МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное
учреждение высшего образования

«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(ФГБОУ ВО «КубГУ»)

Физико-технический факультет

Кафедра оптоэлектроники

Методическое пособие к лабораторным занятиям по дисциплине «Анализ и синтез информационных систем» для обучающихся бакалавриата по направлению подготовки 11.03.02 «Инфокоммуникационные технологии и системы связи»

Составитель: преподаватель Гусев АА.

Краснодар 2017

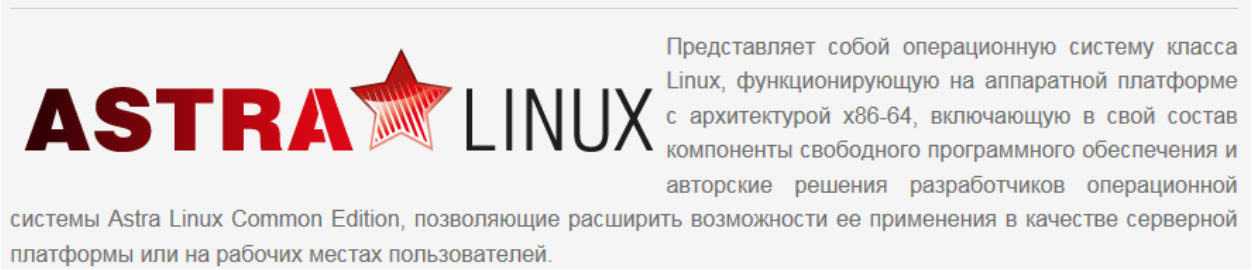
Оглавление

Лабораторная работа № 1. Инсталляция операционной среды GNU/Linux

Цель работы:
– изучить способы инсталляции операционной среды;
– изучить методику устранения неполадок этапа инсталляции операционной среды;
– освоить основные методы планирования организации файловой системы на диске;
– научиться подключать и настраивать периферийное оборудование на этапе инсталляции операционной среды
В процессе выполнения работы студент, руководствуясь методическими указаниями к выполнению данной работы:
– определяет параметры, передаваемые программе-установщику операционной среды;
– разрабатывает план организации файловой системы на диске;
– осуществляет инсталляцию операционной среды;
– предоставляет образ установленной операционной среды для проверки и отвечает на вопросы преподавателя для получения зачета за выполненную работу.

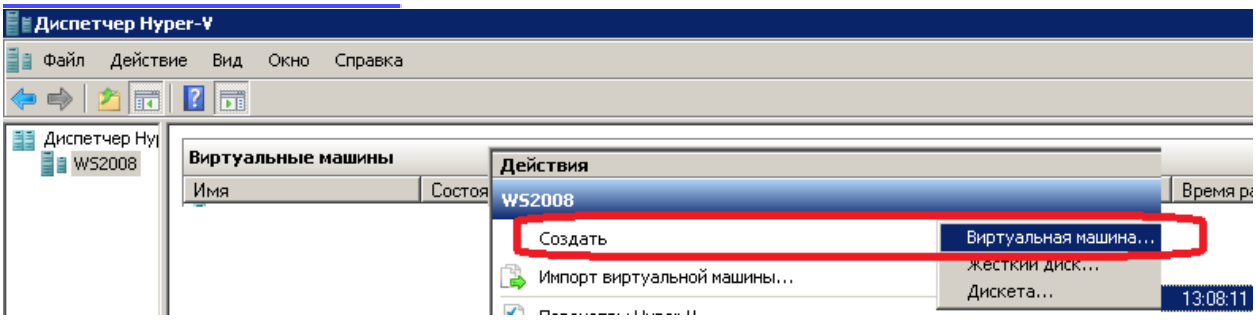
Ход работы

Часть 1. Установка операционной системы
Устанавливаемая операционная система Astra Linux Common Edition.
Представляет собой операционную систему класса Linux, функционирующую на аппаратной платформе с архитектурой x86-64, включающую в свой состав компоненты свободного программного обеспечения и авторские решения разработчиков операционной системы Astra Linux Common Edition, позволяющие расширить возможности ее применения в качестве серверной платформы или на рабочих местах пользователей.
Версия 1.11. релиз "Орел"
Образ установочного диска с контрольными суммами на FTP-сервере
http://mirror.yandex.ru/astra/stable/orel/1.11/iso/orel-1.11-07.07.2017_15.58.iso
http://mirror.yandex.ru/astra/stable/orel/1.11/iso/orel-1.11-07.07.2017_15.58.md5

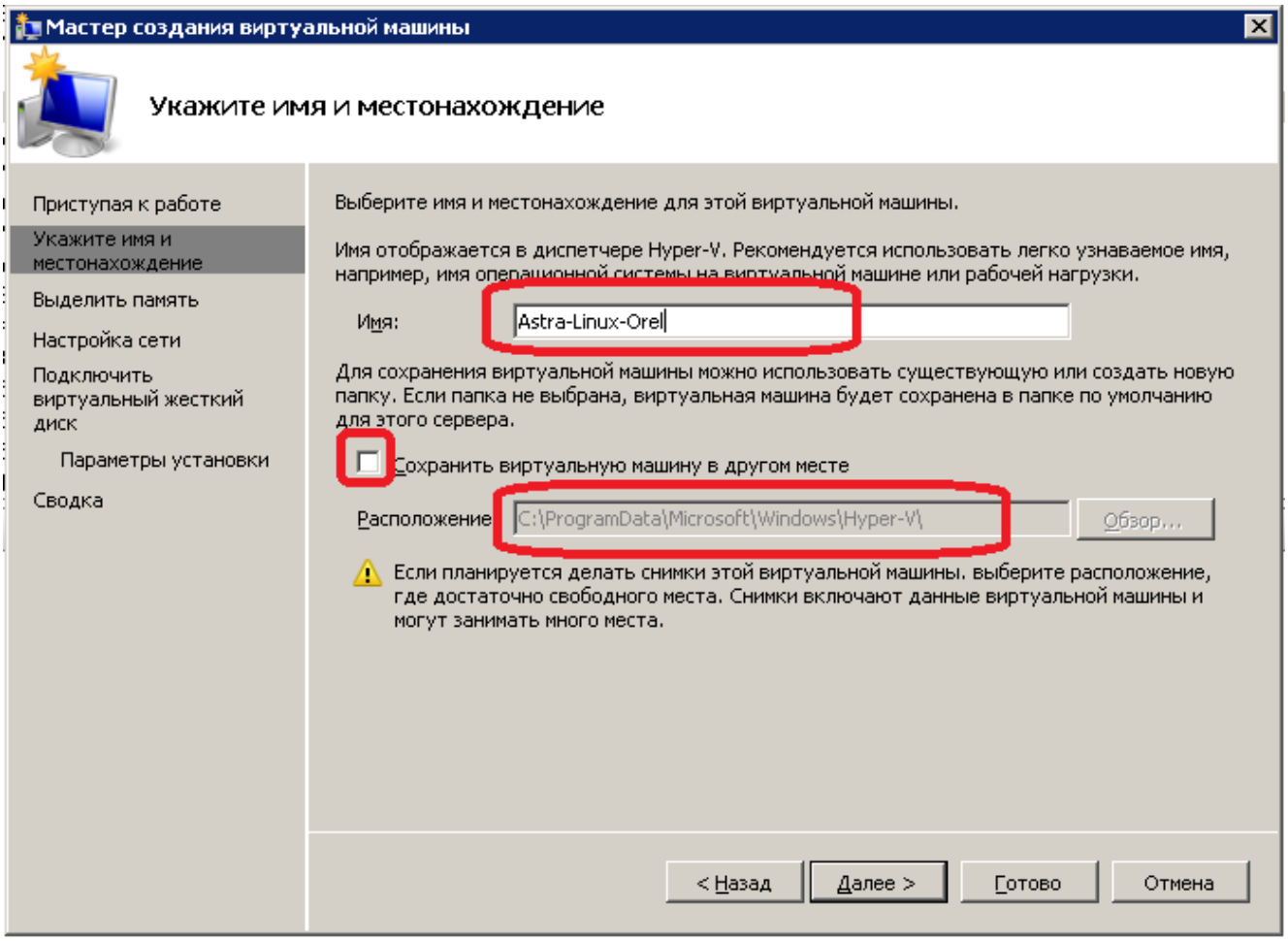


В состав базовой программной платформы входят следующие компоненты:

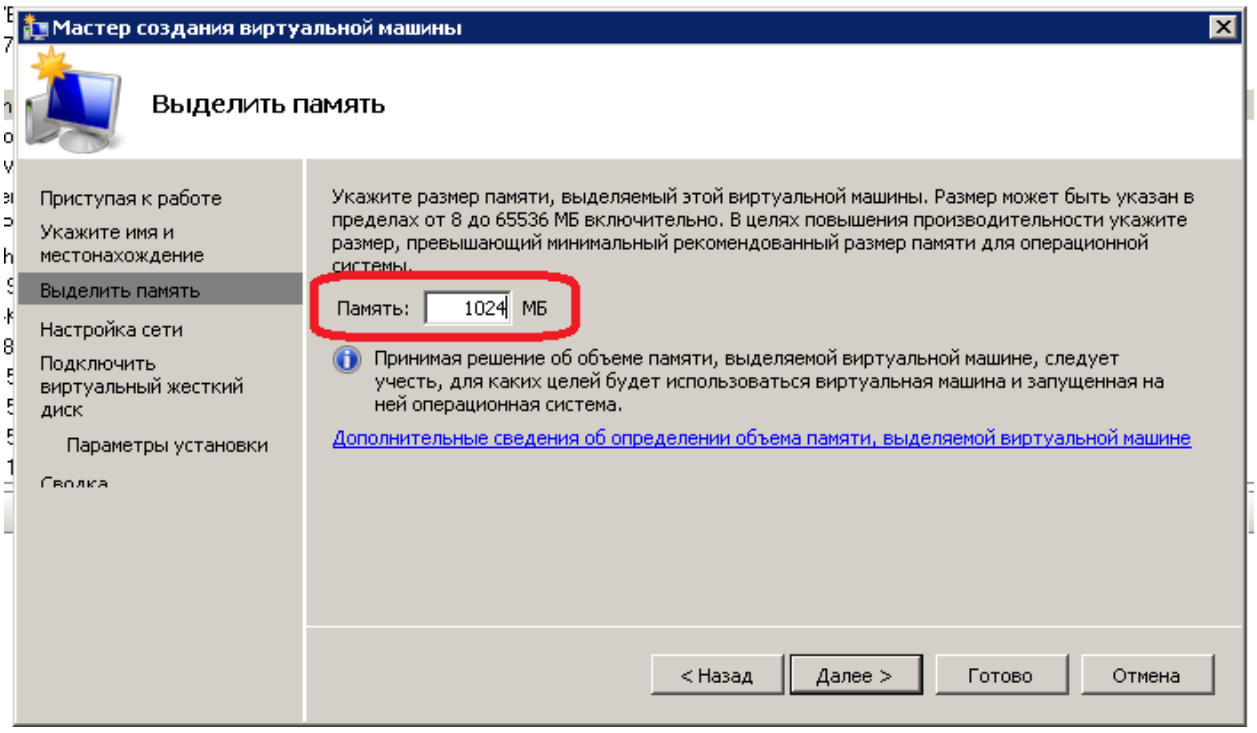
http://astra-linux.ru/soctav.html
Последовательность установки.
1. Предварительные действия по созданию виртуальной машины для установки.
Запустить диспетчер Hyper-V. Выбрать создание виртуальной машины.



2



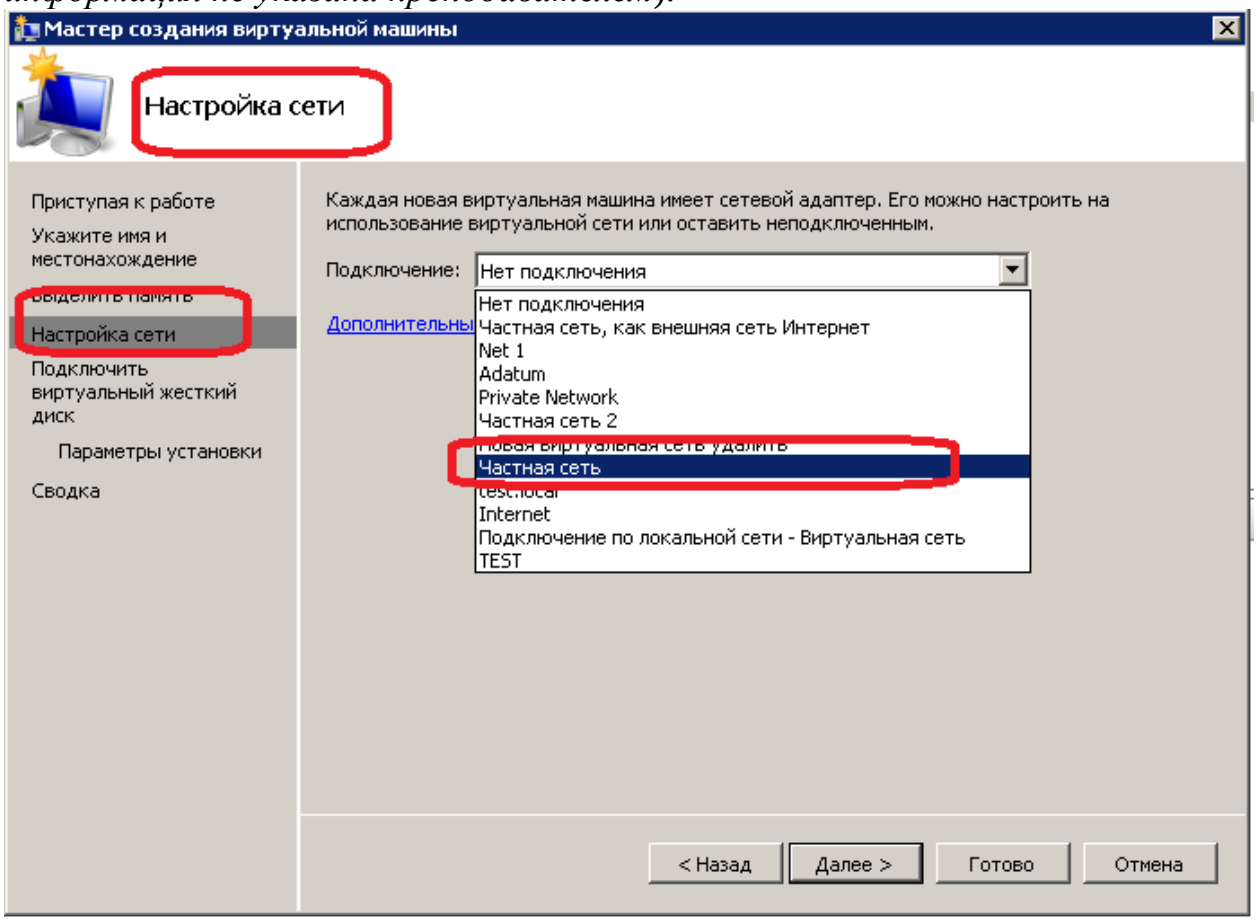
Имя виртуальной машины Astra-Linux-Orel, расположение по умолчанию (*если не указана преподавателем другая информация*).

