

**РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ**

**Факультет физико-математических и естественных наук**

**ОТЧЕТ**

**ПО ЛАБОРАТОРНОЙ РАБОТЕ № 14**

*дисциплина: Операционные системы*

*тема: Средства, применяемые при разработке программного  
обеспечения в ОС типа UNIX/Linux*

Подготовила: Голощапова И.Б.

Группа: НФИбд\_01-20

## 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.

## Библиография

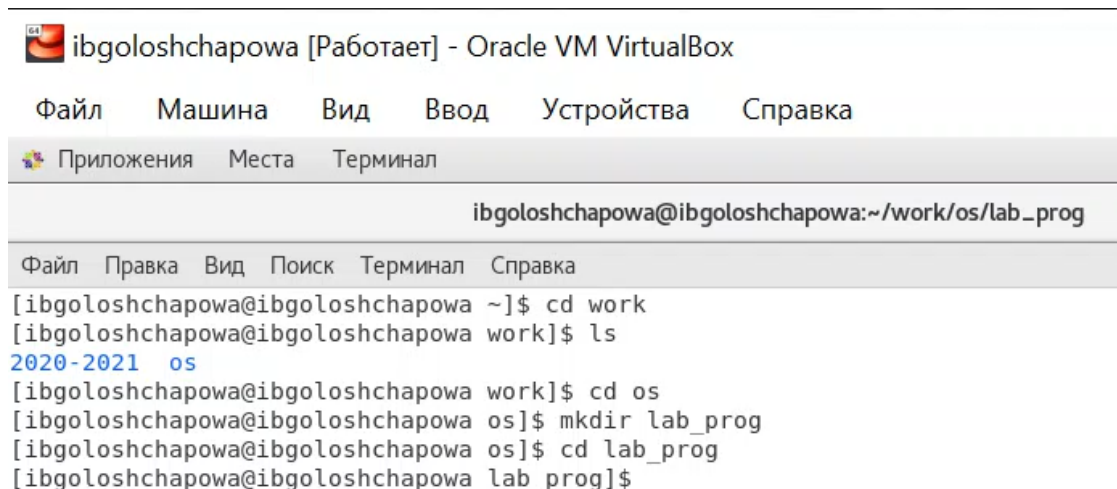
[Командные файлы в Linux](#)

[Википедия - Командная оболочка](#)

[Splint](#)

## Выполнение лабораторной работы

1. В домашнем каталоге создала подкаталог `~/work/os/lab_prog`.

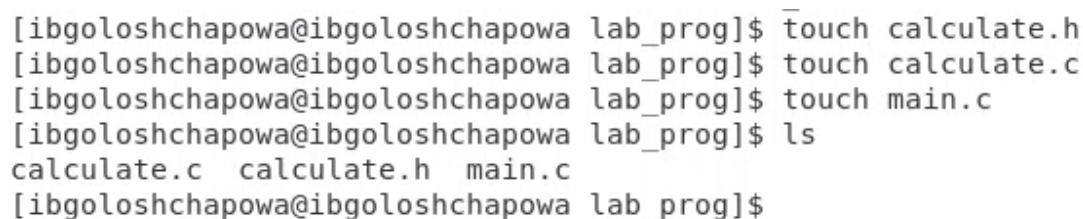


The screenshot shows the Oracle VM VirtualBox interface. The title bar reads "ibgoloshchapowa [Работает] - Oracle VM VirtualBox". Below the title bar is a menu bar with "Файл", "Машина", "Вид", "Ввод", "Устройства", and "Справка". Below the menu bar is a toolbar with "Приложения", "Места", and "Терминал". The main window displays a terminal window with the prompt "ibgoloshchapowa@ibgoloshchapowa:~/work/os/lab\_prog". The terminal shows the following commands and output:

```
Файл  Правка  Вид  Поиск  Терминал  Справка
[ibgoloshchapowa@ibgoloshchapowa ~]$ cd work
[ibgoloshchapowa@ibgoloshchapowa work]$ ls
2020-2021  os
[ibgoloshchapowa@ibgoloshchapowa work]$ cd os
[ibgoloshchapowa@ibgoloshchapowa os]$ mkdir lab_prog
[ibgoloshchapowa@ibgoloshchapowa os]$ cd lab_prog
[ibgoloshchapowa@ibgoloshchapowa lab_prog]$
```

Рис.1 "Создание подкаталога"

2. Создала 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.



