

# **Лабораторная работа № 14**

**Операционные системы**

Гусева Светлана Алексеевна

# **Содержание**

<b>1 Цель работы</b>	<b>5</b>
<b>2 Задание</b>	<b>6</b>
<b>3 Выполнение лабораторной работы</b>	<b>12</b>
<b>4 Выводы</b>	<b>24</b>
<b>5 Контрольные вопросы</b>	<b>25</b>

## **Список таблиц**

# Список иллюстраций

3.1 Создание подкаталогов и файлов. . . . .	12
3.2 Файл calculate.c (начало). . . . .	13
3.3 Файл calculate.c (конец). . . . .	13
3.4 Файл calculate.h. . . . .	14
3.5 Файл main.c. . . . .	14
3.6 Выполнение компиляции программы посредством gcc. . . . .	15
3.7 Makefile. . . . .	15
3.8 Запуск отладчика GDB. . . . .	16
3.9 Выполнение команды run. . . . .	17
3.10 Выполнение make и make clean. . . . .	17
3.11 Выполнение команды list. . . . .	18
3.12 Выполнение команды list 12,15. . . . .	18
3.13 Выполнение команды list calculate.c:20,29. . . . .	19
3.14 Установление точки останова в файле calculate.c на строке номер 21. . . . .	19
3.15 Выведение информации об имеющихся в проекте точках останова. . . . .	20
3.16 Остановка программы в момент прохождения точки останова. . . . .	20
3.17 Значение переменной Numeral. . . . .	21
3.18 Результат выполнения команды display Numeral. . . . .	21
3.19 Удаление точки останова. . . . .	22
3.20 Установка утилиты splint. . . . .	22
3.21 Анализ кода файла calculate.c. . . . .	23
3.22 Анализ кода файла main.c. . . . .	23

# **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.

Реализация функций калькулятора в файле calculate.h:

```
///////////
// calculate.c

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

float
Calculate(float Numeral, char Operation[4])
{
    float SecondNumeral;
    if(strncmp(Operation, "+", 1) == 0)
    {
        printf("Второе слагаемое: ");
        scanf("%f", &SecondNumeral);
    }
}
```

```
return(Numerical + SecondNumerical);

}

else if(strncmp(Operation, "-", 1) == 0)
{
printf("Вычитаемое: ");
scanf("%f", &SecondNumerical);
return(Numerical - SecondNumerical);
}

else if(strncmp(Operation, "*", 1) == 0)
{
printf("Множитель: ");
scanf("%f", &SecondNumerical);
return(Numerical * SecondNumerical);
}

else if(strncmp(Operation, "/", 1) == 0)
{
printf("Делитель: ");
scanf("%f", &SecondNumerical);
if(SecondNumerical == 0)
{
printf("Ошибка: деление на ноль! ");
return(HUGE_VAL);
}
else
return(Numerical / SecondNumerical);
}

else if(strncmp(Operation, "pow", 3) == 0)
{
printf("Степень: ");
}
```

```

scanf("%f", &SecondNumeral);

return(pow(Numerical, SecondNumeral));
}

else if(strncmp(Operation, "sqrt", 4) == 0)
return(sqrt(Numerical));

else if(strncmp(Operation, "sin", 3) == 0)
return(sin(Numerical));

else if(strncmp(Operation, "cos", 3) == 0)
return(cos(Numerical));

else if(strncmp(Operation, "tan", 3) == 0)
return(tan(Numerical));

else
{
printf("Неправильно введено действие ");
return(HUGE_VAL);
}
}

```

Интерфейсный файл calculate.h, описывающий формат вызова функции калькулятора:

```

///////////
// calculate.h

#ifndef CALCULATE_H_
#define CALCULATE_H_

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

#endif /*CALCULATE_H_*/

```

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

```

///////////
// 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("%6.2f\n",Result);
return 0;
}

```

3. Выполните компиляцию программы посредством gcc:

```

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

```

4. При необходимости исправьте синтаксические ошибки.

5. Создайте Makefile со следующим содержанием:

```

#
# 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
```

Поясните в отчёте его содержание. 6. С помощью gdb выполните отладку программы calcul (перед использованием gdb исправьте Makefile):

- Запустите отладчик GDB, загрузив в него программу для отладки: `gdb ./calcul`
- Для запуска программы внутри отладчика введите команду `run: run`
- Для постраничного (по 9 строк) просмотра исходного кода используйте команду `list: list`
- Для просмотра строк с 12 по 15 основного файла используйте `list` с параметрами: `list 12,15`
- Для просмотра определённых строк не основного файла используйте `list` с параметрами: `list calculate.c:20,29`

– Установите точку останова в файле calculate.c на строке номер 21: `list calculate.c:20,27`  
`break 21`  
– Выведите информацию об имеющихся в проекте точка останова: `info breakpoints`  
– Запустите программу внутри отладчика и убедитесь, что программа остановится в момент прохождения точки останова:

`run`

`5`

•

`backtrace`

- Отладчик выдаст следующую информацию:

```
#0 Calculate (Numeral=5, Operation=0x7fffffff280 "-")
```

```
at calculate.c:21
```

```
#1 0x0000000000400b2b in main () at main.c:17
```

а команда backtrace покажет весь стек вызываемых функций от начала программы до текущего места.

