Отчет по лабораторной работе №14

Дисциплина

Филиппова Анна Дмитриевна

Содержание

# Цель работы

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

# Выполнение лабораторной работы

1. В домашнем каталоге создаем подкаталог ~/work/os/lab\_prog с помощью команды «mkdir -p ~/work/os/lab\_prog». (рис. -fig. 1)

Figure 1: Создаем подкаталог

Figure 1: Создаем подкаталог

1. Создаем в каталоге файлы: calculate.h, calculate.c, main.c, используя команды «cd ~/work/os/lab\_prog» и «touch calculate.h calculate.c main.c». 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. (рис. -fig. 2)

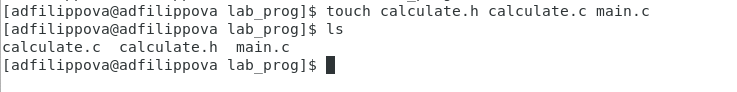


Figure 2: Создаем файлы

Открыв редактор Emacs, приступаем к редактированию созданных файлов. Реализация функций калькулятора в файле calculate.с (рис. -fig. 3) (рис. -fig. 4) (рис. -fig. 5) (рис. -fig. 6)

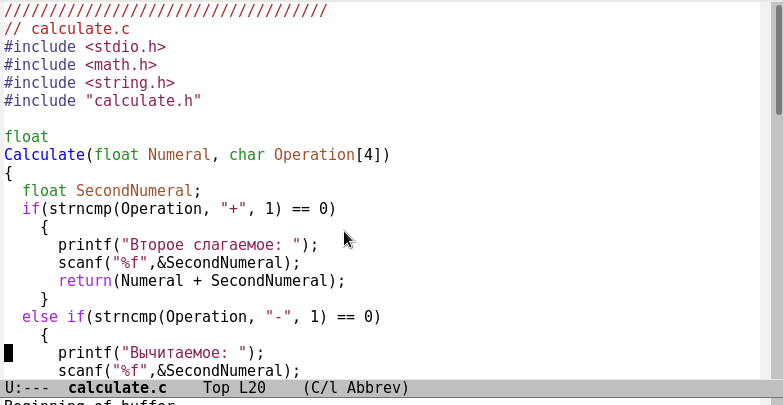


Figure 3: Пишем командные файлы

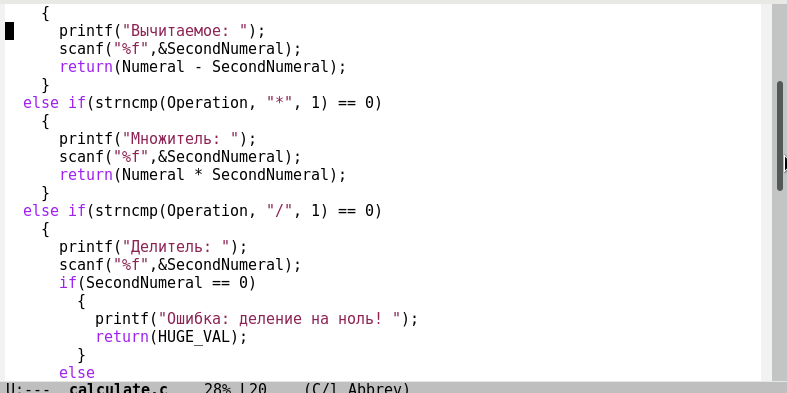


Figure 4: Пишем командные файлы

