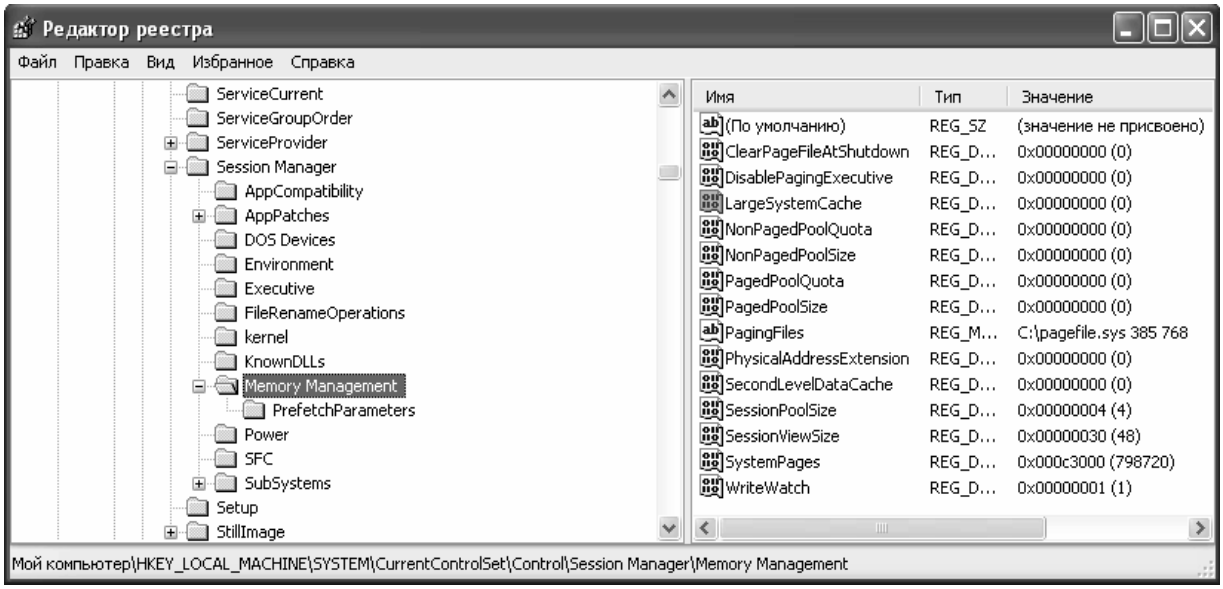


Рис. 4.16. Изменение значения Memory Management.

2. Установить значение параметра DisblePagingExecutive равным 1. Это будет указанием системе на то, что данные следует (в том числе ядро системы и драйверы) хранить в оперативной памяти, а не сбрасывать их на жесткий диск (рис. 4.17).

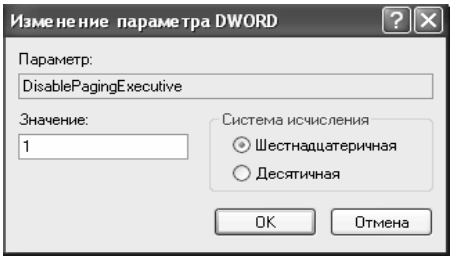


Рис. 4.17. Изменение значения DisblePagingExecutive.

3. Установить значение параметра LargeSystemCache равным 1. В этом случае операционной системе будет выделен участок оперативной памяти минимальным объемом в 4 Мбайт (по умолчанию выделяется 8 Мбайт) для кэширования ядра и окружения системы, что увеличит скорость доступа к этим компонентам. Это укажет Windows на то, что ее ядро следует хранить в оперативной памяти.

4. Создать новое значение типа DWORD с именем IOPageLockLimit. Этот параметр управляет объемом данных при их постраничной передаче и, таким образом, влияет на скорость выполнения операций ввода-вывода. Установить его равным 10000, если объем оперативной памяти не менее 256 Мбайт. Если объем памяти не менее 512 Мбайт, значение параметра может быть равным 40000.

Далее следует опробовать новые параметры, чтобы удостовериться в отсутствии ошибок, связанных с недостатком оперативной памяти. После этого нужно снова открыть окно редактора реестра, найти параметр PagingFiles, удалить из него текст и перегрузить компьютер. Это приведет к установке нулевого размера файла подкачки. После перезагрузки нужно удалить с жесткого диска файл PageFile.sys.

Одним из важных факторов, определяющим работу Windows, является объем перемещаемой и неперемещаемой памяти. Для задания объема неперемещаемой памяти нужно установить значение параметра NonPagedPoolSize (в байтах). После перезагрузки компьютера можно проверить новое значение величины невыгружаемой памяти на вкладке Быстродействие Диспетчера задач (рис. 4.18).