– Посмотрите, чему равно на этом этапе значение переменной Numeral, введя: print Numeral  
На экран должно быть выведено число 5.

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

– Уберите точки останова:

```
info breakpoints
```

```
delete 1
```

7. С помощью утилиты splint попробуйте проанализировать коды файлов calculate.c и main.c.

### 3. Выполнение лабораторной работы

В домашнем каталоге создан подкаталог ~/work/os/lab\_prog и файлы calculate.h, calculate.c, main.c. (рис. 3.1, рис. 3.2, рис. 3.3, рис. 3.4, рис. 3.5).

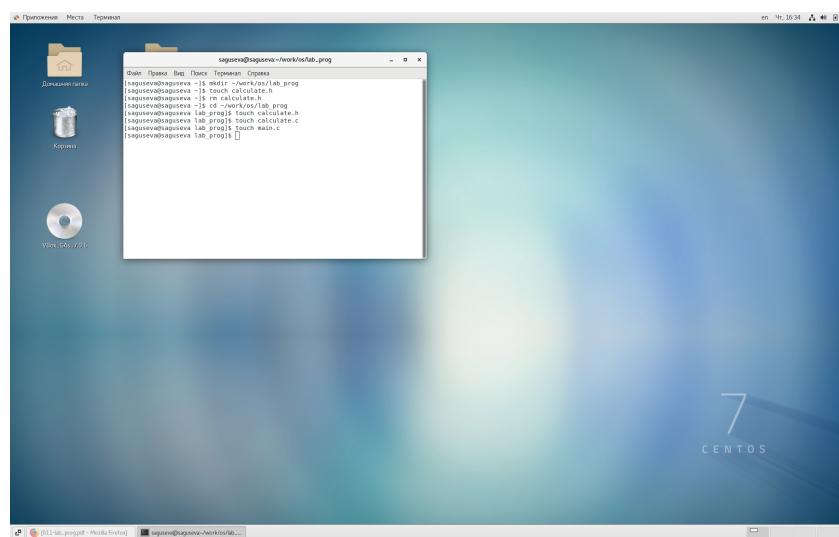


Рис. 3.1: Создание подкаталогов и файлов.

```

#include <math.h>
#include <math.h>
#include <iostream.h>
#include <string.h>

float Calculate(float Numeral, char Operation[4])
{
    float SecondNumeral;
    if(strcmp(Operation, "+", 1) == 0)
    {
        printf("Введи первое значение: ");
        scanf("%f", &SecondNumeral);
        return(Numeral + SecondNumeral);
    }
    else if(strcmp(Operation, "-", 1) == 0)
    {
        printf("Введи первое значение: ");
        scanf("%f", &SecondNumeral);
        return(Numeral - SecondNumeral);
    }
    else if(strcmp(Operation, "*", 1) == 0)
    {
        printf("Введи первое значение: ");
        scanf("%f", &SecondNumeral);
        return(Numeral * SecondNumeral);
    }
    else if(strcmp(Operation, "/", 1) == 0)
    {
        printf("Введи первое значение: ");
        scanf("%f", &SecondNumeral);
        if(SecondNumeral == 0)
        {
            printf("Ошибка! деление на ноль! ");
            return(HUGE_VAL);
        }
        else
            return(Numeral / SecondNumeral);
    }
    else if(strcmp(Operation, "pow", 3) == 0)
    {
        printf("Введи первое значение: ");
        scanf("%f", &SecondNumeral);
        if(SecondNumeral == 0)
        {
            printf("Ошибка! деление на ноль! ");
            return(HUGE_VAL);
        }
        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("Неверно введено действие ");
            return(HUGE_VAL);
        }
    }
}

```

Рис. 3.2: Файл calculate.c (начало).

