Лабораторная работа №14

Дисциплина: Операционные системы

Коновалова Татьяна Борисовна

Содержание

# Цель работы

Цель данной лабораторной 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.

# Задание

1. Сделать отчёт по лабораторной работе №14 в формате Markdown.
2. создать на языке программирования С калькулятора с простейшими функциями.

# Выполнение лабораторной работы

1). В домашнем каталоге создаю подкаталог ~/work/os/lab\_progс помощью команды «mkdir-p~/work/os/lab\_prog» (Рисунок 1 ). Вся необходимая информация про создания каталогов указана в следующем источнике: Программное обеспечение GNU/Linux. Лекция 9. Хранилище и дистрибутив (Г. Курячий, МГУ).

Figure 1: Создание подкаталога

Figure 1: Создание подкаталога

2). 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» (алгоритм действий представлен на рис. 2 ).

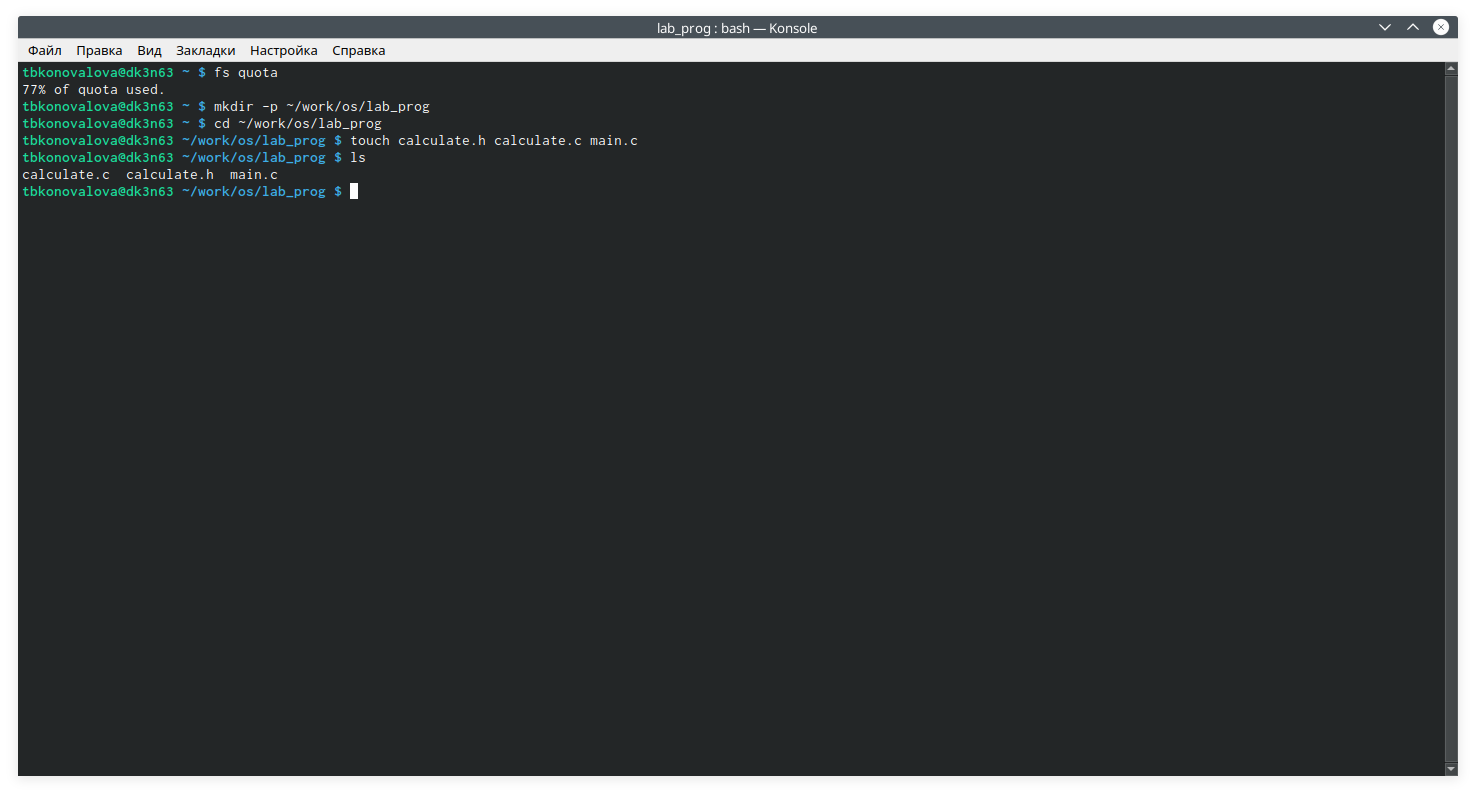


Figure 2: Создание файлов

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 (Программа представлена на рис. 3 , 4 ). Вся необходимая информация про написания программ указана в следующем источнике: Программное обеспечение GNU/Linux. Лекция 10. Минимальный набор знаний (Г. Курячий, МГУ).