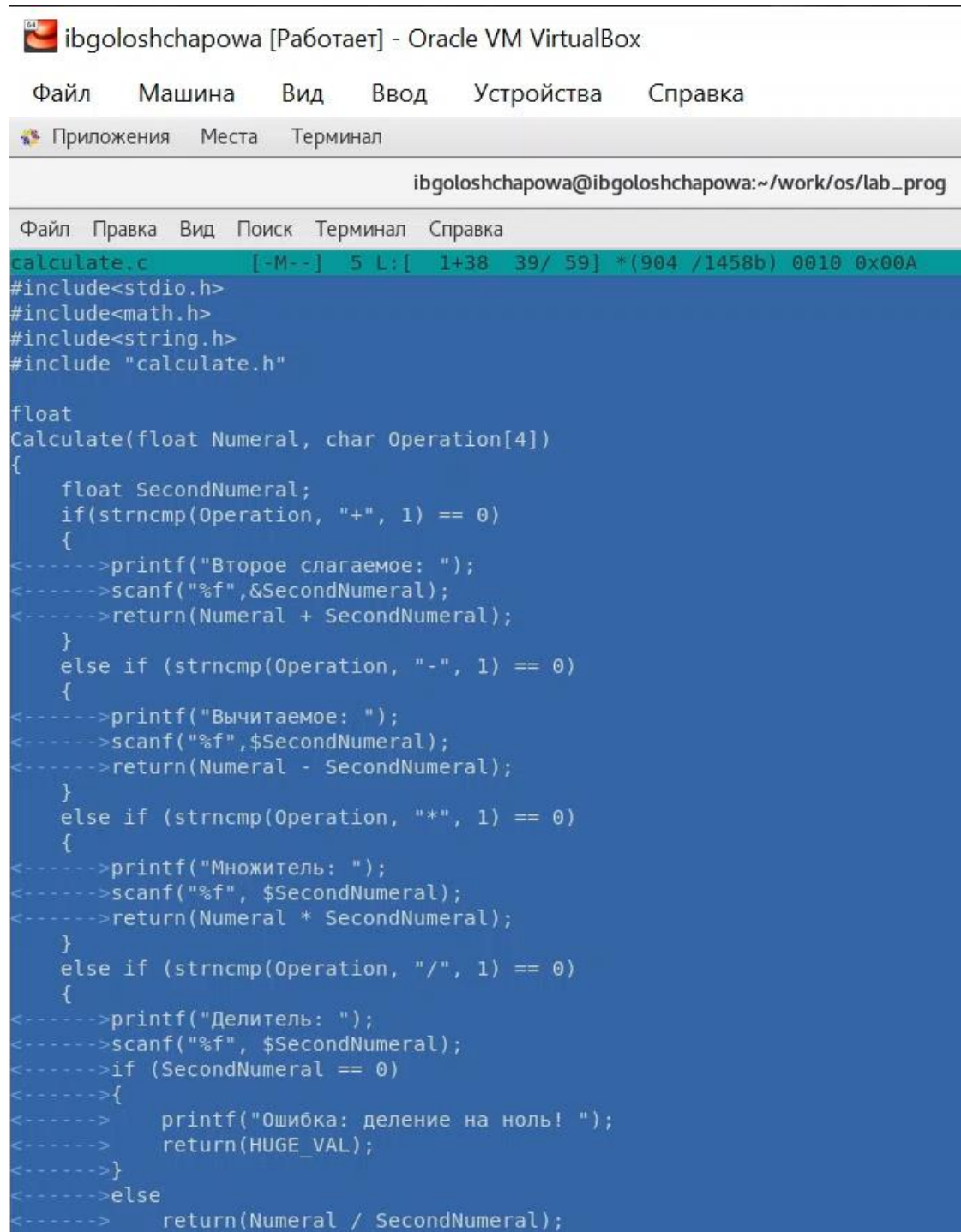
The screenshot shows a terminal window with the prompt "ibgoloshchapowa@ibgoloshchapowa lab\_prog". The terminal shows the following commands and output:

```
[ibgoloshchapowa@ibgoloshchapowa lab_prog]$ touch calculate.h
[ibgoloshchapowa@ibgoloshchapowa lab_prog]$ touch calculate.c
[ibgoloshchapowa@ibgoloshchapowa lab_prog]$ touch main.c
[ibgoloshchapowa@ibgoloshchapowa lab_prog]$ ls
calculate.c calculate.h main.c
[ibgoloshchapowa@ibgoloshchapowa lab_prog]$
```

Рис.2 "Создание файлов"

Реализовала функции калькулятора в файле.

