Отчёт по лабораторной работе №14

Дисциплина: Операционные системы

Матвеева Анастасия Сергеевна

Содержание

# Цель работы

Приобрести простейшие навыки разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linux на примере создания на языке программирования С калькулятора с простейшими функциями.

# Задачи

1. Познакомиться со стандартным средством для компиляции программ в ОС типа UNIX - GCC (GNU Compiler Collection).
2. Познакомиться с отладчиком GDB (GNU Debugger).
3. Познакомиться с утилитой splint.
4. В ходе работы проанализировать и выполнить данные программы.

# Выполнение лабораторной работы

1. В домашнем каталоге создаю подкаталог ~/work/os/lab\_prog с помощью команды «mkdir -p ~/work/os/lab\_prog». Далее создала в каталоге файлы: calculate.h, calculate.c, main.c, используя команды «cd ~/work/os/lab\_prog» и «touch calculate.h calculate.c main.c»(рис. 1)

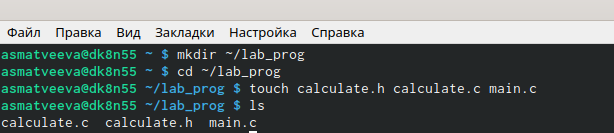


Figure 1: Создание подкаталога и файлов

Это будет примитивнейший калькулятор, способный складывать, вычитать, умножать и делить, возводить число в степень, брать квадратный корень, вычислять sin, cos, tan. При запуске он будет запрашивать первое число, операцию, второе число. После этого программа выведет результат и остановится. Открыв редактор Emacs, приступила к редактированию созданных файлов. Реализация функций калькулятора в файле calculate.с. (рис. 2, 3)

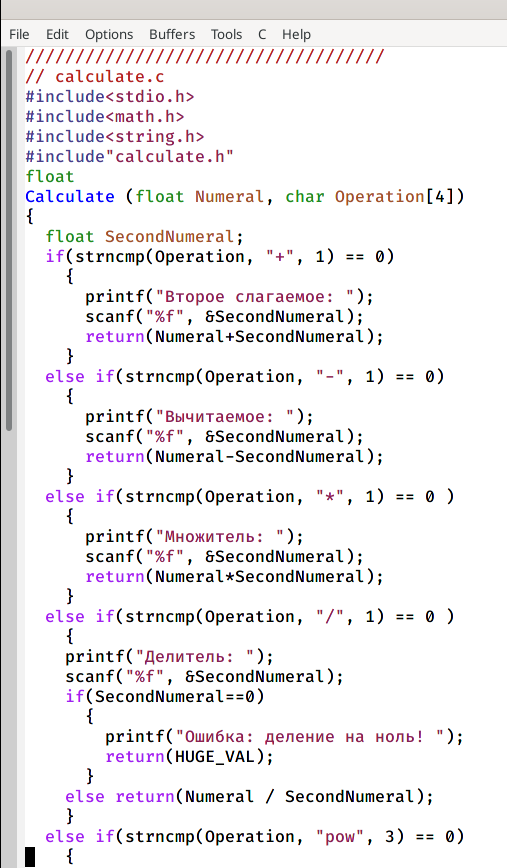


Figure 2: Файл calculate.c

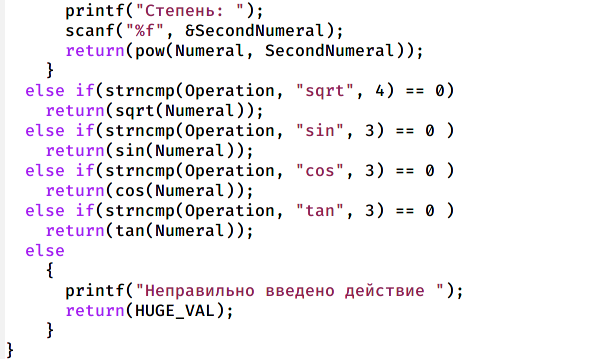


Figure 3: Файл calculate.c

Интерфейсный файл calculate.h,описывающий формат вызова функции калькулятора.(рис. 4)