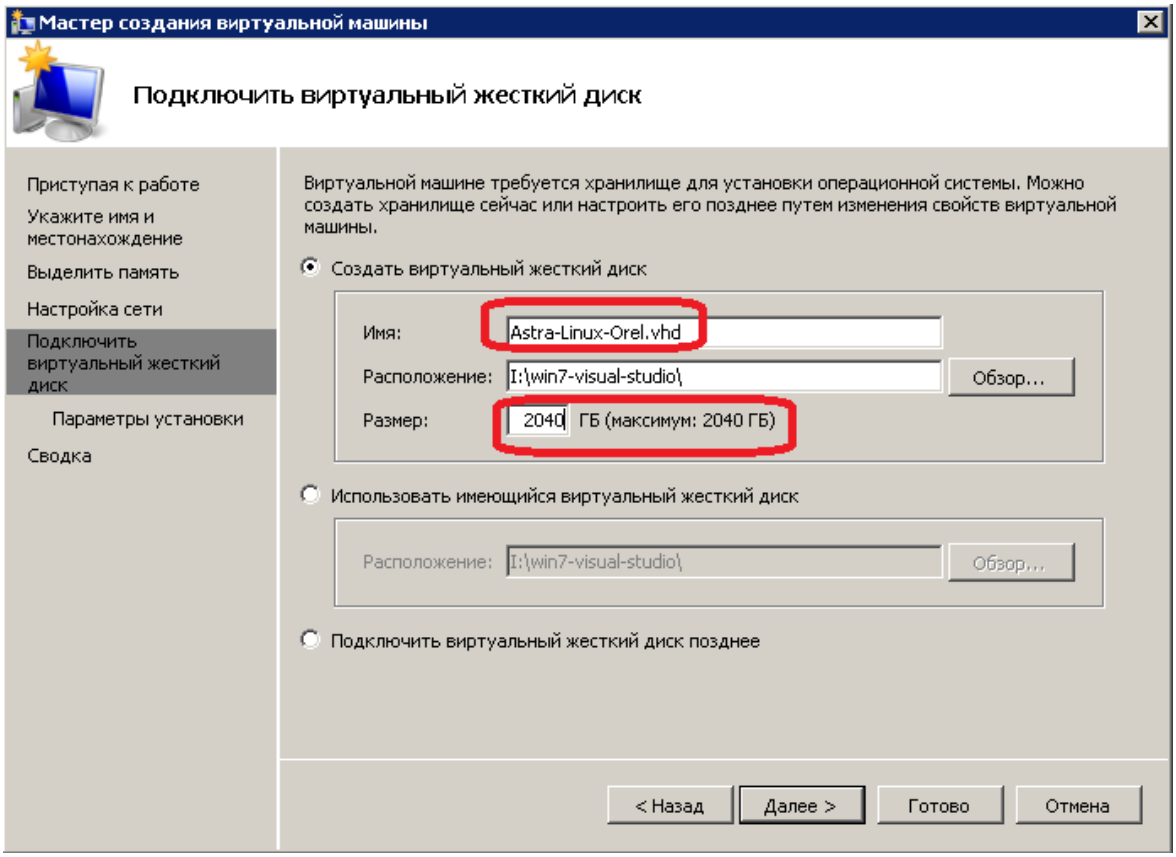


3

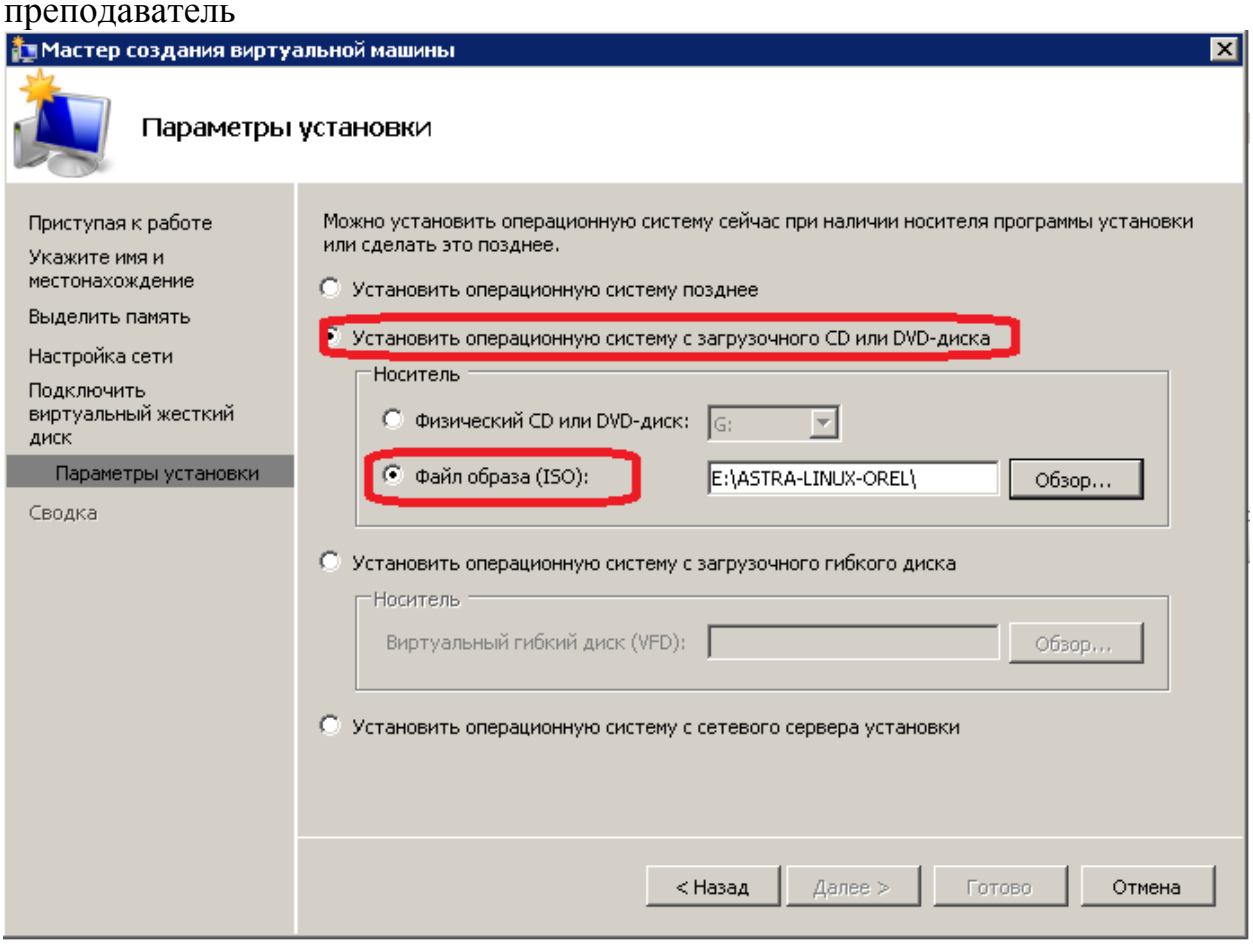
Память 1024. Тип используемой сети – частная (название сети задает преподаватель).

Виртуальный жесткий диск создается максимальной емкости, имя диска Astra-Linux-Orel.vhd, расположение размещения по умолчанию (*если иная информация не указана преподавателем*).





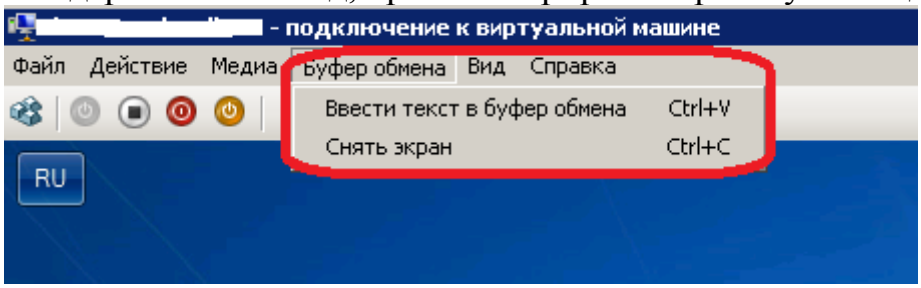
Расположение ISO образа для установки операционной системы указывает преподаватель



После выполнения установки следует запустить на выполнение вирутальную машину.
Для снятия копий экрана при работе с виртуальной машиной следует использовать пункт меню Буфер обмена --> Снять экран (или комбинацию горячих клавиш Ctrl + C).
Изображение копируется в буфер обмена компьютера, затем его можно вставить в отчет (или сразу в документ редактора Word или в графический редактор, выполнить в нем обработку, а затем вставить в отчет).
2. Собственно установка операционной системы.
В процессе установки используются опции, предлагаемые по умолчанию. Имя компьютера **AstraLinux**, если преподавателем не задано иное. Студент, выполняющий работу, создает учетную запись со своей фамилией и именем. Если работа выполняется в составе бригады, то должны быть после установки созданы индивидуальные учетные записи, позволяющие работать с операционной системой всем членам бригады. Пароль для всех создаваемых записей Pa$$w0rd. Цель установки – подготовить рабочее место для пользователя, позволяющее ему выполнять типовые действия (работа с документами, электронной почтой, просмотр ресурсов сети Интернет и пр.). С учетом поставленной задачи критически оценивайте позиции, выбираемые для установки. Например, совершенно очевидно с учетом сказанного, что вариант установки системы в виде конфигурации для обслуживания интернет киоска, выбирать и использовать не следует!

Часть 2. Основные команды для работы с файловой системой

Выполнить все команды, описание которых дано ниже. Запомнить используемые команды. В отчете по лабораторной работе привести названия и содержание команд, проиллюстрировать работу команд скриншотами.



---

1. pwd – Определить местоположение в дереве файловой системы

Команда pwd выводит полное имя директории, которая на настоящий момент является активной.

[igor@anantchenko/root]$ pwd
/root
[igor@anantchenko/root]$

Команда может быть использована как в интерактивном режиме, так и в программах, написанных на одном из скриптовых командных языков (это относится и к любой другой программе). Результат работы этой команды, например, видим в приглашении ко вводу командной строки.

2. man и apropos

Команды man и apropos самые главные команды в любой UNIX – системе. В принципе, можно начинать работать с системой зная только эти две команды. Команда man предназначена для вывода на экран раздела документации по интересующей программе, системном файле, библиотечной функции или логическому устройству системы. Рассмотрим раздел документации по известной команде pwd.

[igor@anantchenko/root]$ man pwd

PWD(1) FSF PWD(1)
NAME
pwd - print name of current/working directory
SYNOPSIS
pwd [OPTION]
DESCRIPTION
Print the full filename of the current working directory. --help display this help and exit --version
output version information and exit
REPORTING BUGS
Report bugs to <bug-sh-utils@gnu.org>.
SEE ALSO
The full documentation for pwd is maintained as a Texinfo manual. If the info and pwd programs are properly installed at your site, the command
info pwd should give you access to the complete manual.
COPYRIGHT Copyright © 1999 Free Software Foundation, Inc. This is free software; see the source for copying conditions. There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. GNU sh-utils 2.0 August 1999 1
[igor@anantchenko/root]$

Можно видеть, что приведена документация, достаточная для понимания того, для чего команда предназначена и какие у нее могут быть параметры. Если команда man предназначена для вывода раздела документации, то команда apropos предназначена для поиска этого раздела. Например, хотим посмотреть все команды, ответственные за работу с директориями.
[igor@anantchenko/usr]$ apropos directory

TIFFCurrentRow, TIFFCurrentStrip, TIFFCurrentTile, TIFFCurrentDirectory, TIFFLastDirectory, TIFFFilen o, TIFFFileName, TIFFGetMode, TIFFIsTiled, TIFFIsByteSwapped, TIFFIsUpSampled, TIFFIsMSB2LSB (3t) - q uery routines TIFFPrintDirectory (3t) - print a description of a TIFFReadDirectory (3t) - get the contents of the next directory in an open TIFFSetDirectory, TIFFSetSubDirectory (3t) - set the current directory for an open TIFFWriteDirectory (3t) - write the current directory in an open Tcl_TranslateFileName (3) - convert file name to native form and replace tilde with home directory
basename (1) - strip directory and suffix from filenames
cd (n) - Change working directory
chdir, fchdir (2) chroot (1) chroot (2) closedir (3) cryptdir (1)
- change working directory - run command or interactive shell with special root directory - change root directory
- close a directory
- encrypt/decrypt all files in a directory - list directory contents
dir (1) directory dirname (1) find (1) getcwd, get_current_dir_name, getwd (3) - Get current working directory
- strip non-directory suffix from file name - search for files in a directory hierarchy getdents (2) - get directory entries getdirentries (3) - get directory entries in a filesystem independent format
ls (1) mcd (1) mdir (1) mdu (1) t mkdir (2) mklost+found (8) - create a lost+found directory on a mounted Linux second extended file system
in a directory DirHandle (3) FindBin (3) dirname (3) finddepth (3) getcwd (3)

[igor@anantchenko/usr]$
- list directory contents - change MSDOS directory '" t
- display an MSDOS directory '" t - display the amount of space occupied by an MSDOS directory '"
- create a directory
mknod (2) mmd (1) mmove (1) mrd (1) opendir (3) pwd (1) pwd (n) readdir (2) readdir (3) rewinddir (3) rmdir (2) scandir, alphasort (3) - scan a directory for matching entries seekdir (3) - set the position of the next readdir() call in the directory stream. telldir (3) - return current location in directory stream.
tixDirSelectDialog (n) - Create and manipulate directory selection dialogs.
- create a directory or special or ordinary file - make an MSDOS subdirectory '" t
- move or rename an MSDOS file or subdirectory '" t - remove an MSDOS subdirectory '" t
- open a directory - print name of current/working directory - Return the current working directory
- read directory entry - read a directory
- reset directory stream - delete a directory

vdir (1) lndir (1x) tree mkdirhier (1x) mkfontdir, fonts.dir, fonts.scale, fonts.alias, encodings.dir (1x) - create an index of X font files
- list directory contents - create a shadow directory of symbolic links to another directory
- makes a directory hierarchy
- supply object methods for directory handles - Locate directory of original perl script - extract just the directory from a path - traverse a directory structure depth-first
- get pathname of current working directory
Из приведенного выше примера видно, что можно успешно выполнять поиск практически любой информации. Если установлен язык интерфейса пользователя русский – то многие разделы документации можно читать по-русски.