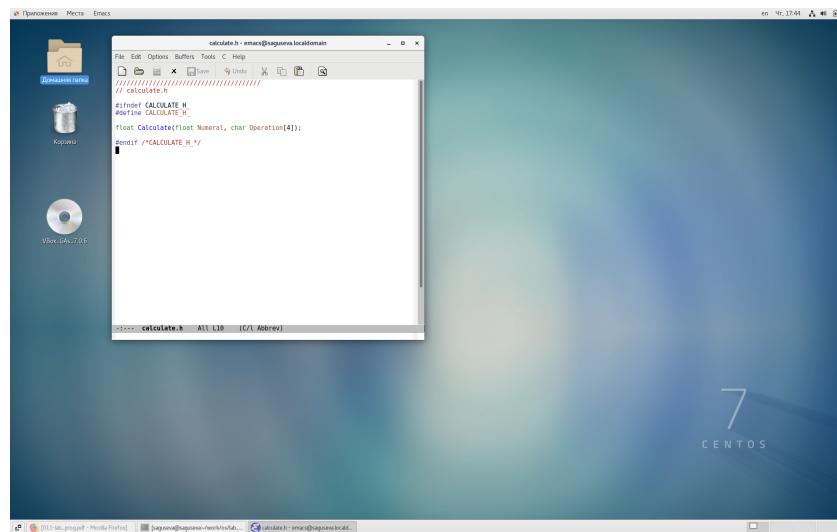
```

#include <math.h>
#include <iostream.h>
#include "calculate.h"

float Calculate(float Numeral, char Operation[4])
{
    float SecondNumeral;
    if(strcmp(Operation, "+", 1) == 0)
    {
        printf("Введи первое значение: ");
        scanf("%f", &SecondNumeral);
        return(Numeral + SecondNumeral);
    }
    else if(strcmp(Operation, "-", 1) == 0)
    {
        printf("Введи первое значение: ");
        scanf("%f", &SecondNumeral);
        return(Numeral - SecondNumeral);
    }
    else if(strcmp(Operation, "*", 1) == 0)
    {
        printf("Введи первое значение: ");
        scanf("%f", &SecondNumeral);
        return(Numeral * SecondNumeral);
    }
    else if(strcmp(Operation, "/", 1) == 0)
    {
        printf("Введи первое значение: ");
        scanf("%f", &SecondNumeral);
        if(SecondNumeral == 0)
        {
            printf("Ошибка! деление на ноль! ");
            return(HUGE_VAL);
        }
        else
            return(Numeral / SecondNumeral);
    }
    else if(strcmp(Operation, "pow", 3) == 0)
    {
        printf("Введи первое значение: ");
        scanf("%f", &SecondNumeral);
        if(SecondNumeral == 0)
        {
            printf("Ошибка! деление на ноль! ");
            return(HUGE_VAL);
        }
        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("Неверно введено действие ");
            return(HUGE_VAL);
        }
    }
}

```

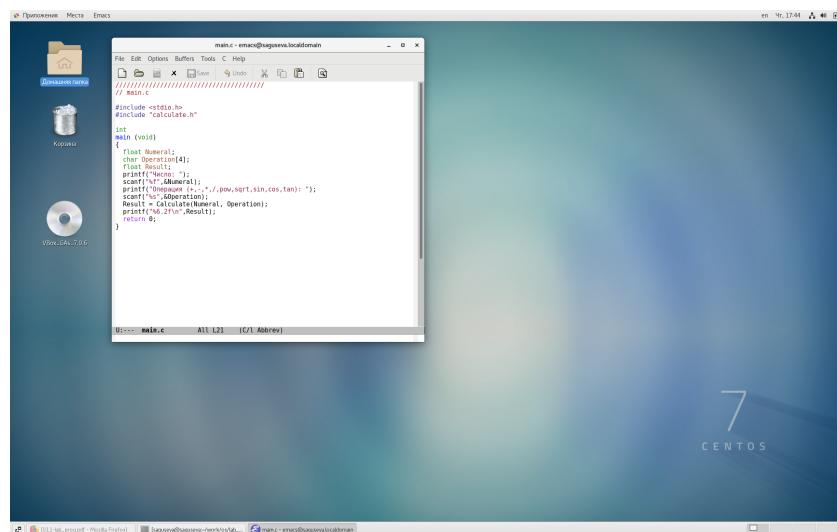
Рис. 3.3: Файл calculate.c (конец).



```
#include <math.h>
#include "calculate.h"

float Calculate(float Numeral, char Operation[4]);
#endif /*CALCULATE_H_*/
```

Рис. 3.4: Файл calculate.h.



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

main()
{
    float Numeral;
    char Operation[4];
    printf("Enter Numeral: ");
    scanf("%f", &Numeral);
    printf("Enter Operation: ");
    scanf("%s", Operation);
    Result = Calculate(Numeral, Operation);
    printf("Result = %f\n", Result);
    return 0;
}
```

Рис. 3.5: Файл main.c.

Выполнение компиляции программы посредством gcc (рис. 3.6).

```

segovia@segovia:~/work/lab_prog
File Права Вид Диск Терминал Окна
[segovia@segovia ~]$ rm calculate.h
[segovia@segovia ~]$ gcc -c calculate.c
[segovia@segovia ~]$ touch calculate.h
[segovia@segovia ~]$ touch main.c
[segovia@segovia ~]$ gcc -c calculate.c
[segovia@segovia ~]$ gcc -c calculate.h -o calculate.o
[segovia@segovia ~]$ gcc -c calculate.c -o calculate.o
[segovia@segovia ~]$ gcc -c main.c -o main.o
[segovia@segovia ~]$ gcc calculate.o main.o -o calcul -lm
[segovia@segovia ~]$ 

```

Рис. 3.6: Выполнение компиляции программы посредством gcc.

Создание Makefile и исправление ошибок (рис. 3.7).

