МЕЖГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛОРУССКО-РОССИЙСКИЙ УНИВЕРСИТЕТ»

Кафедра «Автоматизированные системы управления»

ОПЕРАЦИОННЫЕ СИСТЕМЫ

Методические рекомендации к лабораторным работам для студентов специальности 1-53 01 02 «Автоматизированные системы обработки информации» дневной и заочной форм обучения

Могилев 2023

УДК 004.451
ББК 32.973-018.2
О60

Рекомендовано к изданию
учебно-методическим отделом
Белорусско-Российского университета

Одобрено кафедрой «Автоматизированные системы управления» «24» января 2023 г., протокол № 7

Составитель канд. техн. наук, доц. В. М. Ковальчук

Рецензент канд. техн. наук, доц. В. В. Кутузов

Методические рекомендации содержат базовые сведения о функционировании современных операционных систем.

Учебное издание

ОПЕРАЦИОННЫЕ СИСТЕМЫ

Ответственный за выпуск	А. И. Якимов
Корректор	Т. А. Рыжикова
Компьютерная верстка	М. М. Дударева

Подписано в печать 21.03.2023. Формат 60×84/16. Бумага офсетная. Гарнитура Таймс. Печать трафаретная. Усл. печ. л. 2,79. Уч.-изд. л. 2,94. Тираж 21 экз. Заказ № 351.

Издатель и полиграфическое исполнение:
Межгосударственное образовательное учреждение высшего образования
«Белорусско-Российский университет».
Свидетельство о государственной регистрации издателя,
изготовителя, распространителя печатных изданий
№ 1/156 от 07.03.2019.
Пр-т Мира, 43, 212022, г. Могилев.

Содержание

Введение

Цель методических рекомендаций к лабораторным работам по дисциплине «Операционные системы» заключается в овладении и закреплении студентами практических навыков работы с современными операционными системами.

Цель учебной дисциплины – изучение теоретических основ построения и функционирования современных операционных систем.

Дисциплина «Операционные системы» является неотъемлемой частью современных знаний информационных технологий и связана с рядом других дисциплин типовых учебных планов: «Мобильные приложения для информационных систем», «Основы информационной безопасности», «Системы управления базами данных», «Администрирование и программирование распределенных приложений», «Аппаратно-программное обеспечение ЭВМ и сетей» и др.

Методические рекомендации предназначены для изучения механизмов функционирования операционных систем, понятия процесса как средства описания функционирования любой операционной системы, режимов функционирования операционных систем, аппаратно-программных и информационных ресурсов вычислительной системы.

Полученные при изучении дисциплины знания и навыки будут востребованы при работе с конкретной операционной системой для ПЭВМ, генерации и конфигурировании конкретной операционной системы, использовании механизма прерываний в прикладных программах.

Каждая работа рассчитана на два часа.

Выполнение каждой работы производится в следующем порядке:

1) ознакомится с теоретическими положениями работы;

2) из таблицы «Варианты заданий для выполнения работы» согласно варианту, указанному преподавателем, выбрать задание и исходные данные, выполнить приведенные в нем задания и оформить отчет.

Отчет должен содержать: название и цель работы; постановку задачи; исходные данные; использованные технологии; результаты выполнения; анализ полученных результатов и выводы. В отчете можно привести также ответы на наиболее сложные вопросы, приведенные в конце каждой работы.

1 Лабораторная работа № 1. Изучение интерпретатора команд в ОС Windows

Цель работы: овладеть навыками работы с командной строкой в ОС Windows, научиться работать с потоками ввода/вывода в командной строке.

Методические указания

Интерфейс. Классификация интерфейсов. Как любое техническое устройство, компьютер обменивается информацией с человеком посредством набора определенных правил, обязательных как для машины, так и для человека. Эти правила в компьютерной литературе называются интерфейсом. Интерфейс – это правила взаимодействия операционной системы с пользователями, а также соседних уровней в сети ЭВМ. От интерфейса зависит технология общения человека с компьютером.

Современными видами интерфейсов являются:

– *командный интерфейс.* Командным интерфейс называется потому, что в этом виде интерфейса человек подает «команды» компьютеру, а компьютер их выполняет и выдает результат человеку. Командный интерфейс реализован в виде пакетной технологии и технологии командной строки;

– *WIMP-интерфейс* (Window – окно, Image – образ, Menu – меню, Pointer – указатель). Характерной особенностью этого вида интерфейса является то, что диалог с пользователем ведется не с помощью команд, а с помощью графических образов – меню, окон, других элементов. Хотя и в этом интерфейсе подаются команды машине, но это делается «опосредованно», через графические образы. Такой вид интерфейса называют также «*графическим*» интерфейсом;

– *SILK-интерфейс* (Speech – речь, Image – образ, Language – язык, Knowlege – знание). Этот вид интерфейса наиболее приближен к обычной, человеческой форме общения. В рамках этого интерфейса идет обычный «разговор» человека и компьютера. При этом компьютер находит для себя команды, анализируя человеческую речь и находя в ней ключевые фразы. Результат выполнения команд он также преобразует в понятную человеку форму. Этот вид интерфейса наиболее требователен к аппаратным ресурсам компьютера.

Технология командной строки. При этой технологии в качестве единственного способа ввода информации от человека к компьютеру служит клавиатура, а компьютер выводит информацию человеку с помощью алфавитно-цифрового дисплея (монитора). Эту комбинацию (монитор + клавиатура) стали называть терминалом, или консолью.

Команды набираются в командной строке. Командная строка представляет собой символ приглашения и мигающий прямоугольник – курсор. При нажатии клавиши на месте курсора появляются символы, а сам курсор смещается вправо. Команда заканчивается нажатием клавиши Enter (или Return.) После этого осуществляется переход в начало следующей строки. Именно с этой

позиции компьютер выдает на монитор результаты своей работы. Затем процесс повторяется.

Работа в командной строке операционной системы. После загрузки операционная система готова к работе. Под ее управлением можно запускать различные программы системного и прикладного назначения. Большинство пользователей работают с операционной системой не напрямую, а с помощью файловых (типа Norton Commander и т. п.) или операционных оболочек. Но, во-первых, некоторые программы по разным причинам не выполняются при загруженных оболочках. Во-вторых, вследствие большого количества операционных оболочек трудно ориентироваться в них. В-третьих, не всегда оболочку можно запустить (например, при временном выходе в операционную систему из выполняемой ей же программы.) При временном выходе в DOS поверх программы загружается файл Command.com, и оперативной памяти становится недостаточно для работы операционной оболочки. В-четвертых, при сбоях в операционной системе Microsoft Windows иногда доступна загрузка только в командной строке. Изложенное выше позволяет сделать вывод о необходимости умения пользоваться командной строкой операционной системы.

Приглашение операционной системы. В случае, когда операционная система готова к диалогу с пользователем, она выдает на экран приглашение (таблица 1.1).

Таблица 1.1 – Вид приглашения ОС.

Операционная система	DOS-Windows	UNIX
Вид приглашения	[диск]:\[путь]>	$

Например:

```
C:\WINDOWS\>
C>
```

Внешний вид приглашения можно изменить. Для MS-DOS в качестве элемента приглашения могут быть знаки «=», «|», «$», «>», «<», «текущее время», «текущая дата», «текущий диск и каталог», «текущий диск», «версия MS-DOS», «переход на следующую строку».

Также в системное приглашение MS-DOS можно вставлять другие ASCII-последовательности. В случае установки драйвера ANSI.SYS в файле Config.sys в приглашении также можно использовать и Esc-последовательности.

В системе LINUX есть основное и дополнительное приглашение. Дополнительное приглашение появляется при записи длинной команды, не помещающейся в командной строке. Каждая новая строка начинается с нового дополнительного приглашения.

В приглашении LINUX могут быть любые символы и следующие последовательности (текущая дата, действующая оболочка и ее версия, время суток, имя пользователя, текущий (рабочий) каталог). Для установки желаемого

приглашения необходимо поменять некоторые переменные окружения. Системное приглашение можно сменить с помощью команды PROMPT. Формат команды:

prompt текст

Для сброса всех установок команды PROMPT необходимо в командной строке набрать команду PROMPT без параметров.

Запуск из командной строки. Для того чтобы запустить команду на выполнение, необходимо ввести (набрать) ее имя на клавиатуре, одновременно указав все необходимые параметры и опции, и нажать клавишу Enter. После ее нажатия команда запускается на выполнение и при работе с нею происходит переназначение клавиш и устройств, используемых программой, и пользователь уже работает не с операционной системой, а с прикладной программой. Порядок работы в ней определяется разработчиком и описывается в соответствующей документации, поставляемой вместе с программой.

Редактирование командной строки в различных операционных системах. Для удобства работы с командной строкой операционной системы, например, для корректировки введенной команды, запуска предыдущей, для других целей, применяют клавиши редактирования командной строки (таблица 1.2).

Таблица 1.2 – Клавиши редактирования

Действие	DOS-Windows	UNIX
Ввод текущей строки	Enter	Enter
Ввод символа «конец файла»	Ctrl+Z. F6	–
Стирание предыдущего символа	Backspase	Backspase, Del, Ctrl + n
Удаление текущего символа (на который указывает курсор)	Del	Ctrl + d
Вызов в командную строку предыдущей команды	F3, [стрелка вверх]	! [стрелка вверх]
Включение/выключение режима вставки символов	Ins	–
Очистка всей командной строки	Esc	Cntrl + u

Порядок выполнения работы

1 Работа со стандартными командами ОС Windows.

1.1 Запустите консоль командной строки. Сделать это можно несколькими способами; два из них: в меню Пуск/Программы/Стандартные выберите **«Командная строка»** или в строке Пуск/Выполнить наберите *cmd.*

1.2 Установите необходимые параметры окна командной строки, для этого в системном меню приложения выберите пункт «Свойства». В появившемся диалоге выберите закладку «расположение» и в блоках «размер буфера экрана» и «размер окна» в поле «высота» установите 25.

1.3 В приглашении *«C:\>»* введите *help* – получите список команд.

1.4 Для того чтобы получить помощь по отдельной команде, необходимо ввести команду: *help* [*команда*] или [*команда*] /?.

1.5 Все действия необходимо выполнять в командной строке, для перехода между каталогами используйте команду **cd**, для перехода на другой диск в приглашении командной строки введите [*диск*]: например: *c***:.** Для просмотра содержимого используйте команду *dir.* Для того, чтобы создать новый каталог, используйте команду *md*. За информацией по командам обращайтесь к **help**.

1.6 В каталоге создайте каталог *SPO* и подкаталог вида [*NNN_n*], где *NN_n* – номер Вашей группы и подгруппы, например: *201_1*.

1.7 В каталоге, созданном выше ([*NNN_n*]), создайте подкаталог *LAB1*. Зайдите в каталог *LAB*. Теперь это Ваш рабочий, или текущий, каталог. Все файлы, которые Вы будете создавать должны находиться в нем.

1.8 Измените вид приглашения командной строки так, чтобы она содержала текущие дату и время, например: *01.02.2019 19:44:40,47 D:\ASOI181\NNN_n\LAB1>*.

1.9 Поменяйте цвета консоли с помощью функции *color*, например: *color 70*.

1.10 Верните вид приглашения и цвета консоли к исходному виду.

2 Работа с потоками ввода/вывода.

2.1 Перенаправьте поток вывода команды *help* в файл *help.txt*. В результате в рабочем каталоге должен появиться файл *help.txt.*

2.2 Просмотрите содержимое рабочего каталога (команда *dir*). Там должен быть файл *help.txt*. Для просмотра его содержимого с помощью программы «Блокнот» выберите формат шрифта *Terminal*.

2.3 Для того чтобы вывести на экран содержимое файла необходимо запустить команду *type*, в качестве параметра которой передается имя этого файла, например: *type help.txt.* Как видите, по экрану строки этого файла проносятся с невероятной скоростью, попробуйте выполнить постраничный вывод файла на экран.

2.4 Изучите с помощью *help* действие команды *more*.

2.5 Создайте конвейер из двух команд: help.txt и more. Теперь содержимое файла help.txt можно читать.

2.6 Однако вышеприведенная конструкция нерациональна – дело в том, что команда *more* тоже выводит данные на экран (консоль), так что можно избавится от команды *type*. Перенаправьте поток ввода команды *more*, выберите исходные данные из файла *help.txt.* Теперь имеем тот же результат, как и в предыдущем пункте, а запись команды при этом заметно сократилась.