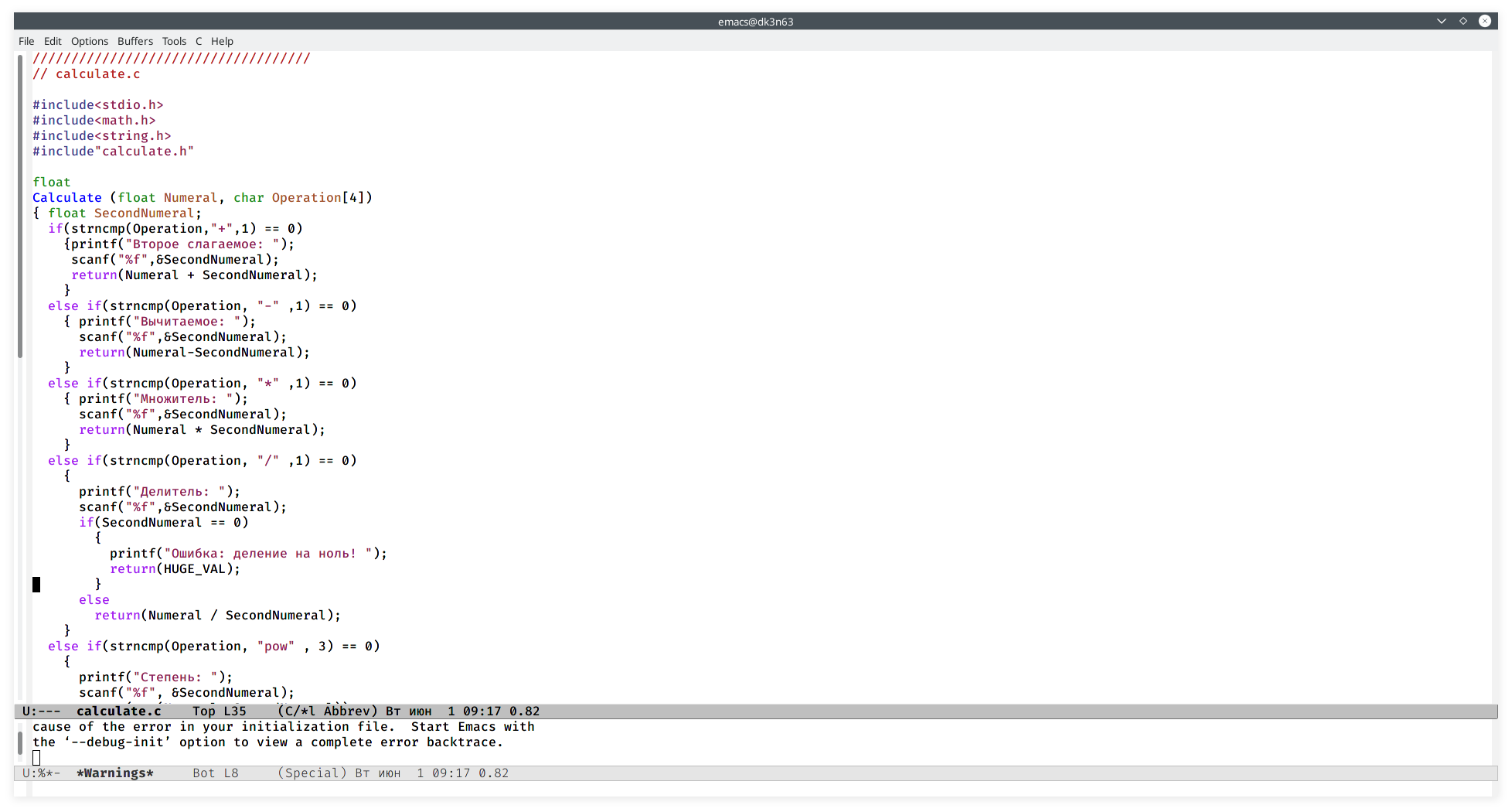


Figure 3: Программа в calculate.c

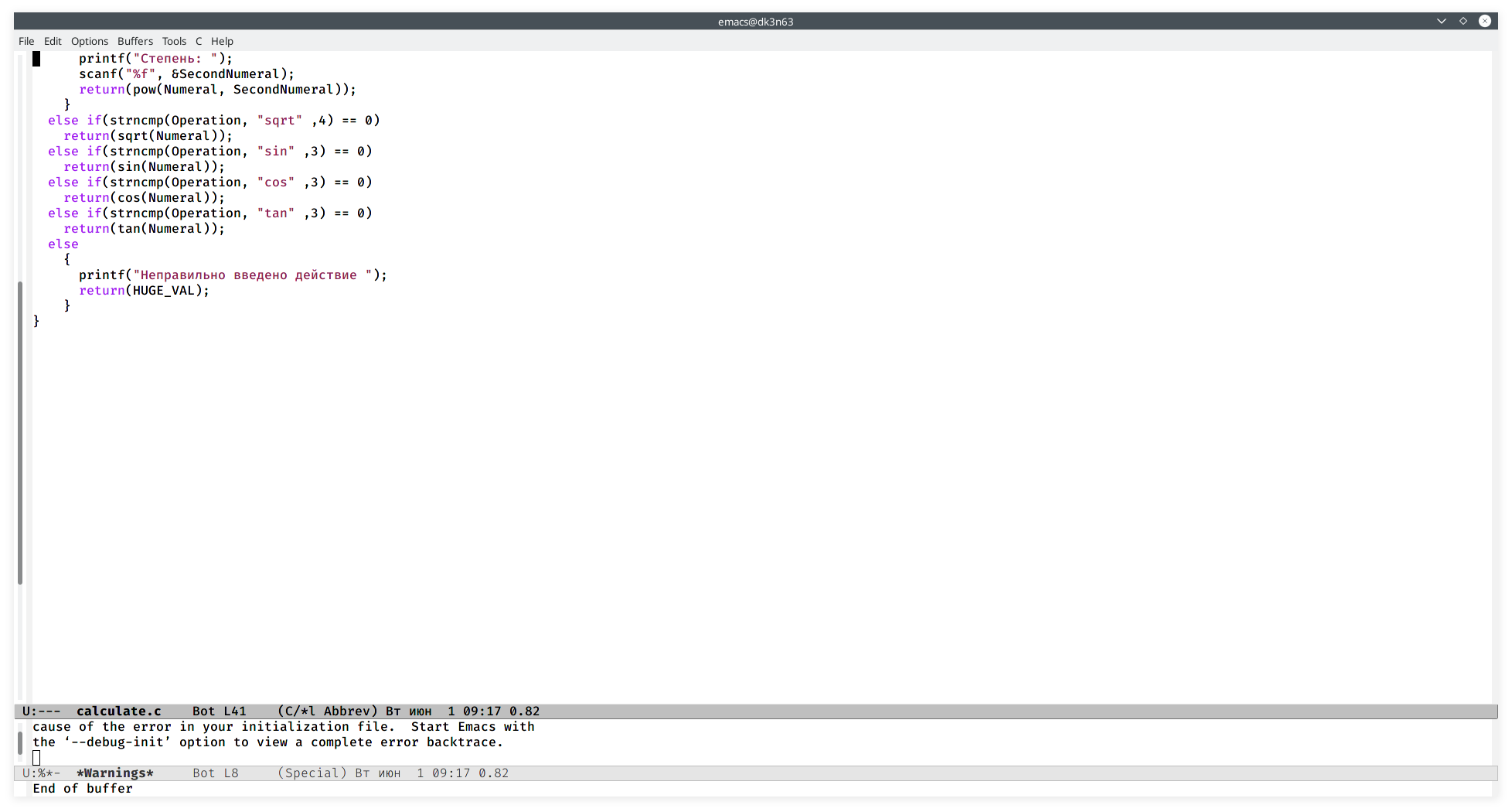


Figure 4: Программа в calculate.c

Интерфейсный файл calculate.h, описывающий формат вызова функции калькулятора (Скриншот 5 ). Подробная информация о написании программ в Linux указана в следующем источнике: Электронный ресурс: Электронный ресурс: https://it.wikireading.ru/34160

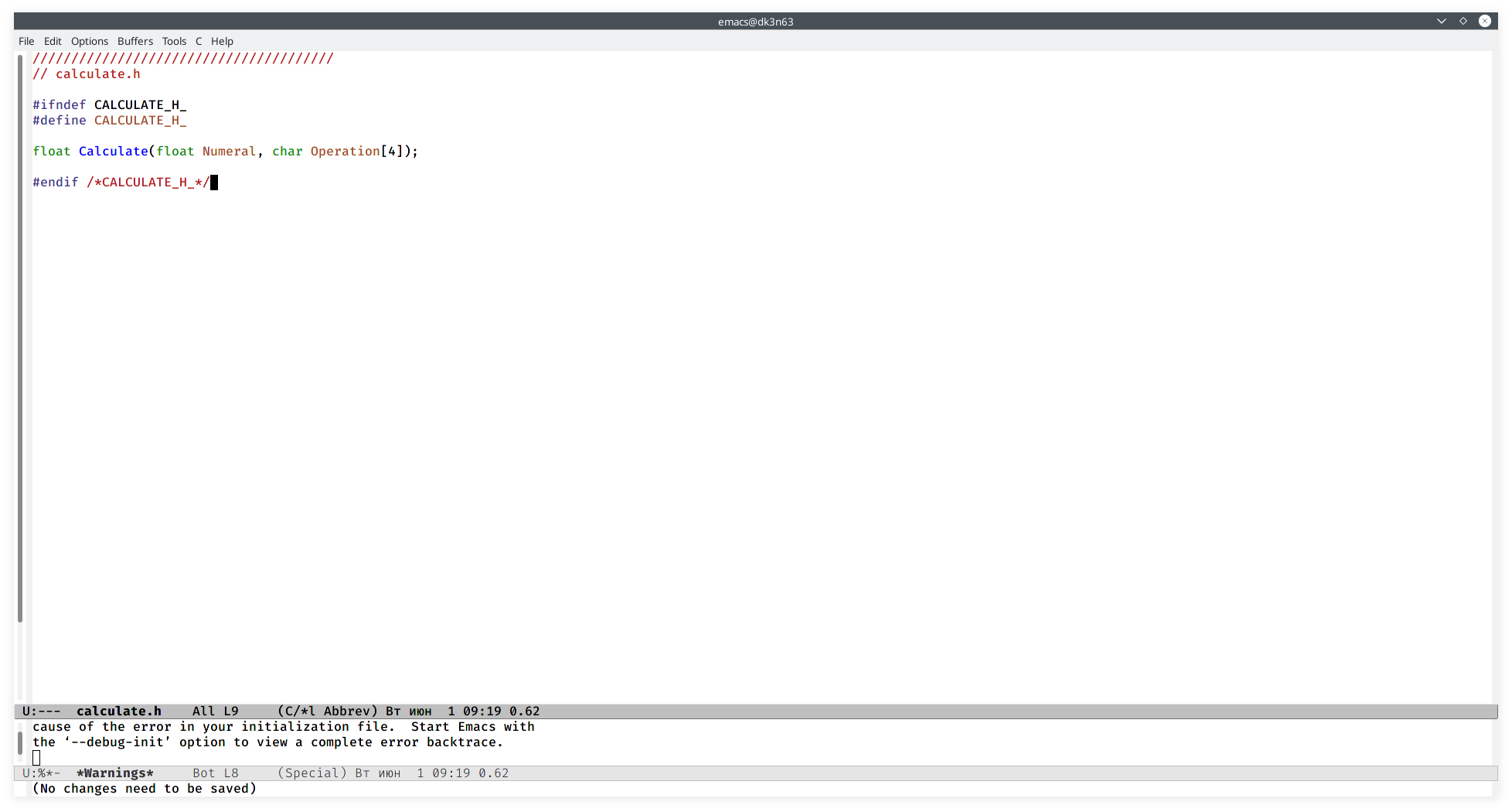


Figure 5: Программа в calculate.h

Основной файл main.c, реализующий интерфейс пользователя к калькулятору (Скриншот 6 ).