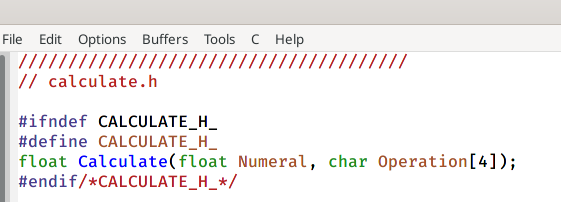


Figure 4: Файл calculate.h

Основной файл main.c,реализующий интерфейс пользователя к калькулятору.(рис. 5)

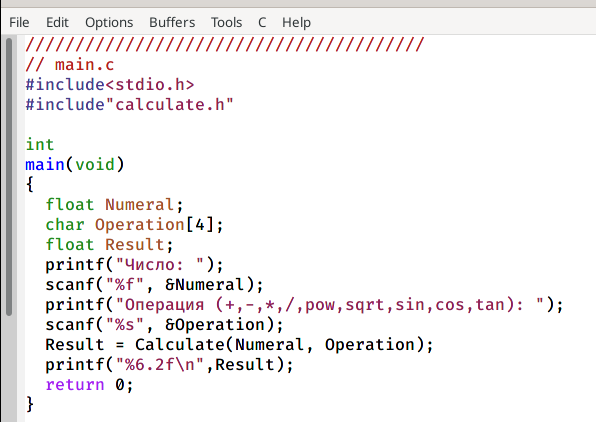


Figure 5: Файл main.c

1. Выполнила компиляцию программы посредством gcc (версия компилятора 8.3.0-19),используя команды «gcc -c calculate.c»,«gcc -c main.c» и «gcc calculate.o main.o -o calcul -lm» (рис. 6)

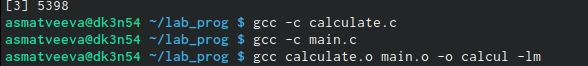


Figure 6: Выполняем компиляцию посредством gcc

1. Так как в ходе предыдущей компиляции программы выявились синтаксические ошибки, поэтому я их исправила и выполнила компиляцию заново. В строке “scanf(“%s”, &Operation);" нужно было убрать знак &, потому что имя массива символов уже является указателем на первый элемент этого массива.(рис. 7)

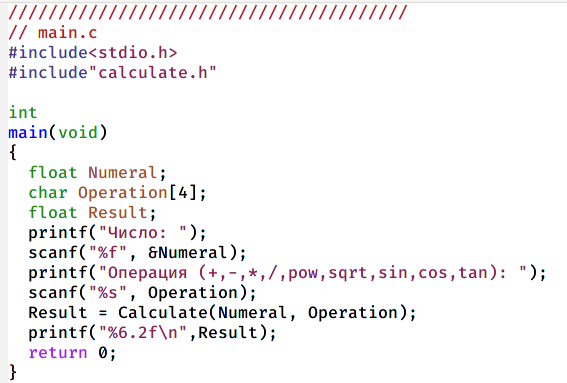


Figure 7: Исправленные ошибки

1. Создала Makefile с необходимым содержанием (рис. 8)

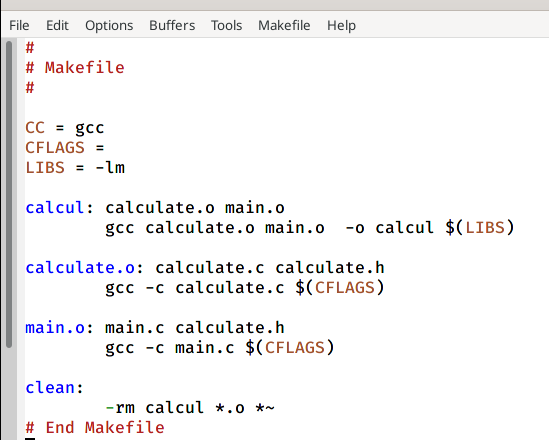


Figure 8: Makefile

Данный файл необходим для автоматической компиляции файлов calculate.c (цель calculate.o), main.c (цель main.o), а также их объединения в оди нисполняемый файл calcul (цель calcul). Цель clean нужна для автоматического удаления файлов. Переменная CC отвечает за утилиту для компиляции. Переменная CFLAGS отвечает за опции в данной утилите. Переменная LIBS отвечает за опции для объединения объектных файлов в один исполняемый файл.