```

Makefile - emacs@segovia.localdomain
File Edit Options Buffers Tools Makefile Help
[Makefile]
CC = gcc
CFLAGS = -g
LIBS = -lm
calcul: calculate.o main.o
    $(CC) calculate.o main.o -o calcul $(LIBS)
calculate: calculate.c calculate.h
    $(CC) -c calculate.c $(CFLAGS)
main: main.c calculate.h
    $(CC) -c main.c $(CFLAGS)
clean:
    rm calcul *.o *
End Makefile

```

Рис. 3.7: Makefile.

Комментарии к Makefile:

**CC = gcc #компилятор**

**CFLAGS = -g #опция gcc для создания отладочной информации**

**LIBS = -lm**

```

#Создание файла calcul
calcul: calculate.o main.o

$(CC) calculate.o main.o -o calcul $(LIBS) #gcc calculate.o main.o -
o calcul -lm

#Создание файла calculate.o

calculate.o: calculate.c calculate.h #

$(CC) -c calculate.c $(CFLAGS) #gcc -c calculate.c -g

#Создание файла main.o

main.o: main.c calculate.h

$(CC) -c main.c $(CFLAGS) #gcc -c main.c -g

#
clean: #удаление всех файлов с разрешением .o

-rm calcul *.o *~ #

```

Запуск отладчика GDB и загрузка в него программы для отладки (рис. 3.8).

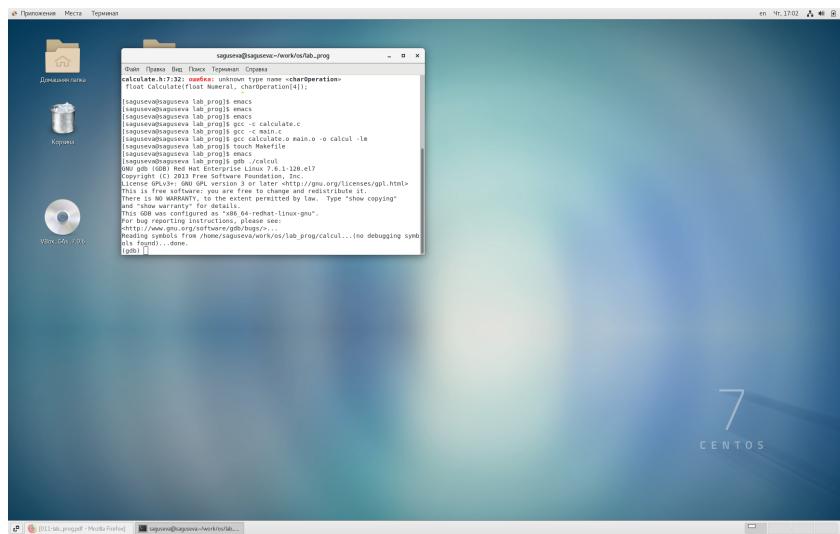


Рис. 3.8: Запуск отладчика GDB.

Для запуска программы внутри отладчика была введена команда run (рис. 3.9).

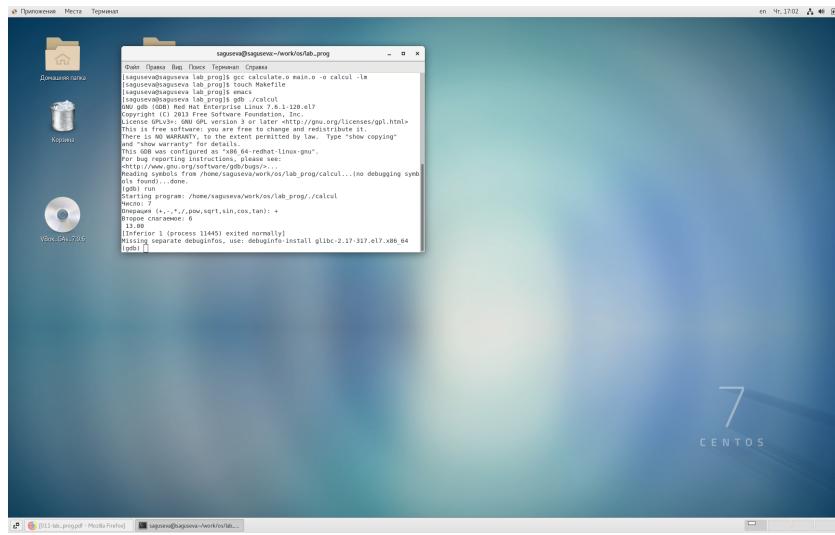


Рис. 3.9: Выполнение команды run.

Выполнение make и make clean (рис. 3.10).

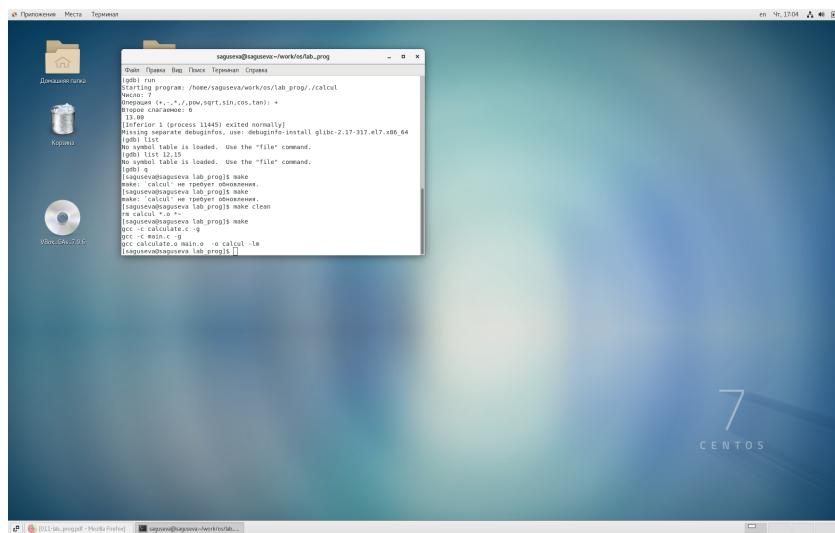


Рис. 3.10: Выполнение make и make clean.

Для постраничного (по 9 строк) просмотра исходного кода была использована команда `list` (рис. 3.11).

The screenshot shows a Linux desktop environment with a blue gradient background featuring the 'CENTOS 7' logo. In the foreground, a terminal window is open with the following text:

```
segurina@segurina:~/work/lab_prog
prc calculate.o main.o -o calcul -le
Copyright (C) 2009 Free Software Foundation, Inc.
GNU gdb (GDB) Red Hat Enterprise Linux 7.4-1.128.el7
Copyright (C) 2014 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This software is distributed under the terms of the GNU General Public License
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
for bug reporting instructions.  Please see
http://www.gnu.org/licenses/gpl.html for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
For bug reporting instructions, please see:
http://www.gnu.org/bugs.html
Reading symbols from /home/segurina/work/os/lab_prog/calcul...done.
(gdb) list
1
2
3
4
5
6
7
8
main (void)
9
10    float Numerical;
11    float Result;
12
13    printf("Value: ");
14    scanf("%f", &Numerical);
15
16    Result = sqrt(sin(cos(tan(Numerical))));
17
18    printf("Result: %f", Result);
19
20(gdb)
```

Рис. 3.11: Выполнение команды list.

Для просмотра строк с 12 по 15 основного файла была использована команда list с параметрами 12,15 (рис. 3.12).

The screenshot shows a Linux desktop environment with a blue gradient background featuring the 'CENTOS 7' logo. In the foreground, a terminal window is open with the following text:

```
segurina@segurina:~/work/lab_prog
prc calculate.o main.o -o calcul -le
Copyright (C) 2009 Free Software Foundation, Inc.
GNU gdb (GDB) Red Hat Enterprise Linux 7.4-1.128.el7
Copyright (C) 2014 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This software is distributed under the terms of the GNU General Public License