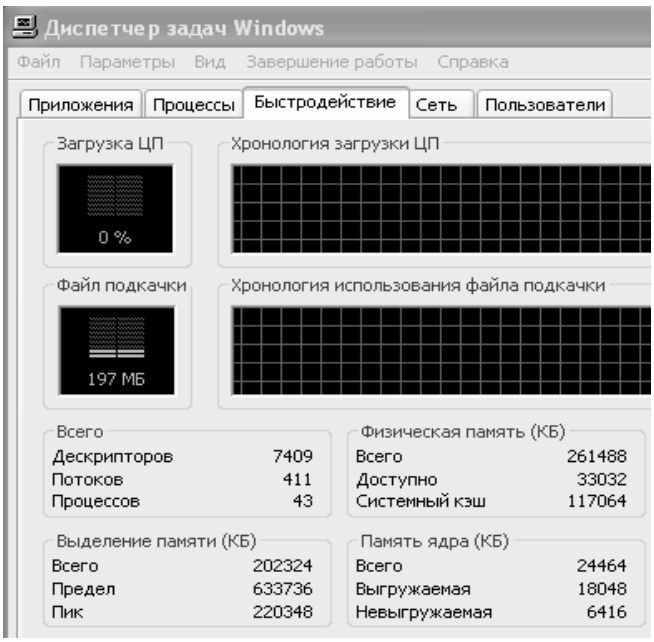


Рис. 4.18. Вкладка Быстродействие программы Диспетчера задач.

4.2.7. Повышение производительности системы

В составе операционной системы Windows XP имеется специальная утилита отладки Doctor Watson, которая предназначена для отслеживания сбоев в приложениях и выдачи соответствующих системных предупреждений. Утилита загружается автоматически с загрузкой системы и чаще всего себя никак не проявляет, а в случае проблем с приложением этот факт и так очевиден. Поэтому можно отказаться от услуг утилиты и высвободить память, занимаемую этой утилитой. Для этого необходимо выполнить следующие действия:

1. Запустить Редактор реестра.
2. Выбрать ветвь HKEY_LOCAL_MACHINE.
3. Перейти к подразделу SOFTWARE\Microsoft\WindowsNT\Current-Version\AeDebug (рис. 4.19).
4. Установить значение 0 для параметра Auto.

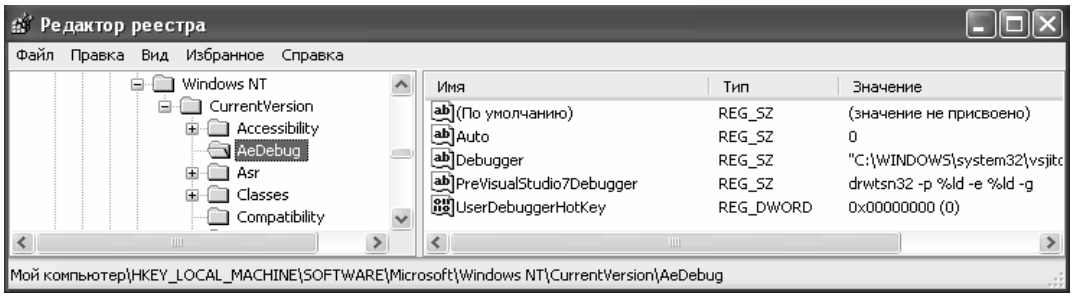


Рис. 4.19. Изменение значения параметра AeDebug.

Хранящиеся в разделе диска NTFS папки, которые содержат значительное количество файлов, открываются в Проводнике очень долго. Это происходит по той

причине, что при каждом обращении к папкам Windows автоматически обновляет системную метку последнего доступа к файлам, что при большом числе файловых объектов занимает много времени. Чтобы отключить эту функцию, нужно в ветви HKEY_LOCAL_MACHINE найти подраздел SYSTEM\CurrentControlSet\Control\FileSystem. Далее нужно создать параметр типа DWORD и присвоить ему значение 1 (рис. 4.20).

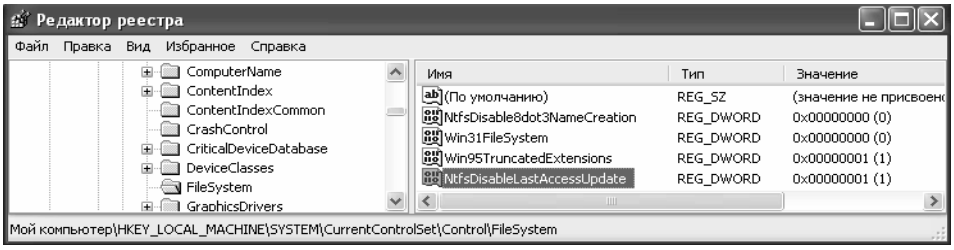


Рис. 4.20. Задание значение параметраNtfsDisableLastAccessUpdate.