1. Далее исправила Makefile.(рис. 9)

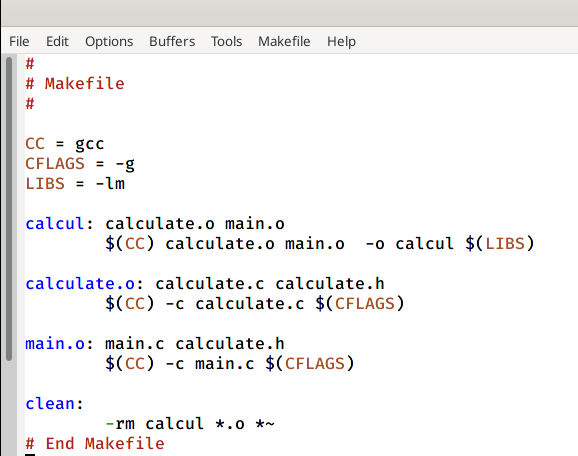


Figure 9: Исправленный Makefile

В переменную CFLAGS добавила опцию -g, необходимую для компиляции объектных файлов и их использования в программе отладчика GDB. Сделала так, что утилита компиляции выбирается с помощью переменной CC. После этого я удалила исполняемые и объектные файлы из каталога с помощью команды «make clean». Выполнила компиляцию файлов, используя команды «make calculate.o», «make main.o», «make calcul».(рис. 10)

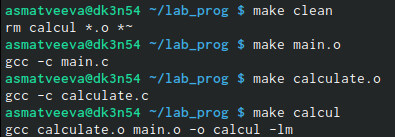


Figure 10: Компиляция файлов

Далее с помощью gdb выполнила отладку программы calcul. Запустила отладчик GDB, загрузив в него программу для отладки, используя команду:«gdb ./calcul». Для запуска программы внутри отладчика ввела команду «run». (рис. 11)

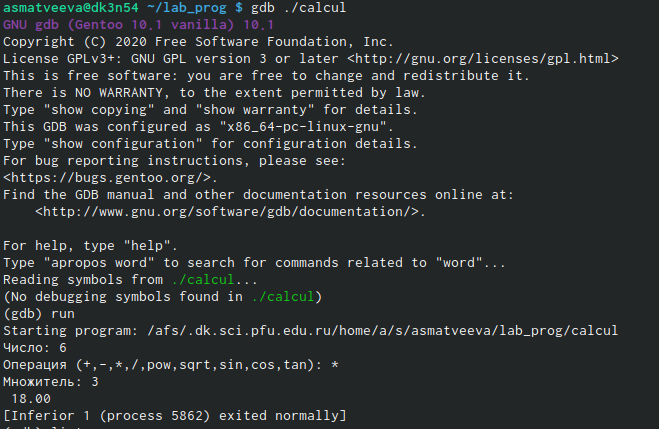


Figure 11: Запуск отладчика gdb и программы

Для постраничного (по 10 строк) просмотра исходного кода использовала команду «list».{ #fig:012 width=70% }

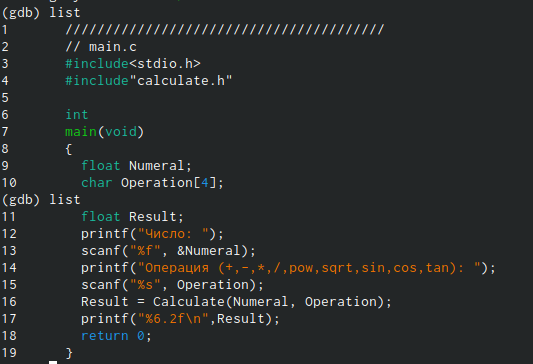


Figure 12: Команда list

Для просмотра строк с 12 по 15 основного файла использовала list с параметрами: “list 12,15”. (рис. 13)

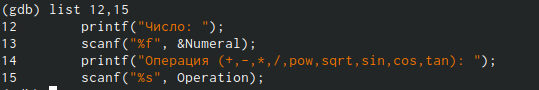


Figure 13: Команда list 12,15

Для просмотра определённых строк не основного файла использовала list с параметрами: “list calculate.c:12,20”. (рис. 14)

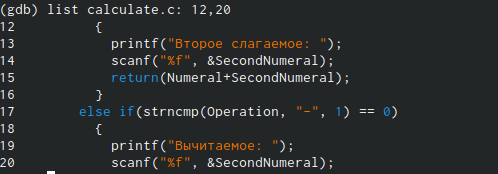


Figure 14: list calculate.c:12,20

Установила точку останова в файле calculate.c на строке номер 21:

"list calculate.c:20,27

break 21". (рис. 15)

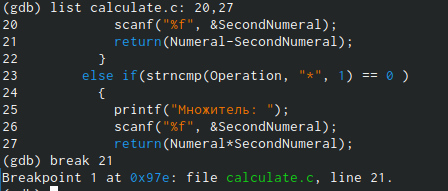


Figure 15: Установка точки останова

Вывела информацию об имеющихся в проекте точках останова: “info breakpoints”. (рис. 16)

Figure 16: Информация о точках останова

Figure 16: Информация о точках останова

Запустила программу внутри отладчика и убедилась, что программа остановится в момент прохождения точки останова:

"run

5

backtrace". (рис. 17)

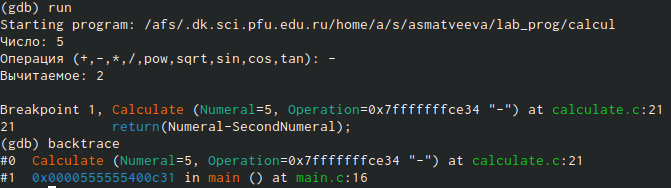


Figure 17: Запуск программы

Посмотрела, чему равно на этом этапе значение переменной Numeral, введя команду «print Numeral». Сравнила с результатом вывода на экран после использования команды «display Numeral». Значения совпадают. (рис. 18)

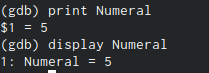


Figure 18: Значение переменной Numeral

Убрала точки останова с помощью команд «info breakpoints» и «delete 1». (рис. 19)

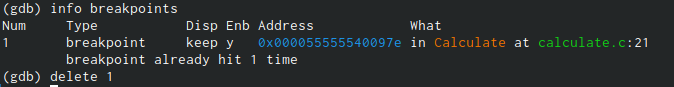


Figure 19: Убираем точки останова

С помощью утилиты splint проанализировала коды файлов calculate.c и main.c. Я воспользовалась командами «splint calculate.c» и «splint main.c». (рис. 20, 21)

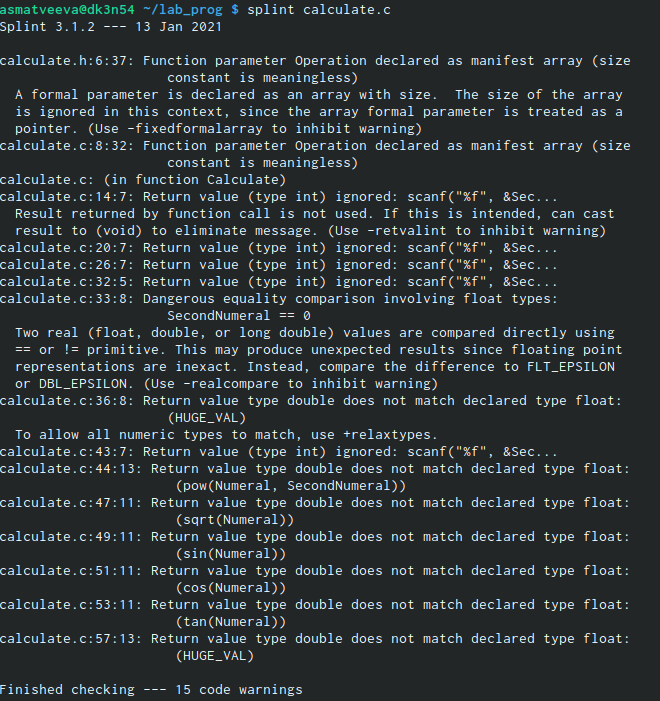


Figure 20: splint calculate.c

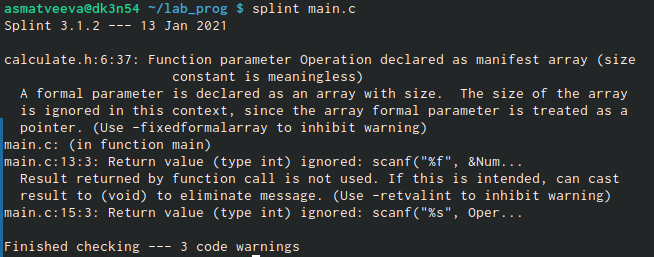


Figure 21: splint main.c

C помощью утилиты splint выяснилось, что в файлах calculate.c и main.c присутствует функция чтения scanf, возвращающая целое число (тип int), но эти числа не используются и нигде не сохранятся. Утилита вывела предупреждениео том, что в файле calculate.c происходит сравнение вещественного числа с нулем. Также возвращаемые значения (тип double) в функциях pow, sqrt, sin, cos и tan записываются в переменную типа float, что свидетельствует о потери данных.