3. ls – Вывод содержимого каталога

Команда ls во многом схожа с командой DOS DIR, в последних версиях Linux добавлено для этой команды второе имя – dir, однако, нерекомендуется пользоваться этим именем. Причина в том, что в старых версиях Linux такое имя отсутствует, как и отсутствует в любых других клонах операционной системы UNIX.
Команда имеет большое количество различных ключей, относящихся к формату вывода каталога. Подробное описание их можно получить, набрав в командной строке man ls. Рассмотрим только самые употребительные варианты.
Вывод полного списка всех файлов с именами, расширениями, правами доступа. Требуется выводить также скрытые файлы (имя которых начинается с точки)

[igor@anantchenko/root]$ ls -al Total 14804
drwxrwxrwx drwxr-xr-x -rw------- -rw------- -rw-r--r-- -rw------- -rw-r--r-- -rw-r--r-- -rw-r--r-- drwx------ -rw-r--r-- drwx------ drwxr-xr-x drwxr-xr-x drwxr-xr-x drwx------ -rw------- drwxr-xr-x drwxr-xr-x -rw-r--r--
11 root 20 root 1 root 1 root 1 mia 1 root 1 mia 1 mia 1 mia 2 root 1 root 3 root 7 root 2 root 2 root 2 root 1 root 3 root 5 root 1 mia
root 4096 root 4096 root 883 root 113 root 1422 root 4013 root 24 root 323 root 264 root 4096 root 182 root 4096 root 4096 root 4096 root 4096 root 4096 root 0 root 4096 root 4096 root 3394
Mar 31 Mar 28 Mar31 Mar28 Mar28 Mar29 Mar 28 Mar 28 Mar 28 Mar21 Mar 22 Mar31 Mar31 Mar24 Mar23 Mar23 Mar 23 Mar24 Mar31 Mar28
21:28 . 20:58 .. 21:28 .ICEauthority 20:53 .Xauthority 20:56 .Xdefaults 13:46 .bash_history 20:56 .bash_logout 20:56 .bash_profile 20:56 .bashrc 17:20 .cedit 1999 .cshrc 21:29 .enlightenment 21:29 .gnome 19:47 .gnome-desktop

23:50 .gnome-help-browser 23:29 .gnome_private 23:15 .lynx_cookies 19:52 .mc 21:29 .netscape 20:56 .screenrc
10
-rw-r--r-- 1 root root -rw------- 1 root root -rw------- 1 root root drwx------ 2 root root
166 Mar 4 1996 .tcshrc 569 Apr 2 19:50 .xsession-errors
[igor@anantchenko/root]$ Вывод краткого списка файлов
[igor@anantchenko/root]$ ls core nsmail То же самое, только вывести имена скрытых файлов [igor@anantchenko/root]$ ls -a . .Xdefaults .bashrc .tcshrc .. .bash_history .cedit
.xsession-errors .ICEauthority .bash_logout .cshrc
.gnome .lynx_cookies .gnome-desktop .mc .gnome-help-browser
15048704 Mar 24 4096 Mar23 23:50 nsmail
.netscape core .Xauthority .bash_profile .enlightenment .gnome_private .screenrc nsmail [igor@anantchenko/root]$
cd – смена текущего каталога.
Пример
[igor@anantchenko/root]$ pwd /root [igor@anantchenko/root]$ cd /usr/bin
[igor@anantchenkobin]$ pwd /usr/bin
[igor@anantchenkobin]$ cd .. [igor@anantchenko/usr]$ pwd
/usr [igor@anantchenko/usr]$ cd . [igor@anantchenko/usr]$ pwd /usr
[igor@anantchenko/usr]$ cd ../root [igor@anantchenko/root]$ pwd
/root [igor@anantchenko/root]$
cd .. – перейти в каталог более низкого уровня.
21:41 Core

---

4. cd . – перейти в текущий каталог (не надо смеяться иногда это бывает нужно)

---

5. more – постраничный просмотр текстового файла

Предоставляет возможность просмотра текстового файла
После вызова выводит на экран первую страницу текстового файла и ждет ввода.
Возможны следующие управляющие выводом команды Пробел – пролистать на страницу вперед h – или ? – посмотреть подсказку по клавишам
z – аналогично пробелу, однако перед вводом можно набрать число строк (аргумент) , которое будет прокручиваться. ВНИМАНИЕ ввод аргумента установит новые умолчания и впоследствии при нажатии пробела будет выводится число строк, указанное в параметре
Enter – выводит одну строчку. Возможен аргумент, тогда прокрутится на большее количество строк.
q – завершить просмотр

s – пропустить количество строк, заданное аргументом
f – пропустить количество экранов, заданное аргументом
b – аналогично f, но крутит назад.
= - показать номер текущей строки
/строка – Найти строку
Для примера можно использовать команды
more /etc/services
more /etc/sendmail.cf
Страницы документации, выдаваемые командой man выводят текст при помощи more

6. grep - найти подстроку

Многоплановая команда. Ознакомиться с документацией (справкой) по ней, выполнить несколько примеров.
Примеры

Поиск подстроки root в файле /etc/passwd
[root@prokosh /etc]# grep root /etc/passwd
root:x:0:0:root:/root:/bin/bash
operator:x:11:0:operator:/root:
mia:x:30:0:Igor Anantchenko:/root:/bin/bash

Поиск подстроки http во всех файлах директории /etc
[root@prokosh /etc]# grep http *
grep: CORBA: Is a directory
grep: X11: Is a directory
grep: charsets: Is a directory
grep: codepages: Is a directory
grep: cron.d: Is a directory
grep: cron.daily: Is a directory
grep: cron.hourly: Is a directory
grep: cron.monthly: Is a directory
grep: cron.weekly: Is a directory grep: default: Is a directory
grep: gtk: Is a directory inetd.conf:linuxconf stream tcp wait root /bin/linuxconf linuxconf --http Binary file ld.so.cache matches grep: logrotate.d: Is a directory lynx.cfg:# is available at http://www.hippo.ru/~hvv/lynxcfg_toc.html lynx.cfg:# STARTFILE can be remote, e.g. http://www.w3.org/default.html , lynx.cfg:# http://www.crl.com/~subir/lynx/lynx_help/lynx_help_main.html lynx.cfg:#HELPFILE:http://www.crl.com/~subir/lynx/lynx_help/lynx_help_main.html
lynx.cfg:DEFAULT_INDEX_FILE:http://www.ncsa.uiuc.edu/SDG/Software/Mos

aic/MetaIndex.html
lynx.cfg:# lynx.cfg:# lynx.cfg:# lynx.cfg:# lynx.cfg:# lynx.cfg:# lynx.cfg:#
lynx.cfg:#
Find RFC 1345 at http://www.ics.uci.edu/pub/ietf/uri/rfc1345.txt . requests to http servers using an Accept-Charsets header. Users can
(http://www.ics.uci.edu/pub/ietf/uri/rfc2068.txt). become
http://www.wfbr.edu:8002/dir/lynx). The prefixes will not be http://www.nyu.edu without testing www.nyu.com). Lynx will try to and use "http://" as the default (e.g., gopher.wfbr.edu or gopher.wfbr. specify a trusted directory for http URLs, which will then be treated as
TRUSTED_EXEC:http://www.wfbr.edu/trusted/
lynx.cfg:#
ALWAYS_TRUSTED_EXEC:http://www.more.net/<tab>/usr/local/kinetic/bin/wh
o.sh lynx.cfg:# lynx.cfg:# lynx.cfg:# lynx.cfg:# lynx.cfg:# lynx.cfg:# (http://www.ics.uci.edu/pub/ietf/uri/rfc1738.txt) lynx.cfg:# in http server
replies. lynx.cfg:#http_proxy:http://some.server.dom:port/
lynx.cfg:#https_proxy:http://some.server.dom:port/
lynx.cfg:#ftp_proxy:http://some.server.dom:port/
lynx.cfg:#gopher_proxy:http://some.server.dom:port/
lynx.cfg:#news_proxy:http://some.server.dom:port/
lynx.cfg:#newspost_proxy:http://some.server.dom:port/
lynx.cfg:#newsreply_proxy:http://some.server.dom:port/
lynx.cfg:#snews_proxy:http://some.server.dom:port/
lynx.cfg:#snewspost_proxy:http://some.server.dom:port/
lynx.cfg:#snewsreply_proxy:http://some.server.dom:port/
lynx.cfg:#nntp_proxy:http://some.server.dom:port/
lynx.cfg:#wais_proxy:http://some.server.dom:port/
lynx.cfg:#finger_proxy:http://some.server.dom:port/
lynx.cfg:#cso_proxy:http://some.server.dom:port/
ALWAYS_TRUSTED_EXEC:http://www.more.net/<tab>show users
TRUSTED_LYNXCGI:<tab>/usr/local/etc/httpd/cgi-bin/
LYNXCGI_DOCUMENT_ROOT:/usr/local/etc/httpd/htdocs
received from https servers never will be sent unencrypted to http https server included a secure attribute for the cookie. The normal Lynx respects RFC 1738
lynx.cfg:# via lynx.cfg:# lynx.cfg:# lynx.cfg:# lynx.cfg:# lynx.cfg:# lynx.cfg:# lynx.cfg:# lynx.cfg:# grep: mail: Is a directory grep: midi: Is a directory mime.types:message/http grep: nmh: Is a directory grep: pam.d: Is a directory grep: pcmcia: Is a directory grep: ppp: Is a directory
to http or https servers if the personal_mail_address has been defined
documents with a http or https base URL, if this would otherwise lead to http and https, since they are less common and may actually be valid in
the http protocol. This is because HTTP servers already specify

http://www.openvms.digital.com/cd/XV310A/ subdirectories. You file, or in src/HTInit.c. See http://www.internic.net/rfc/rfc1524.txt
I use wget for http and ftp transfers via the external command. <url> Any given URL. This can be normal ones like ftp or http or it downloading http and ftp files in the background. In Win95 I use
RULE:Map
http://old.server/*
http://new.server/*
grep: profile.d: Is a directory grep: rc.d: Is a directory grep: security: Is a directory services:www services:https services:https grep: skel: Is a directory grep: smrsh: Is a directory grep: sound: Is a directory grep: sysconfig: Is a directory termcap:# be found at <http://www.ccil.org/~esr/ncurses.html>. grep: vga: Is a directory [root@prokosh /etc]#

Содержимое отчета по лабораторной работе

Отчет выполняется с использованием текстового редактора Word. Формат файла rtf, doc или docx

1. Титульный лист
2. Описание хода выполнения лабораторной работы (Название лабораторной работы, цель работы, последовательность /этапы/ выполнения работы, итоги по проделанной работе /выводы/. Описание выполненных действий сопровождается копиями экрана /скриншотами/ - не менее 10 в отчете.)
3. Письменные ответы на вопросы по теме лабораторной работы (вопросы индивидуальные, выдаются преподавателем).

Лабораторная работа № 2. Конфигурирование операционной среды GNU/Linux

Цель работы:
– изучить способы конфигурирования операционной среды после инсталляции и при изменении набора периферийного оборудования;
– изучить методику устранения неполадок этапа эксплуатации операционной среды;
– освоить основные способы запуска и управления работой прикладных программ;
– научиться подключать и настраивать периферийное оборудование на этапе эксплуатации операционной среды
В процессе выполнения работы студент, руководствуясь методическими указаниями к выполнению данной работы:
– определяет параметры конфигурации операционной среды;
– проводит установку необходимых драйверов при подключении периферийных устройств;
– управляет запуском и выполнением различных прикладных программ в операционной среде;
– предоставляет образ сконфигурированной операционной среды для проверки и отвечает на вопросы преподавателя для получения зачета за выполненную работу.

1. Теоретическая часть

Дистрибутивы *Linux*, в том числе *openSUSE*, состоят из комплекса программ, представленных в формате *пакетов*, и необходимого инструментария по управлению этим комплексом.

Система управления пакетами — это набор инструментов, обеспечивающий унифицированный метод установки, обновления и удаления программного обеспечения в вашей системе. Дистрибутивы *Linux*, включая *openSUSE*, обычно состоят из тысяч отдельных пакетов программ.

Программное обеспечение распространяется через пакеты с прикрепленными метаданными, в которых содержится дополнительная информация: описание назначения пакета, список зависимостей, необходимых для нормальной работы программы, и т.д. Пакеты хранятся в репозиториях, локальных (CD, DVD или жёсткий диск) или сетевых хранилищах. После установки пакета его метаданные сохраняются в локальной базе данных и используются для поиска файлов пакета. Схема взаимодействия показана на рисунке 1.

Libzypp — это реализация подобной системы управления пакетами для *openSUSE*, к ней предоставляется графический интерфейс пользователя *YaST Software* Management (рисунок 2) и интерфейс командной строки *Zypper* (консольный менеджер пакетов, рисунок 3).

