

# **Лабораторная работа №14**

**Дисциплина: Операционные системы**

Егорова Александра

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>4</b>
<b>2</b>	<b>Задание</b>	<b>5</b>
<b>3</b>	<b>Выполнение лабораторной работы</b>	<b>6</b>
<b>4</b>	<b>Выводы</b>	<b>15</b>
<b>5</b>	<b>Контрольные вопросы</b>	<b>16</b>
<b>6</b>	<b>Библиография</b>	<b>21</b>

# List of Figures

3.1	Создание подкаталога . . . . .	6
3.2	Создание файлов . . . . .	6
3.3	Файл calculate.c . . . . .	7
3.4	Файл calculate.c . . . . .	7
3.5	Файл calculate.h . . . . .	8
3.6	Файл main.c . . . . .	8
3.7	Компиляция программы . . . . .	8
3.8	Создание Makefile . . . . .	9
3.9	Исправление Makefile . . . . .	10
3.10	Удаление исполняемых и объектных файлов . . . . .	10
3.11	Отладчик GDB . . . . .	11
3.12	list . . . . .	11
3.13	list 12,15 . . . . .	11
3.14	list calculate.c:20,29 . . . . .	12
3.15	Точка останова в файле . . . . .	12
3.16	Информация точках останова . . . . .	12
3.17	Проверка точки останова . . . . .	12
3.18	Значение переменной Numeral . . . . .	13
3.19	Значение переменной Numeral . . . . .	13
3.20	Убираем точки останова . . . . .	13
3.21	splint . . . . .	14
3.22	splint . . . . .	14

# 1 Цель работы

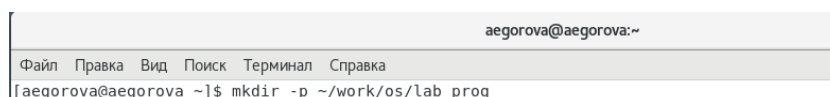
Приобрести простейшие навыки разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linux на примере создания на языке программирования С калькулятора с простейшими функциями.

## 2 Задание

1. В домашнем каталоге создайте подкаталог `~/work/os/lab_prog`.
2. Создайте в нём файлы: `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.
3. Выполните компиляцию программы посредством `gcc`
4. При необходимости исправьте синтаксические ошибки.
5. Создайте `Makefile`
6. С помощью `gdb` выполните отладку программы `calcul`
7. С помощью утилиты `splint` попробуйте проанализировать коды файлов

### 3 Выполнение лабораторной работы

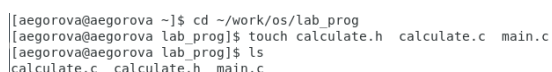
- 1) В домашнем каталоге создаю подкаталог `~/work/os/lab_prog` с помощью команды «`mkdir -p ~/work/os/lab_prog`». (рис. -fig. 3.1)



```
aegorova@aegorova:~  
Файл  Правка  Вид  Поиск  Терминал  Справка  
[aegorova@aegorova ~]$ mkdir -p ~/work/os/lab_prog
```