Контрольные вопросы

1 Классификация интерфейсов.
2 Пакетная технология.
3 Технология командной строки.
4 Графический интерфейс.
5 Запуск и редактирование командной строки.
6 Назначение потоков.

7 Определение и классификация потоков.

2 Лабораторная работа № 2. Изучение архитектуры памяти

Цель работы: изучить архитектуру и задачи управления памяти в ОС.

Методические указания

Основная память (она же ОЗУ) является важнейшим ресурсом, эффективное использование которого решающим образом влияет на общую производительность системы.

Для однозадачных ОС управление памятью не является серьезной проблемой, поскольку вся память, не занятая системой под собственные нужды, может быть отдана в распоряжение единственного пользовательского процесса. Процедуры управления памятью решают следующие задачи.

1 Выделение памяти для процесса пользователя при его запуске и освобождение этой памяти при завершении процесса.

2 Обеспечение настройки запускаемой программы на выделенные адреса памяти.

3 Управление выделенными областями памяти по запросам программы пользователя (например, освобождение части памяти перед запуском порожденного процесса).

Совершенно иначе обстоят дела в многозадачных ОС. Суммарные требования к объему памяти всех одновременно работающих в системе программ, как правило, превышают имеющийся в наличии объем основной памяти. В этих условиях ОС не имеет другого выхода, кроме поочередного вытеснения процессов или их частей на диск, чтобы использовать освободившуюся память на нужды других процессов. Неудачная реализация такого вытеснения может почти полностью застопорить работу ОС, которая большую часть времени будет заниматься записью и чтением с диска.

К основным задачам, которые должна решать подсистема управления памятью многозадачной ОС, добавляются следующие.

1 Предоставление процессам возможностей получения и освобождения дополнительных областей памяти в ходе работы.

2 Эффективное использование ограниченного объема основной памяти для удовлетворения нужд всех работающих процессов, в том числе с использованием дисков как расширения памяти.

3 Изоляция памяти процессов, исключающая случайное или намеренное несанкционированное обращение одного процесса к областям памяти, занимаемым другим процессом.

4 Предоставление процессам возможности обмена данными через общие области памяти.

Понятие «адрес памяти» может рассматриваться с двух точек зрения. С одной стороны, при написании любой программы ее автор либо явно указывает, по каким адресам должны размещаться переменные и команды (так

бывает при программировании на языке ассемблера), либо присвоение конкретных адресов доверяется системе программирования. Те адреса памяти, которые записаны в программе, принято называть ***виртуальными адресами***.

С другой стороны, каждой ячейке памяти компьютера соответствует ее адрес, который должен помещаться на шину адреса при каждом обращении к ячейке. Эти адреса называются ***физическими***.

Распределение с фиксированными разделами. При этом способе распределения памяти администратор системы заранее, при установке ОС, выполняет разбиение всей имеющейся памяти на несколько разделов. Как правило, формируются разделы разных размеров. Допускается также определение большого раздела как суммы нескольких примыкающих друг к другу меньших разделов.

Распределение с динамическими разделами. При такой организации памяти никакого предварительного разбиения не делается. Вся имеющаяся память рассматривается как единое пространство, в котором размещаются загруженные программы. Когда возникает необходимость запустить еще одну программу, система выбирает свободный фрагмент памяти достаточного размера и выделяет его в качестве «динамического раздела» для данной программы. Если не удается найти достаточно большой непрерывный участок памяти, то самым простым решением будет подождать с запуском новой программы, пока не завершится одна из работающих программ. В принципе, можно использовать подкачку, но ее организация в данном случае сложнее, чем в случае фиксированных разделов, поскольку нужно прежде всего выбрать, какая именно из загруженных программ должна быть вытеснена на диск.

Сегментная организация памяти. При сегментной организации вся виртуальная память, используемая программой, разбивается на части, называемые ***сегментами***. Это разбиение выполняется либо самим программистом (если он программирует на языке ассемблера), либо компилятором используемого языка программирования. Размеры сегментов могут быть различными, но в пределах максимального размера, используемого в данной архитектуре. Разбиение обычно производится на логически осмысленные части, такие как сегмент данных, сегмент кода, сегмент стека и т. п. Большая программа может содержать несколько сегментов одного типа, например, несколько сегментов кода или данных.

Таким образом, при сегментной организации у программы нет единого линейного адресного пространства. Виртуальный адрес состоит из двух частей: ***селектора сегмента*** и ***смещения*** от начала сегмента.

Страничная организация памяти. Эта форма организации виртуальной памяти во многом похожа на сегментную. Основные различия заключаются в том, что все страницы, в отличие от сегментов, имеют одинаковые размеры, а разбиение виртуального адресного пространства процесса на страницы выполняется системой автоматически. Типичный размер страницы – несколько килобайт. Для процессоров Pentium, например, страница равна 4 Кб.

Все виртуальные адреса одного процесса относятся к единому линейному пространству, Проще сказать, виртуальный адрес выражается одним числом,

от 0 до некоторого максимума. Старшие разряды двоичного представления этого адреса определяют номер виртуальной страницы, а младшие разряды – смещение от начала страницы. Например, для страниц по 4 Кб смещение занимает 12 младших разрядов адреса.

Физическая память также считается разбитой на части, размеры которых совпадают с размером виртуальной страницы. Эти части называются *физическими страницами* или *страничными кадрами* (page frames). Таблица страниц процесса по структуре похожа на таблицу сегментов. Для каждой виртуальной страницы она содержит режим доступа, флаг присутствия страницы в памяти, номер страничного кадра, флаг чистоты. Если страница отсутствует в памяти, ее данные сохраняются в файле подкачки, который в этом случае чаще называют *страничным файлом* (*page file*).

Сравнение сегментной и страничной **организации.** Оба рассмотренных способа организации виртуальной памяти имеют свои достоинства и недостатки. К преимуществам сегментной организации обычно относят следующие. Легко можно указать режим доступа к сегменту в зависимости от смысла его данных. Например, сегмент кода программы обычно должен быть доступен только для чтения, а сегмент данных может быть доступен и для записи. В том случае, если программа работает с двумя или более структурами данных, каждая из которых может увеличиваться в размерах независимо от других, выделение отдельного сегмента для каждой структуры позволяет освободить программиста от забот, связанных с размещением структур в имеющейся памяти (эти проблемы перекладываются на ОС, которая обязана будет найти место в физической памяти для увеличивающихся сегментов).

Гораздо реже называется еще одна, более прозаическая причина использования сегментов, которая на самом деле в определенный период являлась очень веской. Если в используемой архитектуре компьютера разрядность адреса в командах слишком мала (например, 16 разрядов, как у процессоров i286, что позволяет адресовать всего лишь 64 Кб), а размер программы и ее данных достигает многих мегабайт, то единственное решение – использовать много сегментов по 64 Кб.

Для современных процессоров разрядность адреса составляет 32 или даже 64 бита, что снимает необходимость возиться с большим количеством мелких сегментов. При этом на первый план выходят достоинства страничной организации.

1 Программист не должен вообще думать о разбиении программы и ее данных на части ограниченного размера (сегменты), в его распоряжении единое пространство виртуальных адресов.

2 Исключается возможность фрагментации физической памяти и связанные с этим проблемы.

3 Уменьшается обмен данными с диском, поскольку в него включаются только отдельные страницы, а не целые сегменты.

Порядок выполнения работы

1 Используя монитор ресурсов ОС Windows, определить объем установленной физической памяти, объем виртуальной памяти, величину файла подкачки и его размещение в компьютере.

2 Определить, какое количество физической памяти использует оборудование.

3 Определить, какому процессу требуется наибольшее количество памяти.

4 Определить, какой процесс использует наибольшее количество памяти совместно с другими процессами (разделяемый ресурс).

Контрольные вопросы

1 Какие задачи решают с помощью процедуры управления памятью?
2 Что понимается под виртуальными и физическими адресами памяти?
3 Что понимается под относительной адресацией?
4 Перечислите способы организации памяти.
5 Из каких частей состоит виртуальный адрес?

3 Лабораторная работа № 3. Изучение механизмов взаимодействия операционной системы и устройств ввода-вывода

Цель работы: изучить классы WMI, отвечающие за работу с устройствами.

Методические указания

Устройства ввода-вывода можно условно разделить на две категории: *блочные* устройства и *символьные* устройства. К блочным относятся такие устройства, которые хранят информацию в блоках фиксированной длины, у каждого из которых есть свой собственный адрес. Вся передача данных ведется пакетами из одного или нескольких целых (последовательных) блоков. Важным свойством блочного устройства является то, что оно способно читать или записывать каждый блок независимо от всех других блоков. Блочное устройство – это устройство, которое может содержать файловую систему. Среди наиболее распространенных блочных устройств – жесткие диски, приводы компакт-дисков и флеш-накопители USB.

Другой тип устройств ввода-вывода – символьные устройства. Они выдают или воспринимают поток символов (байт), не относящийся ни к какой блочной структуре. Они не являются адресуемыми и не имеют никакой операции позиционирования. В качестве символьных устройств могут рассматриваться принтеры, сетевые интерфейсы, мыши и множество других устройств, не похожих на дисковые устройства.

Контроллеры устройств. Внешнее устройство обычно состоит из механического и электронного компонента. Электронный компонент называется **контроллером устройства** или адаптером. Он сочетает на одном кристалле функции процессора, содержит ОЗУ. По сути, это однокристальный компьютер, способный выполнять относительно простые задачи. Механический компонент представляет собственно устройство.

Если интерфейс между контроллером и устройством стандартизован (ANSI, IEEE или ISO), то независимые производители могут выпускать совместимые как контроллеры, так и устройства, например, диски IDE или SCSI.

Операционная система обычно имеет дело не с устройством, а с контроллером. Контроллер, как правило, выполняет простые функции, например, при считывании с диска, преобразует поток бит в блоки, состоящие из байт, и осуществляет контроль и исправление ошибок. Проверяется контрольная сумма блока; если она совпадает с указанной в заголовке сектора, то блок считан без ошибок, если нет, то считывается заново. Каждый контроллер имеет несколько регистров памяти, которые используются для взаимодействия с центральным процессором. При помощи этих регистров ОС управляет (считывает, пишет, включает и т. д.) и определяет состояние (готовность) устройства. У многих устройств есть буфер данных (например, видеопамять).

Прерывания. После того как устройство ввода-вывода начало работу, процессор переключается на другие задачи. Чтобы сигнализировать процессору об окончании работы, устройство инициализирует прерывание, выставляя сигнал на выделенную устройству линию шины (а не выделенный провод).

Контроллер прерываний (*англ. Programmable Interrupt Controller*, *PIC*) – микросхема или встроенный блок процессора, отвечающий за возможность последовательной обработки запросов на прерывание от разных устройств. Контроллер прерываний обслуживает поступающие прерывания от устройств.

1 Если необработанных прерываний нет, прерывание выполняется немедленно.

2 Если необработанные прерывания есть, контроллер игнорирует прерывание. Но устройство продолжает удерживать сигнал прерывания на шине до тех пор, пока оно не будет обработано.

Алгоритм работы.

1 Устройство выставляет сигнал прерывания.

2 Контроллер прерываний инициирует прерывание, указывая номер устройства.

3 Процессор начинает выполнять обработку прерывания, вызывая процедуру.

4 Эта процедура подтверждает получение прерывания контроллеру прерываний.

Основные задачи, которые должно решать программное обеспечение ввода-вывода.

1 Независимость от устройств – например, программа, читающая данные из файла не должна задумываться, с чего она читает (CD, HDD и др.). Все проблемы должна решать ОС.

2 Единообразное именование – имя файла или устройства не должны отличаться (в системах UNIX выполняется дословно).

3 Обработка ошибок – ошибки могут быть отловлены на уровне контроллера, драйвера и т. д.

4 Перенос данных – синхронный и асинхронный (в последнем случае процессор запускает перенос данных и переключается на другие задачи до прерывания).

5 Буферизация.

6 Проблема выделенных (принтер) и невыделенных (диск) устройств – принтер должен предоставляться только одному пользователю, а диск многим. ОС должна решать все возникающие проблемы.