Рис. 1. Схема взаимодействия пользователя при установке пакетов

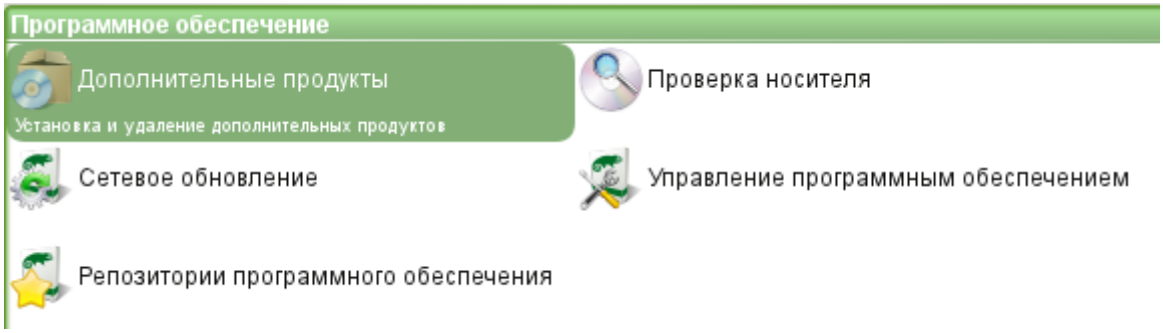


Рис. 2. *YaST*: программное обеспечение

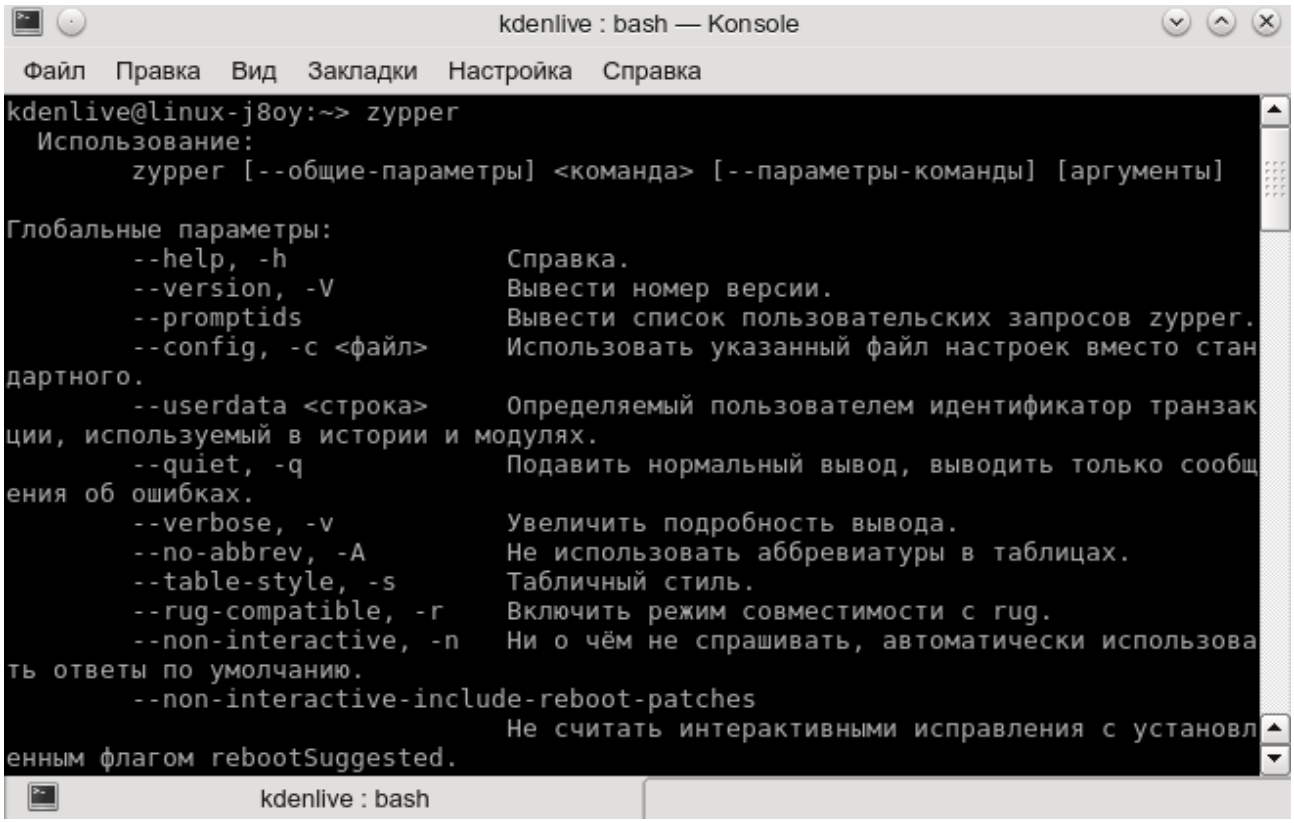


Рис. 3. Консольный менеджер пакетов *Zypper*

1.1 Пакеты

Пакеты — это архивы файлов, содержащие все компоненты приложений (сами приложения, разделяемые библиотеки, пакеты для разработки приложений и т.д.) и инструкции по их запуску и настройке.

Пакет интегрирован в дистрибутив, для которого он был собран, с учётом путей установки, зависимостей, интеграции со средой, скриптов запуска для серверов и т.п. Поэтому всегда следует устанавливать пакеты, собранные именно для определенного дистрибутива и его версии (например, *openSUSE* 13.1). Не стоит применять пакеты других дистрибутивов *Linux* в *openSUSE* и даже пакеты *openSUSE* 13.1 в *openSUSE* 13.2 (хотя работоспособность последних иногда возможна).

1.2 Метаданные пакетов

Пакет также содержит дополнительные сведения, обычно называемые метаданными. В них входят: аннотация, описание, список содержимого пакета, номера версии программы и релиза пакета, когда, где и кем собран пакет, для какой архитектуры предназначена сборка, контрольные суммы файлов, лицензии на программы, сведения о зависимостях и прочее.

1.3 Зависимости пакетов

Важная функция пакетов - описание взаимосвязей приложений. Поскольку приложения требуют для своего выполнения определённого рабочего окружения (других программ, библиотек и т.п.), пакеты могут предоставлять файлы, предназначенные для использования в других пакетах. Зависимости пакетов используются для выражения таких связей.

Использование системы пакетов — важное свойство дистрибутивов *Linux*, обеспечивающее модульный подход к управлению операционной системой и приложениями.

Такой подход очень эффективен для поддержания стабильности и защищённости системы: если исправлена уязвимость в библиотеке, используемой другими приложениями, её обновление закроет данную уязвимость для всех пакетов.

1.4 Форматы пакетов

Программное обеспечение для *Linux* чаще всего распространяется в одном из следующих форматов:

- *tgz* (файлы *tar*, *gzip*). Это просто архивы. Они могут содержать всё, что разработчик считает нужным. Кроме самого формата архива, никаких стандартов на структуру содержимого не существует.
- *deb* (*Debian*). Формат пакетов, принятый в *Debian* и его производных дистрибутивах.

•*rpm* (Менеджер пакетов *RPM*). Созданный *Red Hat* и принятый *LSB (Linux Standard Base* – стандартизация внутренней структуры) в качестве стандарта, *rpm* используется *openSUSE* и многими другими дистрибутивами.

Сам по себе формат пакетов не предоставляет управления зависимостями, а лишь сообщает об их наличии, предоставляя пользователю самому разбираться с установкой необходимых компонентов, если они отсутствуют.

Предположим, что пользователь хочет установить пакет А, который зависит от пакета Б. *RPM* не установит пакет Б автоматически, но сообщит, что он требуется для установки А, и прекратит работу. Пользователь должен сам сперва установить Б, а затем сможет установиться А. Несложно? Теперь представим, что пакет Б зависит от пакетов В и Г, а Г зависит от Д, а Д от ... и так далее. В итоге потребуется вручную отследить все ветви, возможно, немаленького дерева зависимостей. Решение этой проблемы - менеджер пакетов.

1.5 Менеджер пакетов

В современных дистрибутивах, таких как *openSUSE*, установку программ лучше всего делать с помощью менеджера пакетов. Функционируя поверх *RPM*, он получает пакеты из репозиториев (интернет-сервера, *CD, DVD* и т.п.), находит зависимости и устанавливает их в вашу систему.

Менеджер пакетов также упрощает удаление и обновление пакетов. Объём доступного программного обеспечения зависит от репозиториев, которые подключены.

Родным менеджером пакетов *openSUSE* является модуль *YaST Software Management* и программа командной строки *Zypper*, но в дистрибутиве содержатся и другие инструменты управления пакетами. Некоторые из них работают только с *RPM*, другие покрывают более широкий спектр возможностей.

1.6 Репозитории пакетов

Перед установкой пакетов репозитории должны быть доступны в системе — или с дисков, или через интернет.

Репозитории *openSUSE* бывают:

•Официальные – в них входят хорошо протестированные и поддерживаемые пакеты.

•Сторонние – в них содержатся различные дополнительные пакеты, некоторые имеют более новые версии, чем в дистрибутиве, некоторые не могут быть включены в *openSUSE* по лицензионным причинам. Пакеты в таких репозиториях, скорее всего, меньше тестировались, чем официальные.

1.7 Установка программ из пакета RPM

С помощью программы *rpm* можно легко устанавливать, модифицировать, удалять и создавать пакеты программного обеспечения, а также получать о них разнообразную информацию. Все эти дистрибутивы (кроме программы начальной установки) состоят из таких пакетов. Каждый пакет определяется именем программы, номером ее версии и номером версии релиза этой программы дистрибутива, а также архитектурой пакета. Например, *skype-4.3.0.37-suse.i586.rpm*: в этом пакете

- имя - skype,
- номер версии – 4.3.0.37,
- номер релиза - *suse*,
- архитектура - i586.

Чем больше номер версии (или при одинаковых номерах версии - чем больше номер релиза), тем, соответственно, новее пакет. Управлять пакетами можно из командной строки при помощи программы *rpm*, которая имеет следующий синтаксис:

rpm -options rpm_package_name

Установить программу можно, используя опцию -i (опции -v и -h выставлены здесь для того, чтобы включить визуальное отображение процесса установки). Например, для того, чтобы установить skype, наберите:

rpm -ivh skype-4.3.0.37-suse.i586.rpm

Для того чтобы обновить программу (с целью установки более свежей версии), нужно использовать опцию -U, вместо -i, это позволит сохранить все текущие конфигурационные файлы. Если пакета ранее не было в системе, то он будет установлен.

Если необходимо удалить пакет из системы, введите:

rpm -e имя_пакета_без_номера_версии_и_релиза

то есть, например, для пакета *skype*:

rpm -e skype

Если в процессе удаления пакета произойдет нарушение зависимостей, программа *rpm* сообщит об этом.

Также установить программу можно используя графический интерфейс. Для этого необходимо открыть диспетчер файлов и выбрать необходимый *RPM* пакет, затем щелкнуть ЛКМ по пакету и следовать инструкциям на экране (рисунок 4).

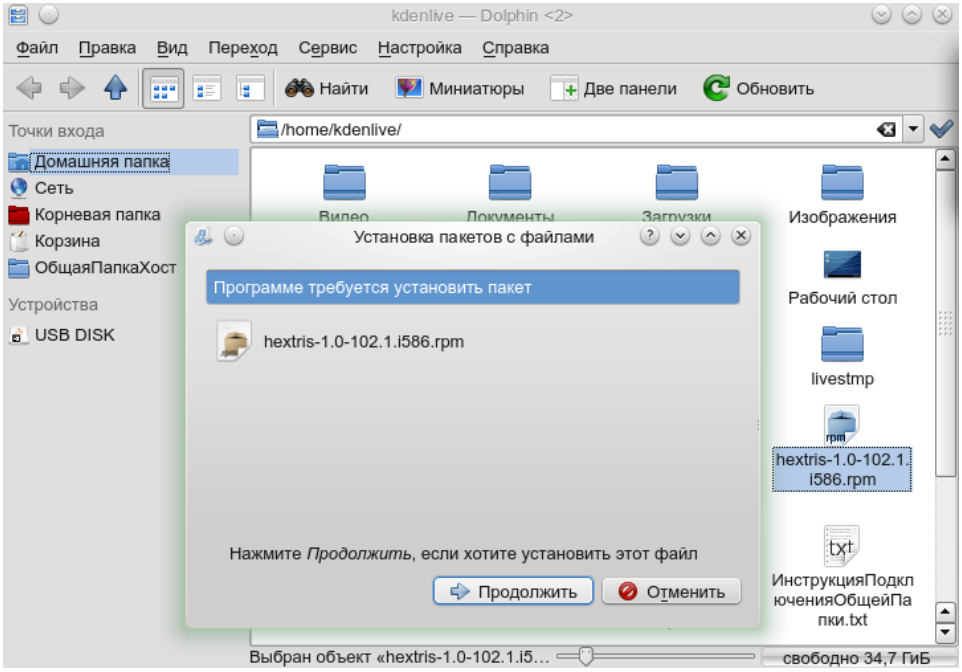


Рис. 4. Установка пакетов с файлами

1.8 Установка программ с использованием *Zypper*

Zypper - средство для управления пакетами в текстовом режиме.

Для того, чтобы найти нужный пакет в подключенных репозиториях используется следующая команда:

zypper se <имя_пакета> — выполнить поиск (*se - search*) пакета

Если нужно найти какую-то программу, но не известно в каком пакете программа находится, то нужно выполнить команду:

zypper wp <имя_программы> - выполнить поиск программы в пакете по названию (опция *wp* – это *what-provides*)

Просмотреть информацию о пакете можно командой:

zypper info <имя пакета>

Установка пакетов из подключенных репозиториев выполняется командой:

zypper in <имя пакета> - данная команда (*in - install*) установит пакет со всеми его зависимостями.

Для удаления пакетов используется опция rm/remove:

zypper rm <имя пакета> - данная команда удалит пакет из системы.

Управление репозиториями возможно черех zypper, для это необходимо набрать команду:

zypper ar URI alias ,

где - URI идентификатор репозитория. alias - это любое понятное вам имя репозитория, позволяющее идентифицировать его и отличить от других.

Удалить репозиторий можно командой:
zypper rr alias

2. Задание на лабораторную работу

Произвести установку и удаление программ с использованием менеджера пакетов.

Произвести настройку драйвера символьного устройства.

3. Методика выполнения задания

1. Запустить операционную систему *openSUSE*
2. Создать папку «*Install*» в домашнем каталоге.
3. Скопировать в данный каталог файлы:

- *xorg-x11-libs-6.8.2-1.EL.66.x86_64.rpm*
- *frozen-bubble-2.212-56.1.i586.rpm*
- *hextris-1.0-102.1.i586.rpm*

4. Произвести установку пакетов разными способами и проанализировать возможные зависимости:

- *xorg-x11-libs-6.8.2-1.EL.66.x86_64.rpm*
- *frozen-bubble-2.212-56.1.i586.rpm*
- *hextris-1.0-102.1.i586.rpm*

5. Получить информацию о пакетах с использованием *zypper*.
6. Добавить репозиторий с помощью YaST:
http://packman.inode.at/suse/openSUSE_13.1/Essentials/
7. Добавить репозиторий с помощью *Zypper*:
http://packman.inode.at/suse/openSUSE_13.1/Multimedia/
8. Удалить пакет с помощью *YaST*:

- *xorg-x11-libs-6.8.2-1.EL.66.x86_64.rpm*

9. Удалить пакет с помощью *zypper*:

- *hextris-1.0-102.1.i586.rpm*

10. Удалить пакет с помощью *RPM*:

- *frozen-bubble-2.212-56.1.i586.rpm*

11. Выполнить поиск программы – текстовый редактор *vi*. Установить/обновить данную программу.