# Выводы

В ходе выполнения данной лабораторной работы я приобрела простейшие навыки разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linuxна примере создания на языке программирования С калькулятора с простейшими функциями.

# Контрольные вопросы

1. Чтобы получить информацию о возможностях программ gcc, make, gdb и др. нужно воспользоваться командой man или опцией -help (-h) для каждой команды.
2. Процесс разработки программного обеспечения обычно разделяется на следующие этапы:

* планирование, включающее сбор и анализ требований к функционалу и другим характеристикам разрабатываемого приложения;
* проектирование, включающее в себя разработку базовых алгоритмов и спецификаций, определение языка программирования;
* непосредственная разработка приложения:

1. кодирование − по сути создание исходного текста программы (возможно в нескольких вариантах); –анализ разработанного кода;
2. сборка, компиляция и разработка исполняемого модуля;
3. тестирование и отладка, сохранение произведённых изменений;

* документирование.

Для создания исходного текста программы разработчик может воспользоваться любым удобным для него редактором текста: vi, vim, mceditor, emacs, geanyи др.

После завершения написания исходного кода программы (возможно состоящей из нескольких файлов), необходимо её скомпилировать и получить исполняемый модуль.

1. Для имени входного файла суффикс определяет какая компиляция требуется. Суффиксы указывают на тип объекта. Файлы с расширением (суффиксом) .c воспринимаются gcc как программы на языке С, файлы с расширением .cc или .C − как файлы на языке C++, а файлы c расширением .o считаются объектными. Например, в команде «gcc-c main.c»: gcc по расширению (суффиксу) .c распознает тип файла для компиляции и формирует объектный модуль − файл с расширением .o. Если требуется получить исполняемый файл с определённым именем (например, hello), то требуется воспользоваться опцией -o и в качестве параметра задать имя создаваемого файла: «gcc -o hello main.c».
2. Основное назначение компилятора языка Си в UNIX заключается в компиляции всей программы и получении исполняемого файла/модуля.
3. Для сборки разрабатываемого приложения и собственно компиляции полезно воспользоваться утилитой make. Она позволяет автоматизировать процесс преобразования файлов программы из одной формы в другую, отслеживает взаимосвязи между файлами.
4. Для работы с утилитой make необходимо в корне рабочего каталога с Вашим проектом создать файл с названием makefile или Makefile, в котором будут описаны правила обработки файлов Вашего программного комплекса.

В самом простом случае Makefile имеет следующий синтаксис:

… : …<команда 1>

…

Сначала задаётся список целей, разделённых пробелами, за которым идёт двоеточие и список зависимостей. Затем в следующих строках указываются команды. Строки с командами обязательно должны начинаться с табуляции.

В качестве цели в Makefile может выступать имя файла или название какого-то действия. Зависимость задаёт исходные параметры (условия) для достижения указанной цели. Зависимость также может быть названием какого-то действия. Команды − собственно действия, которые необходимо выполнить для достижения цели.

Общий синтаксис Makefileимеет вид:

target1 [target2…]:[:] [dependment1…]

[(tab)commands] [#commentary]

[(tab)commands] [#commentary]

Здесь знак # определяет начало комментария (содержимое от знака # и до конца строки не будет обрабатываться. Одинарное двоеточие указывает на то, что последовательность команд должна содержаться в одной строке. Для переноса можно в длинной строке команд можно использовать обратный слэш (). Двойное двоеточие указывает на то, что последовательность команд может содержаться в нескольких последовательных строках.

Пример более сложного синтаксиса Makefile:

#

# Makefile for abcd.c

#

CC = gcc

CFLAGS =

# Compile abcd.c normaly

abcd: abcd.c

$(CC) -o abcd $(CFLAGS) abcd.c

clean:

-rm abcd \*.o \*~

# End Make file for abcd.c

В этом примере в начале файла заданы три переменные: CC и CFLAGS. Затем указаны цели, их зависимости и соответствующие команды. В командах происходит обращение к значениям переменных. Цель с именем cleanпроизводит очистку каталога от файлов, полученных в результате компиляции. Для её описания использованы регулярные выражения.