Драйверы устройств. Для управления каждым подключенным к компьютеру устройством ввода-вывода требуется специальная программа, учитывающая его особенности. Эта программа называется **драйвером устройства**. Обычно она создается производителем устройства и поставляется вместе с этим устройством. Поскольку для каждой операционной системы нужны собственные драйверы, производитель устройства обычно поставляет драйверы для нескольких наиболее популярных операционных систем. Каждый драйвер устройства обычно управляет одним типом устройства или как максимум одним классом родственных устройств. Драйверы должны быть частью ядра (в монолитной системе), чтобы получить доступ к регистрам контроллера. Это одна из основных причин, приводящих к краху операционных систем, потому что драйверы, как правило, пишутся производителями устройств и внедряются в ОС. Драйверы должны взаимодействовать с ОС через стандартные интерфейсы.

Функции, которые выполняют драйверы.

1 Обработка запросов чтения или записи.

2 Инициализация устройства.

3 Управление энергопотреблением устройства.

4 Прогрев устройства (сканера).

5 Включение устройства или запуска двигателя.

Функции программного обеспечения ввода-вывода пространства пользователя.

1 Обращение к системным вызовам ввода-вывода (через библиотечные процедуры).

Форматный ввод-вывод (меняют формат, например, в ASCII).

2 Спулинг (для выделенных устройств) – создается процесс (например, демон печати) и каталог спулера.

Спулинг (подкачка данных) – способ согласования параллельной отправки заданий и их последовательного выполнения – SPOOLING: Simultaneous Peripheral Operation On Line – одновременная работа с периферийными устройствами в интерактивном режиме. Спулинг – буфер для УВВ-типа принтера, который не может принять несколько перекрывающихся потоков данных. При одновременной печати из нескольких приложений их данные накапливаются в отдельных файлах. Когда приложение завершает вывод на

печать, это задание ставится в очередь для вывода на принтер. ОС предоставляет возможность управлять очередью, просматривать очередь, управлять заданиями на печать, удалять задания, останавливать и продолжать процесс печати

Получение информации об устройствах компьютера с помощью технологии Windows Management Instrumentation. Технология WMI (Windows Management Instrumentation) – это созданная фирмой Microsoft реализация модели управления предприятием на базе Web (Web-Based Enterprise Management. WBIIM). Эта технология разработана и принята рабочей группой по управлению распределенными системами (Distributed Management Task Force, DMTF) при участии таких компаний, как BMC Software, Cisco Systems, Intel и Microsoft. Задачей DMTF была разработка таких стандартов для удаленного управления информационной средой предприятия, которые позволили бы управлять всеми физическими и логическими компонентами этой среды из одной точки и не зависели бы при этом от конкретного оборудования, сетевой инфраструктуры, операционной системы, файловой системы и т. д.

Для этого была предложена схема CIM (Common Information Model), которая представляет физическую и логическую структуры компьютерной системы в виде единой расширяемой объектно ориентированной информационной модели и определяет единые интерфейсы для получения информации о любом компоненте этой модели.

Провайдеры WMI. Провайдеры WMI обеспечивают связь между менеджером объектов CIM и управляемыми ресурсами: провайдеры предоставляют для CIMOM данные об управляемом объекте, обрабатывают запросы от управляющих программ и генерируют сообщения о наступлении определенных событий.

При этом провайдер WMI общается с управляемым объектом с помощью специфического API этого объекта, а с CIMOM – посредством стандартного интерфейса прикладного программирования WMI (WMI API). Таким образом, провайдеры скрывают детали внутренней реализации управляемых объектов, позволяя CIMOM обращаться к этим объектам единообразно, используя один и тот же WMI API.

Фактически провайдеры WMI являются серверами COM или DCOM, которые представлены динамическими библиотеками (DLL), находящимися чаще всего в каталоге %SystemRoot%\System32\Wbem. WMI включает в себя множество встроенных провайдеров для операционных систем семейства Windows, которые предназначены для получения данных из известных системных источников, таких как подсистема Win32, журналы событий, системный реестр, системные счетчики производительности (таблица 3.1).

Таблица 3.1 – Стандартные провайдеры WMI

Провайдер	DLL-файл	Описание
Провайдер каталога Active Directory(Active Directory provider)	Dsprov.dll	Позволяет обращаться к объектам Active Directory как к объектам WMI
Провайдер журнала событий (Event Log provider)	Ntevt.dll	Обеспечивает управление журналом событий (выборка по определенному критерию записей для чтения, создание резервных копий и очистка журнала, изменение настроек и т. д.). Также этот провайдер позволяет обрабатывать события, генерируемые журналом (например, добавление в журнал записи определенного типа)
Провайдер системных счетчиков производитель-ности (Perfomance Counter provider)	Wbemperf.dll	Обеспечивает доступ к счетчикам производительности, т. е. к данным, позволяющим численно оценивать производительность системы
Провайдер реестра (Registry provider)	Stdprov.dll	Позволяет читать данные из реестра, создавать и модифицировать там ключи и разделы. Кроме этого, провайдер обеспечивает генерацию события WMI при изменении определенного ключа или ветви реестра
Провайдер SNMP-устройств (SNMP provider)	Snmpincl.dll	Является шлюзом для доступа к системам и устройствам, которые управляются с помощью протокола SNMP (Simple Network Management Protocol)
Провайдер драйверов устройств (WDM provider)	Wmiprov.dll	Позволяет получить доступ к информации низкого уровня о драйверах устройств Windows Driver Model (WDM); в качестве таких устройств могут выступать, например, порты ввода/вывода или сетевые платы
Провайдер подсистемы Win32 (Win32 provider)	Cimwin32.dll	Обеспечивает доступ к информации о компьютере, операционной системе, подсистеме безопасности, дисках, периферийных устройствах, файловых системах, файлах, папках, сетевых ресурсах, принтерах, процессах и т. п.
Провайдер инсталлированных программных продуктов (Windows Installer provider)	Msiprov.dll	Позволяет получить информацию об инсталлированном программном обеспечении

Задачей менеджера объектов CIM (CIMOM) является обеспечение взаимодействия между потребителями сервисов WMI (управляющими приложениями) и провайдерами WMI. CIMOM обрабатывает все запросы, которые поступают от управляющих приложений к WMI, и обеспечивает доставку к этим приложениям информации, полученной в результате выполнения таких запросов.

1 Регистрация провайдеров. Все провайдеры WMI должны быть зарегистрированы с помощью CIMOM; информация о провайдере (например,

тип этого провайдера или путь к библиотеке DLL, которой он представлен) хранится в репозитории CIM.

2 Переадресация запросов. Используя информацию о зарегистрированных провайдерах, CIMOM перенаправляет полученный от управляющего приложения запрос к нужному провайдеру.

3 Доступ к удаленной машине с WMI. Управляющее приложение может обратиться с запросом к любой удаленной машине, на которой установлен WMI. При этом происходит соединение с CIMOM на удаленной машине, после чего все запросы здесь должны обрабатываться точно так же, как и на локальной машине.

4 Обеспечение безопасности. Защита ресурсов WMI состоит в том, что CIMOM проверяет права пользователя, который пытается воспользоваться сервисами WMI на локальном или удаленном компьютере.

5 Обработка запросов управляющих приложений. Потребители WMI обращаются к управляемым объектам с помощью специального языка запросов WMI Query Language (WQL). Если провайдер запрашиваемого объекта не поддерживает напрямую WQL, то CIMOM должен преобразовать этот запрос к тому виду, в котором он сможет быть обработан этим провайдером.

Обработка событий WMI. Поддержка CIMOM этой функции позволяет потребителям WMI создавать обработчики событий, которые возникают при определенном изменении в управляемом объекте (примеры таких событий – снижение объема свободного пространства на жестком диске до заданного значения или запуск на компьютере определенного приложения). Для этого CIMOM периодически опрашивает нужный объект (интервал опроса задается в управляющем приложении) и генерирует событие как только обнаруживает, что заданное заранее условие возникновения события выполнено.

WMI является открытой унифицированной системой интерфейсов доступа к любым параметрам операционной системы, устройствам и приложениям, которые функционируют в ней. Важной особенностью WMI является то, что хранящиеся в нем объекты соответствуют динамическим ресурсам, т. е. параметры этих ресурсов постоянно меняются, поэтому параметры таких объектов не хранятся постоянно, а создаются по запросу потребителя данных. Хранилище свойств объектов WMI называется репозиторием и расположено в системной папке операционной системы Windows (%SystemRoot%\System32\WBEM\Repository\FS).

Для визуального просмотра сведений о устройствах служит компонент ОС Windows «Сведения о системе» (msinfo32.exe), который отображает подробные сведения о конфигурации оборудования, компонентах и программном обеспечении компьютера, включая драйверы. В левой области окна «Сведения о системе» приведен список категорий, а в правой – подробные сведения о каждой из них. К этим категориям относятся следующие.

Аппаратные ресурсы. Общие сведения о компьютере и операционной системе, такие как имя компьютера и его изготовитель, тип используемой BIOS, а также объем установленной памяти.

Класс Win32_BIOS позволяет получить информацию о атрибутах BIOS (базовая система ввода-вывода).

Классы Win32_BaseBoard, *Win32_MotherboardDevice*, *Win32_SystemSlot* позволяют получить информацию о системной плате.

Класс Win32_Bus представляет физические шины.

Класс Win32_OnBoardDevice представляет общие адаптеры, встроенные в системную плату.

Класс Win32_Processor позволяет получить информацию о процессоре. Так как сейчас в большинстве систем находятся процессоры из нескольких ядер, то и экземпляров класс Win32_Processor будет несколько.

Класс Win32_CacheMemory представляет внутреннюю и внешнюю кеш-память в компьютерной системе. Кеш-память – это сверхбыстрая память используемая процессором для временного хранения данных, которые наиболее часто используются.

Классы Win32_PhysicalMemory, *Win32_MemoryDevice*, *Win32_MemoryArray*, *Win32_PhysicalMemoryArray*, *Win32_DeviceMemoryAddress*, *Win32_DMAChannel* позволяют получить информацию о различных видах оперативной памяти. Класс Win32_MemoryDevice предоставляет информацию о начальных и конечных адресах для всех устройств памяти, установленных на компьютере.

Компоненты. Перечень установленных дисководов, звуковых устройств, модемов и других компонентов.

Классы Win32_PortConnector, *Win32_PortResource*, *Win32_ParallelPort*, *Win32_SerialPort*, *Win32_SerialPortConfiguration* позволяют получить информацию о различных видах *портов ввода-вывода.* Порт представляет собой канал передачи данных между устройством и микропроцессором. Порт представляется в микропроцессоре как один или несколько адресов памяти, из которых можно прочитать или в которые можно записать данные. Класс Win32_PortConnector предоставляет информацию о физических портах подключения, таких как Centronics, PS/2 и т. д. Класс Win32_PortResource предоставляет информацию обо всех портах ввода-вывода (I/O ports), найденных на компьютере.

Класс Win32_Keyboard позволяет получить информацию о клавиатуре. Стандартная клавиатура имеет 101 или 102 клавиши. Клавиатуры друг от друга отличаются по типу подключения: Usb, PS/2, блютуз.

Класс Win32_PointingDevice позволяет получить информацию о мыши. Мыши отличаются друг от друга по виду, по типу подключения, по разъемам и размеру.

Класс Win32_SoundDevice содержит сведения о звуковой карте.

Класс CIM_VideoControllerResolution представляет различные видео режимы, которые поддерживает видеоконтроллер. Класс Win32_VideoController содержит сведения о видеокарте.

Классы Win32_NetworkAdapter и Win32_NetworkAdapterConfiguration содержат сведения о сетевых адаптерах. Сетевые адаптеры различаются по типу и разрядности используемой в компьютере внутренней шины данных – ISA, EISA, PCI, MCA. Сетевые адаптеры различаются также по типу принятой в сети сетевой технологии – Ethernet, Token Ring, FDDI и т. п. Некоторые сетевые адаптеры имеют возможность использовать оперативную память ПК в качестве буфера для хранения входящих и исходящих пакетов данных. Базовый

адрес (Base Memory Address) представляет собой шестнадцатеричное число, которое указывает на адрес в оперативной памяти, где находится этот буфер.

Класс Win32_DesktopMonitor предоставляет сведения о мониторе, подключённом к компьютерной системе. Мониторы отличаются друг от друга размером экрана, форматом LSD-матрицы, разрешением LSD-матрицы, поверхностью экрана монитора, яркостью LSD-монитора, контрастностью LSD-матрицы, глубиной цвета матрицы монитора.

Класс Win32_DiskDrive предоставляет сведения о приводах дисков.

Класс Win32_PnPEntity представляет все установленные устройства Plug and Play.