Figure 3.1: Создание подкаталога

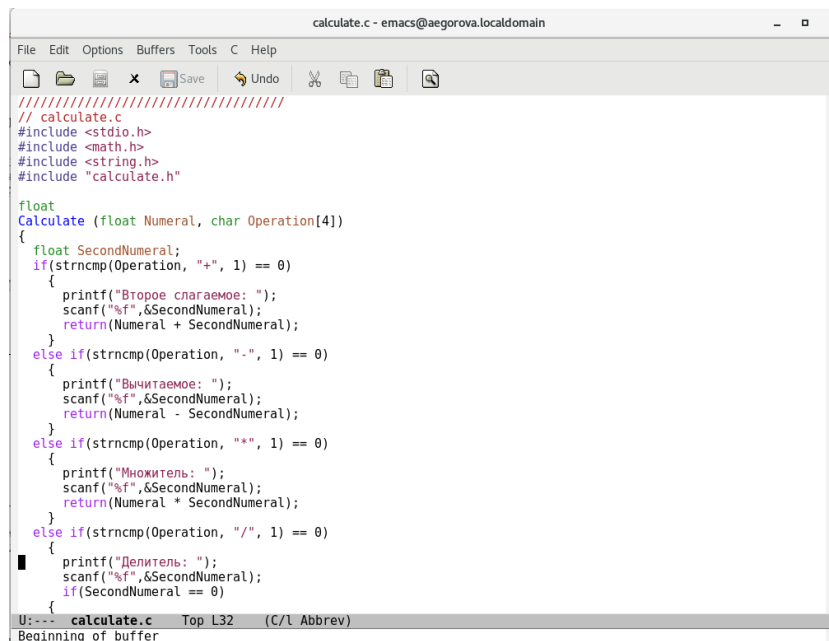
- 2) Создала в каталоге файлы: `calculate.h`, `calculate.c`, `main.c`. Это будет примитивнейший калькулятор, способный складывать, вычитать, умножать и делить, возводить число в степень, брать квадратный корень, вычислять `sin`, `cos`, `tan`. При запуске он будет запрашивать первое число, операцию, второе число. После этого программа выведет результат и остановится. (рис. -fig. 3.2)



```
[aegorova@aegorova ~]$ cd ~/work/os/lab_prog  
[aegorova@aegorova lab_prog]$ touch calculate.h calculate.c main.c  
[aegorova@aegorova lab_prog]$ ls  
calculate.c calculate.h main.c
```

Figure 3.2: Создание файлов

Открыв редактор Emacs, приступила к редактированию созданных файлов. Реализация функций калькулятора в файле `calculate.c` (рис. -fig. 3.3) (рис. -fig. 3.4)

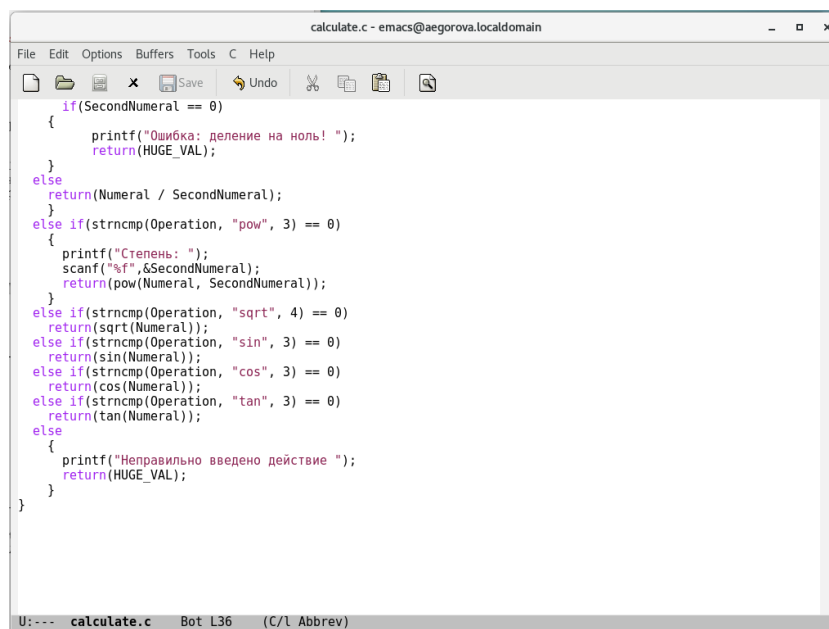


```
calculate.c - emacs@aegorova.localdomain
File Edit Options Buffers Tools C Help
Save Undo
// calculate.c
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "calculate.h"

float
Calculate (float Numeral, char Operation[4])
{
    float SecondNumeral;
    if (strcmp(Operation, "+", 1) == 0)
    {
        printf("Второе слагаемое: ");
        scanf("%f", &SecondNumeral);
        return(Numeral + SecondNumeral);
    }
    else if (strcmp(Operation, "-", 1) == 0)
    {
        printf("Вычитаемое: ");
        scanf("%f", &SecondNumeral);
        return(Numeral - SecondNumeral);
    }
    else if (strcmp(Operation, "*", 1) == 0)
    {
        printf("Множитель: ");
        scanf("%f", &SecondNumeral);
        return(Numeral * SecondNumeral);
    }
    else if (strcmp(Operation, "/", 1) == 0)
    {
        printf("Делитель: ");
        scanf("%f", &SecondNumeral);
        if (SecondNumeral == 0)
        {
            printf("Ошибка: деление на ноль! ");
            return(HUGE_VAL);
        }
        return(Numeral / SecondNumeral);
    }
    else if (strcmp(Operation, "pow", 3) == 0)
    {
        printf("Степень: ");
        scanf("%f", &SecondNumeral);
        return(pow(Numeral, SecondNumeral));
    }
    else if (strcmp(Operation, "sqrt", 4) == 0)
    {
        return(sqrt(Numeral));
    }
    else if (strcmp(Operation, "sin", 3) == 0)
    {
        return(sin(Numeral));
    }
    else if (strcmp(Operation, "cos", 3) == 0)
    {
        return(cos(Numeral));
    }
    else if (strcmp(Operation, "tan", 3) == 0)
    {
        return(tan(Numeral));
    }
    else
    {
        printf("Неправильно введено действие ");
        return(HUGE_VAL);
    }
}

U:--- calculate.c Top L32 (C/l Abbrev)
Beginning of buffer
```