В каждом из трех файлов прописала код на языке программирования C:



The screenshot shows a terminal window titled "ibgoloshchapowa [Работает] - Oracle VM VirtualBox". The window has a menu bar with "Файл", "Машина", "Вид", "Ввод", "Устройства", and "Справка". Below the menu bar is a toolbar with "Приложения", "Места", and "Терминал". The terminal prompt is "ibgoloshchapowa@ibgoloshchapowa:~/work/os/lab\_prog". The terminal content shows the code for "calculate.c" with line numbers 1 to 59. The code includes headers for stdio, math, and string, and defines a function "Calculate" that takes a float "Numeral" and a char array "Operation" of size 4. The function implements addition, subtraction, multiplication, and division, with error handling for division by zero.

```
calculate.c [-M--] 5 L:[ 1+38 39/ 59] *(904 /1458b) 0010 0x00A
#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(Numeral + SecondNumeral);
    }
    else if (strncmp(Operation, "-", 1) == 0)
    {
<----->printf("Вычитаемое: ");
<----->scanf("%f",$SecondNumeral);
<----->return(Numeral - SecondNumeral);
    }
    else if (strncmp(Operation, "*", 1) == 0)
    {
<----->printf("Множитель: ");
<----->scanf("%f", $SecondNumeral);
<----->return(Numeral * SecondNumeral);
    }
    else if (strncmp(Operation, "/", 1) == 0)
    {
<----->printf("Делитель: ");
<----->scanf("%f", $SecondNumeral);
<----->if (SecondNumeral == 0)
<----->{
<----->    printf("Ошибка: деление на ноль! ");
<----->    return(HUGE_VAL);
<----->}
<----->else
<----->    return(Numeral / SecondNumeral);
    }
```

Рис.3 "Файл calculate.c"

```

    }
    else if (strncmp(Operation, "pow", 3) == 0)
    {
<----->printf("Степень: ");
<----->scanf("%f", $SecondNumeral);
<----->return(pow(Numeral, SecondNumeral));
    }
    else if (strncmp(Operaration, "sqrt", 4) == 0)
<----->return(sqrt(Numeral));
    else if (strncmp(Operation, "sin", 3) == 0)
<----->return(sin(Numeral));
    else if (strncmp(Operation, "cos", 3) == 0)
<----->return(cos(Numeral));
    else if (strncmp(Operation, "tan", 3) == 0)
<----->return(tan(Numeral));
    else
    {
<----->printf("Неправильно введено действие ");
<----->return(HUGE_VAL);
    }
}

```

Рис.4 "Файл calculate.c (продолжение кода)"