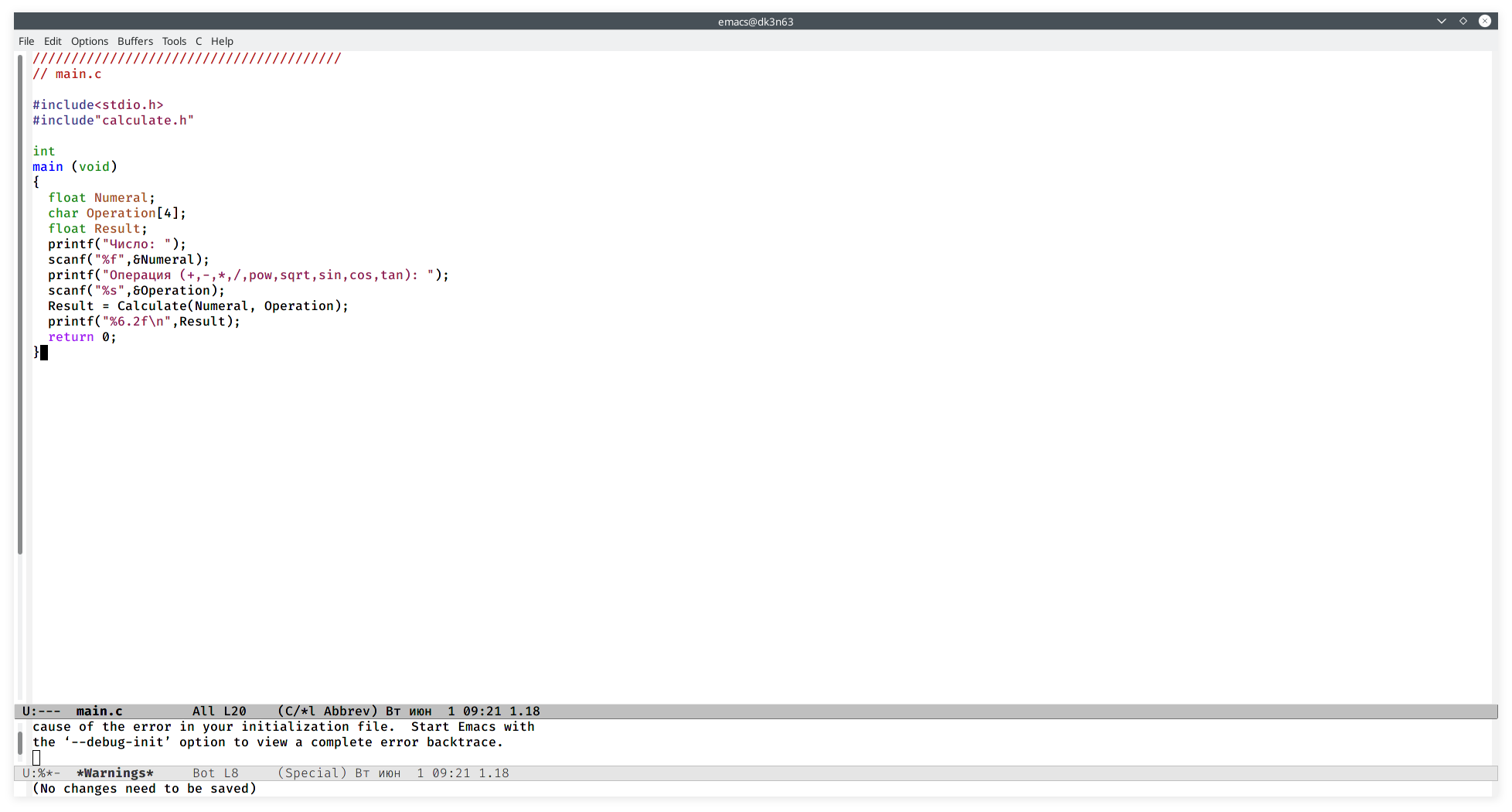


Figure 6: Программа в main.c

3). 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» (алгоритм действий представлен на рис. 7 ).

Figure 7: Компиляция программы

Figure 7: Компиляция программы

4). В ходе компиляции программы никаких ошибок выявлено не было.

5). Создала Makefile с необходимым содержанием (Рисунок 8 ).

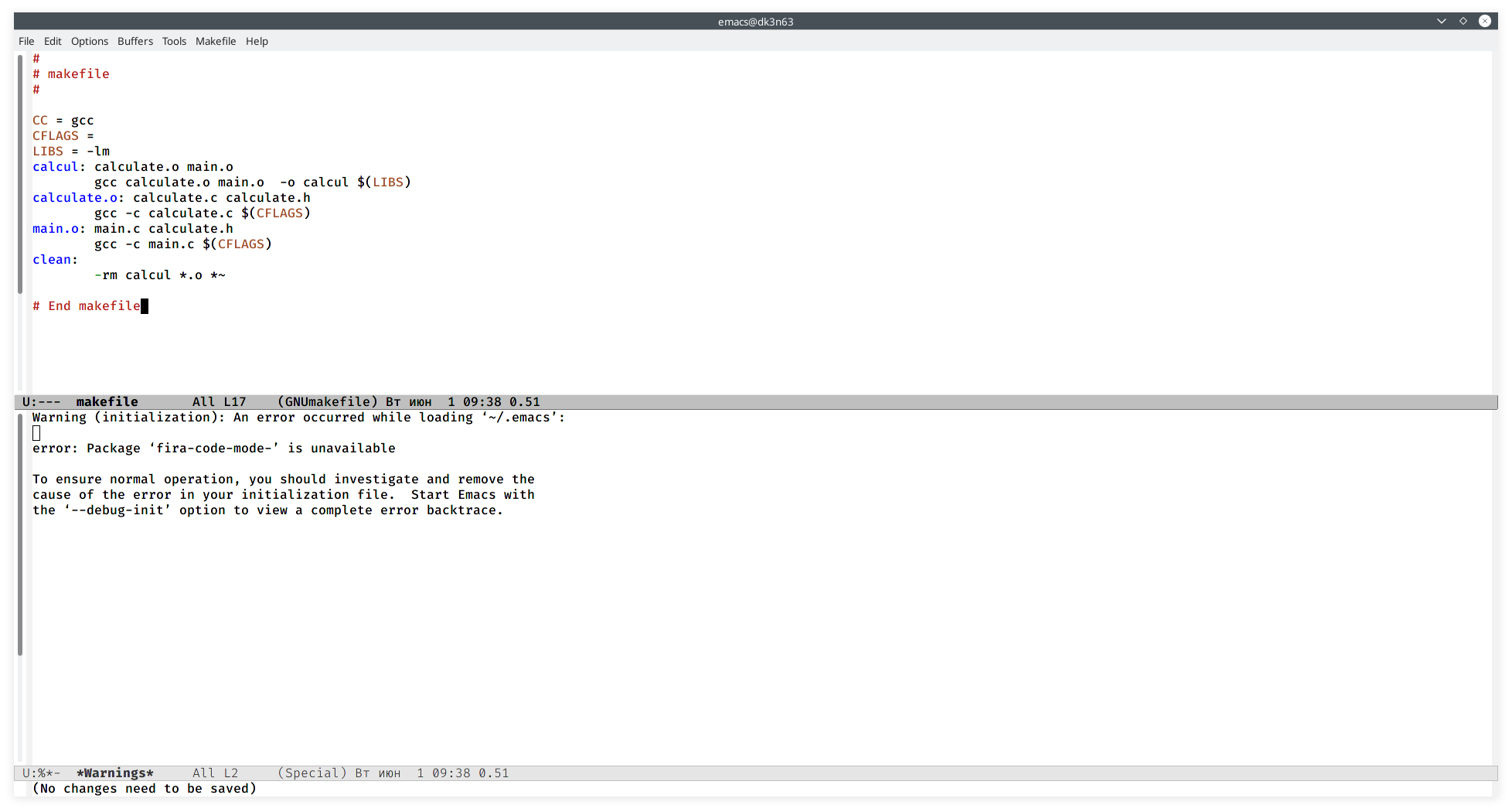


Figure 8: Программа в Makefile

Данный файл необходим для автоматической компиляции файлов calculate.c (цель calculate.o), main.c (цельmain.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.