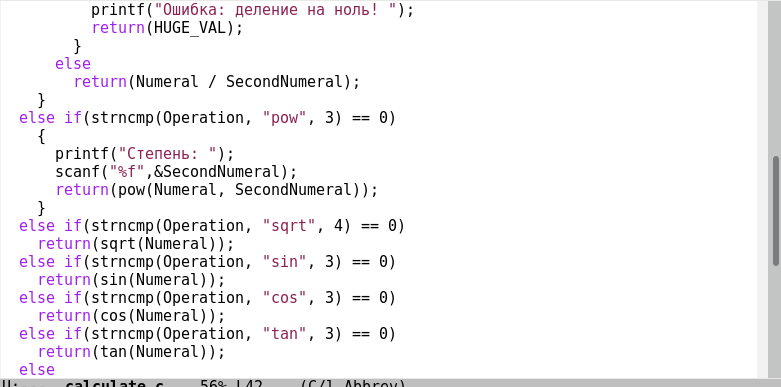


Figure 5: Пишем командные файлы

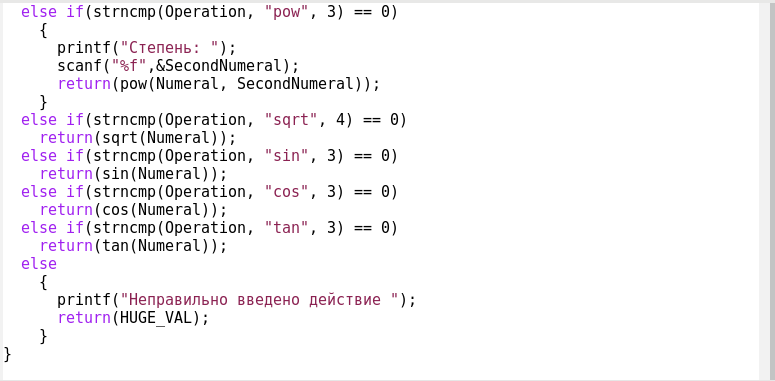


Figure 6: Пишем командные файлы

Интерфейсный файл calculate.h, описывающий формат вызова функции калькулятора. (рис. -fig. 7)

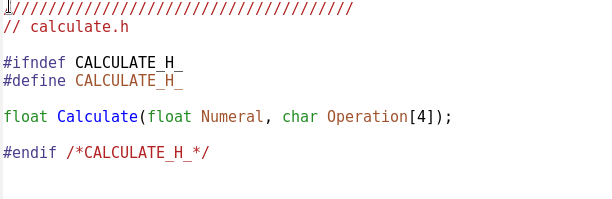


Figure 7: Пишем командные файлы

Основной файл main.c, реализующий интерфейс пользователя к калькулятору. (рис. -fig. 8)

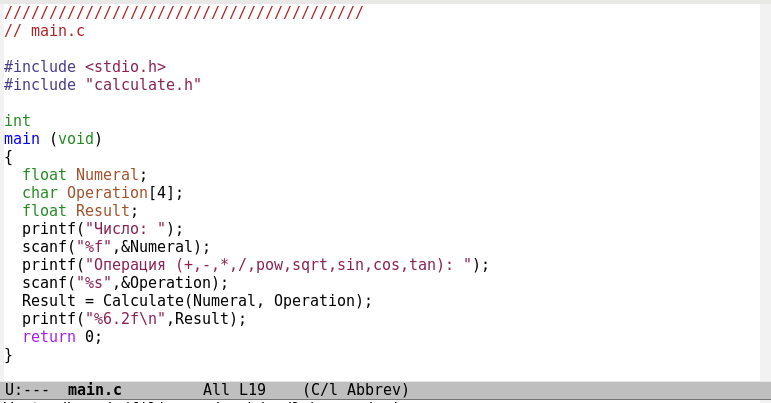


Figure 8: Пишем командные файлы

1. Выполняем компиляцию программы посредством gcc, используя команды «gcc -c calculate.c», «gcc -c main.c» и «gcc calculate.o main.o -o calcul -lm». (рис. -fig. 9)

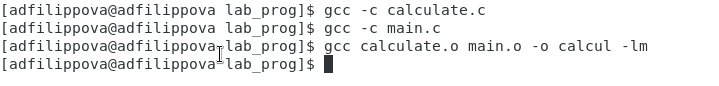


Figure 9: Компиляция программы

1. Ошибок не возникло.
2. Создаем Makefile с нужным содержанием. Данный файл необходим для автоматической компиляции файлов calculate.c (цель calculate.o), main.c (цель main.o), а также их объединения в один исполняемый 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. (рис. -fig. 10) (рис. -fig. 11)