The screenshot shows the Oracle VM VirtualBox interface. The title bar reads "ibgoloshchapowa [Работает] - Oracle VM VirtualBox". Below the title bar are two menus: "Файл" (File) and "Машина" (Machine). The main window has a tab labeled "Терминал" (Terminal). The terminal shows the command prompt "ibgoloshchapowa@ibgoloshchapowa:~/work/os/lab\_prog". Below the terminal, there is a code editor window showing the contents of "calculate.h" and "calculate.c". The code in "calculate.h" includes a header guard, a function prototype for "Calculate", and a comment. The code in "calculate.c" includes the same header guard and the implementation of the "Calculate" function, which uses a switch statement to perform various mathematical operations based on the input string.

```

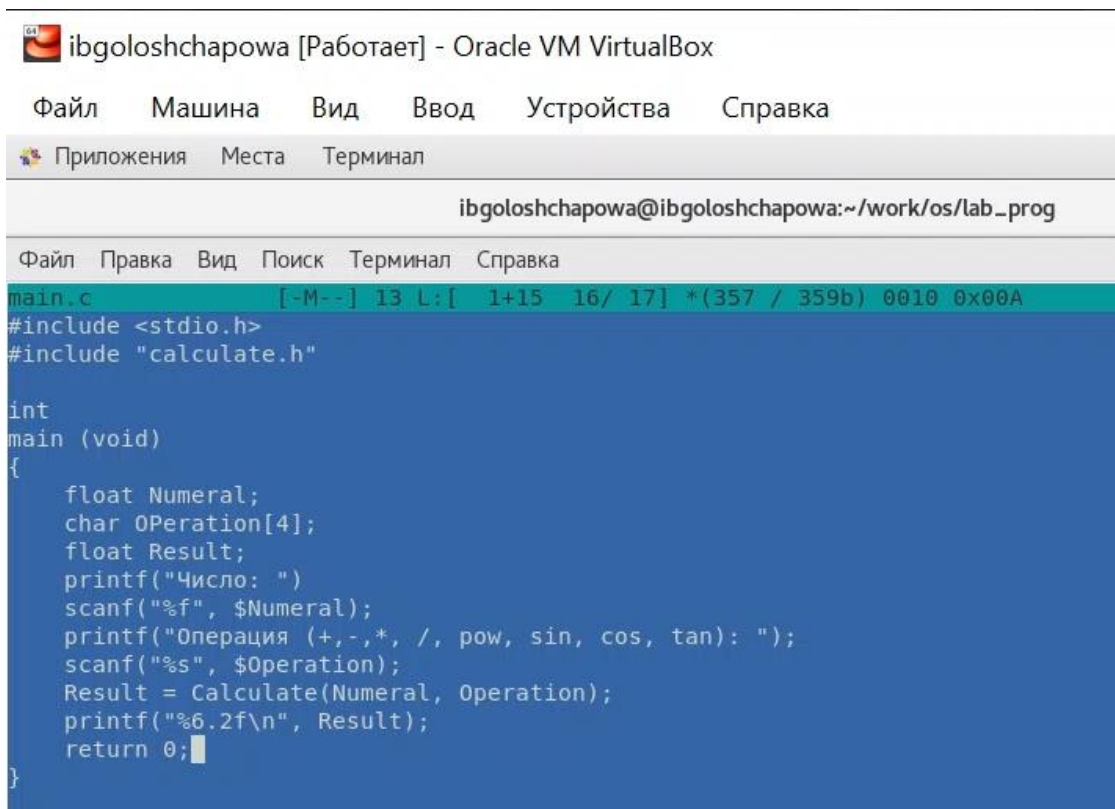
calculate.h      [-----] 23 L:[ 1+ 5  6/  6] *(118 / 118b) <EOF>
#ifndef CALCULATE_H
#define CALCULATE_H

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

#endif /*CALCULATE_H_*/

```

Рис.5 "Файл calculate.p"



```
ibgoloshchapowa [Работает] - Oracle VM VirtualBox
Файл  Машина  Вид  Ввод  Устройства  Справка
Приложения  Места  Терминал
ibgoloshchapowa@ibgoloshchapowa:~/work/os/lab_prog
Файл  Правка  Вид  Поиск  Терминал  Справка
main.c  [-M--] 13 L:[ 1+15 16/ 17] *(357 / 359b) 0010 0x00A
#include <stdio.h>
#include "calculate.h"

int
main (void)
{
    float Numeral;
    char OPeration[4];
    float Result;
    printf("Число: ")
    scanf("%f", $Numeral);
    printf("Операция (+,-,*, /, pow, sin, cos, tan): ");
    scanf("%s", $Operation);
    Result = Calculate(Numeral, Operation);
    printf("%.2f\n", Result);
    return 0;
}
```

Рис.6 "Файл main.c"

3. Выполнила компиляцию программы посредством gcc:

```
gcc -c calculate.c -ggdb
```