6). Далее исправила Makefile (Скриншот 9 ). Подробная информация о написании программ в Linux указана в следующем источнике: Электронный ресурс: https://vunivere.ru/work23597

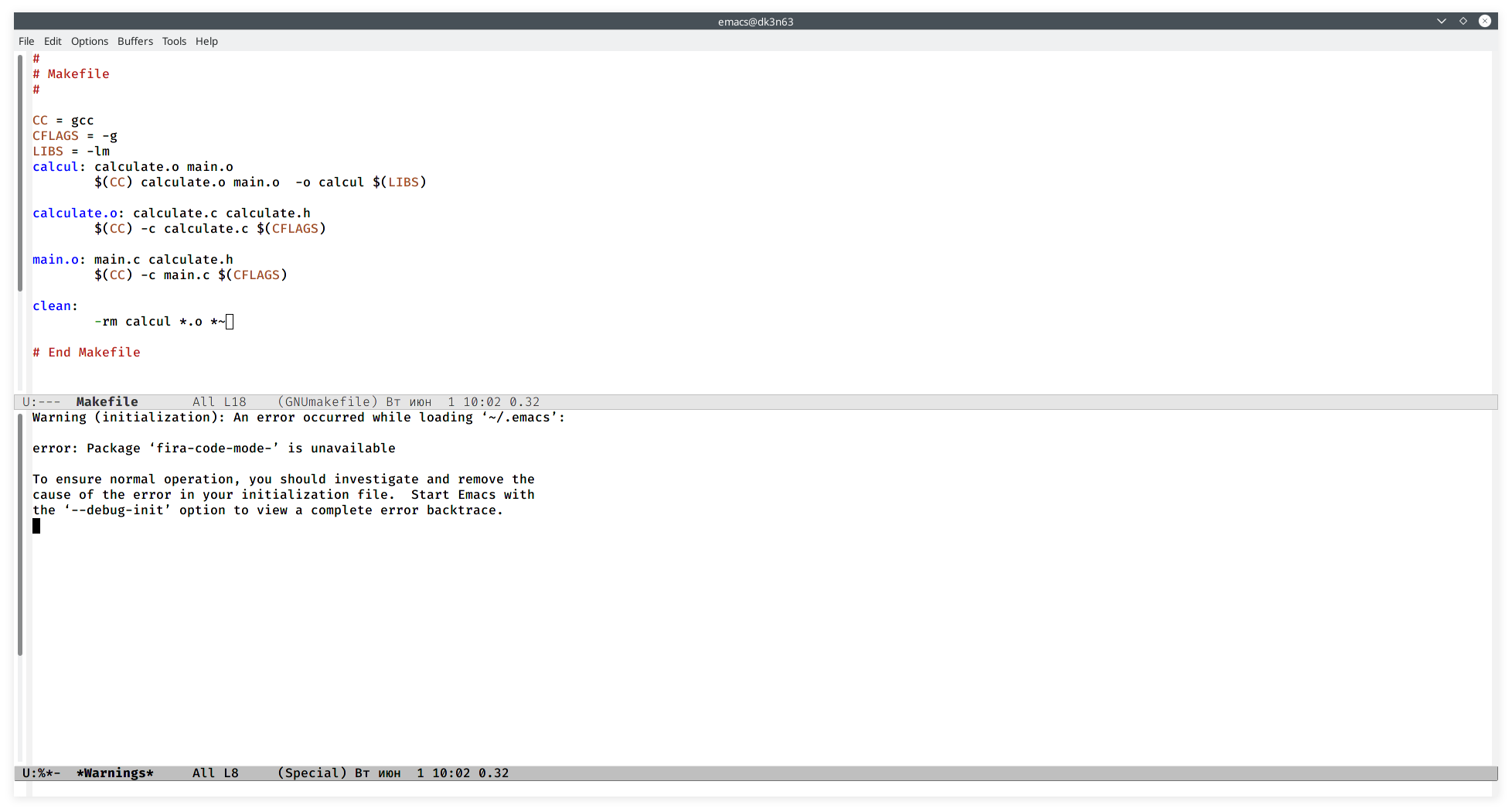


Figure 9: Программа в Makefile

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 clear» (Рисунок 10 ). Выполнила компиляцию файлов, используя команды «make calculate.o», «make main.o», «make calcul» (Рисунок 11 ).

Figure 10: Удаление файлов

Figure 10: Удаление файлов

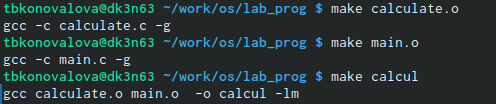


Figure 11: Компиляция файлов

Далее с помощью gdb выполнила отладку программы calcul. Запустила отладчик GDB, загрузив в него программу для отладки, используя команду: «gdb./calcul» (алгоритм действий представлен на рис. 12 ).

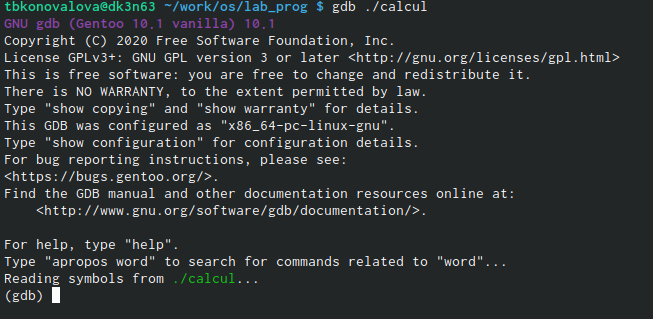


Figure 12: Работа с gdb

Для запуска программы внутри отладчика ввела команду «run» (Рисунок 13 ).

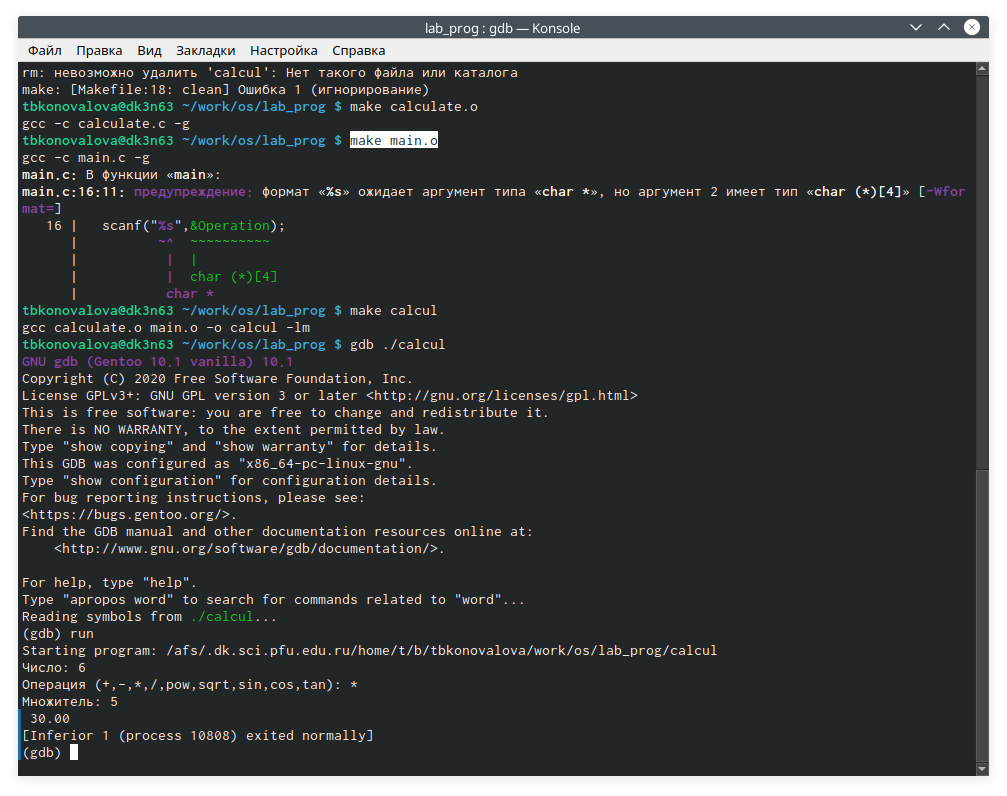


Figure 13: Работа с gdb - run

Для постраничного (по10строк) просмотра исходного кода использовала команду «list» (Рисунок 14 ).

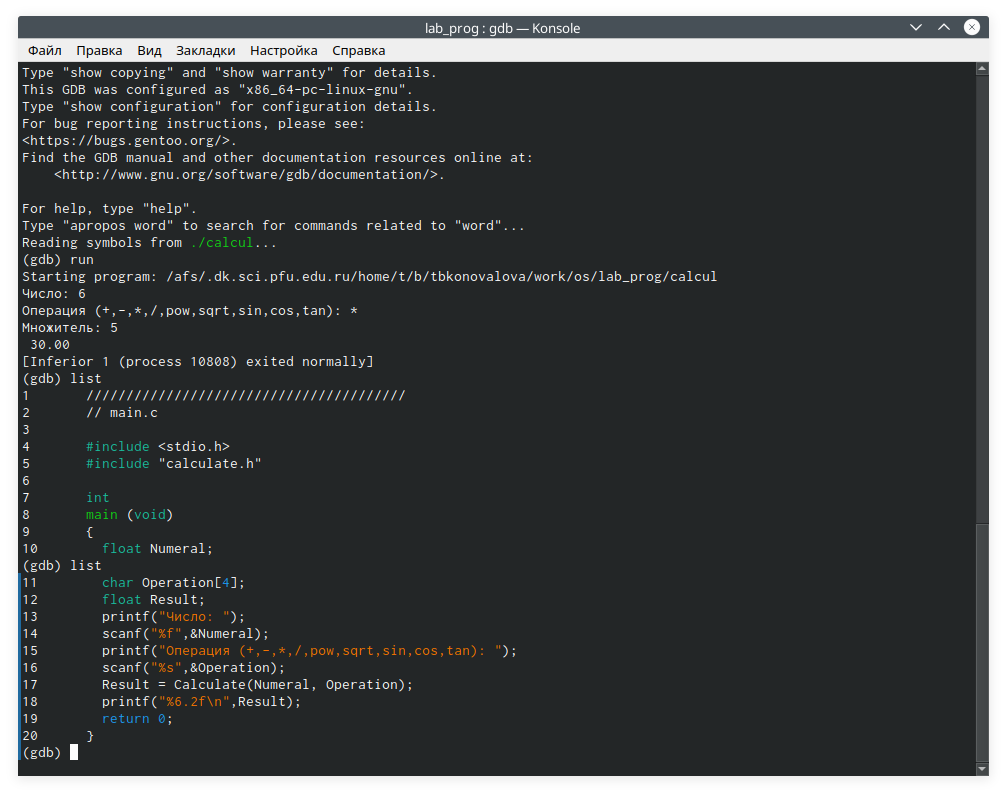


Figure 14: Работа с gdb - list

Для просмотра строк с 12 по 15 основного файла использовала команду «list 12,15» (Рисунок 15 ).