Plug and Play (*сокр. PnP*) – технология, предназначенная для быстрого определения и конфигурирования устройств в компьютере и других технических устройствах. Технология PnP основана на использовании объектно ориентированной архитектуры, ее объектами являются внешние устройства и программы. Операционная система автоматически распознает объекты и вносит изменения в конфигурацию абонентской системы.

Возможности Windows PowerShell для работы с объектами WMI. В PowerShell экземпляры объектов WMI можно получать с помощью командлета *Get-wmiobject* (псевдоним *gwmi*). При этом для обращения к определенному объекту WMI нужно знать наименование класса, к которому он относится (например, *win32_Process* для процессов, запущенных в системе, или *win32_Service* для зарегистрированных на компьютере служб).

Например, для получения информации о BIOS локального компьютера, можно использовать командлет

Get-Wmiobject -Class win32_bios

Если нужно подключиться к подсистеме WMI на другой машине, то ее имя или IP-адрес нужно указать в качестве значения параметра – ComputerName. Список информации о настройках рабочего стола удаленного компьютера

Get-Wmiobject -Class win32_desktop –Computername 10.169.1.15

Работа с любыми внешними объектами в PowerShell производится с помощью системы адаптации типов, поэтому к объектам WMI можно применять любые командлеты для фильтрации, сортировки, группировки и т. д. (рисунок 3.1.).

```
PS C:\Users\Михаил> Get-Wmiobject -Class Win32_DiskDrive  | Where-Object { $_.partitions -eq 1}


Partitions : 1
DeviceID   : \\.\PHYSICALDRIVE2
Model      : UFD 2.0 Silicon-Power4G USB Device
Size       : 3874106880
Caption    : UFD 2.0 Silicon-Power4G USB Device
```

Рисунок 3.1 – Пример применения командлет

Порядок выполнения работы

Выполните одно из заданий, приведенных в таблице 3.2 (вариант задает преподаватель). Для получения полной информации о свойствах и методах класса обращайтесь к MSDN (https://docs.microsoft.com/ru-ru/windows/desktop/CIMWin32Prov/computer-system-hardware-classes) и таблице 3.3. Поиск в MSDN названия того или иного класса позволяет быстро получить перечень всех его свойств и методов.

Таблица 3.2 – Варианты заданий

Вариант	Задание
1	Создайте сценарий WMI, выполняющий запись в файл сведений о материнской плате: производитель, тип первичной шины системной платы, тип вторичной шины системной платы, тип шины (1 – ISA, 5 – PCI Bus, 15 – PNP Bus и т. д.). Используйте классы Win32_BaseBoard, Win32_MotherboardDevice, Win32_OnBoardDevice, Win32_Bus,Win32_SystemSlot
2	Создайте сценарий WMI, выполняющий запись в файл сведений о количестве процессоров и скорости процессора, о размерах кеша 2-го уровня. Используйте классы Win32_Processor, Win32_CacheMemory
3	Создайте сценарий WMI, выполняющий запись в файл сведений о количестве свободной физической памяти, о диапазоне доступных адресов. Используйте классы Win32_PhysicalMemory, Win32_MemoryDevice, Win32_DMAChannel
4	Создайте сценарий WMI, выполняющий запись в файл сведений обо всех портах ввода-вывода (I/O ports), найденных на компьютере. Используйте классы Win32_PortResource, Win32_ParallelPort, Win32_SerialPort
5	Создайте сценарий WMI, выполняющий запись в файл сведений о разрешении экрана, наименование клавиатуры и количество функциональных клавиш. Используйте классы Клавиатура (Win32_Keyboard), Монитор (Win32_DesktopMonitor)
6	Создайте сценарий WMI, выполняющий запись в файл сведений о наименовании, производителе и количестве кнопок мыши, характеристиках шины системной платы. Используйте классы Мышь (Win32_PointingDevice), Шина(Win32_Bus)
7	Создайте сценарий WMI, выполняющий запись в файл сведений о идентификаторах и производителях устройств Plug and Play, наименование и производители аудиоустойств. Используйте классы Аудио (Win32_SoundDevice), устройства Plug and Play (Win32_PnPEntity)
8	Создайте сценарий WMI, выполняющий запись в файл сведений о качестве цветопередачи (количество бит на пиксель), частоте обновления экрана (Гц), описание видеопроцессора. Используйте классы Видео (Win32_VideoController), Класс CIM_VideoControllerResolution
9	Создайте сценарий WMI, выполняющий запись в файл сведений о файловых системах логических дисков, о IP-адресе компьютера. Используйте классы Win32_LogicalDisk, Win32_NetworkAdapterConfiguration)
10	Создайте сценарий WMI, выполняющий запись в файл сведений о том, имеются ли на компьютере CD-ROM, его марка, о устройствах, подключенных к USB портам. Диски (Win32_DiskDrive), CD-ROM (Win32_CDROMDrive) (Win32_USBHub)

Таблица 3.3 – Алфавитный перечень классов провайдера WMI Win32

Имя класса WMI	Описание
Win32_BaseBoard	Управление материнской платой, она также называется motherboard, или системная плата
Win32_BIOS	Управления базовыми сервисами ввода/вывода (Basic input/output services, BIOS)
Win32_BootConfiguration	Управление конфигурацией загрузки (Boot configuration management)
Win32_CDROMDrive	Управление приводом CD-ROM
Win32_ComputerSystem	Управление системой компьютера
WIN32_PROCESSOR	Управление процессором
Win32_ComputerSystem Product	Получение от SMBIOS информации о компьютере как системном продукте
CIM_DataFile	Управление данными файлов (DataFile Management)
WIN32_DCOM Application	Управление приложениями (DCOM Application management)
WIN32_DESKTOP	Управление рабочим столом пользователя (User's Desktop management)
WIN32_DESKTOPMO NITOR	Desktop Monitor management
Win32_DeviceMemory Address	Управление адресами памяти устройств (Device memory addresses management)
Win32_DiskDrive	Управление диском на физическом уровне (Physical disk drive management)
Win32_DiskQuota	Управление квотами NTFS пространства диска (Disk space usage for NTFS volumes)
Win32_DMAChannel	Управление каналами прямого доступа к памяти (Direct memory access, DMA channel management)
Win32_Environment	Управление настройками системного окружения (System environment settings management)
Win32_Directory	Управление директориями файловой системы (Filesystem directory entry management)
Win32_Group	Управление группами учетных записей (Group account management)
Win32_IDEController	Управление контролером диска IDE (IDE Controller management)
Win32_IRQResource	Управление сигналами прерываний (Interrupt request line, IRQ management)
Win32_ScheduledJob	Предоставляет доступ к назначенным заданиям (jobs scheduled) с использованием службы назначенных заданий (schedule service)
Win32_LoadOrderGroup	Управление службами системы, которые задают зависимости запуска (execution dependencies)
Win32_LogicalDisk	Управление дисковыми локальными устройствами хранения (Local storage device management)
Win32_LogonSession	Управление сессиями пользователей (LOGON Sessions)
WIN32_CACHEMEMORY	Управление кешем (Cache memory management)
Win32_LogicalMemory Configuration	Управление памятью системы (как сконфигурирована карта памяти, и как память доступна для приложений и сервисов)
Win32_PhysicalMemory Array	Управление памятью компьютера на физическом уровне (Computer system's physical memory management)
WIN32_NetworkClient	Управление клиентом сети (Network Client management)
Win32_NetworkLoginPr ofile	Управление информацией учетной записи сети для отдельного пользователя

Продолжение таблицы 3.3

Имя класса WMI	Описание
Win32_NetworkProtocol	Управление протоколами и их сетевыми характеристиками
Win32_NetworkConnection	Управление активным сетевым соединением
Win32_NetworkAdapter	Управление сетевым адаптером (Network Interface Controller, NIC)
Win32_NetworkAdapterConfiguration	Управление конфигурацией сетевого адаптера
Win32_NTDomain	Управление доменом NT
Win32_NTLogEvent	Получение доступа к логам событий (Entries in the NT Event Log)
Win32_NTEventlogFile	Управление файлом лога (NT eventlog file management)
Win32_OnBoardDevice	Управление общими адаптерами, установленными в материнскую плату (system board)
Win32_OperatingSystem	Управление инсталлированными операционными системами
Win32_PageFileUsage	Управление виртуальной памятью и её вытеснением на диск
Win32_PageFileSetting	Установка параметров файла виртуальной памяти
Win32_DiskPartition	Управление логическими разделами физического диска (Management of partitioned areas of a physical disk)
Win32_PortResource	Управление портами ввода/вывода (I/O port management)
Win32_PortConnector	Управление физическим соединением портов (Physical connection ports management)
Win32_PrinterConfiguration	Управление конфигурацией принтера
Win32_PrintJob	Управление заданиями принтера
Win32_Process	Управление процессами
Win32_Product	Управление задачей инсталляционных пакетов (Installation package task management)
Win32_QuickFixEngineering	Быстрое исправление ошибок (Quick Fix Engineering)
Win32_QuotaSetting	Установка информации о квотах, установленных для тома диска (Setting information for disk quotas on a volume)
Win32_OSRecoveryConfiguration	Информация, которая была захвачена из памяти при крахе системы
Win32_Registry	Управление реестром системы
Win32_SCSIController	Управление контроллером SCSI
Win32_PerfRawData_PerfNet_Server	Управление информацией о сервере
Win32_Service	Управление прикладными сервисами
Win32_Share	Управление общими сетевыми ресурсами (Shared resource management)
Win32_SoftwareElement	Управление элементами программных продуктов, установленных в системе
Win32_SoftwareFeature	Управление подмножествами программных продуктов SoftwareElement
WIN32_SoundDevice	Управление устройством звука
Win32_StartupCommand	Управление командами автозапуска при входе пользователя в систему
Win32_SystemAccount	Управление учетной записью системы (System account management)
Win32_SystemDriver	Управление системным драйвером для базовой службы (Management of the system driver for a base service)
Win32_SystemEnclosure	Управление физическим доступом к корпусу компьютера (Physical system enclosure management)

Окончание таблицы 3.3

Имя класса WMI	Описание
Win32_SystemSlot	Управление физичесими соединениями, включая порты, слоты и периферийные устройства, и точками проприетарных соединений
Win32_TapeDrive	Управление накопителем на магнитной ленте
Win32_TemperatureProbe	Управление данными от датчика температуры (электронный термометр)
Win32_TimeZone	Управление данными часового пояса (Time zone data management)
Win32_Uninterruptible PowerSupply	Управление источником бесперебойного питания (Uninterruptible power supply, UPS)
Win32_UserAccount	Управление учетными записями пользователей (User account management)
Win32_VoltageProbe	Управление данными сенсора напряжения (электронный вольтметр)
Win32_VolumeQuota Setting	Связывает установки дисковой квоты с определенным дисковым томом
Win32_WMISetting	Управляет рабочими параметрами службы WMI

Контрольные вопросы

1 На какие категории делятся устройства ввода-вывода?

2 Перечислите основные функции контроллера.

3 Какие операции выполняет драйвер в подсистеме ввода-вывода?

4 Что такое прерывание?

5 Какими способами ОС реагирует на прерывания?

6 Перечислите основные задачи программного слоя подсистемы ввода-вывода.

7 Перечислите основные функции драйвера.

8 Перечислите функции программного обеспечения ввода-вывода пространства пользователя.

9 Объясните принципы технологии PnP.

10 Охарактеризуйте назначение и возможности технологии Windows Management Instrumentation.

11 Какие командлеты Windows PowerShell позволяют работать с объектами WMI?

4 Лабораторные работы № 4–5. Сервер сценариев Windows Scripting Host

Цель работы: овладеть навыками работы сервером сценариев ОС Windows, изучить возможности сервера сценариев, режимы выполнения сценариев и методы объекта WscriptShell и WshNetwork.

Методические указания

Долгое время для выполнения однотипных задач в среде Windows и DOS служили командные (пакетные) BAT-файлы. Основным их недостатком были примитивный DOS-интерфейс – отсутствие интерактивности – и довольно ограниченные возможности по работе с WINDOWS (трудность работы в сети, с ярлыками, с реестром и т. д.).

Ситуация изменилась, когда Microsoft разработала Сервер Сценариев (Windows Scripting Host), который должен служить для автоматизации работы с повторяющимися процессами. Сам Windows Scripting Host не является языком как таковым, он только представляет свойства и методы для работы в Windows, которые могут использоваться другими языками сценариев. Наиболее удобными и предназначенными для этого явились ранее разработанные самой Microsoft языки сценариев Visual Basic Scripting Edition (VBScript) и JScript.