12. Удалить репозитории:

http://packman.inode.at/suse/openSUSE_13.1/Essentials/

http://packman.inode.at/suse/openSUSE_13.1/Multimedia/

13. Удалить каталог «*Install*» и все сопутствующие файлы.

14. Убедиться в удалении каталога «*Install*».

15. Изучите исходный код модуля cdev/mephi_cdev.c.

16. Скомпилируйте и загрузите модуль mephi_cdev.ko.

```
$ make
$ sudo insmod mephi_cdev.ko
```

17. Выведите сообщения из буфера ядра:

```
$ dmesg
```

18. Убедитесь в том, что модуль загружен и драйвер зарегистрирован.

19. Определите старший номер, который был выделен драйверу.

```
$ lsmod | head -n2
$grep mephi /proc/devices
```

20. Создайте специальные файлы символьного устройства со старшим номером, который был выделен драйверу, и несколькими младшими номерами.

```
$ sudo mknod -m0666 ./mephi_dev c 247 0
$ sudo mknod -m0666 ./mephi_dev1 c 247 1
$ sudo mknod -m0666 ./mephi_dev2 c 247 2
$ sudo mknod -m0666 ./mephi_dev3 c 247 3
$ ll
```

21. Прочтите данные из созданных специальных файлов символьного устройства.

```
$ cat ./mephi_dev
$ cat ./mephi_dev1
$ cat ./mephi_dev2
$ cat ./mephi_dev3
```

22. Объясните полученные результаты.

23. Удалите модуль.

```
$ sudo rmmod mephi_cdev
```

24. Повторно прочтите данные из специального файла символьного устройства.

```
$ cat ./mephi_dev
```

25. Объясните результат.

26. Выведите сообщения из буфера ядра.

```
$ dmesg | tail
```

27. Повторите загрузку драйвера, но старший номер передайте через параметр major.

```
$ sudo insmod mephi_cdev.ko major=250
```

insmod: ERROR: could not insert module mephi_cdev.ko: Device or resource busy
$ ll /dev | grep 250
crw-------. 1 root root 250, 0 Apr 23 00:21 hidraw0
28. Объясните результат.
29. Подберите свободный старший номер. Повторите загрузку драйвера с передачей номера через параметр major.
30. Изучите сообщения dmesg.
31. Повторно прочтите данные из специального файла символьного устройства.
$ cat ./mephi_dev
32. Объясните результат.
33. Разработка функции записи в драйвер
34. Используя исходный код примера proc/mephi_proc.c, разработайте и реализуйте функцию записи в драйвер.
35. Создание файла устройства из модуля
35. Изучите исходный код модуля https://github.com/efanov/BOOK_KERN_223/blob/master/dev/cdev/dyndev.c.
36. Скомпилируйте и загрузите модуль dyndev.ko.
37. Выведите сообщения из буфера ядра.
38. Убедитесь в том, что модуль загружен и драйвер зарегистрирован. Определите старший номер, который был выделен драйверу.
39.Убедитесь в том, что в каталоге /dev были созданы специальные файлы символьного устройства со старшим номером, который был выделен драйверу, и несколькими младшими номерами.
40. Убедитесь в том, что в /sys/class был создан заданный класс dyn_class и файлы.
41. Изучите содержимое /sys/module/my_dyndev_mod/parameters/major.
42. Прочтите данные из созданных специальных файлов символьного устройства.
43. Удалите модуль.
44. Убедитесь в том, что в каталоге /dev специальные файлы символьного устройства были удалены.
44. Выведите сообщения из буфера ядра.
45.Завершить работу операционной системы.

6. Требования к содержанию и оформлению отчета

Отчет по лабораторной работе должен содержать:
а) титульный лист;
б) описание хода выполнения работы и снимки экрана;
в) заключение по выполненной работе;
г) ответы на контрольные вопросы.

Контрольные вопросы

1. Что такое пакет?
2. Что включает в себя метаданные пакетов?
3. Что такое репозиторий?
4. Какие форматы пакетов существуют?
5. Какие преимущества дает использование системы пакетов?
6. Что подразумевается под зависимостью пакетов?
7. Для чего используется Zypper?

Лабораторная работа № 3. Проверка технического состояния операционной среды и прикладного программного обеспечения.

Цель работы:
– изучить способы проверки технического состояния операционной среды и прикладного программного обеспечения;
– изучить методики оценки вычислительного ресурса операционной среды;

В процессе выполнения работы студент, руководствуясь методическими указаниями к выполнению данной работы:
– определяет параметры конфигурации средств проверки технического состояния операционной среды и прикладного программного обеспечения;
– проводит выполнение необходимых тестов и анализ отчетов;
– предоставляет автоматические отчеты о работе средств проверки технического состояния и отвечает на вопросы преподавателя для получения зачета за выполненную работу.

Теоретическая часть

Поскольку администрирование операционной системы в конечном счете сводится к управлению процессами, каждый раз, когда программа запускается на выполнение, начинается то, что в литературе именуется как *процесс.* Или другими словами – процессом называется выполняемая в данный момент программа или ее потомки. Каждый процесс запускается от имени какого-то пользователя. Процессы, которые стартовали при загрузке, обычно выполняются от имени пользователей root или nobody.

Каждый пользователь может управлять поведением процессов, им запущенных. При этом пользователь root может управлять всеми процессами — как запущенными от его имени, так и процессами, порожденными другими пользователями операционной системы. Управление процессами осуществляется с помощью утилит, а также посредством некоторых команд командной оболочки (shell).

Каждый процесс в системе имеет уникальный номер — идентификационный номер процесса (Process Identification, PID). Этот номер используется ядром операционной системы, а также некоторыми утилитами для управления процессами.

Выполнение процесса на переднем плане и в фоновом режиме

Процессы могут выполняться на переднем плане (foreground) – режим по умолчанию и в фоновом режиме (background).

На переднем плане в каждый момент для текущего терминала может выполняться только один процесс. Однако пользователь может перейти в

другой виртуальный терминал и запустить на выполнение еще один процесс, а на другом терминале еще один и т.д. Процесс переднего плана – это процесс, с которым взаимодействует пользователь, процесс получает информацию с клавиатуры (стандартный ввод) и посылает результаты на экран (стандартный вывод).

Фоновый процесс после своего запуска благодаря использованию специальной команды командной оболочки отключается от клавиатуры и экрана, т.е. не ожидает ввода данных со стандартного ввода и не выводит информацию на стандартный вывод, а командная оболочка не ожидает окончания запущенного процесса, что позволяет пользователю немедленно запустить еще один процесс.

Обычно фоновые процессы требуют очень большого времени для своего завершения и не требуют вмешательства пользователя во время существования процесса. К примеру, компиляция программ или архивирование большого объема информации — кандидаты номер один для перевода процесса в фоновый режим.

Процессы так же могут быть отложенными. Отложенный процесс — это процесс, который в данный момент не выполняется и временно остановлен. После того как пользователь остановил процесс, в дальнейшем может его продолжить как на переднем плане, так и в фоновом режиме. Возобновление приостановленного процесса не изменит его состояния — при возобновлении он начнется с того места, на котором был приостановлен.

Для выполнения программы в режиме переднего плана достаточно просто набрать имя программы в командной строке и запустить ее на выполнение. После этого пользователь может работать с программой.

Для запуска программы в качестве фонового процесса достаточно набрать в командной строке имя программы и в конце добавить знак амперсанта (&), отделенный пробелом от имени программы и ее параметров командной строки, если таковые имеются. Затем программа запускается на выполнение. В отличие от запуска программы в режиме переднего плана получается приблизительно следующее сообщение:

```
/home/student# yes > /dev/null &;
[1] 123
/home/student#
```

Оно состоит из двух чисел и приглашения командной строки. Таким образом, мы запустили программу выполняться в фоновом режиме и получили возможность запустить с той же самой консоли на выполнение еще какую-то программу.

Число [1] означает номер запущенного нами фонового процесса. С его помощью можно будет производить манипуляции с фоновым процессом. Значение 123 показывает идентификационный номер (PID) процесса. Отличия этих двух чисел достаточно существенные. Номер фонового процесса уникален только для пользователя, запускающего данный фоновый процесс. То есть если в системе три пользователя решили запустить фоновый процесс

(первый для текущего сеанса) – в результате у каждого пользователя появится фоновый процесс с номером [1]. Напротив, идентификационный номер процесса (PID) уникален для всей операционной системы и однозначно идентифицирует в ней каждый процесс.

Номер фонового процесса хранится в переменных командной оболочки пользователя и позволяет не забивать голову цифрами типа 2693 или 1294, а использовать переменные вида %1, %2. Однако допускается пользоваться и идентификационным номером процесса.

Для проверки состояния фоновых процессов можно воспользоваться командой командной оболочки – jobs.

```
/home/student# jobs
[1]+  Running yes >/dev/null    &
/home/student#
```

Из вышеприведенного примера видно, что у пользователя в данный момент запущен один фоновый процесс, и он выполняется.

Остановка и возобновление процесса

Помимо прямого указания выполнять программу в фоновом режиме, существует еще один способ перевести процесс в фоновый режим. Для этого необходимо выполнить следующие действия:

- запустить процесс выполняться на переднем плане;
- остановить выполнение процесса;
- продолжить процесс в фоновом режиме.

Для выполнения программы введем ее имя в командной строке и запустим на выполнение. Для остановки выполнения программы необходимо нажать на клавиатуре следующую комбинацию клавиш — <Ctrl>+<Z>. После этого можно увидеть на экране следующее:

```
/home/student#  yes > /dev/null
ctrl+Z
[1]+  Stopped    yes  >/dev/null
/home/student#
```

Получено приглашение командной строки. Для того чтобы перевести выполнение процесса в фоновый режим, необходимо выполнить следующую команду:

```
bg %1
```

Причем необязательно делать это сразу после остановки процесса, главное правильно указать номер остановленного процесса.

Для того чтобы вернуть процесс из в фонового режима выполнения на передний план, достаточно выполнить следующую команду:

```
fg %1
```

В том случае, если необходимо перевести программу в фоновый или, наоборот, на передний план выполнения сразу после остановки процесса,

можно выполнить соответствующую программу без указания номера остановленного процесса.

Существует большая разница между фоновым и остановленным процессом. Остановленный процесс не выполняется и не потребляет ресурсы процесса, однако занимает оперативную память или пространство свопинга. В фоновом же режиме процесс продолжает выполняться.

Для приостановления фонового процесса необходимо переместить процесс на передний план, а затем остановить.

Завершение работы процесса

Существует несколько вариантов завершения работы процесса.

Вариант первый. Если процесс интерактивный, как правило, в документации или прямо на экране написано, как корректно завершить программу.

Вариант второй. В том случае, если не указано как можно завершить текущий
процесс (не фоновый), то можно воспользоваться клавиатурной комбинацией <Ctrl>+<C>. Возможно использовать также комбинацию клавиш <Ctrl>+<Break>. А для остановки фонового процесса можно перевести его на передний план, а затем уже воспользоваться вышеприведенными клавиатурными комбинациями.

Вариант третий и самый действенный. В том случае, если не удалось прекратить выполнение процесса вышеприведенными способами – например, программа зависла или "слетел" терминал — для завершения процесса можно воспользоваться следующими командами: kill, killall.

Команда kill может получать в качестве аргумента как номер процесса, так и идентификационный номер (PID) процесса. Таким образом, команда:

/home/student# kill 123 эквивалентна команде:

/home/student# kill %1

Можно видеть, что не надо использовать "%", когда вы обращаетесь к работе по идентификационному номеру (PID) процесса.

С помощью команды killall можно прекратить выполнение нескольких процессов сразу, имеющих одно и то же имя. Например, команда killall mc прекратит работу всех программ mc, запущенных от имени данного пользователя.

Для того чтобы завершить работу процесса, пользователю надо быть его владельцем. Пользователь root может завершить работу любого процесса в операционной системе.

Программы, используемые для управления процессами

Существует достаточно большое количество утилит, используемых для управления тем или иным способом процессами, исполняемыми в

операционной системе. Рассмотрим только основные утилиты. В табл. 1 приведен список основных программ, тем или иным образом предназначенных для управления процессами.

Таблица1. Программы управления процессами

Программа	Описание
at	Выполняет команды в определенное время
batch	Выполняет команды тогда, когда это позволяет загрузка системы
cron	Выполняет команды по заранее заданному расписанию
crontab	Позволяет работать с файлами crontab отдельных пользователей
kill	Прекращает выполнение процесса
nice	Изменяет приоритет процесса перед его запуском
nohup	Позволяет работать процессу после выхода пользователя из системы
ps	Выводит информацию о процессах
renice	Изменяет приоритет работающего процесса
w	Показывает, кто в настоящий момент работает в системе и с какими про граммами

nohup

Эта утилита позволяет организовать фоновый процесс, продолжающий свою работу даже тогда, когда пользователь отключился от терминала, в отличие от команды &, которая этого не позволяет. Для организации фонового процесса необходимо выполнить следующую команду:

nohup выполняемая_фоновая_команда &

Во вновь запущенном терминале процесс нельзя увидеть с помощью команды jobs, так как команда jobs выводит список процессов текущего терминала, поэтому после подключения к терминалу необходимо использовать команду ps с параметром –A.

ps

Программа ps предназначена для получения информации о существующих в операционной системе процессах. У этой команды есть множество различных опций, но остановимся на самых часто используемых. Для получения подробной информации смотрите man-страницу этой программы.

Простой запуск ps без параметров выдаст список программ, выполняемых на терминале. Обычно этот список очень мал:

```
PID     TTY   TIME      CMD
885     ttyl  00:00:00  login
893     ttyl  00:00:00  bash
```

955 ttyl 00:00:00 ps

Первый столбец – pid (идентификационный номер процесса). Как уже упоминалось, каждый выполняющийся процесс в системе получает уникальный идентификатор, с помощью которого производится управление процессом. Каждому вновь запускаемому на выполнение процессу присваивается следующий свободный PID. Когда процесс завершается, его номер освобождается. Когда достигнут максимальный PID, следующий свободный номер будет взят из наименьшего освобожденного.

Следующий столбец – tty – показывает, на каком терминале процесс выполняется. Запуск команды без параметров ps покажет процессы, выполняемые на текущем терминале.

Столбец time показывает, сколько процессорного времени выполняется процесс. Оно не является фактическим временем с момента запуска процесса, поскольку Linux – это многозадачная операционная система. Информация, указанная в столбце time, показывает время, реально потраченное процессором на выполнение процесса.

