**Отчёт по лабораторной работе №14**

## Дисциплина: Операционные системы

**Джеффри Родригес Сантос**

**Содержание**

1. [Цель работы 5](#_TOC_250004)
2. [Задание 6](#_TOC_250003)
3. [Выполнение лабораторной работы 8](#_TOC_250002)
4. [Контрольные вопросы 20](#_TOC_250001)
5. [Выводы 25](#_TOC_250000)

**Список таблиц**

# Цель работы

### Приобрести простейшие навыки разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linux на примере создания на языке программиро- вания с калькулятора с простейшими

### функциями.

1. Задание
2. В домашнем каталоге создайте подкаталог ~/work/os/lab\_prog.
3. Создайте в нём файлы: calculate.h, calculate.c, main.c. Это будет примитив- нейший 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.
4. Выполните компиляцию программы посредством gcc: gcc

-c calculate.c

gcc -c main.c

gcc calculate.o main.o -o calcul -lm

1. При необходимости исправьте синтаксические ошибки.
2. Создайте Makefile. Поясните в отчёте его содержание.
3. С помощью gdb выполните отладку программы calcul (перед использовани- ем gdb исправьте Makefile ):
   * Запустите отладчик GDB, загрузив в него программу для отладки
   * Для запуска программы внутри отладчика введите команду run
   * Для постраничного (по 10 строк) просмотра исходного код используйте команду list
   * Для просмотра строк с 12 по 15 основного файла используйте list с парамет-

рами

* + Для просмотра определённых строк не основного файла используйте list с параметрами
  + Установите точку останова в файле calculate.c на строке номер 21
  + Выведите информацию об имеющихся в проекте точка останова
  + Запустите программу внутри отладчика и убедитесь, что программа оста- новится в момент прохождения точкиостанова
  + Отладчик выдаст информацию, а команда backtrace покажет весь стек вы- зываемых функций от начала программы до текущего места
  + Посмотрите, чему равно на этом этапе значение переменной Numeral. На экран должно бытьвыведено число 5
  + Сравните с результатом вывода на экран после использования команды
  + Уберите точки останова

1. С помощью утилиты splint попробуйте проанализировать коды файлов calculate.c и main.c.
2. Выполнениелабораторнойработы
3. В домашнем каталоге создаю подкаталог calculate с помощью команды«mkdir calculate».
4. Создал в каталоге файлы: calculate.h, calculate.c, main.c, используя команды«cd calculate» и «touch calculate.h calculate.c main.c» (рис. -fig. 3.1).

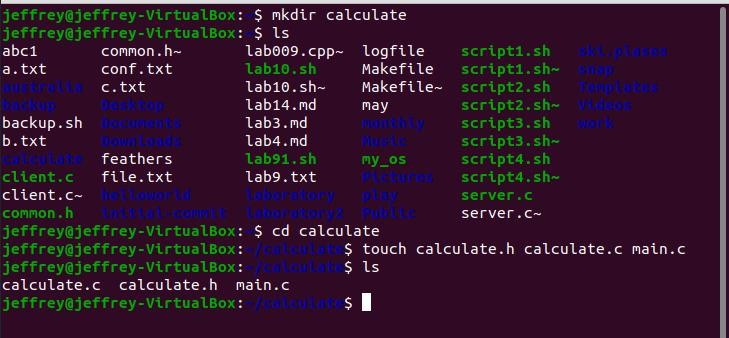
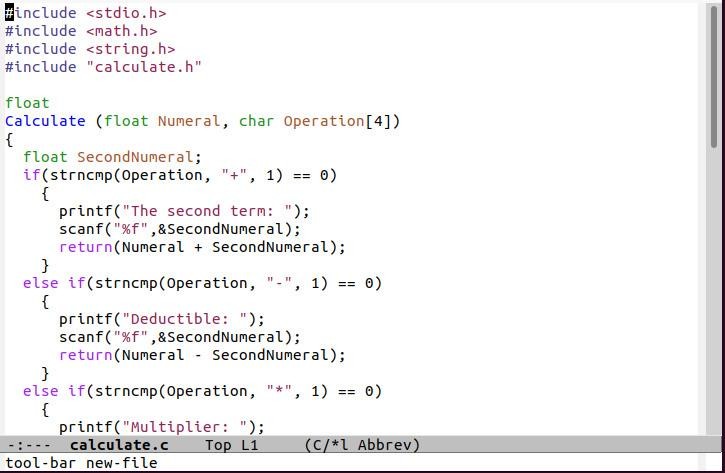


Рис. 3.1: Создал каталог и файлы в нём

Это будет примитивнейший калькулятор, способный складывать, вычитать, умножать и делить, возводить число в степень, брать квадратный корень, вы- числять sin, cos, tan. При запуске он будет запрашивать первое число, операцию, второе число. После этого программа выведет результат и остановится.