Figure 3.3: Файл calculate.c

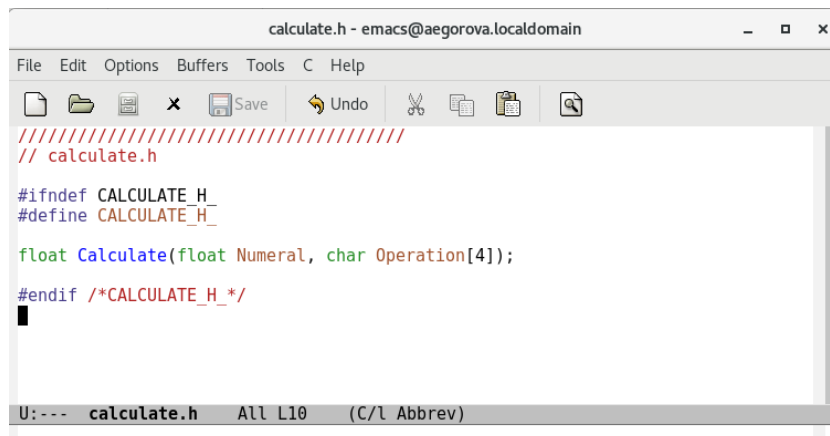


```
calculate.c - emacs@aegorova.localdomain
File Edit Options Buffers Tools C Help
Save Undo
    if (SecondNumeral == 0)
    {
        printf("Ошибка: деление на ноль! ");
        return(HUGE_VAL);
    }
    else
    {
        return(Numeral / SecondNumeral);
    }
    else if (strcmp(Operation, "pow", 3) == 0)
    {
        printf("Степень: ");
        scanf("%f", &SecondNumeral);
        return(pow(Numeral, SecondNumeral));
    }
    else if (strcmp(Operation, "sqrt", 4) == 0)
    {
        return(sqrt(Numeral));
    }
    else if (strcmp(Operation, "sin", 3) == 0)
    {
        return(sin(Numeral));
    }
    else if (strcmp(Operation, "cos", 3) == 0)
    {
        return(cos(Numeral));
    }
    else if (strcmp(Operation, "tan", 3) == 0)
    {
        return(tan(Numeral));
    }
    else
    {
        printf("Неправильно введено действие ");
        return(HUGE_VAL);
    }
}

U:--- calculate.c Bot L36 (C/l Abbrev)
```