Столбец CMD(COMMAND) показывает имя программы. Отображается только имя, опции командной строки не выводятся.

Для получения расширенного списка процессов, выполняемых в системе, используется следующая команда:

ps –ax

Опции, заданные программе в этом примере, заставляют ее выводить не только имена программ, но и список опций, с которыми были запущены программы.

Появился новый столбец - STAT. В этом столбце отображается состояние (status) процесса. Полный список состояний вы можете прочитать в описании программы ps. Опишем ключевые состояния:

– буква R обозначает запущенный процесс, исполняющийся в данный момент времени;

– буква S обозначает спящий (sleeping) процесс — процесс ожидает какое-то событие, необходимое для его активизации;

– буква Z используется для обозначения "зомбированных" процессов (zombied) — это процессы, родительский процесс которых прекратил свое существование, оставив дочерние процессы рабочими.

Если обратить внимание на колонку TTY, то можно заметить, что многие процессы, расположенные в верхней части таблицы, в этой колонке содержат знак "?" вместо терминала. Так обозначаются процессы, запущенные с более не активного терминала. Как правило, это всякие системные сервисы.

Если вы хотите увидеть еще больше информации о выполняемых процессах, попробуйте выполнить команду:

ps -aux

Появились еще следующие столбцы:

- USER - показывает, от имени какого пользователя был запущен данный процесс;
- %CPU, %MEM - показывают, сколько данный процесс занимает соответственно процессорного времени и объем используемой оперативной памяти;
- TIME - время запуска программы.

top

Еще одна утилита, с помощью которой можно получать информацию о запущенных в операционной системе процессах. Для использования достаточно просто запустить команду top на выполнение. Эта утилита выводит на экран список процессов в системе, отсортированных в порядке убывания значений используемых ресурсов.

В начале сообщения, выведенного на экране, идет общесистемная информация — из нее можно узнать время запуска операционной системы, время работы операционной системы от момента последнего перезапуска системы, количество зарегистрированных в данный момент в операционной системе пользователей, а также минимальную, максимальную и среднюю загрузку операционной системы. Помимо этого, отображается общее количество процессов и их состояние, сколько процентов ресурсов системы используют пользовательские процессы и системные процессы, использование оперативной памяти и «свопа».

Далее идет таблица, во многом напоминающая вывод программы ps. Идентификационный номер процесса, имя пользователя — владельца процесса, приоритет процесса, размер процесса, его состояние, используемые процессом оперативная память и ресурс центрального процесса, время выполнения и, наконец, имя процесса.

Утилита top после запуска периодически обновляет информацию о состоянии процессов в операционной системе, что позволяет динамически получать информацию о загрузке системы.

kill

Программа kill предназначена для посылки соответствующих сигналов указанному процессу. Как правило, это бывает тогда, когда некоторые процессы начинают вести себя неадекватно. Наиболее часто программа применяется, чтобы прекратить выполнение процессов.

Для того чтобы прекратить работу процесса, необходимо знать PID процесса либо его имя. Например, чтобы "убить" процесс 123, достаточно выполнить следующую команду:

kill 123

Как обычно, чтобы прекратить работу процесса, пользователю необходимо быть его владельцем. Пользователь root может прекратить работу любого процесса в системе.

Иногда стандартное выполнение программы kill не справляется с поставленной задачей. Обычно это объясняется тем, что данный процесс завис либо выполняет операцию, которую с его точки зрения нельзя прервать немедленно. Для прерывания этого процесса можно воспользоваться следующей командой:

kill -9 123 .

Вообще-то программа kill предназначена для посылки процессам управляющих сигналов, одним из которых является сигнал sigterm (terminate, завершиться). Этот сигнал посылается процессу при выполнении программы kill по умолчанию. Процесс, получивший данный сигнал, должен корректно завершить свою работу (закрыть используемые файлы, сбросить буферы ввода/вывода и т. п.). Ключ -9 указывает программе kill посылать процессу другой тип сигнала — sigkill. Это приводит к тому, что процесс не производит корректного завершения, а немедленно прекращает свою жизнедеятельность. Помимо этих сигналов, в вашем распоряжении целый набор различных сигналов. Полный список сигналов можно получить с помощью команды вызова помощи.

killall

Еще один вариант программы kill. Используется для того, чтобы завершить работу процессов, носящих одно и то же имя. К примеру, в системе запущено несколько программ mc. Для того чтобы одновременно завершить работу этих программ, достаточно всего лишь выполнить следующую команду:

killall mc

Конечно, этим не ограничивается использование данной команды. С ее помощью можно отсылать сигналы группе одноименных процессов. Для получения более подробной информации по этой команде обращайтесь к ее man-странице.

Изменение приоритета выполнения процессов

В операционной системе Linux у каждого процесса есть свой приоритет исполнения. Это очень удобно. Поскольку операционная система многозадачная – то для выполнения каждого процесса выделяется определенное количество времени. Для некоторых задач необходимо выделить побольше, для некоторых можно поменьше. Для этого и предназначен приоритет процесса. Управление приоритетом процесса осуществляется программами nice и renice.

nice

Программа nice позволяет запустить команду с предопределенным приоритетом выполнения, который задается в командной строке. При обычном запуске все задачи имеют один и тот же приоритет, и операционная

система равномерно распределяет между ними процессорное время. Однако с помощью утилиты nice можно понизить приоритет какой-либо задачи, таким образом, предоставляя другим процессам больше процессорного времени. Повысить приоритет той или иной задачи имеет право только пользователь root. Синтаксис использования nice следующий:

nice -*number command*

Уровень приоритета процесса определяется параметром *number,* при этом большее его значение означает меньший приоритет процесса. Значение по умолчанию — 10, и *number* представляет собой число, на которое должен быть уменьшен приоритет.

К примеру, процесс top имеет приоритет, равный -5. Для того чтобы понизить приоритет выполнения процесса на десять, мы должны выполнить следующую команду:

nice 10 top

В результате процесс top имеет приоритет, равный 5.

Только пользователь root может поднять приоритет того или иного процесса, используя для этого *отрицательное* значение параметра *number.*

renice

Программа renice, в отличие от программы nice, позволяет изменить приоритет уже работающего процесса. Формат запуска программы следующий:

renice -*number PID*

В общем, программа renice работает точно так же, как и nice. Уровень приоритета процесса определяется параметром *number,* при этом большее его значение означает меньший приоритет процесса. Значение по умолчанию — 10, и *number* представляет собой число, на которое должен быть уменьшен приоритет процесса.

Только пользователь root может поднять приоритет того или иного процесса, используя для этого *отрицательное* значение параметра number.

Выполнение процессов в заданное время

Одна из основных задач автоматизации администрирования операционной системы — выполнение программ в заданное время или с заданной периодичностью. Для решения этих проблем существует несколько утилит, позволяющих запускать процессы в нужное время.

at

Для запуска одной или более команд в заранее определенное время используется команда at. В этой команде вы можете определить время и дату запуска той или иной команды. Команда at требует, по меньшей мере, двух параметров — время выполнения программы и запускаемую программу с ее параметрами запуска.

Приведенный ниже пример запустит команду на выполнение в 01:01. Для этого введите все, приведенное ниже, с терминала, завершая ввод каждой

строки нажатием клавиши <Enter> и по окончании ввода всей команды — <Ctrl>+<D> для ее завершения.

```
at 1:01
    ls
echo “Time is 1:01”
```

Помимо времени, в команде at может быть также определена и дата запуска программы на выполнение.

Пользователь root может без ограничения применять практически любые команды. Для обычных пользователей права доступа к команде at определяются файлами /etc/at.allow и /etc/at.deny. В файле /etc/at.allow содержится список тех, кому разрешено использовать команду at, а в файле /etc/at.deny находится список тех, кому ее выполнять запрещено.

batch

Команда batch в принципе аналогична команде at. Более того, batch представляет собой псевдоним команды at -b. Пользователь хочет запустить резервное копирование вечером. Однако в это время система очень занята, и выполнение резервирования системы практически парализует ее работу. Для этого и существует команда batch — ее использование позволяет операционной системе самой решить, когда наступает подходящий момент для запуска задачи в то время, когда система не сильно загружена.

Формат команды batch представляет собой просто список команд для выполнения, следующих в строках за командой; заканчивается список комбинацией клавиш <Ctrl>+<D>. Можно также поместить список команд в файл и перенаправить его на стандартный ввод команды batch.

cron

Cron — это программа, выполняющая задания по расписанию, но, в отличие от команды at, она позволяет выполнять задания неоднократно. Вы определяете времена и даты, когда должна запускаться та или иная программа. Времена и даты могут определяться в минутах, часах, днях месяца, месяцах года и днях недели.

Программа cron запускается один, раз при загрузке системы. При запуске cron проверяет очередь заданий at и задания пользователей в файлах crontab. Если для запуска не было найдено заданий — следующую проверку cron произведет через минуту.

Практическая часть

1. Запустите программу yes в фоновом режиме с подавлением потока вывода.
2. Запустите программу yes на переднем плане с подавлением потока вывода. Приостановите выполнение программы. Заново запустите программу yes с теми же параметрами, и завершите ее выполнение.
3. Запустите программу yes на переднем плане без подавления потока вывода.

Приостановите выполнение программы. Заново запустите программу yes с теми же параметрами, и завершите ее выполнение.

4. Проверьте состояния процессов, воспользовавшись командой jobs.
5. Переведите процесс, который у вас выполняется в фоновом режиме на передний план и остановите его.
6. Переведите любой ваш процесс с подавлением потока вывода в фоновый режим.
7. Проверьте состояния процессов, воспользовавшись командой jobs. Обратите внимание, что процесс стал выполняющемся (Running) в фоновом режиме.
8. Запустите процесс в фоновом режиме, таким образом, чтобы он продолжил свою работу даже после отключения от терминала.
9. Закройте окно и заново запустите консоль. Убедитесь, что процесс продолжил свою работу.
10. Получите информацию о запущенных в операционной системе процессах с помощью утилиты top.
11. Запустите еще три программы yes в фоновом режиме с подавлением потока вывода.
12. «Убейте» два процесса: для одного используйте его PID, а для другого его идентификатор конкретного задания.
13. Попробуйте послать сигнал 1 (SIGHUP) процессу, запущенному с помощью nohup и обычному процессу.
14. Запустите еще несколько программ yes в фоновом режиме с подавлением потока вывода.
15. Завершите их работу одновременно, используя команду killall.
16. Запустите программу yes в фоновом режиме с подавлением потока вывода. Используя утилиту nice, запустите программу yes с теми же параметрами и с приоритетом, большим на 5. Сравните абсолютные и относительные приоритеты у этих двух процессов.
17. Используя утилиту renice, измените приоритет у одного из потоков yes таким образом, чтобы у обоих потоков приоритеты были равны.
18. Сделайте так, чтобы в xx минут xx часов автоматически выполнилась утилита ls и вывелась строка текста «Lab rab 3. Zadanie 18». Учтите, что вывод будет осуществляться не на экран, а в файл: /var/spool/mail/student (xx минут xx часов – ближайшие несколько минут).
19. Сделайте так, чтобы в xx минут xx часов каждую пятницу автоматически выполнилась утилита ps и вывелась строка текста «Lab rab 3. Zadanie 19». Учтите, что вывод будет осуществляться не на экран, а в файл: /var/spool/mail/student (xx минут xx часов – ближайшие несколько минут).
20. Сделайте так, чтобы в xx минут xx часов каждый xx месяц автоматически выполнилась утилита ps и вывелась строка текста «Lab rab 3. Zadanie 20». Учтите, что вывод будет осуществляться не на экран, а в файл: /var/spool/mail/student (xx минут xx часов – ближайшие несколько минут, xx месяц – текущий месяц).

Контрольные вопросы.

1. Объясните, что произойдет, если запустить программу yes в фоновом режиме без подавления потока вывода.
2. Объясните разницу между действием сочетаний клавиш ^Z и ^C.
3. Опишите, что значит каждое поле вывода команды jobs.
4. Назовите главное отличие утилиты top от ps.
5. Почему процесс, запущенный с помощью nohup не «убивается» сигналом 1?

Лабораторная работа № 4. Простейшие классы и объекты C++.

Цель работы:
– изучить основные понятия объектно-ориентированного программирования на практике на примере языка C++;
– изучить способы объявления классов и создания объектов в C++;
– освоить основные способы организации доступа к данным объекта;
– научиться создавать простейшие классы для структурированного хранения данных элементарных типов;
В процессе выполнения работы студент, руководствуясь методическими указаниями к выполнению данной работы:
– определяет способ организации программы в виде системы классов и объектов;
– проводит разработку программы с применением принципов объектно-ориентированного подхода;
– управляет запуском и выполнением разработанной программы;
– предоставляет исходный код программы и отвечает на вопросы преподавателя для получения зачета за выполненную работу.

Задание

Необходимо спроектировать и запрограммировать на языке C++ классы фигур, согласно вариантов задания (таблица 1).
Классы должны удовлетворять следующим правилам:
• Должны иметь общий родительский класс Figure.
• Должны иметь общий виртуальный метод Print, печатающий параметры фигуры и ее тип в стандартный поток вывода cout.
• Должный иметь общий виртуальный метод расчета площади фигуры – Square.
• Должны иметь конструктор, считывающий значения основных параметров фигуры из стандартного потока cin.
• Должны быть расположенны в раздельных файлах: отдельно заголовки (.h), отдельно описание методов (.cpp).
Программа должна позволять вводить фигуру каждого типа с клавиатуры, выводить параметры фигур на экран и их площадь.

Варианты задания (фигуры вращения)

Таблица 1

Вариант	Фигура №1	Фигура № 2	Фигура № 3
1.	Треугольник	Квадрат	Прямоугольник
2.	Квадрат	Прямоугольник	Трапеция
3.	Прямоугольник	Трапеция	Ромб

4.	Трапеция	Ромб	5-угольник
5.	Ромб	5-угольник	6-угольник
6.	6-угольник	8-угольник	Треугольник
7.	8-угольник	Треугольник	Квадрат
8.	Треугольник	Квадрат	Прямоугольник
9.	Прямоугольник	Трапеция	Ромб
10.	Ромб	5-угольник	6-угольник
11.	8-угольник	3-угольник	Ромб
12.	Трапеция	Ромб	5-угольник
13.	Квадрат	Прямоугольник	Трапеция
14.	Трапеция	Ромб	5-угольник
15.	5-угольник	6-угольник	8-угольник
16.	Квадрат	Прямоугольник	Трапеция
17.	Треугольник	Квадрат	Прямоугольник
18.	5-угольник	8-угольник	Треугольник
19.	Квадрат	Прямоугольник	8-угольник
20.	Квадрат	Ромб	5-угольник
21.	Ромб	Треугольник	Трапеция

Полезный пример

Данный пример демонстрирует основные возможности языка C++, которые понадобится применить в данной лабораторной работе. Пример не является решением варианта лабораторной работы.