Раньше языки VBScript и JScript по своим возможностям были очень близки к Visual Basic for Applications – они также могли быть вызваны только из MS Internet Explorer и нескольких других программ Microsoft, которые их поддерживали.

С появлением Windows Scripting Host появилась возможность создавать для них отдельные сценарии, которые можно запускать и без Internet Explorer.

Также преимуществом Windows Scripting Host является то, что для запуска сценариев требуется мало памяти, и то, что файлы сценариев могут быть практически любого размера (содержать десятки тысяч строк).

Еще одним преимуществом Windows Scripting Host является отсутствие среды разработки – не нужны компиляторы, редактирование сценариев может производиться в любом текстовом редакторе, способном работать с текстовыми файлами.

Сервер сценариев предназначен для автоматизации повторяющихся задач и во многом по сравнению с обычными языками программирования обладает достаточно скромными возможностями. Но по сравнению с пакетными файлами DOS он обладает более широкими возможностями, такими как:

– вывод сообщений на экран;

– запуск других программ;

– работать с сетевыми дисками;

– устанавливать принтеры;

– работать с переменными среды;

– работать с реестром.

В Windows 2000 и последующих версиях Windows Scripting Host установлен по умолчанию. Отключить его использование можно только удалением ассоциаций с его файлами. Установленный Windows Script Host поддерживает несколько видов файлов: vbs, vbe, js, jse, wsf, wsc и wsh. Все они (кроме vbe и jse) являются простыми текстовыми файлами и могут редактироваться в любом текстовом редакторе.

Файлы .vbs и .js являются файлами, написанными на языке сценариев MS Visual Basic Script и MS JScript соответственно.

Файлы vbe и jse – это vbs- и js-файлы зашифрованные с помощью программы MS Script Encoder.

Файлы с расширением .wsf – это файлы, содержащие XML-разметку для работы с WSH.

Файлы wsc – Windows Script Components (WSC) – позволяют упаковывать сценарии для использования их в качестве COM-компонентов. По сути, это те же wsf-файлы, еще и содержащие COM-компоненты.

Файлы wsh являются файлами настроек Сервера Сценариев.

Для запуска сценариев в составе Windows Scripting Host служат файлы WScript.exe (диалоговый режим) и CScript.exe (режим командной строки). Они находятся в каталоге C:\WINDOWS\system32\. WScript.exe служит для запуска сценариев из Windows. Используя его, можно запускать сценарии подобно обычным приложениям Windows. Вот несколько способов.

1 Запускать сценарий как обычное приложение двойным щелчком мыши, выделить и нажать Enter и т. д.

2 Ввести имя файла сценария и путь к нему в окне в окне «Выполнить» (RUN) меню «Пуск» (Start).

3 Ввести в строку окна «Выполнить» WScript.exe и имя файла сценария (с указанием пути к нему). При этом можно использовать параметры запуска WScript.exe.

CScript.exe – это версия Windows Scripting Host, которая используется для запуска сценариев из командной строки.

Синтаксис: *CScript* [*параметры*] *имя_файла. расширение* [*аргументы*]

Для запуска сценариев как с помощью CScript.exe, так и с WScript.exe используют параметры командной строки согласно таблице 4.1.

Таблица 4.1 – Параметры командной строки

Параметр	Версия WSH	Описание
//B	1.0	Пакетный режим (подавляется вывод информации, запросов и сообщений об ошибках)
//D	2.0	Включить активную отладку
//E:*язык*	2.0	Указать язык сценария для исполнения файла
//H:CScript	1.0	Заменить исполняемый сервер сценариев на CScript.exe
//H:WScript	1.0	Заменить исполняемый сервер сценариев на WScript.exe
//I	1.0	Диалоговый режим (противоположный //B) (по умолчанию)
//Job:xxxx	2.0	Выполнить задание xxxx WSF-файла
//Logo	1.0	Отображать заставку (по умолчанию)
//Nologo	1.0	Не выводить заставку
//S	1.0	Запомнить параметры текущей командной строки для данного пользователя
//T:nn	1.0	Задать время исполнения сценария в секундах
//X	2.0	Выполнить сценарий под отладчиком
//U	2.0	Применять кодировку Unicode для перенаправленного консольного ввода-вывода

В работе WSH используются девять объектов: WScript (не путать с WScript.exe), WshArguments, WshEnvironment, WshNetwork, WshShell, WshShortcut, WshSpecialFolders, WshUrlShortcut и FileSystemObject.

Объект WScript является главным объектом Windows Script Host. Он служит для создания объектов и выполняет служебные задачи связанные с ними, содержит сведения о сервере сценариев и о запущенных сценариях.
Объект WshArguments служит для работы с аргументами среды.

WshEnvironment – работает переменными среды.

WshNetwork – используется при работе с сетевым окружением; содержит информацию для сети о данном компьютере, позволяет подключать сетевые принтеры и диски.

WshShell – служит для работы с переменными среды Windows, запускает другие программы, работает с реестром и т. д.

WshShortcut – создает ярлыки.

WshSpecialFolders – используется для доступа к специальным папкам Windows, таким как меню Пуск, Рабочий стол, Мои документы и т. д.

WshUrlShortcut – еще один объект для создания ссылок, но обладающий более ограниченными возможностями, чем WshShortcut.

Отличается FileSystemObject-объект. Как таковой он, не является объектом WSH и дочерним объектом WScript, но занимает важное место в создании сценариев, используется для работы с файлами.

Объект TextStream используется для работы с содержанием текстовых файлов.

Методы объекта WScript. CreateObject создает экземпляр объекта ActiveX. Синтаксис: object.CreateObject(strProgID[,strPrefix]), где object – объект WScript, StrProgID – класс к которому принадлежит объект.

Например, создадим объект WshShell.

Set WshShell = WScript.CreateObject(“WScript.Shell”)

ConnectObject – позволяет подключить исполняемый сценарий к существующему объекту, его событиям.

DisconnectObject – отключается от объекта, с которым был соединен сценарий методом ConnectObject.

GetObject – получает объект, который уже создан и находится в другом файле.

Echo – выводит диалоговое окно с сообщением пользователю. При использовании CScript.exe выводит строку с текстом.

Sleep – переводит сценарий в неактивное состояние на заданное время (в миллисекундах), после чего продолжает его работу.

Quit – завершает работу сценария. Не обязателен.

Windows Scripting Host имеет два вида диалоговых окон: простое (метод Echo) и управляющее (метод Popup)

Метод Echo объекта WScript отображает сообщение в диалоговом окне, если используется WScript.exe, или выводит строку с текстом, если используется CScript.exe, по своим возможностям дублируя команду echo bat-файлов.

Синтаксис следующий: object.Echo [[Arg1] [,Arg2] [,Arg3] ...], где object – объект WScript, Arg1, Arg2, Arg3 ... – данные, которые должны быть выведены на экран. Для перевода строки используется константа vbCrLf.

Управляющее окно Popup имеет те же возможности вывода информации, что и окно созданное с помощью метода Echo, но вместе с тем обладает дополнительными возможностями, расширяющими его возможности и сферу его применения. Метод Popup является методом объекта WshShell, и для его использования должен быть создан объект WshShell.

Синтаксис: intButton = object.Popup(strText,[WaitSec],[strTitle],[natType], где object – объект WshShell , strText – само сообщение в данном окне, WaitSec – время (в секундах), по истечении которого окно закроется, strTitle – заголовок окна. Если отсутствует, то заголовок окна будет по умолчанию «Сервер сценариев», natType – параметр, определяющий картинку и кнопку в данном окне.

В сценариях VBScript, кроме использования диалоговых окон Windows, Script Host может использовать собственные диалоговые окна MsgBox (окно вывода информации) и InputBox (окно ввода информации). Параметры, используемые в данных функциях, аналогичны применяемым в языке VBA.

Методы и свойства объекта WshShell. Объект WshShell служит для работы с переменными среды Windows, специальными папками, запускает

другие программы, создает ярлыки и т. д. Для его использования его необходимо создать методом CreateObject:

Set WshShell = CreateObject("WScript.Shell")

Информация о системе. Переменные среды или переменные окружения (англ. environment variables) – текстовые переменные операционной системы, хранящие данные о ряде настроек системы. Переменные среды Microsoft Windows делятся на две категории:

1) переменные среды пользователя – указывают путь до пользовательских директорий;

2) системные переменные – хранят данные о некоторых директориях операционной системы и конфигурации компьютера.

Список переменных среды Windows можно узнать, набрав в командной строке: set или нажав кнопку «Переменные среды» на вкладке «Дополнительно» в диалоговом окне «Свойства системы» («Мой компьютер» → «Свойства»).

Свойство Environment объекта WshShell позволяет работать с системными переменными среды. Для этого создается объект WshEnvironment.

Синтаксис: WshShell.Environment([strType]), где strType – может принимать параметры «System», «User», «Volatile» или «Process».

Значение «Volatile» используется для работы с данными, передаваемыми другими программами, а «System», «User» и «Process» используются для работы с системными данными (таблица 4.2).

Таблица 4.2 – Системных данных среды, доступных с помощью значений «System», «User» и «Process»

Значение	Описание	Место присутствия		
		System	User	Process
NUMBER_OF_PROCESSORS	Количество процессоров на данном компьютере	X	–	X
PROCESSOR_ARCHITECTURE	Тип процессора	X	–	X
PROCESSOR_IDENTIFIER	Расширенные данные о процессоре	X	–	X
PROCESSOR_LEVEL	Поколение процессора	X	–	X
OS	Операционная система	X	–	X
COMSPEC	Путь к файлу командной строки (cmd.exe или command.com)	X	–	X
HOMEPATH	Каталог по умолчанию для пользователей	–	–	X

Окончание таблицы 4.2

Значение	Описание	Место присутствия		
		System	User	Process
HOMEDRIVE	Первый локальный диск (обычно C:)	–	–	X
PATH	Заданные системные пути	X	X	X
PATHEXT	Исполняемые файлы (.exe, .com и т. д.)	X	–	X
SYSTEMDRIVE	Диск, на котором находится директория с операционной системой	–	–	X
SYSTEMROOT	Директория Windows	–	–	X
WINDIR	Директория Windows	X	_	X
TEMP или TMP	Папка для временных файлов	_	X	X

Объект Environment, как и все коллекции WSH, имеет свойство Count, в котором хранится число элементов коллекции, и метод.

Порядок выполнения работы

1 Создайте с помощью Блокнота файл, выводящий сообщение. Например, введите текст:

WScript.Echo "Это первый сценарий"

или

WScript.Echo "Это первый сценарий";

2 Сохраните файл с любым из допустимых расширений (.vbs или .js).

3 Запустите сценарий на выполнение в диалоговом режиме и режиме командной строки.

4 Задайте настройки для сценария, для чего выполните следующие действия. Щелкните на нем правой кнопкой мыши и в окне свойств файла выберите вкладку «Сценарий». Измените настройки на этой вкладке, поставив или сняв любой флажок, чтобы кнопка «Вернуть установки по умолчанию» стала доступна. После этого щелкните OK. Появился файл с расширением wsh.

5 В дальнейшем, если необходимо использовать измененные настройки, нужно вместо файла с расширением .vbs или .js запускать файл с расширением .wsh.

6 Откройте файл с расширением .wsh с помощью Блокнота. Там будет примерно следующий текст:

[ScriptFile]
Path=C:\Мои документы\Пример1.VBS
[Options]

Timeout=10
DisplayLogo=1

Параметр Path в разделе *[ScriptFile]* содержит путь к файлу, для которого используется wsh-файл. Настройки в разделе *[Options]* – это сами настройки, для которых, собственно, и создан файл настройки.

Timeout – определяет время, отведенное для выполнения сценария.

DisplayLogo – отвечает за вывод эмблемы Windows Script Host при запуске в командном режиме. Если изменить его на 0, то эмблема отображаться не будет.

BatchMode – включает/выключает пакетный режим. Если ему присвоить значение 1, то сценарий будет выполнятся в пакетном режиме – без вывода информации на экран и сообщений об ошибках.

7 Создайте файлы сценариев, которые выводят сведения о системе согласно варианту по таблице 4.3.