1. Во время работы над кодом программы программист неизбежно сталкивается с появлением ошибок в ней. Использование отладчика для поиска и устранения ошибок в программе существенно облегчает жизнь программиста. В комплект программ GNU для ОС типа UNIX входит отладчик GDB (GNUDebugger).

Для использования GDB необходимо скомпилировать анализируемый код программы таким образом, чтобы отладочная информация содержалась в результирующем бинарном файле. Для этого следует воспользоваться опцией -gкомпилятора gcc:

gcc -c file.c -g

После этого для начала работы с gdb необходимо в командной строке ввести одноимённую команду, указав в качестве аргумента анализируемый бинарный файл:

gdb file.o

1. Основные команды отладчика gdb:

* backtrace − вывод на экран пути к текущей точке останова (по сути вывод − названий всех функций)
* break − установить точку останова (в качестве параметра может быть указан номер строки или название функции)
* clear − удалить все точки останова в функции
* continue − продолжить выполнение программы
* delete − удалить точку останова
* display − добавить выражение в список выражений, значения которых отображаются при достижении точки останова программы
* finish − выполнить программу до момента выхода из функции
* info breakpoints − вывести на экран список используемых точек останова
* info watchpoints − вывести на экран список используемых контрольных выражений
* list − вывести на экран исходный код (в качестве параметра может быть указано название файла и через двоеточие номера начальнойи конечной строк)
* next − выполнить программу пошагово, но без выполнения вызываемых в программе функций
* print − вывести значение указываемого в качестве параметра выражения
* run − запуск программы на выполнение
* set − установить новое значение переменной
* step − пошаговое выполнение программы
* watch − установить контрольное выражение, при изменении значения которого программа будет остановлена

Для выхода из gdb можно воспользоваться командой quit (или её сокращённым вариантом q) или комбинацией клавиш Ctrl-d. Более подробную информацию по работе с gdb можно получить с помощью команд gdb -h и man gdb.

1. Cхема отладки программы показана в 6 пункте лабораторной работы.
2. При запуске компилятор выдал ошибку в строке:

scanf(“%s”, &Operation);

нужно убрать знак &, потому что имя массива символов уже является указателем на первый элемент этого массива.

1. Система разработки приложений UNIX предоставляет различные средства, повышающие понимание исходного кода. К ним относятся:

* cscope − исследование функций, содержащихся в программе,
* lint − критическая проверка программ, написанных на языке Си.

1. Утилита splint анализирует программный код, проверяет корректность задания аргументов использованных в программе функций и типов возвращаемых значений, обнаруживает синтаксические и семантические ошибки.

В отличие от компилятора C анализатор splint генерирует комментарии с описанием разбора кода программы и осуществляет общий контроль, обнаруживая такие ошибки, как одинаковые объекты, определённые в разных файлах, или объекты, чьи значения не используются в работе программы, переменные с некорректно заданными значениямии типами и многое другое. # Библиография

1. https://esystem.rudn.ru/pluginfile.php/1142099/mod\_resource/content/2/011-lab\_prog.pdf
2. Кулябов Д.С. Операционные системы: лабораторные работы: учебное пособие / Д.С. Кулябов, М.Н. Геворкян, А.В. Королькова, А.В. Демидова. — М. : Изд-во РУДН, 2016. — 117 с. — ISBN 978-5-209-07626-1 : 139.13; То же [Электронный ресурс]. — URL: http://lib.rudn.ru/MegaPro2/Download/MObject/6118.
3. Робачевский А.М. Операционная система UNIХ [текст] : Учебное пособие / А.М. Робачевский, С.А. Немнюгин, О.Л. Стесик. — 2-е изд., перераб. и доп. — СПб. : БХВ-Петербург, 2005, 2010. — 656 с. : ил. — ISBN 5-94157-538-6 : 164.56. (ЕТ 60)
4. Таненбаум Эндрю. Современные операционные системы [Текст] / Э. Таненбаум. — 2-е изд. — СПб. : Питер, 2006. — 1038 с. : ил. — (Классика Computer Science). — ISBN 5-318-00299-4 : 446.05. (ЕТ 50)