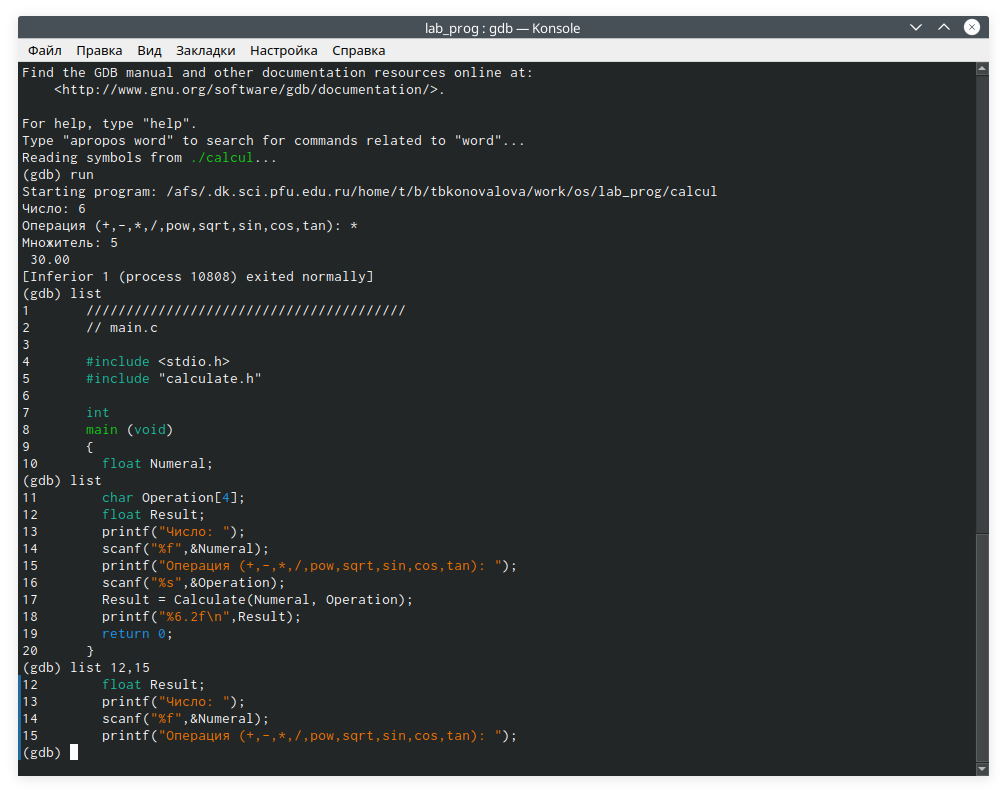


Figure 15: Работа с gdb - list 12,15

Для просмотра определённых строк не основного файла использовала команду «list calculate.c:20,29» (Рисунок 16 ).

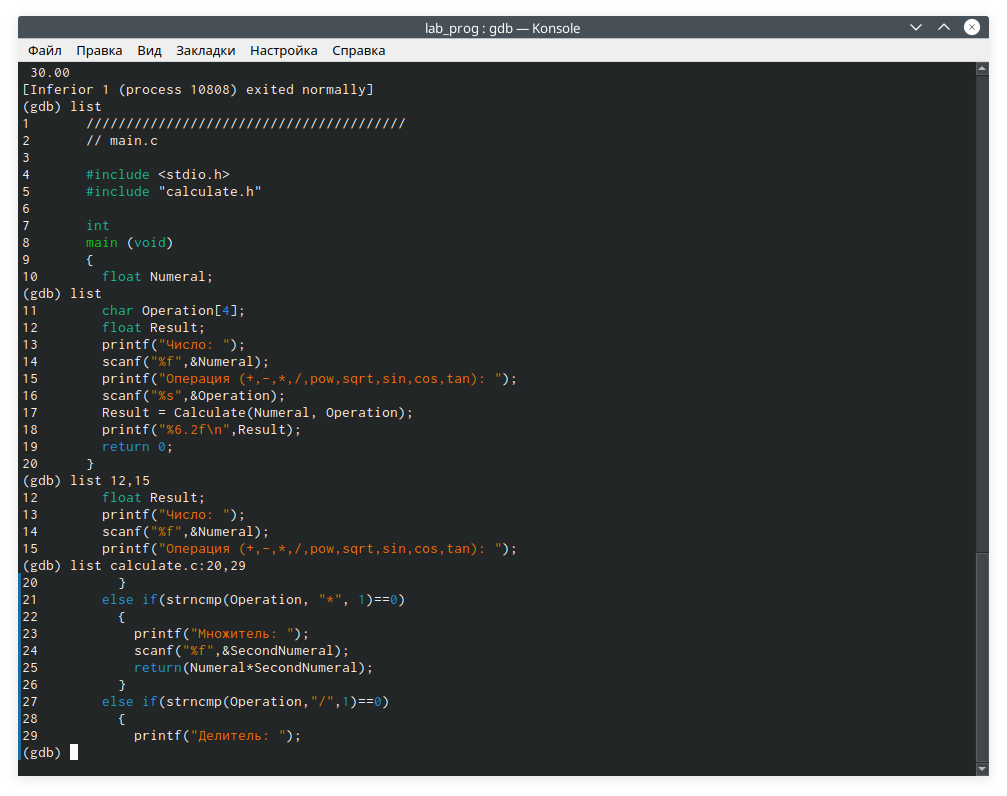


Figure 16: Работа с gdb - list calculate.c:20,29

Установила точку останова в файле calculate.c на строке номер 21, используя команды «list calculate.c:20,27» и «break 21» (Рисунок 17 ).

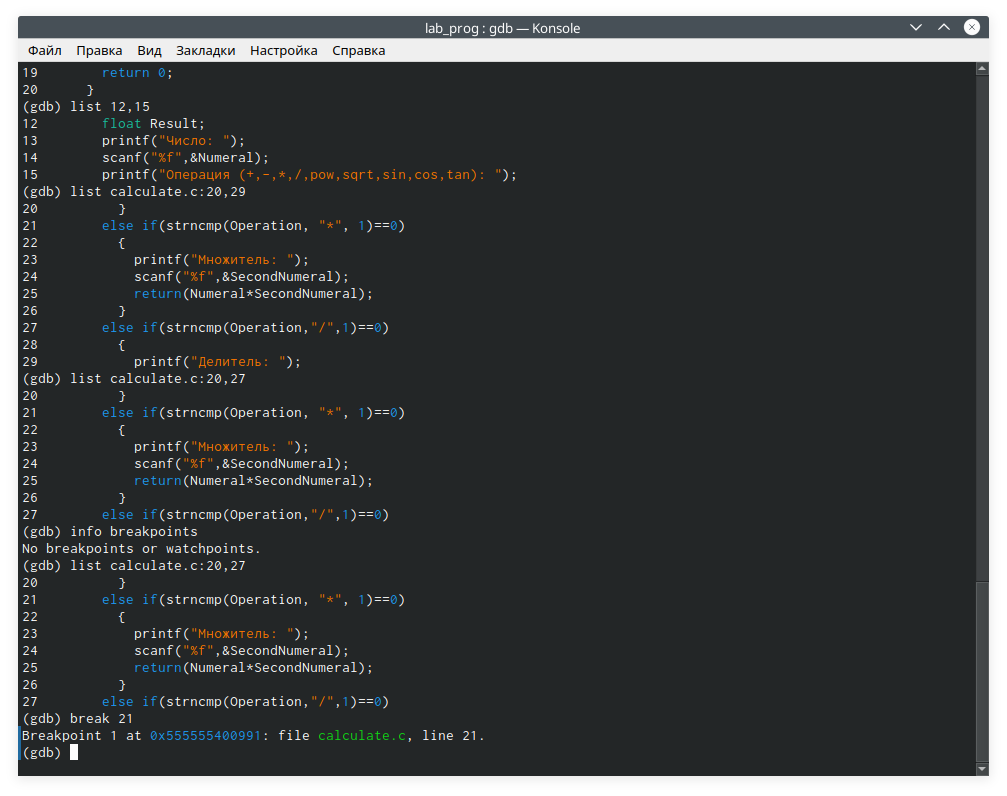


Figure 17: Работа с gdb - list calculate.c:20,27

Вывела информацию об имеющихся в проекте точках останова с помощью команды «info breakpoints» (Скриншот 18 ).

