Отчёта по лабораторной работе №14

Операционый Систем

Коне Сирики НФИБД-01-20

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 # Задание

-Этапы разработки приложений -Компиляция исходного текста и построение исполняемого файла -Тестирование и отладка Анализ исходного текста программы

# Выполнение лабораторной работы

Ход работы: 1. В домашнем каталоге создала подкаталог ~/work/os/lab\_prog.

(рис. 1)

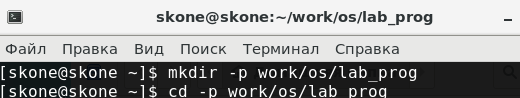


Figure 1: рисунок 1

1. Создал в нём файлы: calculate.h, calculate.c, main.c. Это примитивнейший калькулятор, способный складывать, вычитать, умножать и делить, возводить число в степень, брать квадратный корень, вычислять sin, cos, tan. При запуске он запрашивает первое число, операцию, второе число. После этого программа выводит результат и останавливается. Код для программы приведён в задании к лабораторной работе.

(рис. 2)

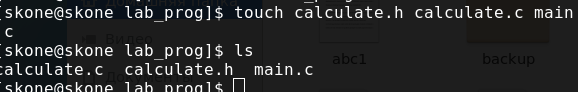


Figure 2: рисунок 2

Реализация функций калькулятора в файле calculate.h:

(рис. 3)

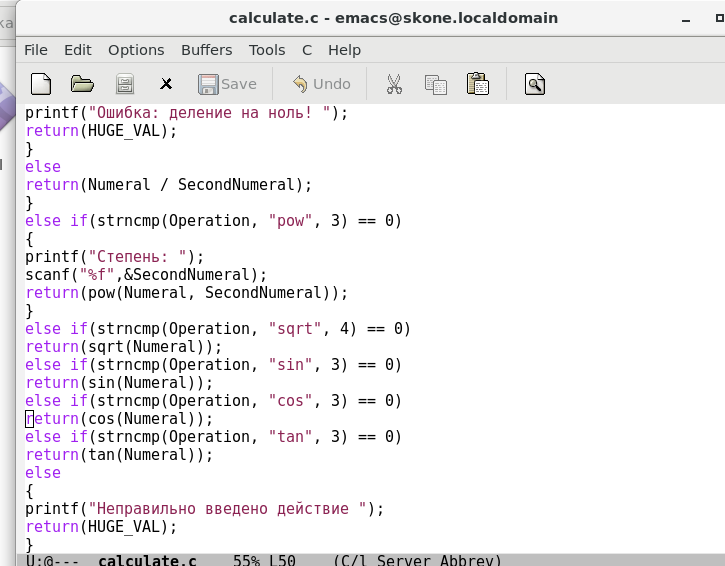


Figure 3: рисунок 3

Интерфейсный файл calculate.h, описывающий формат вызова функциикалькулятора:

(рис. 4)

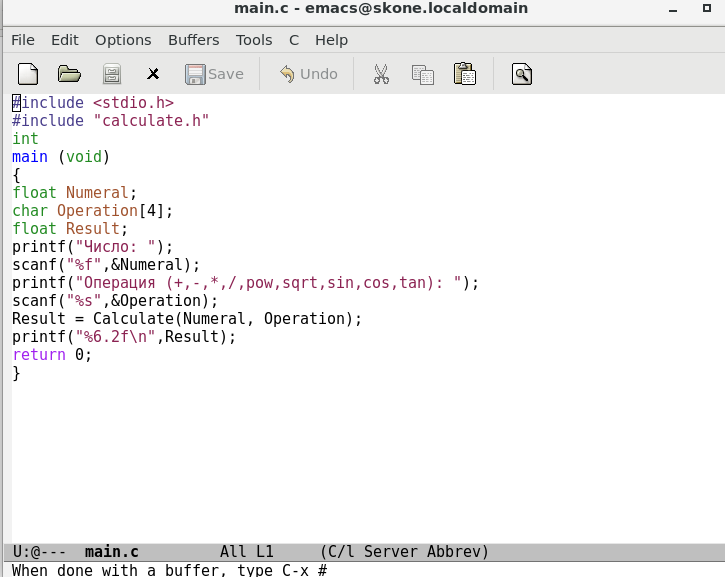


Figure 4: рисунок 4

Основной файл main.c, реализующий интерфейс пользователя к калькулятору:

(рис. 5)