Открыв редактор Emacs, приступил к редактированию созданных файлов.

Реализация функций калькулятора в файле calculate.с (рис. -fig. 3.2) (рис. -fig. 3.3).



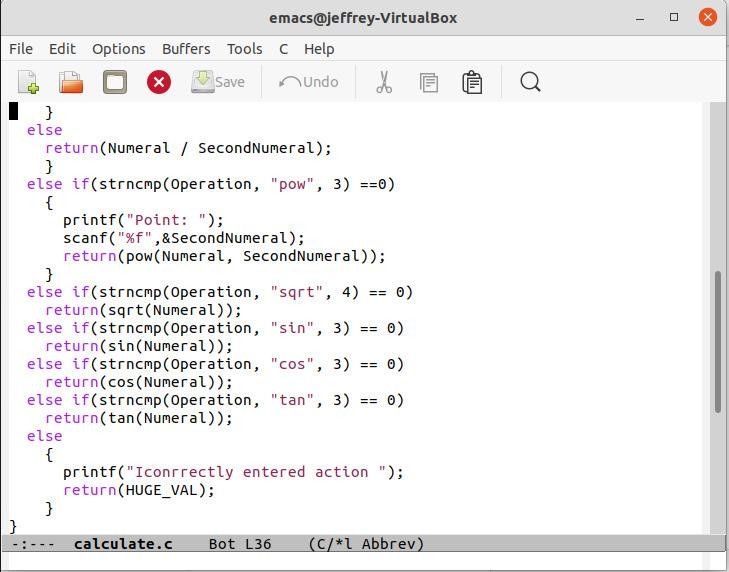


Рис. 3.2: Реализация функций калькулятора в файле calculate.с

Интерфейсный файл calculate.h, описывающий формат вызова функциикаль- кулятора (рис. -fig. 3.4).

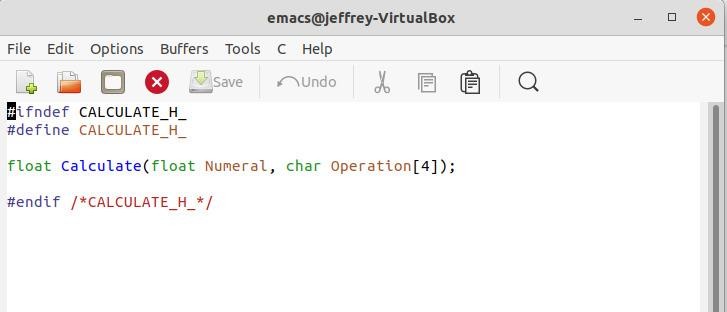


Рис. 3.4: Интерфейсный файл calculate.h

Основной файл main.c, реализующий интерфейс пользователя к калькулятору (рис. -fig. 3.5).

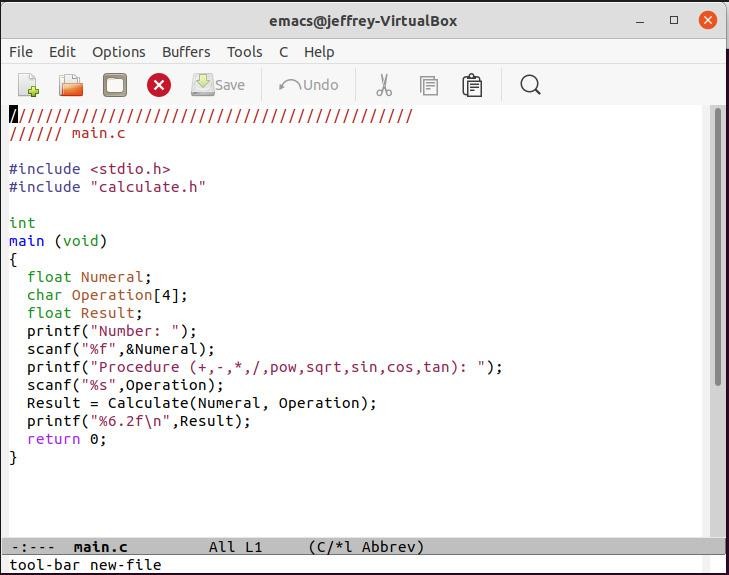


Рис. 3.5: Основной файл main.c

1. Выполнил 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. 3.6).