Листинг файла Figure.h

```
#ifndef FIGURE_H
#define FIGURE_H
class Figure {
public:
     virtual double Square() = 0;
     virtual void Print() = 0;
     virtual ~Figure() {};
};
#endif /* FIGURE_H */
```

Листинг файла Triangle.h

```
#ifndef TRIANGLE_H
#define TRIANGLE_H
#include <cstdlib>
#include <iostream>
#include "Figure.h"
class Triangle : public Figure{
public:
     Triangle();
```

```
     Triangle(std::istream &is);
     Triangle(size_t i,size_t j,size_t k);
     Triangle(const Triangle& orig);
     double Square() override;
     void Print() override;
     virtual ~Triangle();
private:
     size_t side_a;
     size_t side_b;
     size_t side_c;
};
#endif /* TRIANGLE_H */
```

Листинг файла Triangle.cpp

```
#include "Triangle.h"
#include <iostream>
#include <cmath>
Triangle::Triangle() : Triangle(0, 0, 0) {
}
Triangle::Triangle(size_t i, size_t j, size_t k) : side_a(i),
side_b(j),
side_c(k) {
     std::cout << "Triangle created: " << side_a << ", " <<
side_b << ", " <<
side_c << std::endl;
}
Triangle::Triangle(std::istream &is) {
     is >> side_a;
     is >> side_b;
     is >> side_c;
}
Triangle::Triangle(const Triangle& orig) {
     std::cout << "Triangle copy created" << std::endl;
     side_a = orig.side_a;
     side_b = orig.side_b;
     side_c = orig.side_c;
}
double Triangle::Square() {
double p = double(side_a + side_b + side_c) / 2.0;
return sqrt(p * (p - double(side_a))*(p - double(side_b))*(p -
double(side_c)));
}
void Triangle::Print() {
     std::cout << "a=" << side_a << ", b=" << side_b << ", c="
<< side_c <<
     std::endl;
}
Triangle::~Triangle() {
     std::cout << "Triangle deleted" << std::endl;
}
```

Листинг файла Main.cpp

```
#include <cstdlib>
#include "Triangle.h"
int main(int argc, char** argv) {
     Figure *ptr = new Triangle(std::cin);
     ptr->Print();
     std::cout << ptr->Square() << std::endl;
     delete ptr;
     return 0;
}
```

Лабораторная работа № 5. Разработка классов.

Цель работы:
– изучить способы защиты данных при разработке классов в C++;
– изучить способы инициализации классов в конструкторе;
– освоить основные способы взаимодействия с объектами класса;
В процессе выполнения работы студент, руководствуясь методическими указаниями к выполнению данной работы:
– определяет способ организации класса с учетом требований по корректному и безопасному хранению данных;
– проводит разработку программы на основе разработанных классов;
- проводит отладку программы
– предоставляет исходный код программы и отвечает на вопросы преподавателя для получения зачета за выполненную работу.

Задание

Необходимо спроектировать и запрограммировать на языке C++ класс-контейнер первого уровня, содержащий одну фигуру (колонка фигура 1), согласно вариантов задания (реализованную в ЛР4).
Классы должны удовлетворять следующим правилам:
• Требования к классу фигуры аналогичны требованиям из лабораторной работы 4.
• Классы фигур должны иметь переопределенный оператор вывода в поток std::ostream (<<).
Оператор должен распечатывать параметры фигуры (тип фигуры, длины сторон, радиус и т.д).
• Классы фигур должны иметь переопределенный оператор ввода фигуры из потока std::istream (>>).
Оператор должен вводить основные параметры фигуры (длины сторон, радиус и т.д).
• Классы фигур должны иметь операторы копирования (=).
• Классы фигур должны иметь операторы сравнения с такими же фигурами (==).
• Класс-контейнер должен содержать объекты фигур “по значению” (не по ссылке).
• Класс-контейнер должен иметь метод по добавлению фигуры в контейнер.
• Класс-контейнер должен иметь методы по получению фигуры из контейнера (определяется структурой контейнера).
• Класс-контейнер должен иметь метод по удалению фигуры из контейнера (определяется структурой контейнера).
• Класс-контейнер должен иметь перегруженный оператор по выводу контейнера в поток std::ostream (<<).
• Класс-контейнер должен иметь деструктор, удаляющий все элементы контейнера.

• Классы должны быть расположенны в раздельных файлах: отдельно заголовки (.h), отдельно описание методов (.cpp).
Нельзя использовать:
• Стандартные контейнеры std.
• Шаблоны (template).
• Различные варианты умных указателей (shared_ptr, weak_ptr).
Программа должна позволять:
• Вводить произвольное количество фигур и добавлять их в контейнер.
• Распечатывать содержимое контейнера.
• Удалять фигуры из контейнера.

Полезный пример

Данный пример демонстрирует основные возможности языка C++, которые понадобится применить в данной лабораторной работе. Пример не является решением варианта лабораторной работы.

Листинг файла TStack.h

```
#ifndef TSTACK_H
#define TSTACK_H
#include "Triangle.h"
#include "TStackItem.h"
class TStack {
public:
     TStack();
     TStack(const TStack& orig);
     void push(Triangle &&triangle);
     bool empty();
     Triangle pop();
     friend std::ostream& operator<<(std::ostream& os,const
TStack& stack);
     virtual ~TStack();
private:
     TStackItem *head;
};
#endif /* TSTACK_H */
```

Листинг файла TStack.cpp

```
#include "TStack.h"
TStack::TStack() : head(nullptr) {
}
TStack::TStack(const TStack& orig) {
head = orig.head;
}
std::ostream& operator<<(std::ostream& os, const TStack& stack)
{
```

```
     TStackItem *item = stack.head;
     while(item!=nullptr)
     {
          os << *item;
          item = item->GetNext();
     }
     return os;
}
void TStack::push(Triangle &&triangle) {
     TStackItem *other = new TStackItem(triangle);
     other->SetNext(head);
     head = other;
}
bool TStack::empty() {
     return head == nullptr;
}
Triangle TStack::pop() {
     Triangle result;
     if (head != nullptr) {
     TStackItem *old_head = head;
     head = head->GetNext();
     result = old_head->GetTriangle();
     old_head->SetNext(nullptr);
     delete old_head;
}
     return result;
}
TStack::~TStack() {
     delete head;
}
```

Листинг файла TStackItem.h

```
#ifndef TSTACKITEM_H
#define TSTACKITEM_H
#include "Triangle.h"
class TStackItem {
public:
     TStackItem(const Triangle& triangle);
     TStackItem(const TStackItem& orig);
     friend std::ostream& operator<<(std::ostream& os, const
TStackItem& obj);
     TStackItem* SetNext(TStackItem* next);
     TStackItem* GetNext();
     Triangle GetTriangle() const;
     virtual ~TStackItem();
private:
     Triangle triangle;
     TStackItem *next;
};
#endif /* TSTACKITEM_H */
```

Листинг файла TStackItem.cpp

```
#include "TStackItem.h"
#include <iostream>
TStackItem::TStackItem(const Triangle& triangle) {
     this->triangle = triangle;
     this->next = nullptr;
     std::cout << "Stack item: created" << std::endl;
}
TStackItem::TStackItem(const TStackItem& orig) {
     this->triangle = orig.triangle;
     this->next = orig.next;
     std::cout << "Stack item: copied" << std::endl;
}
TStackItem* TStackItem::SetNext(TStackItem* next) {
     TStackItem* old = this->next;
     this->next = next;
     return old;
}
Triangle TStackItem::GetTriangle() const {
     return this->triangle;
}
TStackItem* TStackItem::GetNext() {
     return this->next;
}
TStackItem::~TStackItem() {
     std::cout << "Stack item: deleted" << std::endl;
     delete next;
}
std::ostream& operator<<(std::ostream& os, const TStackItem&
obj) {
     os << "[" << obj.triangle << "]" << std::endl;
     return os;
}
```

Листинг файла Triangle.h

```
#ifndef TRIANGLE_H
#define TRIANGLE_H
#include <cstdlib>
#include <iostream>
class Triangle {
public:
     Triangle();
     Triangle(size_t i,size_t j,size_t k);
     Triangle(const Triangle& orig);
     Triangle& operator++();
     double Square();
     friend Triangle operator+(const Triangle& left,const
Triangle& right);
     friend std::ostream& operator<<(std::ostream& os, const
Triangle& obj);
```

```
     friend std::istream& operator>>(std::istream& is, Triangle&
obj);
     Triangle& operator=(const Triangle& right);
     virtual ~Triangle();
private:
     size_t side_a;
     size_t side_b;
     size_t side_c;
};
#endif /* TRIANGLE_H */
```

Листинг файла Triangle.cpp

```
#include "Triangle.h"
#include <iostream>
#include <cmath>
Triangle::Triangle() : Triangle(0, 0, 0) {
}
Triangle::Triangle(size_t i, size_t j, size_t k) : side_a(i),
side_b(j), side_c(k) {
     std::cout << "Triangle created: " << side_a << ", " <<
side_b << ", " <<
     side_c << std::endl;
}
Triangle::Triangle(const Triangle& orig) {
     std::cout << "Triangle copy created" << std::endl;
     side_a = orig.side_a;
     side_b = orig.side_b;
     side_c = orig.side_c;
}
double Triangle::Square(){
     double p = double(side_a + side_b + side_c) / 2.0;
     return sqrt(p * (p - double(side_a))*(p -
double(side_b))*(p -
double(side_c)));
}
Triangle& Triangle::operator=(const Triangle& right) {
     if (this == &right) return *this;
     std::cout << "Triangle copied" << std::endl;
     side_a = right.side_a;
     side_b = right.side_b;
     side_c = right.side_c;
     return *this;
}
Triangle& Triangle::operator++() {
     side_a++;
     side_b++;
     side_c++;
     return *this;
}
Triangle operator+(const Triangle& left,const Triangle& right) {
     return
```

```
Triangle(left.side_a+right.side_a,left.side_b+right.side_b,left.
side_c+right.side_c);
}
Triangle::~Triangle() {
     std::cout << "Triangle deleted" << std::endl;
}
std::ostream& operator<<(std::ostream& os, const Triangle& obj)
{
     os << "a=" << obj.side_a << ", b=" << obj.side_b << ", c="
<< obj.side_c
<< std::endl;
     return os;
}
std::istream& operator>>(std::istream& is, Triangle& obj) {
     is >> obj.side_a;
     is >> obj.side_b;
     is >> obj.side_c;
     return is;
}
```

Листинг файла Main.cpp

```
#include <cstdlib>
#include <iostream>
#include "Triangle.h"
#include "TStackItem.h"
#include "TStack.h"
// Simple stack on pointers
int main(int argc, char** argv) {
     TStack stack;
     stack.push(Triangle(1,1,1));
     stack.push(Triangle(2,2,2));
     stack.push(Triangle(3,3,3));
     std::cout << stack;
     Triangle t;
     t = stack.pop(); std::cout << t;
     t = stack.pop(); std::cout << t;
     t = stack.pop(); std::cout << t;
     return 0;
}
```

Лабораторная работа № 6. Классы для работы с динамическими структурами данных.

Цель работы:
– изучить классы стандартной библиотеки C++, управляющие работой стека, очереди.
– изучить способы динамических структур данных различных типов на языке C++.
– научиться создавать подходящие динамические структуры для решения прикладной задачи;
В процессе выполнения работы студент, руководствуясь методическими указаниями к выполнению данной работы:
– определяет способ организации хранения данных программы в виде системы динамических структур данных;
– проводит разработку программы с применением принципов динамического хранения данных;
– управляет запуском и выполнением разработанной программы;
– предоставляет исходный код программы и отвечает на вопросы преподавателя для получения зачета за выполненную работу.

Задание

1.Опишите стек с целочисленным информационным полем. Заполните его длинами строк, считанных из файла. Распечатайте на экране содержимое стека. Укажите номер и длину последней самой короткой строки файла.

2.Разработайте программу, с помощью которой можно определить наибольший допустимый размер очереди с вещественным информационным полем. Найдите этот размер (число элементов в очереди).

3.Опишите очередь с вещественным информационным полем, и заполните ее элементами с клавиатуры. Выполните циклический сдвиг элементов в очереди так, чтобы в ее начале был расположен наибольший элемент.

4.Разработайте программу, с помощью которой можно определить наибольший допустимый размер стека с вещественным информационным полем. Найдите этот размер (число элементов в стеке). Сравните с наибольшим допустимым размером очереди с аналогичным информационным полем.

Указания к выполнению работы.

Каждое задание необходимо решить в соответствии с изученными методами формирования, вывода и обработки данных очередей и стеков в языке С++. Обработку очередей или стеков следует выполнить на основе базовых алгоритмов: поиск, вставка элемента, удаление элемента, удаление всей динамической структуры. При объявлении списков выполните комментирование используемых полей. Задачи 2 и 4 носят исследовательский характер, поэтому при составлении отчета к ним следует подробно описать предлагаемый метод оценки максимального размера очереди или стека. Программу для решения каждого задания необходимо разработать методом процедурной абстракции, оформив комментарии к коду.

Следует реализовать каждое задание в соответствии с приведенными этапами:

- изучить словесную постановку задачи, выделив при этом все виды данных;
- сформулировать математическую постановку задачи;
- выбрать метод решения задачи, если это необходимо;
- разработать графическую схему алгоритма;
- записать разработанный алгоритм на языке С++;
- разработать контрольный тест к программе;
- отладить программу;
- представить отчет по работе.

Требования к отчету.

Отчет по лабораторной работе должен соответствовать следующей структуре.

1) Титульный лист. Словесная постановка задачи. В этом подразделе проводится полное описание задачи. Описывается суть задачи, анализ входящих в нее физических величин, область их допустимых значений, единицы их измерения, возможные ограничения, анализ условий при которых задача имеет решение (не имеет решения), анализ ожидаемых результатов.

2) Математическая модель. В этом подразделе вводятся математические описания физических величин и математическое описание их взаимодействий. Цель подраздела – представить решаемую задачу в математической формулировке.
3) Алгоритм решения задачи. В подразделе описывается разработка структуры алгоритма, обосновывается абстракция данных, задача

разбивается на подзадачи. Схема алгоритма выполняется по ЕСПД (ГОСТ 19.003-80 и ГОСТ 19.002-80).