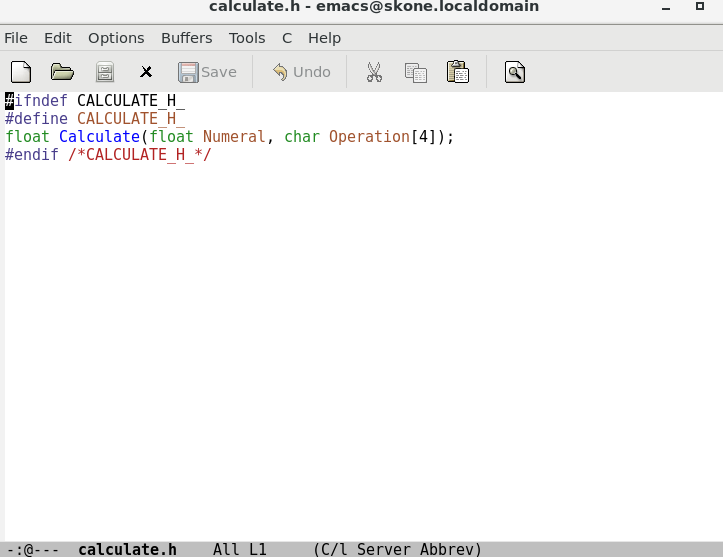


Figure 5: рисунок 5

1. Выполнил компиляцию программы посредством gcc: gcc -c calculate.c gcc -c main.c gcc calculate.o main.o -o calcul –lm

(рис. 6)

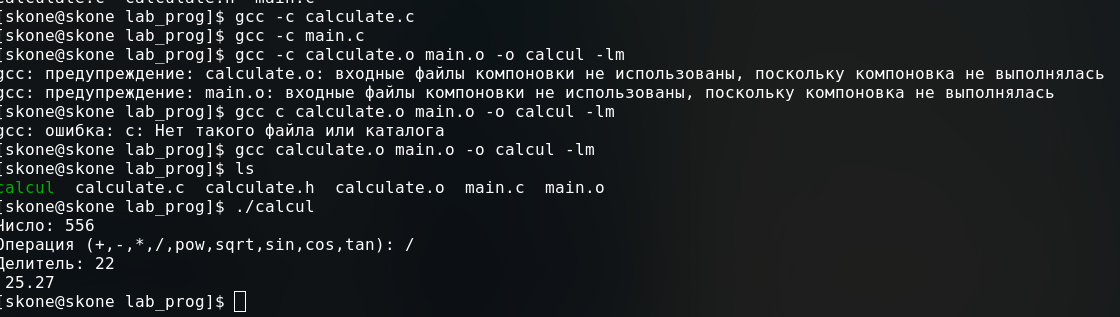


Figure 6: рисунок 6

1. Исправил синтаксические ошибки: Scanf(“%s”,&Operation) -> scanf(“%s”,Operation)
2. Создал Makefile со следующим содержанием: # # Makefile #

CC = gcc #задание переменных CFLAGS = LIBS = -lm

calcul: calculate.o main.o gcc calculate.o main.o -o calcul $(LIBS) #получение исполняемого файла

calculate.o: calculate.c calculate.h gcc -c calculate.c $(CFLAGS) #компилирование файла calculate.c

main.o: main.c calculate.h gcc -c main.c $(CFLAGS) #компилирование файла main.c

clean: -rm calcul *.o* ~ #очистка каталога от файлов, появившихся в процессе компиляции # End Makefile

(рис. 7)

Figure 7: рисунок 7

Figure 7: рисунок 7

(рис. 8)

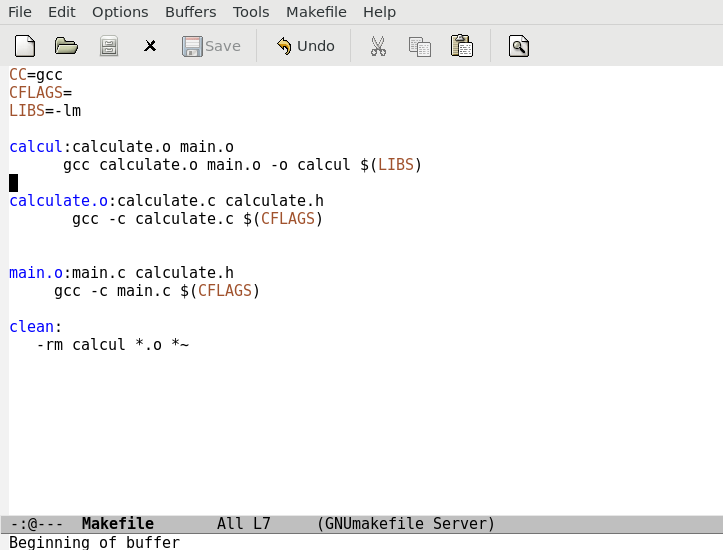


Figure 8: рисунок 8

1. С помощью gdb выполнил отладку программы calcul (перед использованием gdb исправила Makefile: CFLAGS=-g):