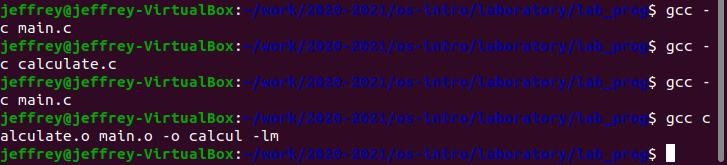


Рис. 3.6: Выполнил компиляцию программы посредством gcc

1. В ходе компиляции программы никаких ошибок выявлено не было. Создал Makefile с необходимым содержанием (рис. -fig.
2. Данный файл необходим для автоматической компиляции файлов

calculate.c (цель calculate.o), main.c (цель main.o), а также их объединения в один испол- няемый файл calcul (цель calcul). Цель clean нужна для

автоматического удаления файлов. Переменная CC отвечает за утилиту для компиляции. Переменная CFLAGSотвечаетзаопциивданной утилите.

Переменная LIBS отвечает за опции для объединения объектных файлов в один исполняемый файл.

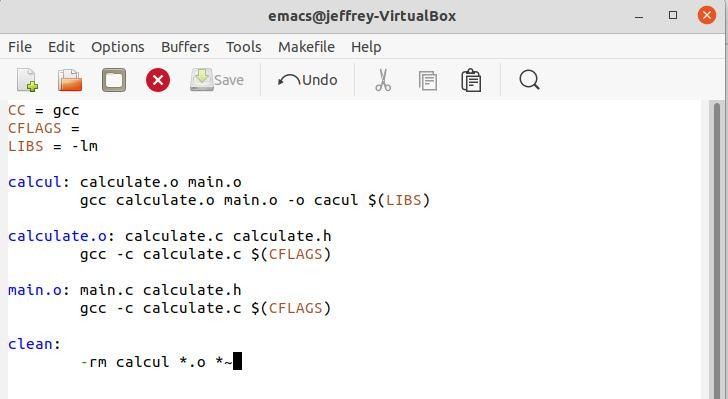


Рис. 3.7: Создал Makefile с необходимым содержанием

1. Далее исправил Makefile (рис. -fig. 3.8). В переменную CFLAGS добавил опцию -g, необходимую для компиляции объектных файлов и их использо- вания в программе отладчика GDB. Сделал так, что утилита компиляции выбирается с помощью переменной CC.

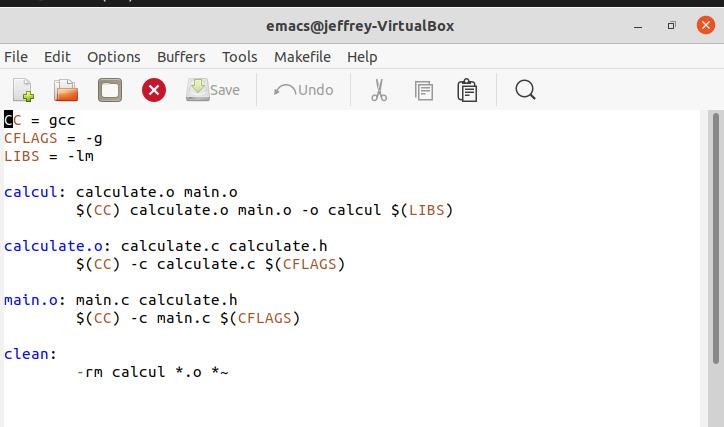


Рис. 3.8: далее исправил Makefile

После этого я удалил исполняемые и объектные файлы из каталога с помощью команды«makeclean».Выполнилкомпиляциюфайлов,используякоманды«make calculate.o», «make main.o», «male calcul» (рис.fig. 3.9).



Рис. 3.9: Используем команды make

Далее с помощью gdb выполнил отладку программы calcul. Запустил отладчик GDB, загрузив в него программу для отладки, используя команду: «gdb ./calcul» (рис. -fig. 3.10).