4) Листинг программы. Подраздел должен содержать текст программы на языке программирования С++
5) Контрольный тест. Подраздел содержит наборы исходных данных и полученные в ходе выполнения программы результаты.
6) Выводы по лабораторной работе.
7) Ответы на контрольные вопросы.

Контрольные вопросы

1.В чем преимущества и недостатки организации структур в виде стека?
2.В чем преимущества и недостатки организации структур в виде очереди?
3.Для моделирования каких реальных задач удобно использовать стек? А для каких очередь?
4.Какое значение хранит указатель на стек?
5.Какое значение хранит указатель на очередь?
6.Какие существуют ограничения на тип информационного поля стеки и очереди?
7.С какой целью в программах выполняется проверка на пустоту стека и очереди?
8.При работе со стеком или очередью доступны позиции ограниченного числа элементов. Возможна ли ситуация записи новых элементов стека или очереди на уже занятые собственными элементами участки памяти (запись себя поверх себя)? Ответ обоснуйте.
9.С какой целью в программах выполняется удаление стека и очереди по окончании работы с ними? Как изменится работа программы, если операцию удаления не выполнять?

Лабораторная работа № 7. Наследование. Потоки.

Цель работы:
– изучить механизм наследования и управления потоками выполнения программы на языке C++.
– изучить способы объявления наследования классов и организации потоков в C++.
– освоить основные способы управления доступом к данным класса при наследовании;
В процессе выполнения работы студент, руководствуясь методическими указаниями к выполнению данной работы:
– определяет способ организации программы в виде системы классов и объектов с использованием наследования для сокращения размеров программы и потоков для распараллеливания подзадач;
– проводит разработку программы с применением принципов наследования и управления потоками;
– управляет запуском и выполнением разработанной программы;
– предоставляет исходный код программы и отвечает на вопросы преподавателя для получения зачета за выполненную работу.

Задание 1

Написать программу, использующую потоки cout и cin для ввода/вывода. Программа должна использовать операции new и delete для работы с динамической памятью.

В программе должна быть функция создания и ввода динамического целочисленного массива. Используя манипуляторы, выведите на экран результат функции, обрабатывающей массив согласно своему варианту, отформатировав его к правой стороне выделенного поля (10 позиций), не более 4-х знаков после запятой, а символ-заполнитель установите *. Не забудьте освободить динамическую память.

Задние 2

Выполните предыдущее задание для прямоугольного двумерного динамического массива.

Варианты (задание 1 и 2)

1.Найдите полусумму максимального и минимального элементов
2.Найдите отношение суммы четных к сумме нечетных элементов массива
3.Найти среднее арифметическое положительных элементов массива.

4.Найдите отношение количества отрицательных элементов к количеству положительных элементов массива
5.Найдите среднее арифметическое четных элементов массива.
6.Найдите отношение минимального значения к максимальному
7.Найдите среднее арифметическое отрицательных нечетных элементов массива
8.Найдите среднее геометрическое модулей максимального и минимального значений массива
9.Найти среднее арифметическое отрицательных элементов массива.
10.Найдите отношение количества четных элементов к количеству нечетных элементов массива
11.Найдите среднее арифметическое отрицательных четных элементов массива
12.Найдите среднее арифметическое положительных элементов массива.
13.Найдите отношение максимального значения к минимальному
14.Найдите среднее арифметическое отрицательных нечетных элементов массива
15.Найдите отношение суммы модулей отрицательных к сумме положительных элементов массива
16.Найдите минимальное значение среди положительных элементов (если таких в массиве нет, то результат – ноль)
17.Найдите максимальное значение среди отрицательных элементов (если таких в массиве нет, то результат – ноль)
18.Найдите количество элементов массива, имеющих максимальное значение

Задание 3

Написать программу, использующую потоки cout и cin для ввода/вывода и перегрузку функций. Программа должна содержать одноименные функции, описанные ниже, и демонстрировать их возможности.

Варианты (3)
1.Написать функции, вычисляющие произведение числа на число, скалярное и векторное произведение двух векторов (во втором случае входным параметром будет еще и адрес выходного массива; значения входных массивов не должны изменяться, а результат должен помещаться в другой массив).
2.Написать функции, вычисляющие сумму двух чисел, двух векторов из трех элементов и двух матриц 3x3 (значения входных массивов не должны изменяться, а результат должен помещаться в другой массив).
3.Написать функции, вычисляющие модуль целого числа, модуль действительного числа, длину вектора из трех элементов и определитель

матрицы 3x3.
4.Написать функции, находящие расстояние между точками, минимальное расстояние между точкой и прямой, прямой и точкой, расстояние между параллельными прямыми.

Задание 4

Создать класс Динамический массив. Класс должен содержать
•конструкторы (по количеству элементов, по массиву и количеству элементов), в том числе конструктор копирования;
•деструктор;
•перегруженную операцию присваивания;
•функцию добавления элемента в конец массива и в заданную позицию;
•функцию удаления элемента из заданной позиции;
•функция подсчета количества вхождений заданного значения в массив.

В классе должна быть предусмотрена операция помещения в поток и операция извлечения из потока элементов объекта.

Задание 5

Создать класс-наследник для класса Динамический массив из предыдущего задания. Обеспечить корректное наследование, корректные конструкторы и деструкторы.

Задание 6

Добавить операции и функции, описанные в индивидуальном задании.

Варианты

1. Класс Стек (элементы char)

функции pop, push(x), top, empty - соответственно извлечение из начала, добавление в начало, ссылка на первый элемент, проверка на пустоту; перегрузка операций << (push) и >>(pop) соответственно.

2. Класс Очередь (элементы char)

функции pop, push(x), front, back, empty - соответственно извлечение из начала, добавление в конец, ссылки на первый и последний элементы, проверка на пустоту; перегрузка операций << (push) и >>(pop) соответственно.

3. Класс Вектор (элементы double)

все операции с векторами: сложение, вычитание, домножение на число, скалярное произведение, проверка коллинеарности и параллельности двух векторов; обязательная перегрузка операций, считывание вектора из потока до нулевого элемента (перегрузка >>).

4. Класс Прямая (элементы double) y=kx+b

все операций: сложение, вычитание, домножение на число, параллельный перенос (y=f(x)+a), отражение (y=-f(x)), нахождение точки пересечения прямых, нахождение точки пересечения с Ox, угол наклона к Ox, проверка коллинеарности и параллельности двух векторов; обязательная перегрузка операций, считывание коэффициентов из потока (перегрузка >>), прямая в 3-хмерном пространстве.

5. Класс Многочлен (элементы int)

все операции: сложение, вычитание, домножение на число, добавление числа; обязательная перегрузка операций, считывание вектора из потока до нулевого элемента (перегрузка >>), перегрузка деления многочлена на многочлен.

6.Класс Упорядоченная последовательность (элементы int, все элементы упорядочены по неубыванию)

все операции с множествами с сохранением упорядоченности: сложение, вычитание, пересечение (*), функция проверки на подмножество, модуль - среднее арифметическое последовательности; обязательная перегрузка операций, функции проверки на арифметическую или геометрическую прогрессии.

7. Класс Множество (элементы int, обратите внимание, что в множестве значения элементов внутри одного объекта не должны повторяться)

все операций с множествами: сложение, вычитание, пересечение (*), модуль - количество элементов; обязательная перегрузка операций, функции проверки на подмножество.

8. Класс Подмножество (элементы bool, множество задается статическим компонентным данным класса - массивом из неповторяющихся элементов, объект показывает какие элементы множества входят в данное подмножество - true на соответствующей позиции, false - иначе)

все операции с множествами: сложение, вычитание, пересечение (*), модуль -

количество элементов; обязательная перегрузка операций, функция проверки на подмножество.

9. Класс Дек(элементы bool).функции popF, popB,pushF(x),pushB(x), front, back,empty - соответственно извлечение из начала или конца, добавление в начало или конец, ссылки на первый и последний элементы, проверка на пустоту; перегрузка операций << (pushB) и >>(popB) соответственно

Требования к отчету.

Отчет по лабораторной работе должен соответствовать следующей структуре.

1) Титульный лист. Словесная постановка задачи. В этом подразделе проводится полное описание задачи. Описывается суть задачи, анализ входящих в нее физических величин, область их допустимых значений, единицы их измерения, возможные ограничения, анализ условий при которых задача имеет решение (не имеет решения), анализ ожидаемых результатов.

2) Математическая модель. В этом подразделе вводятся математические описания физических величин и математическое описание их взаимодействий. Цель подраздела – представить решаемую задачу в математической формулировке.
3) Алгоритм решения задачи. В подразделе описывается разработка структуры алгоритма, обосновывается абстракция данных, задача разбивается на подзадачи. Схема алгоритма выполняется по ЕСПД (ГОСТ 19.003-80 и ГОСТ 19.002-80).
4) Листинг программы. Подраздел должен содержать текст программы на языке программирования C++
5) Контрольный тест. Подраздел содержит наборы исходных данных и полученные в ходе выполнения программы результаты.
6) Выводы по лабораторной работе.

Лабораторная работа № 8. Обработка исключительных ситуаций в C++.

Цель работы:

– освоить основные способы управления исключительными ситуациями времени выполнения программы в C++;
– научиться создавать обработчики основных типов исключений;
В процессе выполнения работы студент, руководствуясь методическими указаниями к выполнению данной работы:
– определяет способ обработки исключительных ситуаций, возникающих в программе;
– проводит разработку программы с применением обработки исключительных ситуаций;
– управляет запуском и выполнением разработанной программы;
– предоставляет исходный код программы и отвечает на вопросы преподавателя для получения зачета за выполненную работу.

Теоретическая часть

Обработка исключительных ситуаций

Средства обработки ошибочных ситуаций позволяют передать обработку исключений из кода, в котором возникло исключение, некоторому другому программному блоку, который выполнит в данном случае некоторые определенные действия. Таким образом, основная идея данного механизма состоит в том, что функция проекта, которая обнаружила непредвиденную ошибочную ситуацию, которую она не знает, как решить, генерирует сообщение об этом (бросок исключения). А система вызывает по этому сообщению программный модуль, который перехватит исключение и отреагирует на возникшее нештатное событие. Такой программный модуль называют «обработчик» или перехватчик исключительных ситуаций. И в случае возникновения исключения в его обработчик передаётся произвольное количество информации с контролем ее типа. Эта информация и является характеристикой возникшей нештатной ситуации.

Обработка исключений в C++ это обработка с завершением. Это означает, что исключается невозможность возобновления выполнения программы в точке возникновения исключения.

Для обеспечения работы такого механизма были введены следующие ключевые слова:

try - проба испытания;

catch - перехватить (обработать);

throw - бросать.

Кратко рассмотрим их назначение.

try - открывает блок кода, в котором может произойти ошибка; это обычный составной оператор:

```
try
 {
код
};
```

Код содержит набор операций и операторов, который и будет контролироваться на возникновение ошибки. В него могут входить вызовы функции пользователя, которые компилятор так же возьмет на контроль. Среди данного набора операторов и операций обязательно указывают операцию броска исключения: throw.

Операция броска throw имеет следующий формат:

throw выражение;

где - «выражение» определяет тип информации, которая и описывает исключение (например, конкретные типы данных).

catch - сам обработчик исключения, который перехватывает информацию:

```
catch ( тип параметр)
 {
код
 }
```

Через параметр обработчику передаются данные определенного типа, описывающие обрабатываемое исключение.

Код определяет те действия, которые надо выполнить при возникновении данной конкретной ситуации. В C++ используют несколько форм обработчиков. Такой обработчик получил название параметризованный специализированный перехватчик

Перехватчик должен следовать сразу же после блока контроля, т.е. между обработчиком и блоком контроля не должно быть ни одного оператора. При этом в одном блоке контроля можно вызывать исключения разных типов для разных ситуаций, поэтому обработчиков может быть несколько.

В этом случае их необходимо расположить сразу же за контролирующим блоком последовательно друг за другом.

Кроме того, запрещены переходы, как извне в обработчик, так и между обработчиками.

Можно воспользоваться универсальным или абсолютным обработчиком:

```
catch ( . . . )
 {
код
}
```

где (...) - означают способность данного перехватчика обрабатывать информацию любого типа. Такой обработчик располагают последним в пакете специализированных обработчиков. Тогда, если исключение не будет перехвачено специализированными обработчиками, то буде выполнен последний - универсальный.

В случае не возникновения исключения, набор обработчиков будет обойден, т.е. проигнорирован.

Если же исключение было брошено, при возникновении критической ситуации, то будет вызван конкретный перехватчик при совпадении его параметра с выражением в операторе броска, т.е. управление будет передано найденному обработчику. После выполнения кода вызванного обработчика, управление передается оператору, который расположенный за последним перехватчиком, или проект корректно завершает работу.

Существенное отличие вызова конкретного обработчика от вызова обычной функции заключается в следующем: при возникновении исключения и передаче управления определенному обработчику, система осуществляет вызов всех деструкторов для всех объектов классов, которые были созданы с момента начала контроля и до возникновения исключительной ситуации с целью их уничтожения.

Блоки try, как составные блоки могут быть вложены:

```
try {
 ...
 try
 {
 ...
 }
 ...
 }
```

тогда, в случае возникновения исключения в некотором текущем блоке, поиск обработчика последовательно продолжается в блоках, предшествующих уровней вложенности с продолжением вызова деструкторов.

Задание

Перепишите программу из лабораторной работы № 7 с учетом обработки исключительных ситуаций.

Отчет по лабораторной работе должен соответствовать следующей структуре.

1) Титульный лист. Словесная постановка задачи. В этом подразделе проводится полное описание задачи. Описывается суть задачи, анализ входящих в нее физических величин, область их допустимых значений, единицы их измерения, возможные ограничения, анализ условий при которых задача имеет решение (не имеет решения), анализ ожидаемых результатов.

2) Математическая модель. В этом подразделе вводятся математические описания физических величин и математическое описание их взаимодействий. Цель подраздела – представить решаемую задачу в математической формулировке.
3) Алгоритм решения задачи. В подразделе описывается разработка структуры алгоритма, обосновывается абстракция данных, задача разбивается на подзадачи. Схема алгоритма выполняется по ЕСПД (ГОСТ 19.003-80 и ГОСТ 19.002-80).
4) Листинг программы. Подраздел должен содержать текст программы на языке программирования C++
5) Контрольный тест. Подраздел содержит наборы исходных данных и полученные в ходе выполнения программы результаты.
6) Выводы по лабораторной работе.