(рис. 9)

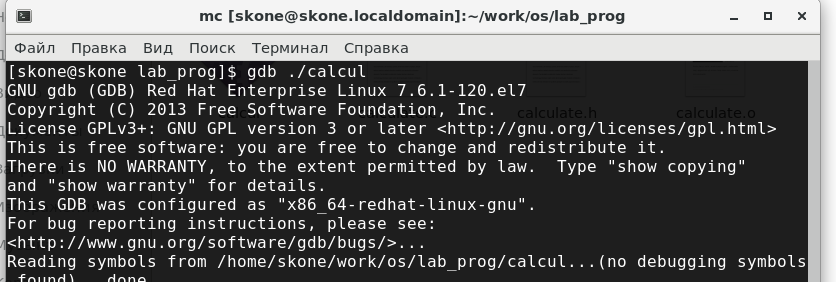


Figure 9: рисунок 9

– Запустил отладчик GDB, загрузив в него программу для отладки: gdb ./calcul

(рис. 10)

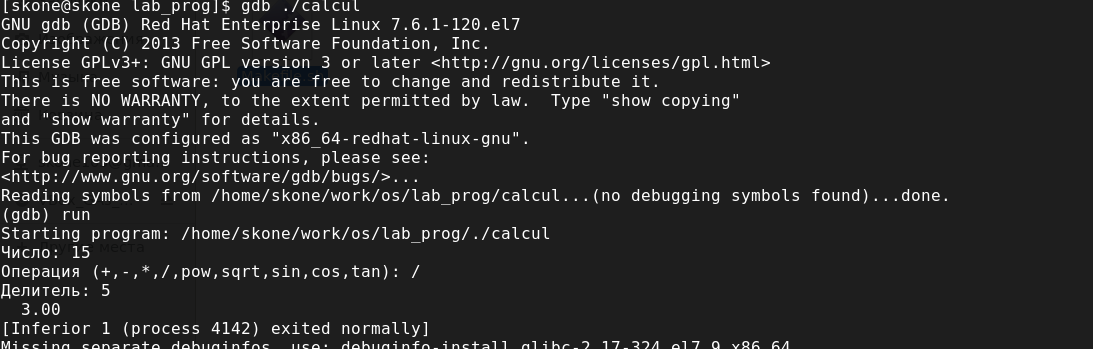


Figure 10: рисунок 10

– Для запуска программы внутри отладчика ввёл команду run: run

(рис. 11)

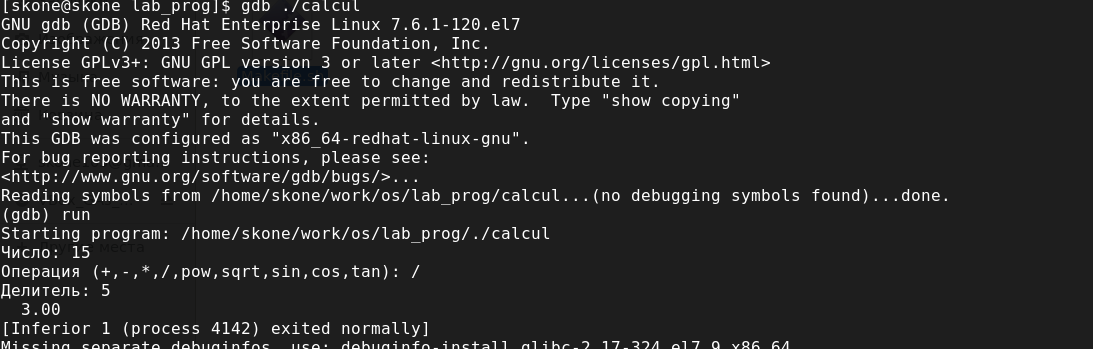


Figure 11: рисунок 11

– Для постраничного (по 9 строк) просмотра исходного код использовал команду list: list

(рис. 12)