Значительного ускорения работы пользовательской оболочки Windows XP, можно добиться, отключив стандартными средствами все визуальные эффекты (тень под курсором мыши, фоновые рисунки, темы, анимацию и т. д.). Для получения доступа к опциям визуальных настроек следует выполнить команды Панель управления – Система – Дополнительно – Параметры быстродействия. Затем на вкладке Визуальные эффекты нужно убрать или поставить флажки напротив нужных опций. Также в этом отношении эффективно уменьшение времени задержки при разворачивании меню и отключение анимации при сворачивании-разворачивании окон.

Используя системный реестр это можно сделать следующим образом. Выбрать ветвь HKEY_CURRENT_USER и перейти к подразделу ControlPanel\Desktop. Найти параметр MenuShowDelay (по умолчанию он равен 400) и установить его значение равным 0 (рис. 4.21).

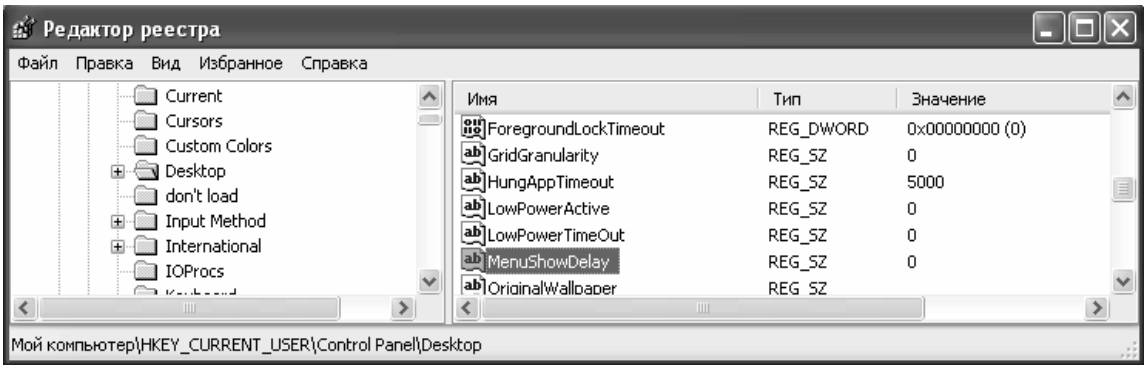


Рис. 4.21. Установление параметра MenuShowDelay.

Однако в этом случае список будет появляться мгновенно даже при быстром перемещении курсора над кнопкой. Это может быть неудобно, поэтому нужно определить оптимальное значение экспериментальным путем. Далее перейти в подраздел Windows

Metrics и установить значение параметра MinAnimate равным 0 (рис. 4.22) и перезагрузить компьютер.

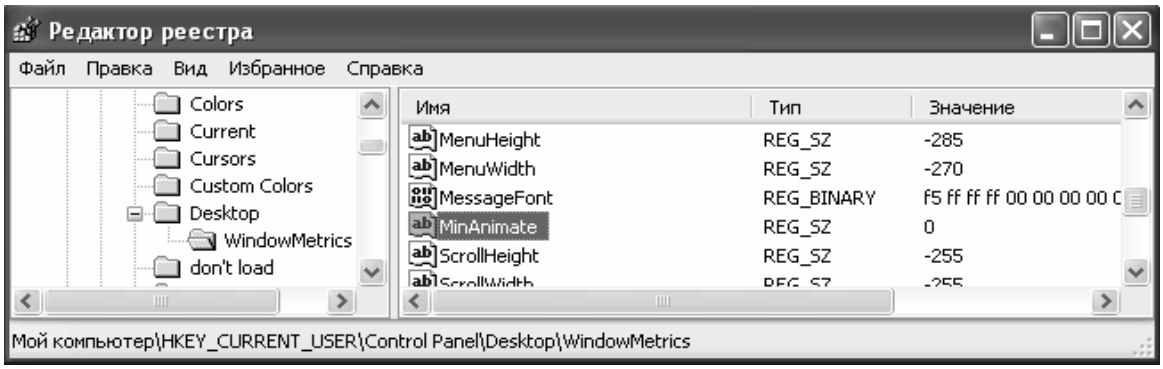


Рис. 4.22. Установление значения параметра MinAnimate.