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
for bug reporting instructions.  Please see
http://www.gnu.org/bugs.html
This GDB was configured as "x86_64-redhat-linux-gnu".
For bug reporting instructions, please see:
http://www.gnu.org/bugs.html
Reading symbols from /home/segurina/work/os/lab_prog/calcul...done.
(gdb) list 12,15
12
13    float Result;
14
15    printf("Result: %f", Result);
16
17    Result = sqrt(sin(cos(tan(Numerical));
18
19    printf("Result: %f", Result);
20
21(gdb)
```

Рис. 3.12: Выполнение команды list 12,15.

Для просмотра определённых строк не основного файла была использована команда list с параметрами calculate.c:20,29 (рис. 3.13).

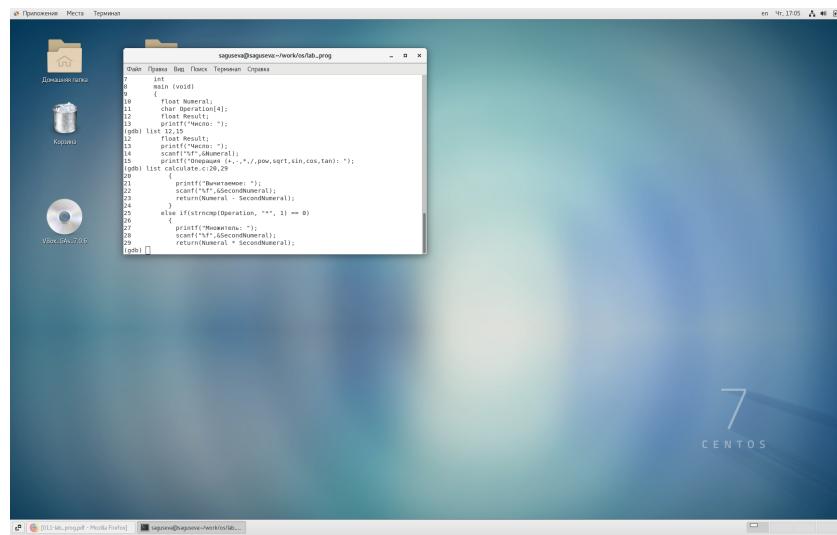


Рис. 3.13: Выполнение команды list calculate.c:20,29.

Была установлена точка останова в файле calculate.c на строке номер 21 (рис. 3.14).

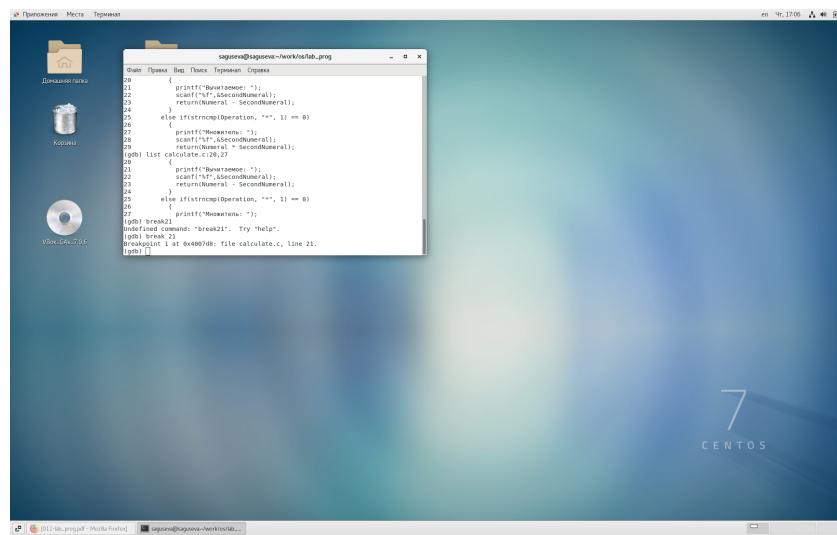


Рис. 3.14: Установление точки останова в файле calculate.c на строке номер 21.

Выведение информации об имеющихся в проекте точках останова (рис. 3.15).

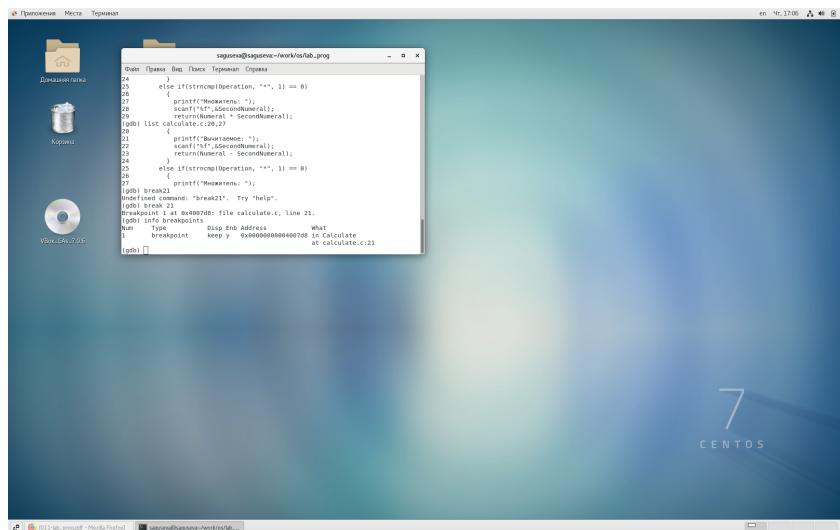


Рис. 3.15: Выведение информации об имеющихся в проекте точках останова.

Была запущена программа внутри отладчика. Программа остановилась в момент прохождения точки останова (рис. 3.16).

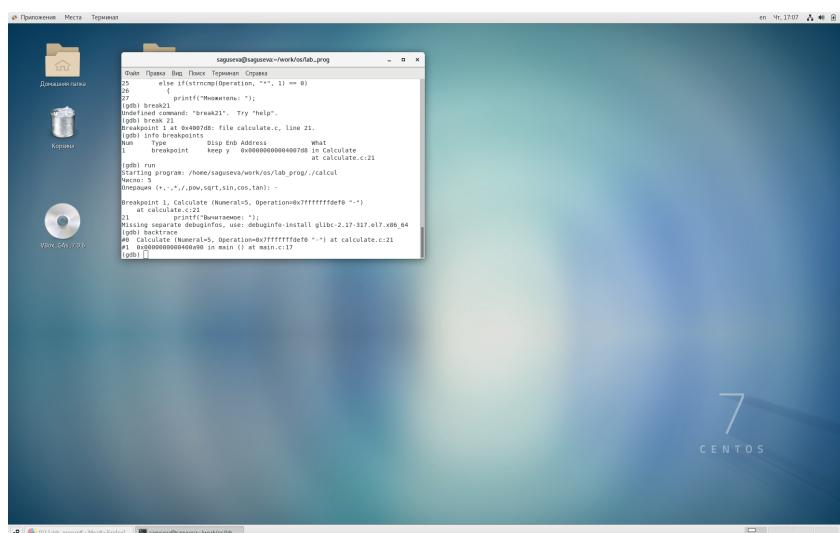


Рис. 3.16: Остановка программы в момент прохождения точки останова.

Просмотр значения переменной Numeral на этом этапе (рис. 3.17).