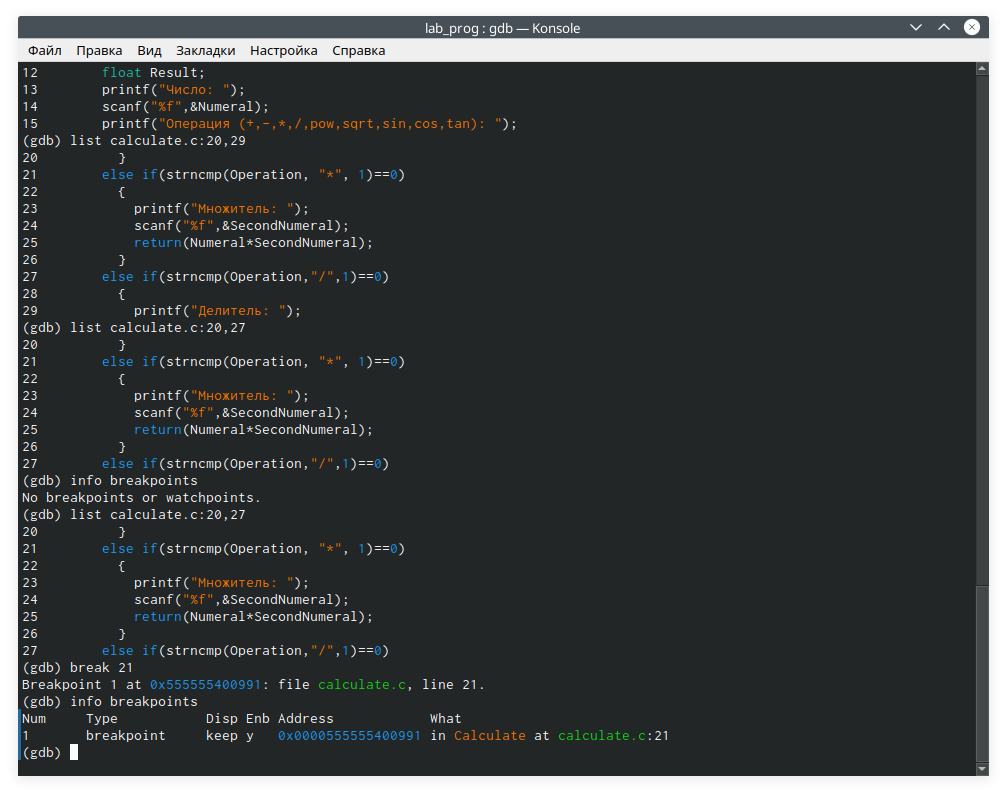


Figure 18: Работа с gdb - info breakpoints

Запустила программу внутри отладчика и убедилась, что программа остановилась в момент прохождения точки останова. Использовала команды «run», «5», «\*» и «backtrace» (Скриншот 19 ).

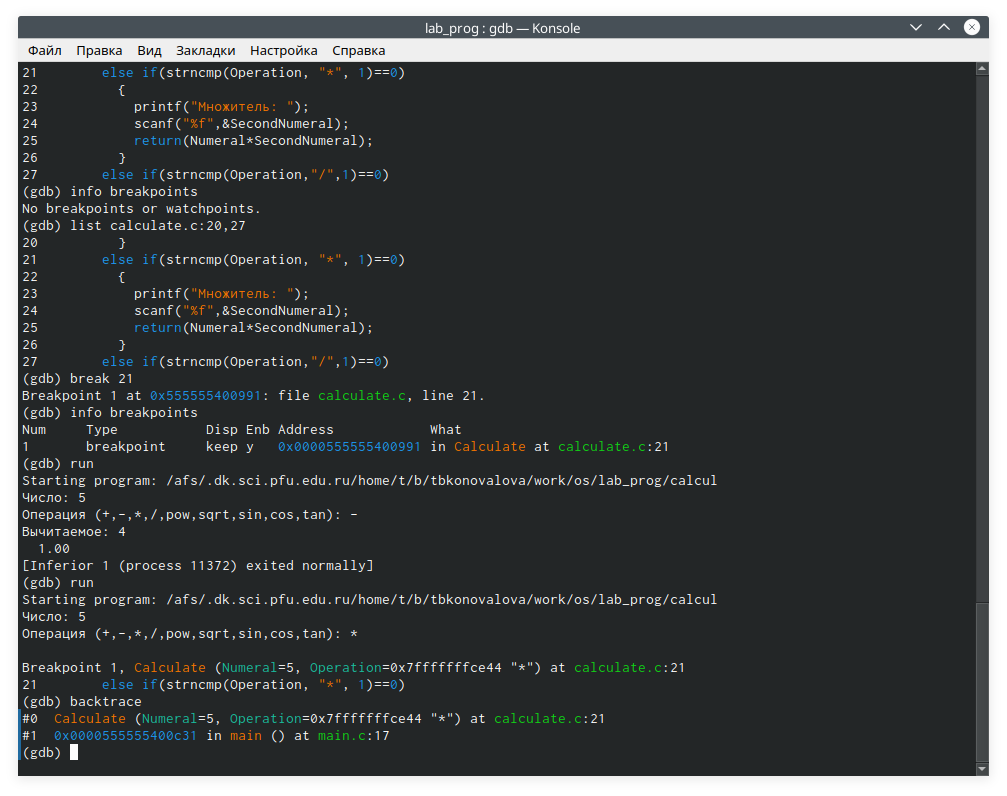


Figure 19: Работа с gdb - run

Посмотрела, чему равно на этом этапе значение переменной Numeral, введя команду «print Numeral» (Скриншот 20 ).

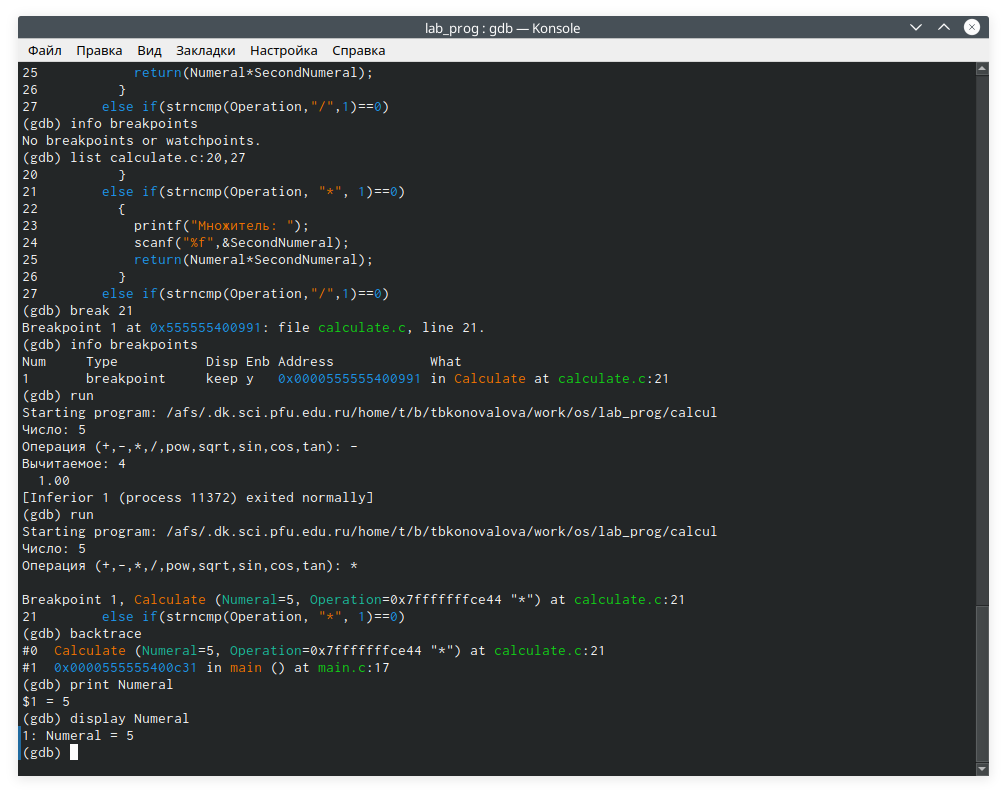


Figure 20: Работа с gdb - print Numeral

Сравнила с результатом вывода на экран после использования команды «display Numeral». Значения совпадают (Скриншот 21 ).