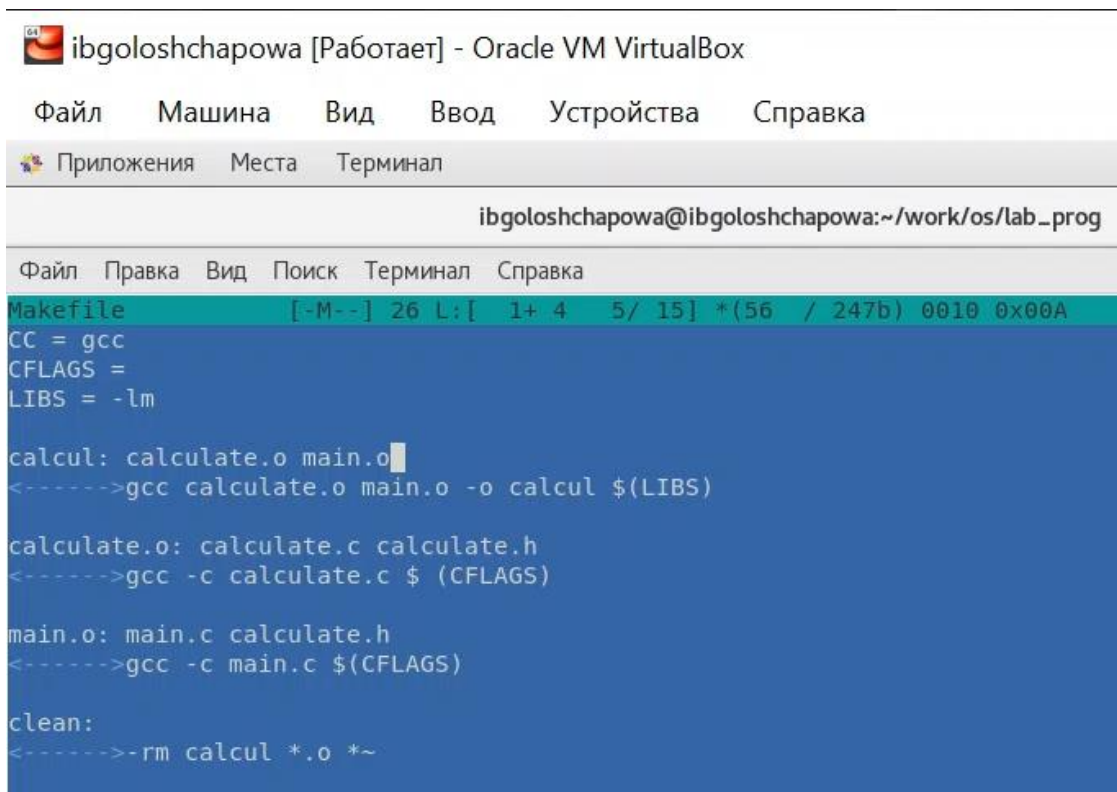
```
gcc -c main.c -ggdb
```

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

```
[ibgoloshchapowa@ibgoloshchapowa lab_prog]$ gcc -c calculate.c -ggdb
[ibgoloshchapowa@ibgoloshchapowa lab_prog]$ gcc -c main.c -ggdb
[ibgoloshchapowa@ibgoloshchapowa lab_prog]$ gcc calculate.o main.o -o calcul -lm -ggdb
[ibgoloshchapowa@ibgoloshchapowa lab_prog]$ gdb ./calcul
GNU gdb (GDB) Red Hat Enterprise Linux 7.6.1-120.el7
```

Рис.7 "Компиляция файлов"

4. Исправила имеющиеся синтаксические ошибки.
5. Создала Makefile со следующим содержанием:



The screenshot shows the Oracle VM VirtualBox interface. The title bar reads "ibgoloshchapowa [Работает] - Oracle VM VirtualBox". The menu bar includes "Файл", "Машина", "Вид", "Ввод", "Устройства", and "Справка". Below the menu bar is a toolbar with "Приложения", "Места", and "Терминал". The terminal window title is "ibgoloshchapowa@ibgoloshchapowa:~/work/os/lab\_prog". The terminal content shows a Makefile and compilation commands:

```
Makefile      [-M--] 26 L:[ 1+ 4 5/ 15] *(56 / 247b) 0010 0x00A
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 *
```

Рис.8 "Makefile"

6. С помощью gdb выполнила отладку программы calcul (перед использованием gdb исправила Makefile):

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

– Для запуска программы внутри отладчика ввела команду run:

```
[ibgoloshchapowa@ibgoloshchapowa 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/ibgoloshchapowa/work/os/lab_prog/calcul...done.
(gdb) run
Starting program: /home/ibgoloshchapowa/work/os/lab_prog/./calcul
Число: 6
Операция (+,-,*, /, pow, sin, cos, tan): /
Делитель: 3
2.00
[Inferior 1 (process 8903) exited normally]
Missing separate debuginfos, use: debuginfo-install glibc-2.17-317.el7.x86_64
(gdb)
```

Рис.9 "run"



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

```
1      #include <stdio.h>
2      #include "calculate.h"
3
4      int
5      main (void)
6      {
7          float Numeral;
8          char Operation[4];
9          float Result;
10         printf("Число: ");
(gdb)
```