```

sugorov@sugorov:~/work/lab/lab_prog
Breakpoint 1 at 0x8048070: file calculate.c, line 21.
(gdb) run
Starting program: /home/sugorov/work/os/lab_prog./calculate
Numeral: 5
(gdb)

```

Рис. 3.17: Значение переменной Numeral.

Результат выполнения команды `display Numeral` (рис. 3.18).

```

sugorov@sugorov:~/work/lab/lab_prog
Breakpoint 1 at 0x8048070: file calculate.c, line 21.
(gdb) run
Starting program: /home/sugorov/work/os/lab_prog./calculate
Numeral: 5
(gdb) display Numeral
Numeral = 5
(gdb)

```

Рис. 3.18: Результат выполнения команды `display Numeral`.

Удаление точки останова (рис. 3.19).

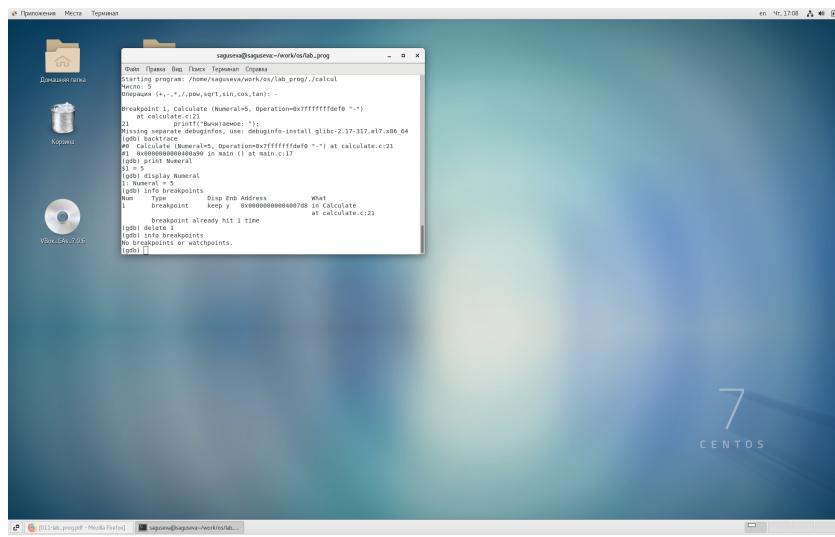


Рис. 3.19: Удаление точки останова.

Установка утилиты `splint` (рис. 3.20).

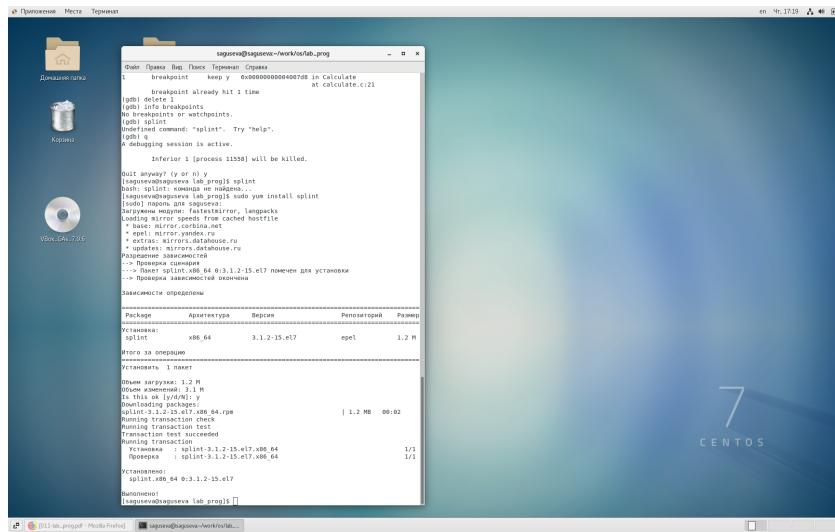


Рис. 3.20: Установка утилиты splint.

Анализ кода файла calculate.c с помощью утилиты split (рис. 3.21).

Рис. 3.21: Анализ кода файла calculate.c.

Анализ кода файла main.c с помощью утилиты split (рис. 3.22).

Рис. 3.22: Анализ кода файла main.c.

## **4. Выводы**

В процессе выполнения лабораторной работы мной были преобретены 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. Как получить информацию о возможностях программ gcc, make, gdb и др.?

С помощью команд man и info.

2. Назовите и дайте краткую характеристику основным этапам разработки приложений в UNIX.

- создание исходного кода программы;
- представляется в виде файла
- сохранение различных вариантов исходного текста;
- анализ исходного текста;
- компиляция исходного текста и построение исполняемого модуля;
- тестирование и отладка;
- проверка кода на наличие ошибок
- сохранение всех изменений, выполняемых при тестировании и отладке.

3. Что такое суффикс в контексте языка программирования? Приведите примеры использования.

Суффикс - это составная часть имени файла. Например, в имени файла calculate.c, ".c" - это суффикс.

4. Каково основное назначение компилятора языка С в UNIX?

Основное назначение компилятора с языка Си заключается в компиляции всей программы в целом и получении исполняемого модуля.

5. Для чего предназначена утилита make?

Для автоматизации преобразования файлов из одной формы в другую.

6. Приведите пример структуры Makefile. Дайте характеристику основным элементам этого файла.