Таблица 4.3 – Варианты индивидуального задания

Вариант	Сведения о системе
1	Операционная система, директория Windows
2	Расширенные данные о процессоре, поколение процессора
3	Заданные системные пути, директория Windows
4	Тип процессора, операционная система
5	Количество процессоров на данном компьютере, заданные системные пути
6	Диск, на котором находится директория с операционной системой, каталог по умолчанию для пользователей
7	Первый локальный диск, поколение процессора
8	Путь к файлу командной строки, папка для временных файлов
9	Тип процессора, заданные системные пути
10	Заданные системные пути, поколение процессора

Контрольные вопросы

1 Для чего предназначен сервер сценариев?
2 Какими возможностями обладает сервер сценариев?
3 Как запускается сервер сценариев?

5 Лабораторная работа № 6. Изучение архитектуры ОС семейства Windows. Управление сервисами, процессами и потоками

Цель работы: овладеть навыками работы с процессами и потоками в Windows /2000/7.

Методические указания

Архитектура ОС Windows имеет модульную структуру и состоит из двух основных уровней – компонентов, работающих в режиме пользователя,

и компонентов режима ядра. Программы и подсистемы, работающие в режиме пользователя, имеют ограничения на доступ к системным ресурсам. Режим ядра имеет неограниченный доступ к системной памяти и внешним устройствам. Ядро системы NT называют гибридным ядром или макроядром. Архитектура включает в себя само ядро, уровень аппаратных абстракций (HAL), драйверы и ряд служб (Executives), которые работают в режиме ядра (Kernel-mode drivers) или в пользовательском режиме (User-mode drivers)

Пользовательский режим Windows состоит из подсистем, передающих запросы ввода-вывода соответствующему драйверу режима ядра посредством менеджера ввода-вывода. Есть две подсистемы на уровне пользователя: подсистема окружения (запускает приложения, написанные для разных операционных систем) и интегрированная подсистема (управляет особыми системными функциями от имени подсистемы окружения).

Процесс в Windows состоит из следующих компонентов.

1 Структура данных, содержащая всю информацию о процессе, в том числе список открытых дескрипторов различных системных ресурсов, уникальный идентификатор процесса, различную статистическую информацию и т. д.

2 Адресное пространство – диапазон адресов виртуальной памяти, которым может пользоваться процесс.

3 Исполняемая программа и данные, проецируемые на виртуальное адресное пространство процесса.

Поток (thread) – сущность внутри процесса, получающая процессорное время для выполнения. В каждом процессе есть минимум один поток.

В Windows реализована система вытесняющего планирования на основе приоритетов, в которой всегда выполняется поток с наибольшим приоритетом, готовый к выполнению. Выбранный для выполнения поток работает в течение некоторого периода, называемого квантом. Квант определяет, сколько времени будет выполняться поток, пока операционная система не прервет его. По окончании кванта операционная система проверяет, готов ли к выполнению другой поток с таким же (или большим) уровнем приоритета. Если таких потоков не оказалось, текущему потоку выделяется еще один квант. Однако поток может не полностью использовать свой квант. Как только другой поток с более высоким приоритетом готов к выполнению, текущий поток вытесняется, даже если его квант еще не истек.

Процессорное время выделяется потокам в соответствии с их уровнем приоритета. В Windows существует 32 уровня приоритета, от 0 до 31. Они группируются так:

31–16 – уровни реального времени;

15–1 – динамические уровни;

0 – системный уровень, зарезервированный для потока обнуления страниц (zero-page thread).

При создании процесса, ему назначается один из шести классов приоритетов, приведенных в таблице 5.1.

Таблица 5.1 – Классы приоритетов процессов

Класс	Флаг в функции Create Process	Числовой уровень
Realtime (реального времени)	REALTIME_PRIORITY_CLASS	24
High (высокий)	HIGH_PRIORITY_CLASS	13
Above normal* (выше нормального)	ABOVE_NORMAL_ PRIORITY_CLASS	10
Normal (нормальный)	NORMAL_PRIORITY_CLASS	8 (7–9)
Below normal* (Ниже нормального)	BELOW_NORMAL_ PRIORITY_CLASS	6
Idle (простаивающий)	IDLE_PRIORITY_CLASS	4

Порядок выполнения работы

Разработайте сценарий WMI, выполняющий запись в текстовый файл сведений об службах, согласно варианту:

1) сведений о службах (Уникальный идентификатор службы, Полное описание службы, Тип службы) ОС, которые могут быть приостановлены;

1) сведений о службах ОС (Учетная запись, от имени которой запускается служба, Полное описание службы, Тип службы), которые не взаимодействуют с рабочим столом пользователей;

2) сведений о службах ОС (Уникальный идентификатор службы, Краткое описание службы, Полный путь к бинарному файлу, соответствующему службе), которые являются интерактивными процессами;

3) ОС, имеющих тип Kernel Driver (Уникальный идентификатор службы, Имя службы, способ загрузки службы);

4) сведений о службах ОС, которые взаимодействуют с рабочим столом пользователей;

5) сведений о службах ОС, которые загружаются автоматически (Способ загрузки службы, Уникальный идентификатор службы, Имя службы);

6) сведений о службах ОС (Полное описание службы, Полный путь к бинарному файлу, Уникальный идентификатор службы) соответствующих службе о службах ОС, которые загружаются автоматически;

7) сведений о службах ОС (Полное описание службы, Уникальный идентификатор службы, Текущее состояние службы), которые могут быть запущены вручную;

8) сведений о службах ОС, имеющих тип Share Process (Уникальный идентификатор службы, Имя службы, способ загрузки службы);

9) сведений о службах ОС, которые имеют тип File System Driver (способ загрузки службы, уникальный идентификатор службы, Имя службы).

Контрольные вопросы

1 Из чего состоят процесс и поток в среде Windows?

2 Какие классы приоритетов применяются в ОС Windows?

3 В чем отличие базового и текущего приоритетов?

6 Лабораторная работа № 7. Изучение команд для работы с файловой системой ОС семейства UNIX

Цель работы: изучить работу с виртуальными машинами, установку ОС семейства UNIX.

Методические указания

Концепция виртуальной машины доводит подход, основанный на уровнях абстракции, до своего логического завершения. Согласно данной концепции, совокупность аппаратуры и ОС трактуется как аппаратура (машина). Виртуальная машина предоставляет интерфейс, полностью аналогичный интерфейсу обычной машины без базового программного обеспечения.

Идея естественной виртуализации: поверх аппаратного уровня (физический сервер) располагается уровень монитора виртуальных машин VMM (гипервизор). Гипервизор полностью эмулирует компьютер и способен поддерживать выполнение более чем одной операционной системы. На VMM выполняются так называемые гостевые операционные системы (guest OS) виртуальных машин, непосредственно поддерживающие работу приложений.

Платформа VirtualBox представляет собой настольную систему виртуализации для Windows, Linux и Mac OS хостов, поддерживающую операционные системы Windows, Linux, OS/2 Warp, OpenBSD и FreeBSD в качестве гостевых.

Краткая характеристика ОС семейства UNIX. ОС UNIX является одной из наиболее распространенных операционных систем современных компьютеров. Своим поразительным долголетием UNIX обязана таким качествам, как хорошо продуманная логическая организация, возможность достаточно простого переноса на компьютеры с другой архитектурой, а также доступность исходных текстов системы (по крайней мере в ранний период развития UNIX).

ОС Linux – это многопользовательская, многозадачная, многотерминальная операционная система (ОС) из семейства UNIX, под управлением которой могут одновременно выполняться несколько задач. Она предназначена для работы на серверах и рабочих станциях, обеспечивает подключение дополнительных терминалов и допускает этом режиме использование графических оболочек.

ОС Linux является сетевой операционной системой для 32- или 64-разрядных платформ. Она обеспечивает маштабируемость в диапазоне от игровых приставок (Sony Play Station) до кластерных серверов Internet. ОС Linux не связана с конкретной моделью компьютеров. Её ядро реализовано на языке высокого уровня (языке СИ), что позволяет достаточно легко переносить эту систему с одной платформы на другую. Система распространяется по лицензии GNU либо подобным свободным лицензиям, обеспечивается как коммерческое, так и свободное сопровождение через Internet. Поставка исходных модулей системы обеспечивает возможность адаптации прикладных программ в случае перехода на другую платформу и дает возможность контроля кодов, реализующих несанкционированный доступ.

В разработке системы приняло участие большое количество специалистов, зарегистрировавших свои авторские права, что дает гарантии ее немонополизации.

Все действия в ОС UNIX оформлены как процессы. Процесс представляет собой совокупность выполняемых программ или одну выполняемую программу, которые вызываются при исполнении системной команды. Процесс может породить один или несколько других процессов, которые могут выполняться параллельно. ОС Linux поддерживает многопроцессорную архитектуру для параллельного выполнения процессов.

Работа в консоли – самый быстрый и удобный интерфейс для решения ряда задач. Независимо от того, какой дистрибутив используется, базовые команды будут одни и те же. Нельзя забывать и о том, что текстовый режим устойчивее графического.

Поскольку графический интерфейс Linux – это, по сути, дела обычная прикладная программа, то ее неработоспособность не приводит к общему краху системы. Если пользователь не боится текстового режима, то он быстро внесет необходимые изменения в соответствующий конфигурационный файл и заново запустит систему. В противном случае придется прибегнуть к полной ее переустановке, что значительно дольше.

Перейти в режим командной строки можно двумя способами. Первый – активация текстовой консоли. Для этого следует нажать комбинацию клавиш Ctrl+Alt+F[номер консоли]. Появится строка приглашения на регистрацию в системе, где нужно последовательно набрать логин и пароль. Второй способ – открытие консоли непосредственно в оконном менеджере. При этом пользователь продолжает работать в графическом режиме. Важно понимать, что и в первом, и во втором случаях все запущенные программы будут продолжать нормально функционировать.

Для переключения текстовых консолей нужно нажимать не Ctrl+Alt+F [номер консоли], а просто Alt+F[номер консоли] – клавиша Ctrl применяется только в графическом режиме.

При работе в текстовом режиме визуальный маркер начала строки может быть двух видов:

1) *знак диеза* (*#*) – указывает на то, что пользователь работает под именем root и ему открыты все файлы системы. В этом случае необходимо проявлять особую осторожность – необдуманные действия чреваты серьезными последствиями;

2) *знак доллара* (*$*) – обычный пользователь.

Концепция ограничения прав доступа приводит к тому, что некоторые команды, выполнение которых подразумевает расширенные полномочия, откажутся запускаться. Причем специфика UNIX такова, что никаких подсказок или пояснений на экране не появится – предполагается, что человек полностью контролирует систему и нисколько не нуждается в помощи программ, от которых требуется только беспрекословное выполнение распоряжений хозяина.

Пользовательские оболочки Linux принято делить на две категории: оконные менеджеры и интегрированные графические среды. Первые предоставляют потребителю только механизм управления визуальными

объектами, тогда как вторые включают в себя дополнительное ПО. XWindow (именно Window, а не Windows – обратите на это внимание) – графическая среда для UNIX-систем. В ее основу легла клиент-серверная модель, только реализована она в пределах одной рабочей станции. Для передачи данных используется специальный протокол сетевой связи (X Network Protocol).

Формат команд в ОС UNIX следующий:

имя команды [аргументы] [параметры] [метасимволы]

Имя команды может содержать любое допустимое имя файла; аргументы – одна или несколько букв со знаком минус (–); параметры – передаваемые значения для обработки; метасимволы интерпретируются как специальные операции. В квадратных скобках [] указываются необязательные части команд.

Команда

touch имя_файла

изменяет время последней модификации файла на текущее. Побочный эффект: если файла нет, он создается с нулевым размером.

Команда

>имя_файла

создает файл нулевого размера (буквально: в этот файл перенаправляется вывод из ниоткуда).

Команда создания каталога

mkdir имя_директории

создает новый каталог. Если используется в следующем формате (**`mkdir имя_директории1 имя_директории2 имя_директории3`**), создаст папки: имя_директории1, имя_директории2 и имя_директории3.

Команда копирования файла в другой файл или каталог

cp *файл-источник файл-или-каталог-приемник*

Если файл-приемник существует, он будет удален (то есть копирование производится поверх). Каталог-приемник должен существовать. UNIX рассчитана на профессиональных пользователей. Предупреждений по поводу удаления файлов не выводится.