Figure 3.4: Файл calculate.c

Интерфейсный файл calculate.h , описывающий формат вызова функции-калькулятора. (рис. -fig. 3.5)



```
calculate.h - emacs@aegorova.localdomain
File Edit Options Buffers Tools C Help
////////////////////
// calculate.h

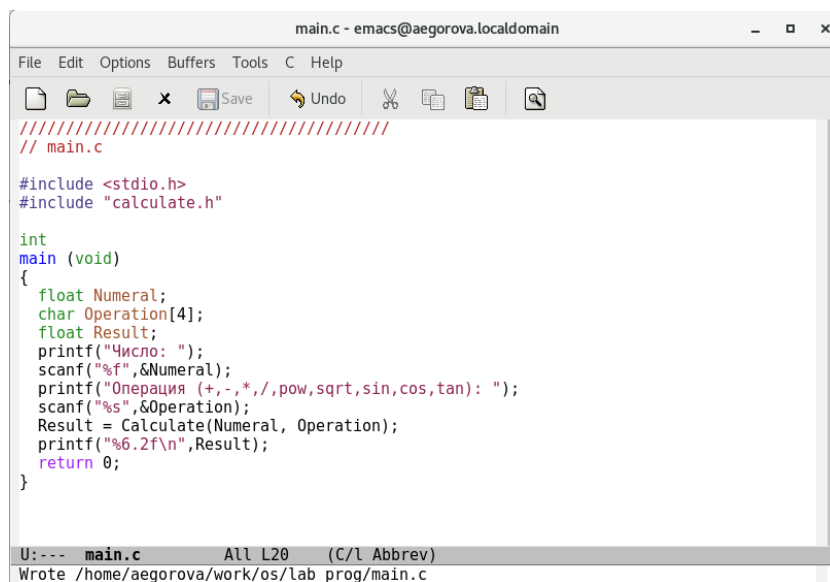
#ifndef CALCULATE_H_
#define CALCULATE_H_

float Calculate(float Numeral, char Operation[4]);

#endif /*CALCULATE_H_*/
█
U:--- calculate.h All L10 (C/l Abbrev)
```

Figure 3.5: Файл calculate.h

Основной файл main.c , реализующий интерфейс пользователя к калькулятору.  
(рис. -fig. 3.6)



```
main.c - emacs@aegorova.localdomain
File Edit Options Buffers Tools C Help
////////////////////
// main.c

#include <stdio.h>
#include "calculate.h"

int
main (void)
{
    float Numeral;
    char Operation[4];
    float Result;
    printf("Число: ");
    scanf("%f",&Numeral);
    printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): ");
    scanf("%s",&Operation);
    Result = Calculate(Numeral, Operation);
    printf("%.2f\n",Result);
    return 0;
}
U:--- main.c All L20 (C/l Abbrev)
Wrote /home/aegorova/work/os/lab_prog/main.c
```

Figure 3.6: Файл main.c

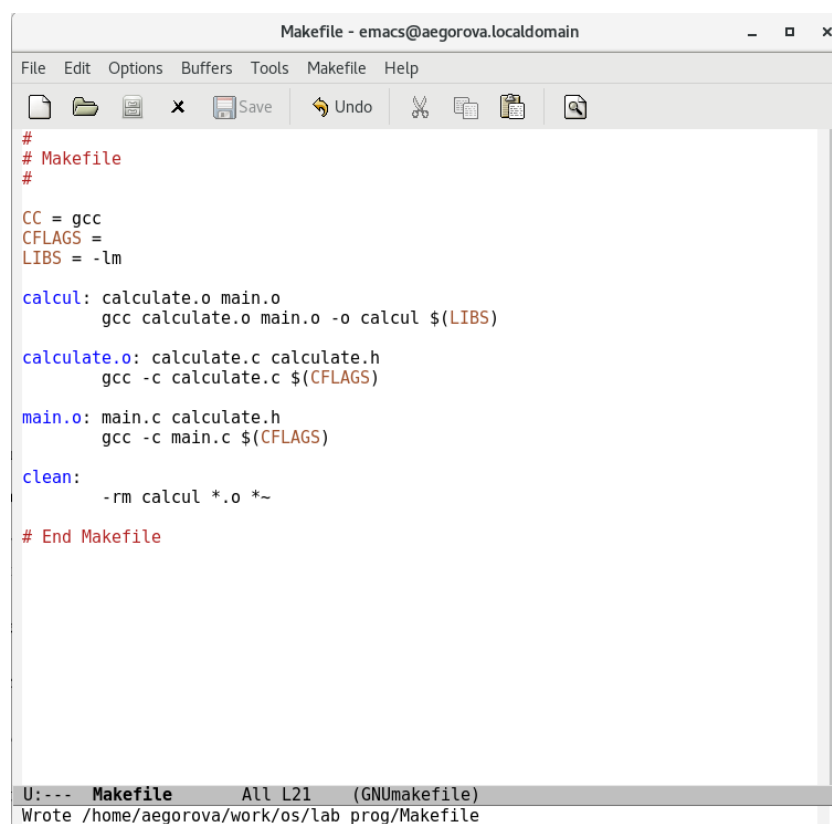
3) Выполните компиляцию программы посредством gcc : gcc -c calculate.c; gcc  
-c main.c; gcc calculate.o main.o -o calcul -lm. (рис. -fig. 3.7)

```
[aegorova@aegorova lab_prog]$ gcc -c calculate.c
[1] Done emacs
[2]- Done emacs
[3]+ Done emacs
[aegorova@aegorova lab_prog]$ gcc -c main.c
[aegorova@aegorova lab_prog]$ gcc calculate.o main.o -o calcul -lm
```