```
CC = gcc #компилятор
CFLAGS = -g #опция gcc для создания отладочной информации
LIBS = -lm
#Создание файла calcul
calcul: calculate.o main.o
    $(CC) calculate.o main.o -o calcul $(LIBS) #gcc calculate.o main.o -o calcul -lm
#Создание файла calculate.o
calculate.o: calculate.c calculate.h #
    $(CC) -c calculate.c $(CFLAGS) #gcc -c calculate.c -g
#Создание файла main.o
main.o: main.c calculate.h
    $(CC) -c main.c $(CFLAGS) #gcc -c main.c -g
#
clean: #удаление всех файлов с разрешением .o
    -rm calcul *.o *~ #
```

7. Назовите основное свойство, присущее всем программам отладки. Что необходимо сделать, чтобы его можно было использовать?

Для отладки программ обычно применяют три способа:

1. Пошаговая отладка программ с заходом в подпрограммы;
2. Пошаговая отладка программ с выполнением подпрограммы как одного оператора;

### 3. Выполнение программы до точки останова.

Пошаговая отладка программ заключается в том, что выполняется один оператор программы и, затем контролируются те переменные, на которые должен был воздействовать данный оператор.

Если в программе имеются уже отлаженные подпрограммы, то подпрограмму можно рассматривать, как один оператор программы и воспользоваться вторым способом отладки программ.

Если в программе существует достаточно большой участок программы, уже отлаженный ранее, то его можно выполнить, не контролируя переменные, на которые он воздействует. Использование точек останова позволяет пропускать уже отложенную часть программы. Точка останова устанавливается в местах, где необходимо проверить содержимое переменных или просто проконтролировать, передаётся ли управление данному оператору.

Практически во всех отладчиках поддерживается это свойство (а также выполнение программы до курсора и выход из подпрограммы). Затем отладка программы продолжается в пошаговом режиме с контролем локальных и глобальных переменных, а также внутренних регистров микроконтроллера и напряжений на выводах этой микросхемы.

Следует отметить такое достоинство программы, как то, что она сразу показывает, где именно допущена та или иная ошибка, выделяя строку.

### 8. Назовите и дайте основную характеристику основным командам отладчика gdb.

break или b - создание точки останова;

info или i - вывести информацию, доступные значения: break, registers, frame, locals, args;

run или r - запустить программу;

continue или c - продолжить выполнение программы после точки останова;

step или s - выполнить следующую строчку программы с заходом в функцию;

next или n - выполнить следующую строчку без захода в функцию;

print или p - вывести значение переменной;

backtrace или bt - вывести стек вызовов;

x - просмотр содержимого памяти по адресу;

`ptype` - просмотр типа переменной;  
`h` или `help` - просмотр справки по команде;  
`q` или `quit` - выход из программы.

9. Опишите по шагам схему отладки программы, которую Вы использовали при выполнении лабораторной работы.

- 1) компиляция программы;
- 2) исправление ошибок в программе;
- 3) загрузка программы в `gdb`;
- 4) выполнение программы отладчиком.

10. Прокомментируйте реакцию компилятора на синтаксические ошибки в программе при его первом запуске.

Ошибки отсутствуют.

11. Назовите основные средства, повышающие понимание исходного кода программы.

- `cscope` - исследование функций, содержащихся в программе;
- `splint` — критическая проверка программ, написанных на языке Си.

12. Каковы основные задачи, решаемые программой `splint`.

- 1) Проверка корректности задания аргументов всех использованных в программе функций, а также типов возвращаемых ими значений;
- 2) Поиск фрагментов исходного текста, корректных с точки зрения синтаксиса языка Си, но малоэффективных с точки зрения их реализации или содержащих в себе семантические ошибки;
- 3) Общая оценка мобильности пользовательской программы.