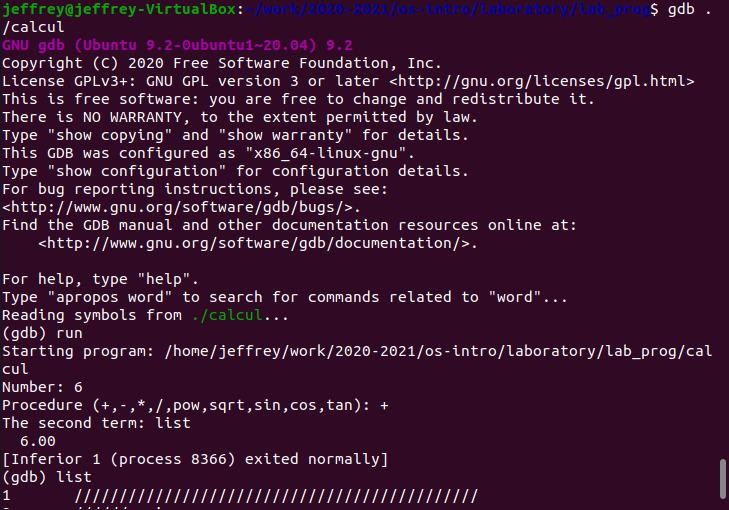


Рис. 3.10: Запустил отладчик GDB

Для запуска программы внутри отладчика ввёл команду «run» (рис. -fig. 3.11).

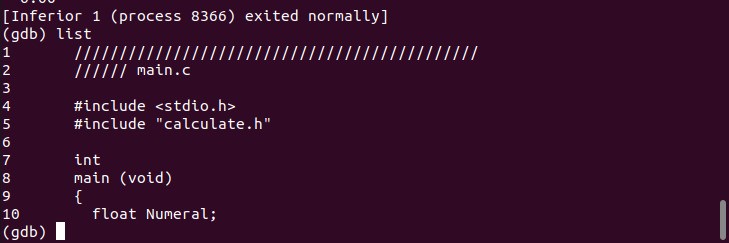


Рис.

3.12: Использовал команду «list

Для просмотра строк с 12 по 15 основного файла использовал команду «list 12,15» (рис. -fig. 3.13).

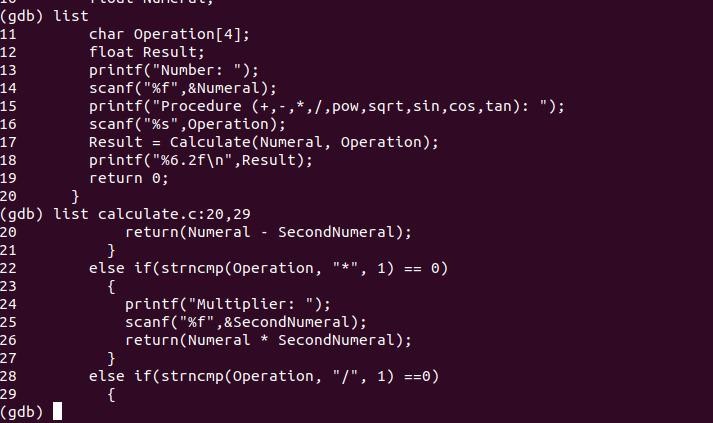


Рис. 3.13: Просмотр строк с 12 по 15

Для просмотра определённых строк не основного файла использовал команду«list calculate.c:20,29» (рис. -fig. 3.14).

Рис. 3.14: Просмотр определённых строк не основного файла

Установил точку останова в файле calculate.c на строке номер 18, используя команды «list calculate.c:15,22» и «break 18» (рис. -fig. 3.15).

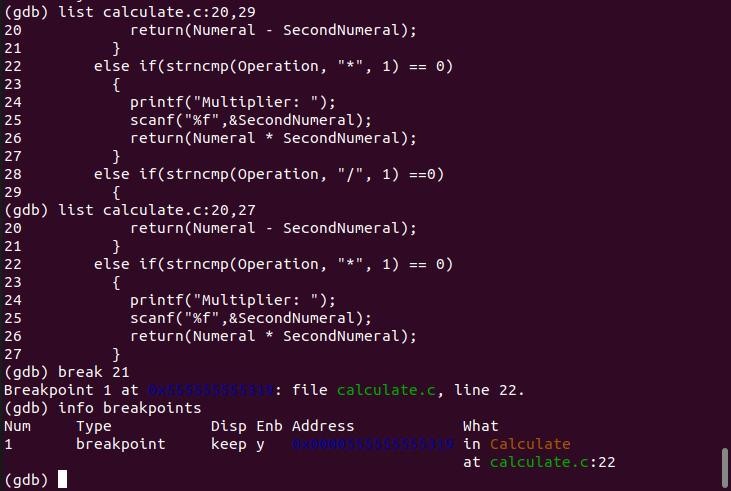


Рис. 3.14: Просмотр определённых строк не основного файла

Установил точку останова в файле calculate.c на строке номер 18, используя команды «list calculate.c:15,22» и «break 18» (рис. -fig. 3.15).