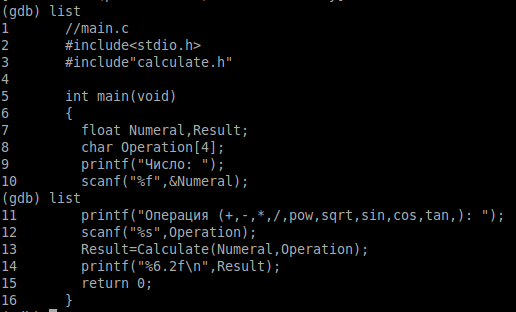


Figure 12: рисунок 12

– Для просмотра строк с 12 по 15 основного файла использовал list с параметрами: list 12,15

(рис. 13)

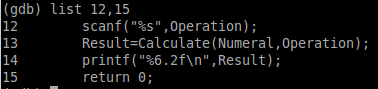


Figure 13: рисунок 13

– Для просмотра определённых строк не основного файла использовал list с параметрами: list calculate.c:20,29

(рис. 14)

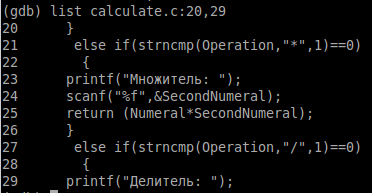


Figure 14: рисунок 14

– Установил точку останова в файле calculate.c на строке номер 21:break 21

(рис. 15)

Figure 15: рисунок 15

Figure 15: рисунок 15

– Вывёл информацию об имеющихся в проекте точка останова: info breakpoints

(рис. 16)

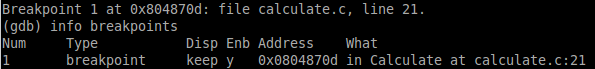


Figure 16: рисунок 16

– Запустил программу внутри отладчика и убедилаь, что программа останавливается в момент прохождения точки останова: run 5 \* backtrace

– Отладчик выдал следующую информацию: #0 Calculate (Numeral=5, Operation=0xbffff2d8 "\*") at calculate.c:21

(рис. 17)

Figure 17: рисунок 17

Figure 17: рисунок 17

#1 0x08048956 in main () at main.c:13 а команда backtrace показала весь стек вызываемых функций от начала программы до текущего места. – Посмотрела, чему равно на этом этапе значение переменной Numeral, введя: print Numeral На экран было выведено число 5.

(рис. 18)

Figure 18: рисунок 18

Figure 18: рисунок 18

– Сравнила с результатом вывода на экран после использования команды: display Numeral

(рис. 19)

Figure 19: рисунок 19

Figure 19: рисунок 19

– Убрал точки останова: info breakpoints delete 1

(рис. 20)

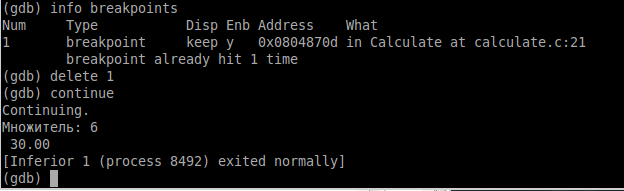


Figure 20: рисунок 20

1. С помощью утилиты splint попробовала проанализировать коды файлов calculate.c и main.c.

(рис. 21)

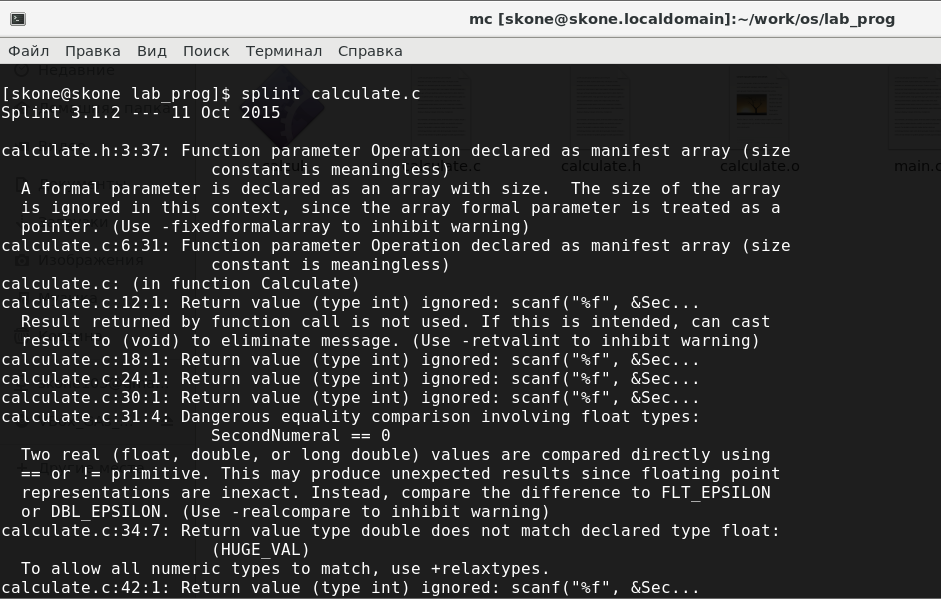


Figure 21: рисунок 21

Например, split отмечает отсутствие необходимости указывать размер массива, который является аргументом функции. В некоторых математических функциях возвращается double, а не float.