Figure 3.7: Компиляция программы



- 4) Синтаксические ошибки не найдены.
- 5) Создала Makefile с необходимым 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. 3.8)



```
# Makefile
#
CC = gcc
CFLAGS =
LIBS = -lm

calcul: calculate.o main.o
    gcc calculate.o main.o -o calcul $(LIBS)

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

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

clean:
    -rm calcul *.o *~

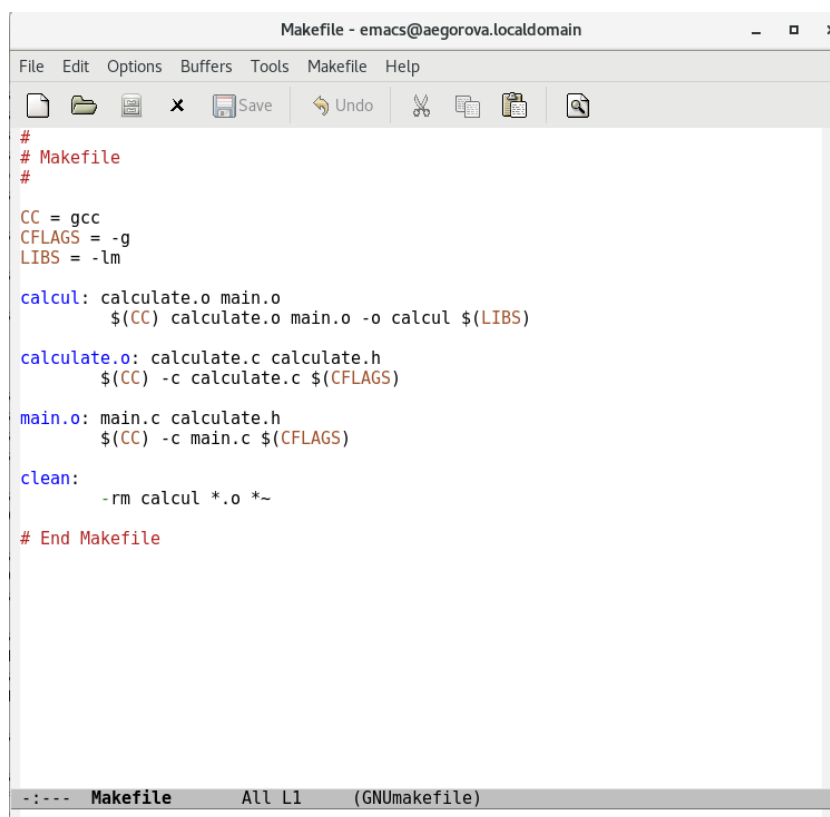
# End Makefile
```

U:--- Makefile All L21 (GNUmakefile)  
Wrote /home/aegorova/work/os/lab\_prog/Makefile