Figure 10: Создание файла

Figure 10: Создание файла

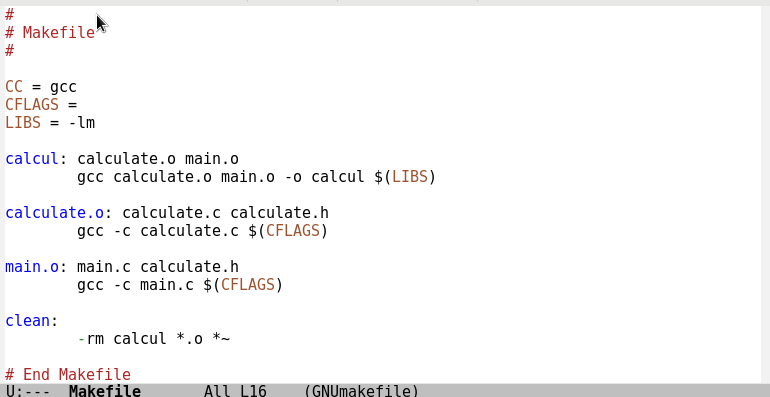


Figure 11: Пишем Makefile

1. Исправляем Makefile перед использованием gdb. В переменную CFLAGS добавляем опцию -g, необходимую для компиляции объектных файлов и их использования в программе отладчика GDB. Делаем так, что утилита компиляции выбирается с помощью переменной CC. (рис. -fig. 12)

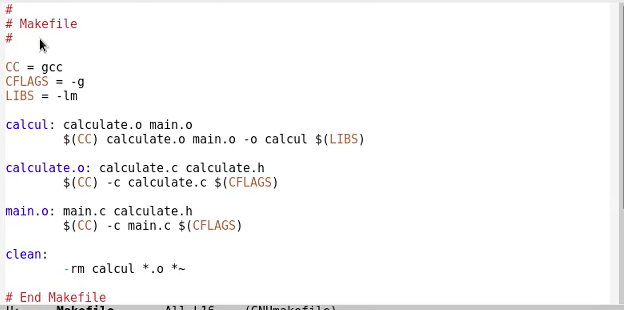


Figure 12: Исправляем Makefile

После этого удаляем исполняемые и объектные файлы из каталога с помощью команды «make clear». Выполняем компиляцию файлов, используя команды «make calculate.o», «make main.o», «male calcul». (рис. -fig. 13)

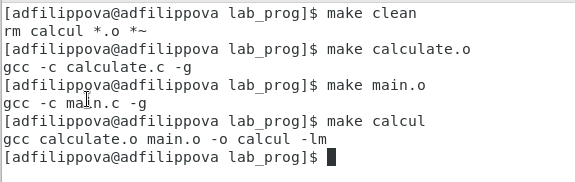


Figure 13: Компиляция файлов

Далее с помощью gdb выполняем отладку программы calcul. Запускаем отладчик GDB, загрузив в него программу для отладки, используя команду: «gdb ./calcul». (рис. -fig. 14)

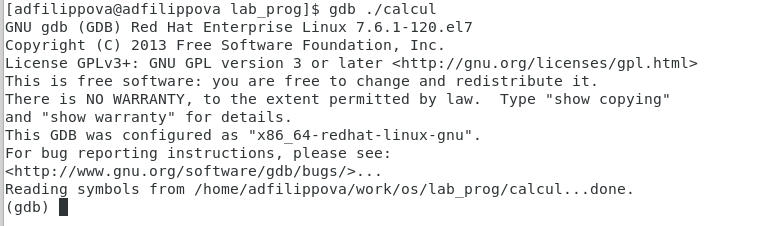


Figure 14: Запуск отладчика

Для запуска программы внутри отладчика вводим команду «run». (рис. -fig. 15)

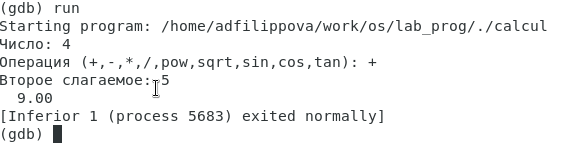


Figure 15: run

Для постраничного (по 9 строк) просмотра исходного кода используем команду «list». (рис. -fig. 16)

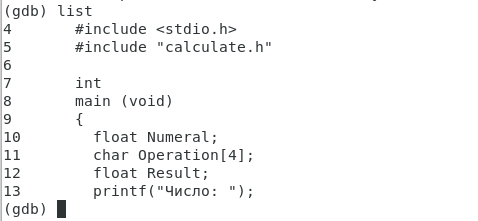


Figure 16: list

Для просмотра строк с 12 по 15 основного файла используем команду «list 12,15». (рис. -fig. 17)