(рис. 22)

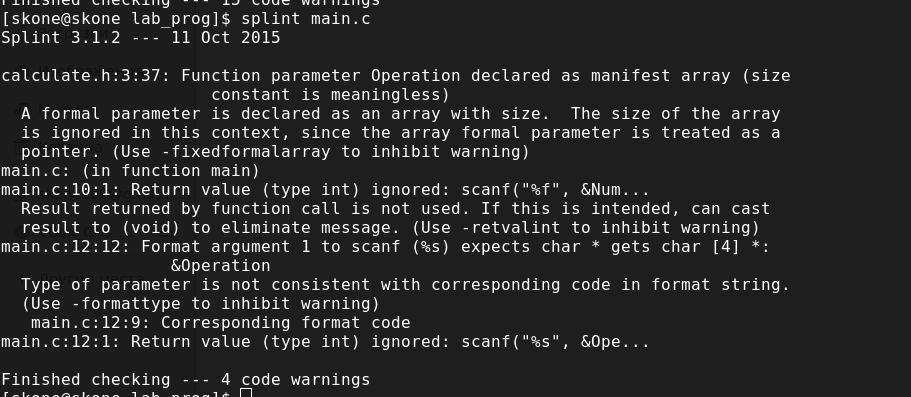


Figure 22: рисунок 22

Контрольные вопросы:

1. Как получить информацию о возможностях программ gcc, make, gdb и др.? Прочитать man-файл.
2. Назовите и дайте краткую характеристику основным этапам разработки приложений в UNIX. 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: – 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.
3. Что такое суффикс в контексте языка программирования? Приведите примеры использования. Суффикс – это расширение файла. Позволяет определить тип файла, т.е., что с ним можно делать. Например, 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.
4. Каково основное назначение компилятора языка С в UNIX? Компиляция файлов с исходным кодом в объектные модули и получение исполняемых файлов.
5. Для чего предназначена утилита make? Make позволяет автоматизировать процесс преобразования файлов программы из одной формы в другую, отслеживает взаимосвязи между файлами.
6. Приведите пример структуры Makefile. Дайте характеристику основным элементам этого файла. Общий синтаксис Makefile имеет вид: target1 [target2…]:[:] [dependment1…] [(tab)commands] [#commentary] [(tab)commands] [#commentary] 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. После # пишутся комментарии, они не обрабатываются.
7. Назовите основное свойство, присущее всем программам отладки. Что необходимо сделать, чтобы его можно было использовать? Программы отладки позволяют найти и устранить ошибки в программе. Чтобы использовать их, необходимо скомпилировать анализируемый код программы таким образом, чтобы отладочная информация содержалась в результирующем бинарном файле.
8. Назовите и дайте основную характеристику основным командам отладчика gdb. Основные команды: 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 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
9. Опишите по шагам схему отладки программы, которую Вы использовали при выполнении лабораторной работы. Запуск отладчика Запуск программы в отладчике Просмотр исходного кода основного и не основного файлов Установка точек останова и просмотр информации о них Запуск программы с установленными точками останова. Просмотр стека вызываемых функций при достижении точки останова Просмотр значения переменной в момент достижения точки останова Удаление точек останова
10. Прокомментируйте реакцию компилятора на синтаксические ошибки в программе при его первом запуске. Компилятор выводит найденные им ошибки с комментариями. Это такие ошибки, которые могут повлиять на работу программы.
11. Назовите основные средства, повышающие понимание исходного кода программы. Само по себе грамотное написание кода (с переносами и отступами в нужных местах) уже повышает его понимание. Также этой цели служат комментарии.
12. Каковы основные задачи, решаемые программой splint? Эта утилита 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 та- кие ошибки, как одинаковые объекты, определённые в разных файлах, или объекты, чьи значения не используются в работе программы, переменные с некорректно за- данными значениями и типами и многое другое

# Выводы

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