Figure 3.8: Создание Makefile

- 6) С помощью gdb выполняю отладку программы calcul (перед использованием gdb исправила 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. (рис. -fig. 3.9)



```
#
# Makefile
#
CC = gcc
CFLAGS = -g
LIBS = -lm

calcul: calculate.o main.o
    $(CC) calculate.o main.o -o calcul $(LIBS)

calculate.o: calculate.c calculate.h
    $(CC) -c calculate.c $(CFLAGS)

main.o: main.c calculate.h
    $(CC) -c main.c $(CFLAGS)

clean:
    -rm calcul *.o *~

# End Makefile
```

Figure 3.9: Исправление Makefile

После этого удалила исполняемые и объектные файлы из каталога с помощью команды «make clear». Выполнила компиляцию файлов. (рис. -fig. 3.10)

```
[aegorova@aegorova lab_prog]$ make clean
rm calcul *.o *~
[aegorova@aegorova lab_prog]$ make calculate.o
gcc -c calculate.c -g
[1]+  Done                  emacs
[aegorova@aegorova lab_prog]$ make main.o
gcc -c main.c -g
[aegorova@aegorova lab_prog]$ make calcul
gcc calculate.o main.o -o calcul -lm
```

Figure 3.10: Удаление исполняемых и объектных файлов

Запускаю отладчик GDB, загрузив в него программу для отладки. Для запуска программы внутри отладчика ввела команду run. (рис. -fig. 3.11)

```
[aegorova@aegorova lab_prog]$ gdb ./calcul
GNU gdb (GDB) Red Hat Enterprise Linux 7.6.1-120.el7
Copyright (C) 2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/aegorova/work/os/lab_prog/calcul...done.
(gdb) run
Starting program: /home/aegorova/work/os/lab_prog/./calcul
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): +
Второе слагаемое: 3
      8.00
[Inferior 1 (process 4536) exited normally]
```

Figure 3.11: Отладчик GDB

Для постраничного (по 9 строк) просмотра исходного код использую команду list. (рис. -fig. 3.12)

```
(gdb) list
4      #include <stdio.h>
5      #include "calculate.h"
6
7      int
8      main (void)
9      {
10     float Numeral;
11     char Operation[4];
12     float Result;
13     printf("Число: ");
```

Figure 3.12: list

Для просмотра строк с 12 по 15 основного файла использую list с параметрами. (рис. -fig. 3.13)

```
(gdb) list 12,15
12     float Result;
13     printf("Число: ");
14     scanf("%f",&Numeral);
15     printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): ");
```

Figure 3.13: list 12,15

Для просмотра определённых строк не основного файла использую list с параметрами. (рис. -fig. 3.14)

```
(gdb) list calculate.c:20,29
20     printf("Вычитаемое: ");
21     scanf("%f",&SecondNumeral);
22     return(Numeral - SecondNumeral);
23 }
24 else if(strncmp(Operation, "*", 1) == 0)
25 {
26     printf("Множитель: ");
27     scanf("%f",&SecondNumeral);
28     return(Numeral * SecondNumeral);
29 }
```

Figure 3.14: list calculate.c:20,29

Устанавливаю точку останова в файле calculate.c на строке номер 21 (list calculate.c:20,27; break 21). (рис. -fig. 3.15)

```
(gdb) list calculate.c:20,27
20     printf("Вычитаемое: ");
21     scanf("%f",&SecondNumeral);
22     return(Numeral - SecondNumeral);
23 }
24 else if(strncmp(Operation, "*", 1) == 0)
25 {
26     printf("Множитель: ");
27     scanf("%f",&SecondNumeral);
(gdb) break 21
Breakpoint 1 at 0x4007e7: file calculate.c, line 21.
```

Figure 3.15: Точка останова в файле

Вывела информацию об имеющихся в проекте точках останова. (рис. -fig. 3.16)

```
(gdb) info breakpoints
Num   Type             Disp Enb Address            What
1      breakpoint      keep y   0x00000000004007e7 in Calculate at calculate.c:21
```