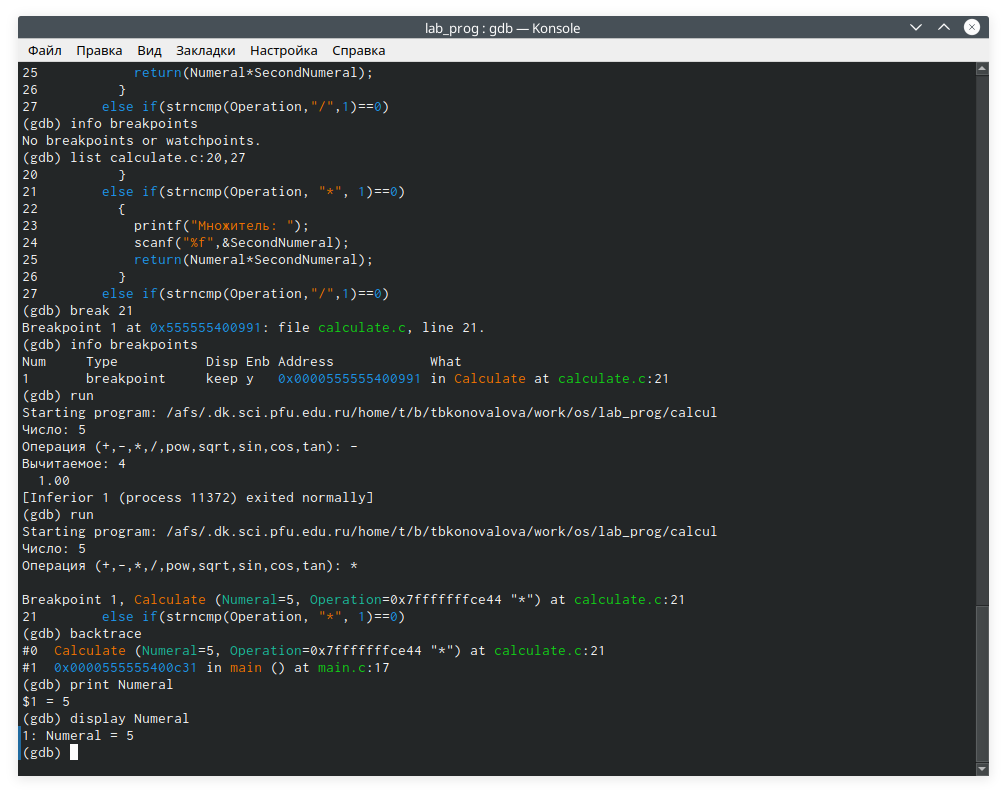


Figure 21: Работа с gdb - display Numeral

Убрала точки останова с помощью команд «info breakpoints» и «delete1» (алгоритм действий представлен на рис. 22 ).

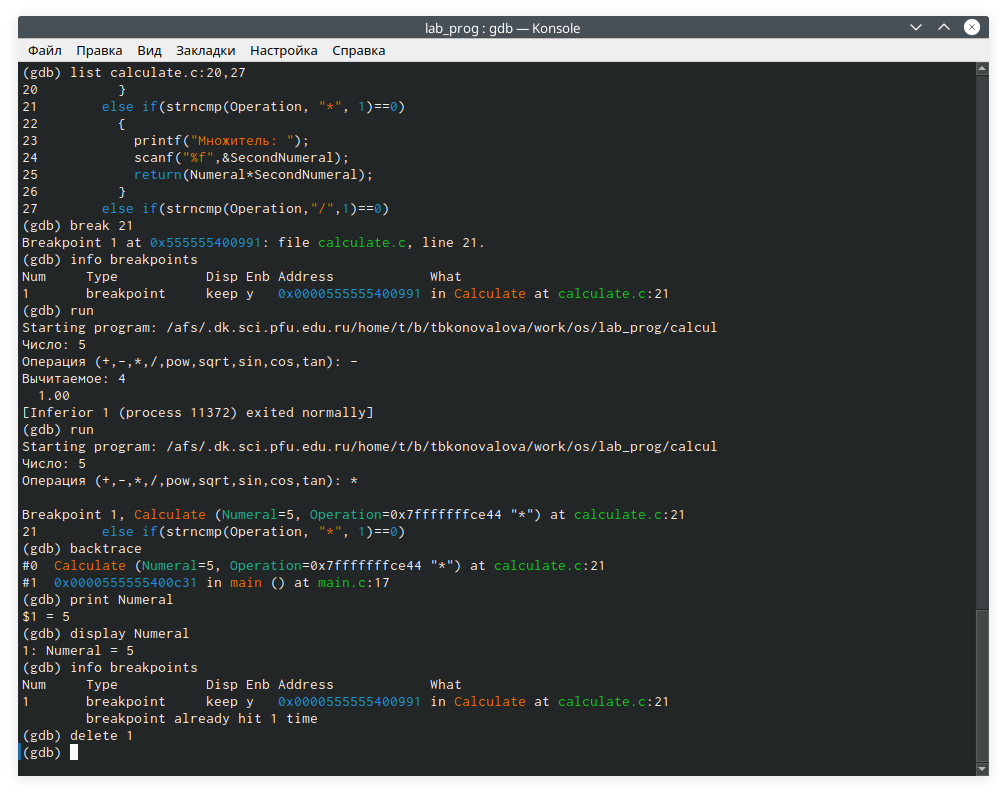


Figure 22: Работа с gdb - info breakpoints

7). Далее воспользовалась командами «splint calculate.c» и «splint main.c» (алгоритм действий представлен на рис. 23 , 24 ).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.