Команда рекурсивного копирование каталога в другой каталог

cp-R каталог-источник каталог-приемник

Команда перемещения/переименования файла или каталога

mv *файл-или-каталог-источник файл-или-каталог-приемник*

используется для перемещения или переименования файлов или каталогов. Если в качестве аргументов заданы имена двух файлов, то имя первого файла будет изменено на имя второго (*mv file1 file_1*). Если последний аргумент является именем существующего каталога, то *mv* перемещает все заданные файлы в этот каталог (*mv file ./dir/*).

Если последний аргумент не является каталогом и задано более чем два файла, то будет выдано сообщение об ошибке.

Ключи, использующиеся с *mv*:

– *f* – не запрашивать подтверждения операций;

– *i* – выводить запрос на подтверждение операции, когда существует файл, в который происходит переименование или перемещение;

– –, – завершает список ключей. Применяется для использования с файлами, имена которых начинаются на –.

Символическая связь – особый тип файла, содержащий имя другого файла (в листинге *ls-l* такие файлы обозначаются буквой *l* в первой колонке). Чтение-запись в файл-связь на самом деле приводят к чтению-записи в файл, на который он ссылается. Например, при выводе на экран содержимого символической связи появятся данные файла, на который эта символическая связь ссылается.

Создание символической связи:

ln-s существсвующий_файл файл_связь

(Файл с именем *файл_связь* не должен существовать).

Жесткие связи. В файловой системе UNIX имя файла является указателем на индексный дескриптор (*i-node*), который содержит атрибуты файла и массив адресов дисковых блоков, в которых находятся данные файла (рисунок 6.1). Однако файл может иметь несколько имен. Дескриптор содержит только счетчик числа этих имен, значение которого показывается во второй колонке листинга *ls-l*.

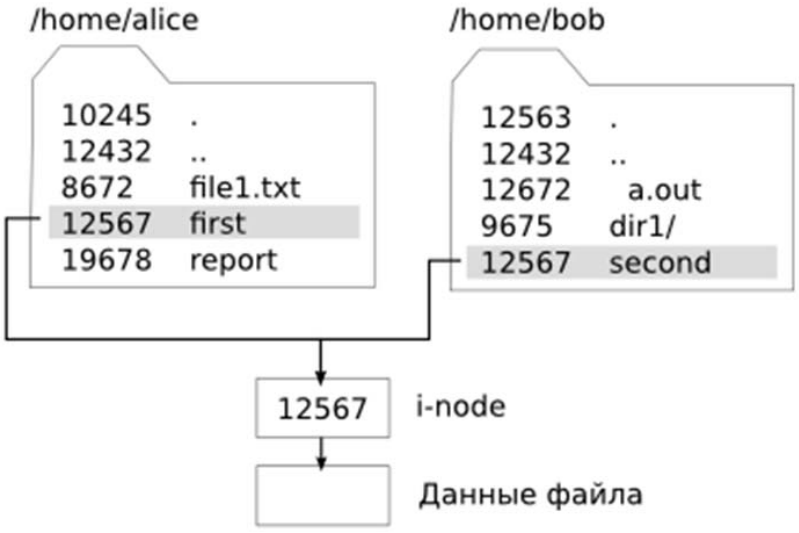

Рисунок 6.1 – Организация жесткой связи

Создание жесткой связи – это создание еще одного имени, ссылающегося на тот же самый индексный дескриптор:

```
ln сущестсвующее_имя_файла новое_имя_файла
```

Команды удаления файлов и каталогов:

rm имя_файла

удаляет файл, но не каталог;

rmdir имя_каталога

удаляет каталог, только если он пуст;

rm -r имя_каталога

удаляет каталог рекурсивно (то есть со всеми подкаталогами), но просит подтверждения при удалении файлов, в которые у вас нет права записи (поскольку для удаления файла достаточно иметь право записи в его каталог);

rm -rf имя_каталога

удаляет каталог рекурсивно и молча.

Удаление файла на самом деле представляет собой уменьшение на единицу счетчика его имен в индексном дескрипторе; соответствующий системный вызов называется *unlink(2)*. Физически файл удаляется системой, если он закрыт и если счетчик имен равен нулю. Это свойство используется программистами при создании временных файлов: после создания файла функцией *open* сразу же вызывается *unlink*, сбрасывающий счетчик имен в 0. Пока файл открыт, его можно использовать (через файловую переменную, возвращенную функцией open; имени у файла уже нет). После вызова *close* файл автоматически уничтожается.

Команда

file имя_файла [имя_файла ...]

определяет тип файла путем анализа его содержимого.

Расширение файла (часть имени после точки) в Unix не имеет никакого специального значения. Например, для запуска файла требуется не расширение типа *.exe*, а право на исполнение.

Команда

cat имя_файла

выводит содержимое файла на терминал.

Команда

more имя_файла

пейджер (выводит файл постранично, с остановом). Фактически команда *man* формирует текст справки и пропускает его через *more*. Поэтому команды управления прокруткой текста (см. выше) в описании *man*.

Команда

tail [-число] имя_файла

выводит последние 10 строк файла или сколько указано.

Команда

head [-число] имя_файла

выводит первые 10 строк файла или сколько указано.

Порядок выполнения работы

1 Установите виртуальную машину и ОС семейства UNIX. Выполните в командной строке перечисленные ниже действия. Протоколируйте процесс выполнения (с помощь скриншотов или записывая команды) и представьте их в отчете.

2 Получите справку о команде *ls* с использованием разных ключей.

3 Создайте двумя способами два файла (с помощью *touch* и >). Не забудьте просмотреть содержимое каталога, где вы создали файлы.

4 В домашнем каталоге создайте два подкаталога *d1* и *d2* с помощью отдельных команд для каждого каталога и одной команды для обоих каталогов.

5 Создайте в домашнем каталоге текстовый файл *test.txt* и скопируйте его в заранее созданный каталог *d1*.

6 Рекурсивно скопируйте каталог *d1* в каталог *d2*. Просмотрите содержимое домашнего и *d2* каталогов.

7 Создайте файл abc.txt и измените его имя на *cba.txt*. Далее переместите этот файл в каталог *d2*. Создайте также каталог *d3* и переместите его в каталог *d2*.

8 Создайте текстовый файл *b.txt* и символическую связь для него *c.txt.* Просмотрите содержимое рабочего каталога и обратите внимание на файл с.*txt* и его размер: 5 – это число символов в имени «*b.txt*».

9 Создайте жесткую связь для файла *b.txt – d.txt*. Просмотрите содержимое рабочего каталога и обратите внимание на значение второй колонки для *b.txt* и *d.txt* и размер этих файлов, заметьте также время последней модификации. Файлы *b.txt* и *d.txt* абсолютно равноправны, т. к. это два разных имени одного и того же физического файла. С помощью команды *touch* измените время

последней модификации файла *d.txt* и выведите данные об обоих файлах (обратите внимание на время последней модификации). Далее выведите *номера индексных дескрипторов* этих файлов и обратите внимание на их значения.

10 Удалите файлы *b.txt*, *c.txt* и *d.txt*. Рекурсивно удалите каталог *d1* в каталоге *d2* (с просьбой подтверждения удаления), а также каталог *d3* из каталога *d2* (также рекурсивно, но без запроса подтверждения). Удалите оставшееся содержимое каталога d2 и удалите *ПУСТОЙ* каталог.

Контрольные вопросы

1 Что такое виртуальная машина и каковы ее возможности?
2 Что считается файлами в ОС UNUX?
3 Какие типы файлов существуют в ОС UNUX?
4 Объясните назначение связей с файлами и способы их создания.
5 Какие методы создания и удаления файлов, каталогов знаете?

7 Лабораторная работа № 8. Создание и выполнение командных файлов в ОС UNIX

Целью работы: изучение методов создания и выполнения командных файлов на языке *Shell*-интерпретатора.

Методические указания

В предыдущих лабораторных работах взаимодействие с командным интерпретатором *Shell* осуществлялось с помощью командной строки. Однако *Shell* является также и языком программирования, который применяется для написания командных файлов (*shell*-файлов). Командные файлы также называются скриптами и сценариями. *Shell*-файл содержит одну или несколько выполняемых команд (процедур), а имя файла в этом случае используется как имя команды.

Простой пример shell-файла. Для создания сценариев можно использовать любой текстовый редактор. Создайте новый файл и введите следующие строки:

```
#!/bin/bash
echo "It is my first shell-file!!!"
# комментарий
ls $HOME
echo "Done!"
```

Первая строка указывает текущей оболочке, какую программу следует использовать для интерпретации файла. В данном случае это оболочка *bash*. Это сделано для того, что, если сценарий вызывается в пределах другой оболочки или файлового менеджера, они будут «знать», что для выполнения этого сценария требуется оболочка *bash*. Вторая строка – это первая команда в

сценарии. Команда *echo* используется для вывода на экран простой информационной строки. Третья строка – комментарий. Далее следуют ещё две команды – команда *ls*, которая берёт имя каталога в качестве параметра, и, наконец, команда *echo*, выводящая на экран информацию об успешном выполнении работы.

Как любой язык программирования, командный язык *shell* поддерживает переменные. Тип их – строковый. Для обозначения переменных *Shell* используется последовательность букв, цифр и символов подчеркивания; переменные не могут начинаться с цифры.

Оператор присваивания выглядит так:

$имя_переменной=значение

Имя должно начинаться с буквы и может состоять из латинских букв, цифр, знака подчеркивания. Если значение переменной содержит специальные символы в имени файла, то при указании его имени в команде этот символ нужно экранировать знаком «\» (обратный слеш) или заключать все имя в двойные кавычки.

Вот ряд символов, которые имеют специальное значение для командного интерпретатора, и их использование не рекомендуется:

~ ! @ # $ & % * ( ) [ ] { } ' " \ : ; > < пробел

Переменные вида *$n*, где *n* – целое число, используются для идентификации позиций элементов в командной строке с помощью номеров, начиная с нуля.

Операция подстановки значения переменной обозначается символом *$*. Вывести значение переменной можно командой *echo*:

```
$ var=”Это моя переменная!”
$ echo var     # выводит имя переменной var
$ echo $var    # выводит значение переменной “Это моя переменная!”
```

Установив перед именем переменной знак **$**, мы сообщаем интерпретатору, что нужно заменить ее значением.

Для вывода переменных можно использовать команду *printf*:

```
PI=3,14159265358979
printf "Число пи с точностью до 2 знака после запятой = %1.2f" $PI
printf "Число пи с точностью до 9 знака после запятой = %1.9f" $PI
```

Команда *expr* (*express* – выражать) вычисляет выражение *expression* и записывает результат в стандартный вывод. Элементы выражения разделяются пробелами; символы, имеющие специальный смысл в командном языке, нужно экранировать. Строки, содержащие специальные символы, заключают в апострофы. Используя команду *expr*, можно выполнять сложение, вычитание,

умножение, деление, взятие остатка, сопоставление символов и т. д.

Сложение, вычитание:

b =190
a =` expr 200 - $b `

где ` – обратная кавычка (левая верхняя клавиша). Умножение *, деление /, взятие остатка %.

Встроенные команды являются частью интерпретатора и не требуют для своего выполнения проведения последовательного поиска файла команды и создания новых процессов.

Команда для работы с данными

echo [ключи] параметры

копирует свои параметры на стандартный вывод (но с учетом специальных символов, если они имеются). Если не задан ключ *–n*, то в конце выдачи добавляется перевод строки. Если задан ключ *–e*, то в выдаваемой строке можно использовать обозначения некоторых «непечатных» символов с помощью знака \. В частности, *n* означает перевод строки, *t* – символ табуляции, *a* – звонок, *a* *nnn*, где *nnn* – от одной до трех восьмеричных цифр, или *xnnn* (*nnn* – от одной до трех шестнадцатеричных цифр) означает соответствующий символ кода ASCII.

Команда

more [файл]

выводит файл-параметр (или, в его отсутствие, стандартный ввод) порциями, умещающимися на экране. Для вывода следующей порции нужно нажать клавишу «пробел».

Команда

less [файл]

просмотра файла. Позволяет перемещаться по файлу вперед и назад. Для выдачи сводки по командам перемещения следует ввести *h*, для выхода из просмотра нужно ввести *q*.

Командой

wc [ключи] [файлы]

для каждого параметра-файла (или для стандартного ввода) выдается строка, содержащая, в зависимости от ключа, число строк в файле (ключ *–l*), число слов (ключ *–w*) или число символов (ключ *–c*). По умолчанию (без ключей) выдаются все три числа.