Figure 3.16: Информация точках останова

Запустила программу внутри отладчика и убедилась, что программа остановилась в момент прохождения точки останова. (рис. -fig. 3.17)

```
(gdb) run
Starting program: /home/aegorova/work/os/lab_prog/./calcul
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): -
Breakpoint 1, Calculate (Numeral=5, Operation=0x7fffffffdd90 "-") at calculate.c:21
21     scanf("%f",&SecondNumeral);
(gdb) backtrace
#0 Calculate (Numeral=5, Operation=0x7fffffffdd90 "-") at calculate.c:21
#1 0x0000000000400a90 in main () at main.c:17
```

Figure 3.17: Проверка точки останова

Посмотрела, чему равно на этом этапе значение переменной Numeral. (рис. -fig. 3.18)

```
(gdb) print Numeral
$1 = 5
```

Figure 3.18: Значение переменной Numeral

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

```
(gdb) display Numeral
1: Numeral = 5
```

Figure 3.19: Значение переменной Numeral

Убрала точки останова. (рис. -fig. 3.20)

```
(gdb) info breakpoints
Num   Type             Disp Enb Address          What
1     breakpoint        keep y   0x00000000004007e7 in calculate at calculate.c:21
      breakpoint already hit 1 time
(gdb) delete 1
```

Figure 3.20: Убираем точки останова

7) С помощью утилиты splint попробовала проанализировать коды файлов calculate.c и main.c. С помощью утилиты splint выяснилось, что в файлах calculate.c и main.c присутствует функция чтения scanf, возвращающая целое число, но эти числа не используются и нигде не сохраняются. 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 потере данных. (рис. -fig. 3.21) (рис. -fig. 3.22)

```
[aegorova@aegorova lab_prog]$ splint calculate.c
Splint 3.1.2 --- 11 Oct 2015

calculate.h:7:37: Function parameter Operation declared as manifest array (size
        constant is meaningless)
    A formal parameter is declared as an array with size. The size of the array
    is ignored in this context, since the array formal parameter is treated as a
    pointer. (Use -fixedformalarray to inhibit warning)
calculate.c:9:31: Function parameter Operation declared as manifest array (size
        constant is meaningless)
calculate.c: (in function Calculate)
calculate.c:15:7: Return value (type int) ignored: scanf("%f", &Sec...
    Result returned by function call is not used. If this is intended, can cast
    result to (void) to eliminate message. (Use -retvalint to inhibit warning)
calculate.c:21:7: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:27:7: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:33:7: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:34:10: Dangerous equality comparison involving float types:
        SecondNumeral == 0
    Two real (float, double, or long double) values are compared directly using
    == or != primitive. This may produce unexpected results since floating point
    representations are inexact. Instead, compare the difference to FLT_EPSILON
    or DBL_EPSILON. (Use -realcompare to inhibit warning)
calculate.c:37:10: Return value type double does not match declared type float:
        (HUGE_VAL)
```

Figure 3.21: splint

```
[aegorova@aegorova lab_prog]$ splint main.c
Splint 3.1.2 --- 11 Oct 2015

calculate.h:7:37: Function parameter Operation declared as manifest array (size
        constant is meaningless)
    A formal parameter is declared as an array with size. The size of the array
    is ignored in this context, since the array formal parameter is treated as a
    pointer. (Use -fixedformalarray to inhibit warning)
main.c: (in function main)
main.c:14:3: Return value (type int) ignored: scanf("%f", &Num...
    Result returned by function call is not used. If this is intended, can cast
    result to (void) to eliminate message. (Use -retvalint to inhibit warning)
main.c:16:14: Format argument 1 to scanf (%s) expects char * gets char [4] *:
        &Operation
    Type of parameter is not consistent with corresponding code in format string.
    (Use -formattype to inhibit warning)
    main.c:16:11: Corresponding format code
main.c:16:3: Return value (type int) ignored: scanf("%s", &Ope...

Finished checking --- 4 code warnings_
```

Figure 3.22: splint

## 4 Выводы

В ходе выполнения данной лабораторной работы 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.

## 5 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: 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 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.
- 3) 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 C, файлы с расширением .cc или .C – как файлы на языке C++, а файлы с 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».

- 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 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 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 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.

- 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 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`
- 8) 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 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`.

- 9) Схема 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 массива символов уже является указателем на первый элемент этого 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 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 значениями и типами и многое другое.

## 6 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 UNIX [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)