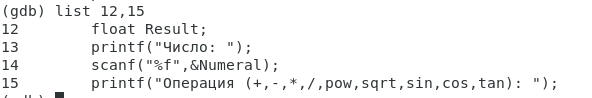


Figure 17: list 12,15

Для просмотра определённых строк не основного файла используем команду «list calculate.c:20,29». (рис. -fig. 18)

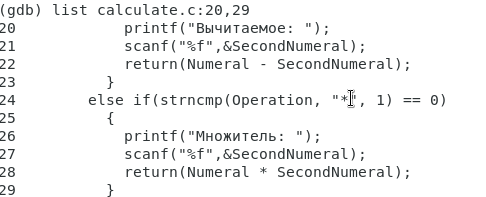


Figure 18: list calculate.c:20,29

Устанавливаем точку останова в файле calculate.c на строке номер 21,используя команды «list calculate.c:20,27» и «break 21». (рис. -fig. 19)

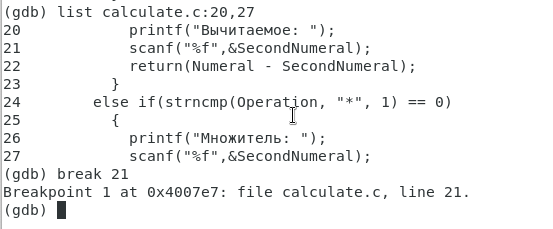


Figure 19: Точка останова

Выводим информацию об имеющихся в проекте точках останова с помощью команды «info breakpoints». (рис. -fig. 20)

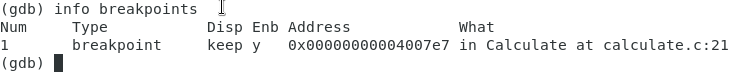


Figure 20: Информация о точках останова

Запускаем программу внутри отладчика и убедилась, что программа остановилась в момент прохождения точки останова. Использовала команды «run», «5», «−» и «backtrace». (рис. -fig. 21)

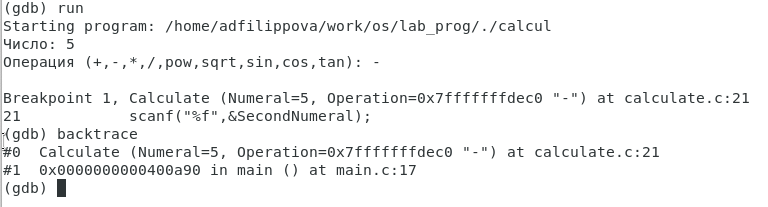


Figure 21: Проверка точки останова

Посматриваем, чему равно на этом этапе значение переменной Numeral,введя команду «print Numeral». (рис. -fig. 22)

Figure 22: Информация о точках останова

Figure 22: Информация о точках останова

Сравниваем с результатом вывода на экран после использования команды «display Numeral». Значения совпадают. (рис. -fig. 23)

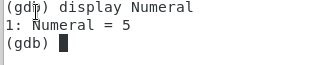


Figure 23: Информация о точках останова

Убираем точки останова с помощью команд «info breakpoints» и «delete 1». (рис. -fig. 24)

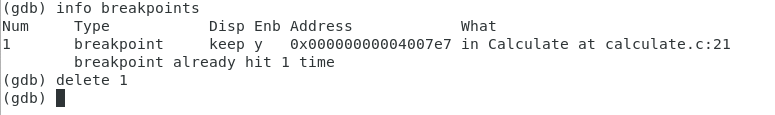


Figure 24: Убираем точку оастанова

1. С помощью утилиты splint анализируем коды файлов calculate.c и main.c. Предварительно устанавливаем данную утилиту с помощью команды «yum install splint». Далее используем команду «splint calculate.c» и «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 предупреждение о 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. (рис. -fig. 25) (рис. -fig. 26) (рис. -fig. 27)