Вывел информацию об имеющихся в проекте точках останова с помощью команды «info breakpoints» (рис.-fig. 3.16).

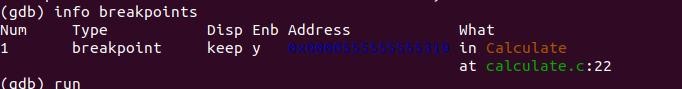


Рис. 3.16: Вывел информацию об имеющихся в проекте точках останова

### Запустил программу внутри отладчика и убедился, что программа остановилась в момент прохождения точки останова. Использовал команды «run»,«5», «−» и «backtrace» (рис. -fig. 3.17).

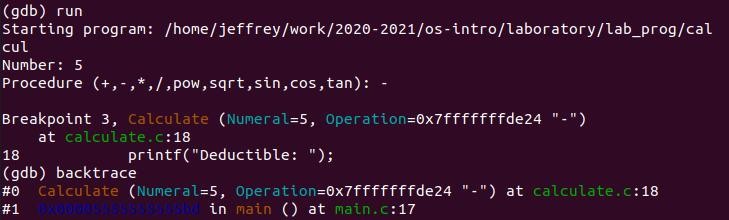


Рис. 3.17: Запустил программу внутри отладчика до точки останова Посмотрел, чему равно на этом этапе значение переменной Numeral, введя

команду «print Numeral» (рис. -fig. 3.18).



Рис. 3.18: Посмотрел, чему равно Numeral

Сравнил с результатом вывода на экран после использования команды«display Numeral». Значения совпадают (рис. -fig. 3.19).



Рис. 3.19: Сравнил с результатом вывода на экран

Убрал точку останова с помощью команд «info breakpoints» и «delete 3» (рис.-fig. 3.20).



Рис. 3.20: Убрал точку останова

1. 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 разрабатываемогоприложения;
* проектирование, включающее в себя разработку базовых алгоритмов и спецификаций, определение языкапрограммирования;
* непосредственная разработка приложения:
* кодирование − по сути создание исходного текста программы (возможно в нескольких вариантах);
* анализ разработанного кода;
* сборка, компиляция и разработка исполняемого модуля;
* тестирование и отладка, сохранение произведённых изменений;
* документирование.

Для создания исходного текста программы разработчик может воспользо- ваться любым удобным для него редактором текста: vi, vim, mceditor, emacs, geany и др.

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

* 1. Для имени входного файла суффикс определяет какая компиляция требует- ся.Суффиксыуказываютнатипобъекта.Файлысрасширением(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 (суф- фиксу) .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 воспользо- ваться опцией -o и в качестве параметра задать имя создаваемого файла:

«gcc -o hello main.c».

* 1. Основное назначение компилятора языка Си в UNIX заключается вкомпи- ляции всей программы и получении исполняемого файла/модуля.
  2. Длясборкиразрабатываемого приложенияи собственно компиляции по- лезно воспользоваться утилитой make. Она позволяет автоматизировать процесс преобразования файлов программы из одной формы в другую, отслеживает взаимосвязи междуфайлами.
  3. 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>

…

Сначалазадаётсясписокцелей, разделённыхпробелами, закоторым идёт двоеточие и список зависимостей. Затем в следующих строках указыва- ются 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 достиженияцели.

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.

* 1. Во время работы над кодом программы программист неизбежно сталкива- ется с появлением ошибок в ней. Использование отладчика для поиска и устранения ошибок в программе существенно облегчает жизнь програм- миста. 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 компилятораgcc:

gcc -c file.c -g

После этого дляначалаработысgdbнеобходимо вкомандной строке вве- сти 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 отоб- ражаются при достижении точки остановапрограммы
* 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 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 ко- торого программа будет остановлена

Для выхода из gdb можно воспользоваться командой quit (или её сокра- щённым 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 про- граммы 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.
  4. Утилита splint анализирует программный код, проверяет корректность задания аргументов использованных в программе функций и типов возвра- щаемых значений, обнаруживает синтаксические и семантические ошибки. В отличие от компилятора C анализатор splint генерирует комментарии с описанием разбора кода программы и осуществляет общий контроль, обна- руживая такие ошибки, как одинаковые объекты, определённые в разных файлах, или объекты, чьи значения неиспользуются в работепрограммы, переменные с некорректно заданными значениями и типами и многое другое.

1. Выводы

В ходе выполнения данной лабораторной работы я приобрёл простейшие навыки разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linux на примере создания на языке программирования С калькулятора с простейши- ми функциями.