Команда

head [ключи] [файл]

выдает указанное число первых строк файла-параметра или стандартного ввода. По умолчанию выдаются 10 строк. Ключ *–n* число указывает иное число строк. Ключ *–c* размер указывает, что вместо определенного числа строк следует выдать указанное число начальных байтов, при этом размер можно также указывать в килобайтах (для этого запись размера нужно завершить суффиксом *k*), в мегабайтах (суффикс *m*) или в стандартных блоках по 512 байт (суффикс *b*).

Команда

tail [ключи] [файл]

выдает на стандартный вывод несколько последних строк файла-параметра или стандартного ввода. По умолчанию выдаются 10 строк. Ключи *–n* число и *–c* размер действуют так же, как для команды *head*. Если перед числом строк или байтов записан знак +, то соответствующее число указывает, сколько надо пропустить от начала файла, в противном случае – сколько нужно оставить в конце файла.

Команда

grep [ключи] образец [список_файлов]

выполняет поиск заданной строки-образца в указанных файлах или в стандартном вводе. Если образец содержит пробелы, его следует заключить в кавычки. По умолчанию на стандартный вывод выдаются все строки, содержащие образец. Если проверяется несколько файлов, то перед выводимыми строками выдается имя файла.

При задании образца можно использовать регулярные выражения, задающие шаблон поиска строки. Синтаксис и семантика регулярных выражений напоминают использование подстановочных знаков при поиске в Microsoft Word. Основные символы, используемые при записи регулярных выражений, приведены в таблице 7.1.

Таблица 7.1 – Основные символы, используемые при записи регулярных выражений

Символ	Описание символа
.	Любой символ
\w	Любая буква или цифра
\W	Любой символ, кроме букв и цифр
[символы]	Любой из перечисленных символов. Можно задавать диапазоны через знак – (например, [0–9] соответствует любой цифре)
[^символы]	Любой символ, кроме перечисленных. Например, [^A-Za-z] означает любой символ, кроме латинских букв
^	Начало строки
$	Конец строки
выраж*	Выражение присутствует ноль или более раз
выраж?	Выражение присутствует ноль или один раз
выраж+	Выражение присутствует один или более раз
выраж1выраж2	Последовательное соединение строк, соответствующих выражениям выраж1 и выраж2
выраж1|выраж2	Строка, соответствующая либо выраж1, либо выраж2
(…)	Используются для группировки выражений
\символ	Экранирует специальный символ, т. е. делает его обычным

Например, регулярное выражение *'^A([0-9]+|[^0-9])B?'* означает: «В начале строки должна стоять буква А, за которой может следовать либо одна или несколько цифр, либо ровно один символ, отличный от цифры. После этого должна следовать буква В». То же самое условие можно записать и проще: *'^A.[0-9]*B?'*. Регулярные выражения рекомендуется заключать в апострофы, чтобы *Shell* не попытался интерпретировать некоторые знаки как свои специальные символы.

Команда *grep* может иметь ряд ключей, некоторые из них приведены в таблице 7.2

Таблица 7.2 – Ключи команды *grep*

Ключ	Действие ключа команды
–c	Выдается только число подходящих строк, а не сами строки
–n	Перед каждой строкой выводится ее номер в файле
–i	Игнорируется различие строчных и прописных букв
–h	Отменяется выдача имен файлов
–v	Выдаются только строки, которые НЕ содержат образца
–r	Ищет во всех файлах указанного каталога и его подкаталогов

Ключи команды определяют способ сортировки. По умолчанию строки сортируются по возрастанию, как в словаре. Некоторые ключи приведены в таблице 7.3.

Таблица 7.3 – Ключи команды, определяющие способ сортировки

Ключ	Способ сортировки
–b	Игнорируются пробелы и табуляции в начале строки
–f	Игнорируется различие между прописными и строчными буквами
–n	Поля рассматриваются как числа (возможно, со знаком и десятичной точкой) и сравниваются по числовому значению
–r	Сортировка ведется по убыванию
–t символ	Указанный символ рассматривается как разделитель полей (вместо пробела и табуляции)
–o файл	Результат записывается в указанный файл (вместо стандартного вывода)
–u	Из нескольких строк с одинаковыми значениями сравниваемых полей сохраняется только одна

Команда *cmp [-l |-s] файл1 файл2 [смещение1 [смещение2]]* сравнивает данные в двух файлах. Если файлы идентичны, возвращает код завершения 0, если различаются – код 1, если произошла ошибка (например, файл не найден) – код, больший 1. По умолчанию выдает номер байта и номер строки, в которых найдено первое различие. Нумерация начинается с 1. Если один файл совпадает с начальной частью другого, выдается сообщение о найденном конце файла. Если файлы идентичны, ничего не выдается. Если задан ключ *–l*, то для каждого различия выдается номер байта и различающиеся значения. С ключом *–s* не выдается ничего (только код завершения).

Величины смещений указывают, сколько байт надо пропустить от начала каждого файла, прежде чем начать сравнение.

Простейший способ создать небольшой файл – использовать команду *echo* с перенаправлением стандартного вывода:

$ echo -e "Hello!\nHow are you?" > hello

Другой вариант – использовать команду *cat*, опять-таки с перенаправлением вывода. Текст файла можно задать как стандартный ввод, содержащийся в тексте команды (в режиме «документ здесь»):

```
$ cat > hello << _TEXT_
> Hello!
> How are you?
> _TEXT_
```

Здесь в качестве ограничителя текста можно использовать любое слово, не встречающееся в этом тексте. Еще один вариант – команда с перенаправлением стандартного вывода, но без перенаправления стандартного ввода. После ввода команды «*cat > hello*» *Shell* будет принимать вводимые строки текста, пока пользователь не введет комбинацию *Ctrl+D* (конец файла).

Чтобы выдать список всех имен файлов текущего каталога, можно с равным успехом использовать либо команду «*echo* *», либо команду *ls* без

параметров. Имена будут выданы в одну строку и разделены пробелами. Если нужен иной (не текущий) каталог, можно указать его имя, например: «*echo home/student/**» или «*ls home/student*».

Команда «*ls -l*» выдает имена файлов по одному в строке, сопровождая имена основной информацией о файлах.

Если нужно выдать лишь имена файлов, соответствующие некоторому шаблону, то следует указать в команде требуемый шаблон, например: «*ls a*[123]*» – выдать все имена файлов, начинающиеся с буквы *a* и заканчивающиеся одной из цифр 1, 2 или 3.

Команда *cat* выдает уже не имена, а содержимое заданных файлов. Например, «*cat * > sumfile*» означает: «объединить содержимое всех файлов текущего каталога и записать результат в файл *sumfile*».

По заданному имени входа пользователя выдается, в зависимости от указанного ключа, либо его полное имя, либо идентификатор пользователя, либо идентификатор группы пользователя. Допускается задание сразу нескольких ключей. Например, скрипт носит имя *userinfo*, а допустимые ключи – *n* (полное имя), *u* (идентификатор пользователя) и *g* (идентификатор группы), то где синтаксис вызова *userinfo [-nug] имя_входа.*

Порядок выполнения работы

1 По номеру варианта определить номера задач (таблицы 7.4 и 7.5), для которых разработать *shell*-программы. Привести в отчете код и тестирование программы с различными наборами данных.

Таблица 7.4 – Определение номеров задач

Вариант	Номер задачи
1	1, 2, 9, 16, 17
2	1, 3, 8, 15, 18
3	1, 4, 11, 14, 21
4	1, 5, 7, 13, 20
5	1, 6, 10, 12, 19
6	1, 2, 11, 13, 21
7	1, 4, 9, 12, 19
8	1, 5, 10, 15, 20
9	1, 3, 7, 16, 18
10	1, 6, 8, 14, 17

Таблица 7.5 – Содержание задач

Номер задачи	Содержание задачи
1	Вывод на экран списка параметров командной строки с указанием номера каждого параметра
2	Формирование файла со списком файлов в домашнем каталоге, вывод на экран этого списка в алфавитном порядке и общего количества файлов
3	Формирование файла со списком файлов в домашнем каталоге, вывод на экран этого списка в порядке возрастания времени создания файла и общего количества файлов
4	Формирование файла со списком файлов в домашнем каталоге, вывод на экран этого списка в порядке возрастания времени создания файла и общего количества файлов
5	Формирование файла со списком файлов в домашнем каталоге, вывод на экран этого списка в порядке убывания времени обращения к файлу и общего количества файлов
6	Формирование файла со списком файлов в домашнем каталоге, вывод на экран этого списка в порядке возрастания времени изменения файла и общего количества файлов
7	Запрос и ввод имени каталога, переход в этот каталог, формирование файла с листингом каталога и возвращение в исходный каталог
8	Запрос и ввод имени пользователя, сравнение с текущим логическим именем пользователя и вывод сообщения: верно/неверно
9	Запрос и ввод числового идентификатора пользователя, сравнение с текущим идентификатором пользователя и вывод сообщения: верно/неверно
10	Запрос и ввод имени файла в текущем каталоге и количества дней. Вывод сообщения о файле: обновлялся / не обновлялся
11	Запрос и ввод имени файла (задается полный путь) и его типа, сравнение с действительным типом файла и вывод сообщения: совпадает / не совпадает
12	Циклическое чтение системного времени и выполнение очистки экрана в заданный момент
13	Циклическое чтение системного времени и переход в корневой каталог в заданный момент
14	Циклический просмотр списка файлов и выдача сообщения при появлении заданного имени в списке
15	Циклический просмотр списка каталогов и выдача сообщения при появлении заданного имени в списке
16	Циклическое чтение системного времени и в заданный момент создание каталога
17	Для каждого из файлов, перечисленных в списке параметров, создать отдельный подкаталог своего домашнего каталога и переместить туда файл. В случае, если нельзя выполнить перемещение (нельзя удалить файл), запрашивать пользователя, выполнять ли копирование или пропустить файл. Имена подкаталогов строить путем добавления к имени домашнего каталога чисел 1, 2, 3 и т. д.
18	Создать вручную «телефонный справочник», состоящий из нескольких записей, содержащих три поля: фамилия, адрес, номер телефона. Поля записи разделять знаком табуляции. Составить скрипт, который по заданной фамилии или адресу, или номеру телефона (в зависимости от указанного ключа) выдает значения двух других полей соответствующей записи

Окончание таблицы 7.5

Номер задачи	Содержание задачи
19	Провести копирование из одного каталога (источника) в другой каталог (приемник) всех файлов, имена которых удовлетворяют заданному шаблону. В зависимости от заданного ключа, запрашивать подтверждение копирования либо для каждого файла, либо только в случае замены существующего файла, либо никогда
20	Выполнить в диалоге настройку поиска файла: запросить и ввести шаблон имени, начальный каталог поиска, тип файла, число дней после изменения файла или после обращения к нему. Выполнить поиск и вывести имена найденных файлов
21	Найти в указанном каталоге все файлы, содержащие заданную строку. Для каждого найденного файла запросить действие, которое необходимо выполнить: удалить файл, запретить доступ к нему прочих пользователей или оставить, как есть

Контрольные вопросы

1 Какие функции выполняет командный интерпретатор *Shell*?
2 Какое назначение имеют *shell*-файлы?
3 Как создать *shell*-файл и сделать его выполняемым?
4 Какие типы переменных используются в *shell*-файлах?
5 В чем заключается анализ цепочки символов?
6 Какие встроенные команды используются в *shell*-файлах?
7 Как производится управление программами?
8 Назовите операторов создания циклов.
9 Как задаются и выполняются простые и сложные команды?

Список литературы

1 Операционные системы. Основы UNIX: учебное пособие / А. Б. Вавренюк [и др.] – Москва: ИНФРА-М, 2021. – 160 с.

2 **Беспалов, Д. А.** Операционные системы реального времени и технологии разработки кросс-платформенного программного обеспечения: учебное пособие: в 2 ч. / Д. А. Беспалов, С. М. Гушанский, Н. М. Коробейникова – Ростов-на-Дону; Таганрог: Южн. федер. ун-т, 2019. – Ч.1. – 139 с.

3 **Беспалов, Д. А.** Операционные системы реального времени и технологии разработки кросс-платформенного программного обеспечения: учебное пособие: в 2 ч. / Д. А. Беспалов, С. М. Гушанский, Н. М. Коробейникова. – Ростов-на-Дону; Таганрог: Южн. федер. ун-т, 2019. – Ч. 2. – 168 с.