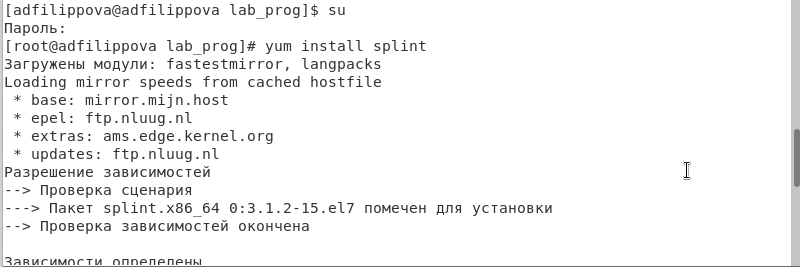


Figure 25: Установка splint

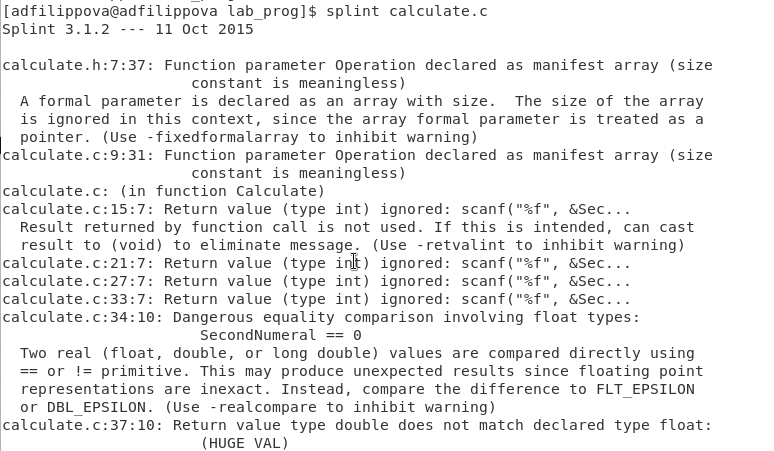


Figure 26: Splint

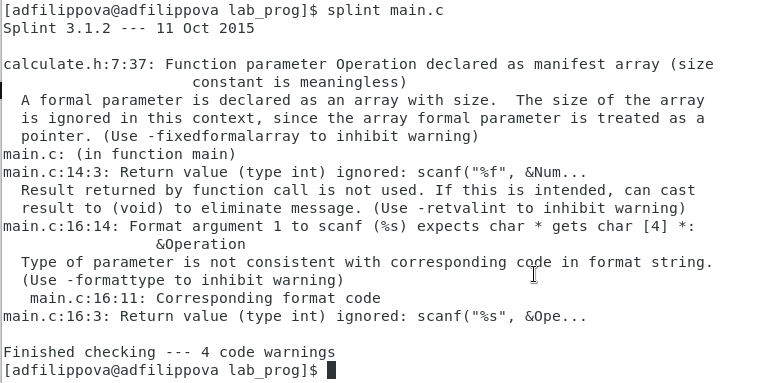


Figure 27: Splint

# Выводы

Я приобрела простейшие навыки разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linux на 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 кфункционалу и другим характеристикам разрабатываемого приложения;
* проектирование, включающее в себя разработку базовых алгоритмов и спецификаций, определение языка программирования;
* непосредственная разработка приложения: o кодирование − по сути создание исходного текста программы (возможно в нескольких вариантах); – анализ разработанного кода; o сборка, компиляция и разработка исполняемого модуля; o 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, geany и 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 вкомпиляции 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 Makefile имеет 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 водной 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 Makefile 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 clean производит очистку каталога от файлов, полученных в результате компиляции. Для её описания использованы регулярные выражения.
5. 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 (GNU Debugger).Для использования GDB необходимо скомпилировать анализируемый код программы таким образом, чтобы отладочная информация содержалась в результирующем бинарном файле. Для этого следует воспользоваться опцией -g компилятора 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
6. 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 начальной и конечной строк)
* next − выполнить программу пошагово, но без выполнениявызываемых 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 первом запуске компилятор не выдал никаких ошибок, но в коде программы main.c допущена ошибка, которую компилятор мог пропустить (возможно, из-за версии 8.3.0-19): в строке scanf(“%s”, &Operation); нужно убрать знак &, потому что имя массива символов уже является указателем на первый элемент этого массива.
3. 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 работепрограммы, переменные с некорректно заданными значениями и типами и многое другое.

# Библиография

1. 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.
2. 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)
3. 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)