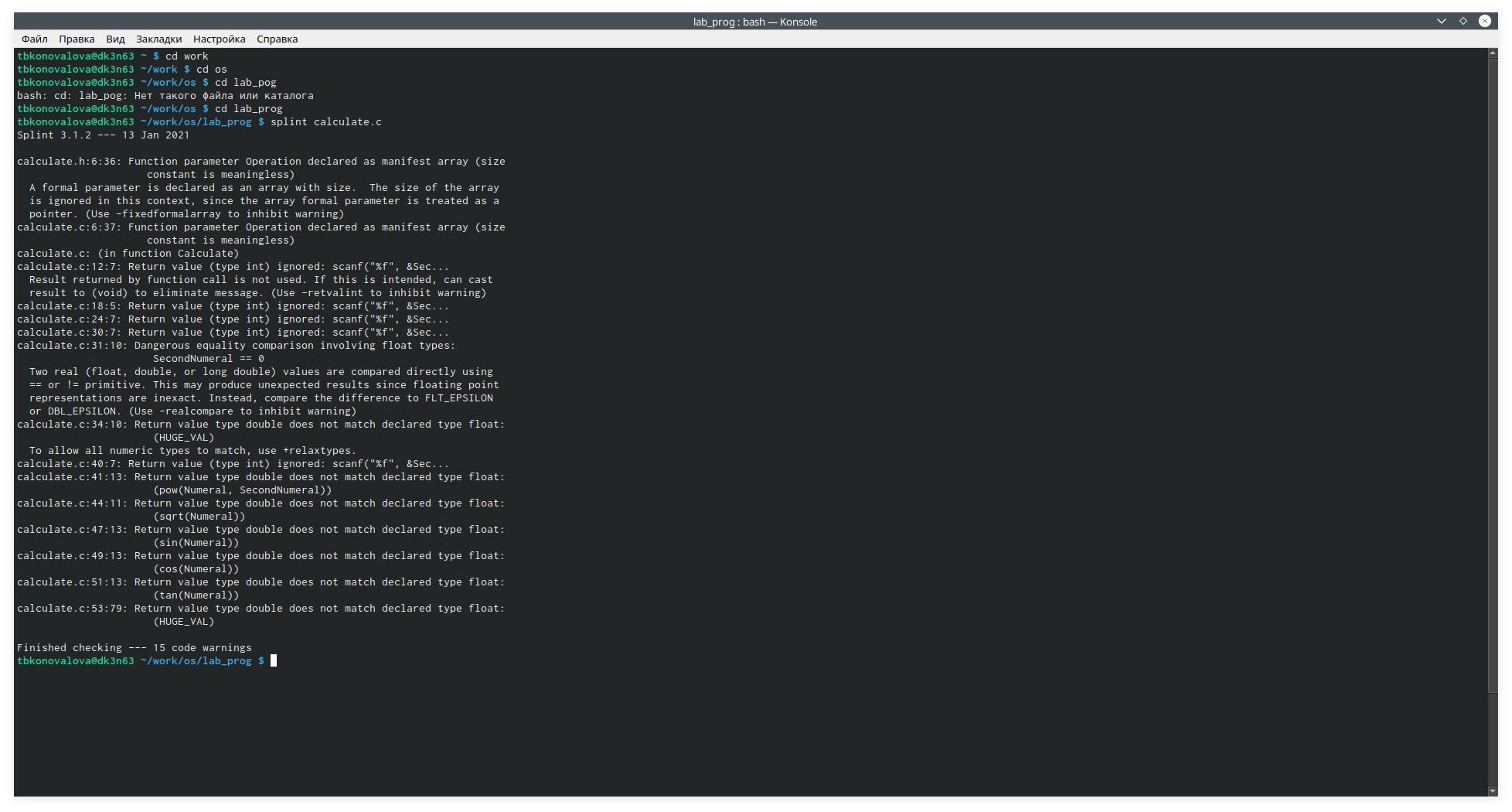


Figure 23: Результат команды splint calculate.c

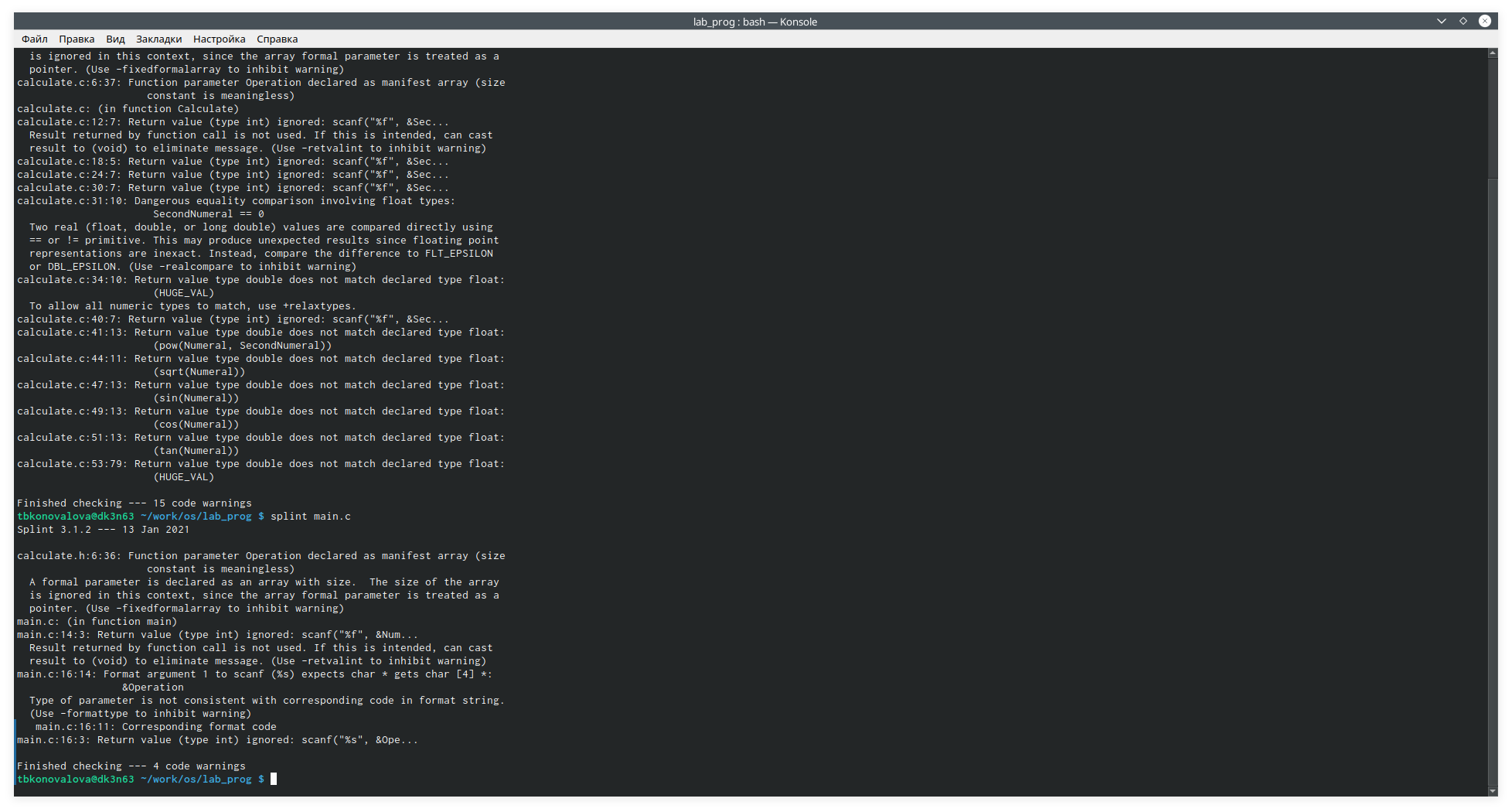


Figure 24: Результат конмады splint main.c

Контрольные вопросы:

1). Чтобы получить информацию о возможностях программ gcc, make, gdbи др.нужно воспользоваться командой manили опцией -help(-h)для каждой команды.

2). Процесс разработки программного обеспечения обычно разделяется на следующие этапы:

1. планирование, включающее сбор и анализ требований к функционалу и другим характеристикам разрабатываемого приложения;
2. проектирование, включающее в себя разработку базовых алгоритмов и спецификаций, определение языка программирования;
3. непосредственная разработка приложения: oкодирование −по сути создание исходного текста программы (возможно в нескольких вариантах); –анализ разработанного кода; oсборка, компиляция и разработка исполняемого модуля; oтестирование и отладка, сохранение произведённых изменений;
4. 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.

3). Для имени входного файла суффикс определяет какая компиляция требуется. Суффиксы указывают на тип объекта. Файлы с расширением (суффиксом) .cвоспринимаются gccкак программы на языке С, файлы с расширением .ccили .C−как файлы на языке C++, а файлы cрасширением .oсчитаются объектными.Например, в команде «gcc-cmain.c»:gccпо расширению (суффиксу) .cраспознает 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 -oи в качестве параметра задать имя создаваемого файла: «gcc-ohellomaiВ ходе выполнения данной лабораторной работы я приобрелапростейшие навыки разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linuxна примере создания на языке программирования С калькулятора с простейшими функциями.n.c».

4). Основное назначение компилятора языка Си в UNIX заключается в компиляции всей программы и получении исполняемого файла/модуля.

5). 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.

6). Для работы с утилитой makeнеобходимо в корне рабочего каталога с Вашим проектом создать файл с названием makefileили 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 Makefileможет 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 = gccCFLAGS =# Compile abcd.c normalyabcd: abcd.c$(CC) -o abcd $(CFLAGS) abcd.cclean:-rm abcd *.o* ~# EndMakefileforabcd.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.

7). 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 GNUдля ОС типа UNIXвходит 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-cfile.c-g. После этого для начала работы с gdbнеобходимо в командной строке ввести одноимённую команду, указав в качестве аргумента анализируемый бинарный файл: gdbfile.o

8). Основные команды отладчика gdb: 1. backtrace − вывод на экран пути к текущей точке останова (по сутивывод − названий всех функций); 2. break − установить точку останова (в качестве параметра можетбыть указан номер строки или название функции); 3. clear − удалить все точки останова в функции; 4. continue − продолжить выполнение программы; 5. delete − удалить точку останова; 6. display − добавить выражение в список выражений, значения которых отображаются при достижении точки останова программы; 7. finish − выполнить программу до момента выхода из функции; 8. info breakpoints −вывести на экран список используемых точек останова; 9. info watchpoints −вывести на экран список используемых контрольных выражений; 10. list − вывести на экран исходный код (вВ ходе выполнения данной лабораторной работы я приобрелапростейшие навыки разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linuxна примере создания на языке программирования С калькулятора с простейшими функциями. качестве параметра может быть указано название файла и через двоеточие номера начальнойи конечной строк); 11. next − выполнить программу пошагово, но без выполнения вызываемых в программе функций; 12. print − вывести значение указываемого в качестве параметра выражения; 13. run − запуск программы на выполнение; 14. set − установить новое значение переменной; 15. step − пошаговое выполнение программы; 16. watch − установить контрольное выражение, при изменении значения которого программа будет остановлена. Для выхода из gdbможно 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-hи mangdb.

9). Cхема отладки программы показана в 6 пункте лабораторной работы.

10). 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 массивасимволов уже является указателемна первый элементэтого массива.

11). 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.

12). 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 Cанализатор splintгенерирует 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 работt программы, переменные с некорректно заданными значениямии типами и многое другое.

# Библиография

1. Программное обеспечение GNU/Linux. Лекция 9. Хранилище и дистрибутив (Г. Курячий, МГУ)
2. Программное обеспечение GNU/Linux. Лекция 10. Минимальный набор знаний (Г. Курячий, МГУ)
3. Программное обеспечение GNU/Linux. Лекция 11. udev, DBus, PolicyKit (Г. Курячий, МГУ)
4. Электронный ресурс: https://vunivere.ru/work23597
5. Электронный ресурс: https://it.wikireading.ru/34160

# Выводы

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