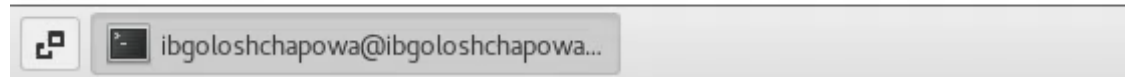


Рис.10 "list"

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

```
(gdb) list 12,15
12         printf("Операция (+,-,*, /, pow, sin, cos, tan): ");
13         scanf("%s", &Operation);
14         Result = Calculate(Numeral, Operation);
15         printf("%.2f\n", Result);
(gdb)
```

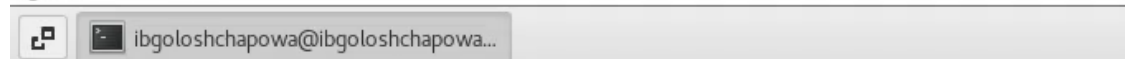


Рис.11 "list 12,15"

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

```

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

```



Рис.12 "list calculate.c:20,29"

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

list calculate.c:20,27

break 21

```

20         return(Numeral - SecondNumeral);
21     }
22     else if (strncmp(Operation, "*", 1) == 0)
23     {
24         printf("Множитель: ");
25         scanf("%f", &SecondNumeral);
26         return(Numeral * SecondNumeral);
27     }
(gdb) break 21
Breakpoint 1 at 0x400810: file calculate.c, line 21.
(gdb)

```

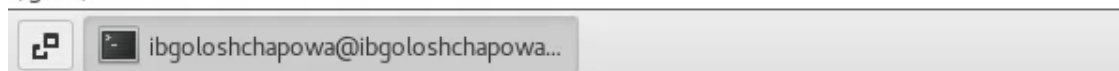


Рис.13 "break 21"

– Вывела информацию об имеющихся в проекте точка останова:

info breakpoints

```

(gdb) info breakpoints
Num   Type             Disp Enb Address                  What
1     breakpoint       keep y   0x0000000000400810 in Calculate at calculate.c:21
(gdb)

```

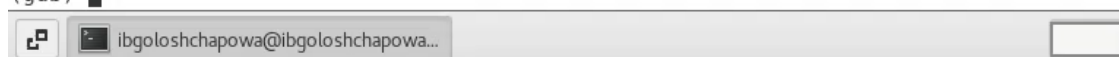


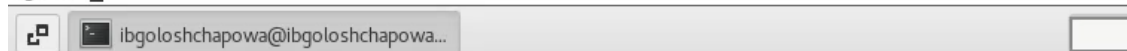
Рис.14 "info breakpoints"

– Запустила программу внутри отладчика и убедилась, что программа остановилась в момент прохождения точки останова:



```
(gdb) run
Starting program: /home/ibgoloshchapowa/work/os/lab_prog/./calcul
Число: 5
Операция (+,-,*, /, pow, sin, cos, tan): /

Breakpoint 1, Calculate (Numeral=5, Operation=0x7fffffffde90 "/") at calculate.c:22
22         else if (strcmp(Operation, "*") == 0)
(gdb) █
```




*Рис.15 "запуск с точкой останова"*

– Посмотрела, чему равно на этом этапе значение переменной Numeral, введя:

`print Numeral`

На экран должно быть выведено число 5.

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

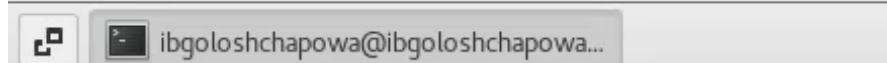


*Рис.16 "print Numeral"*

– Сравнила с результатом вывода на экран после использования команды:

`display Numeral`

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



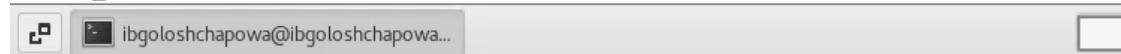
*Рис.17 "display Numeral"*

– Убрала точки останова:

`info breakpoints`

delete 1

```
(gdb) info breakpoints
Num      Type           Disp Enb Address                  What
1        breakpoint     keep y   0x0000000000400810 in Calculate at calculate.c:21
          breakpoint already hit 1 time
(gdb) delete 1
(gdb)
```



*Рис.18 "delete 1"*

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

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

calculate.h:4:37: 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:7:31: Function parameter Operation declared as manifest array (size
                    constant is meaningless)
calculate.c: (in function Calculate)
calculate.c:13:2: 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:19:2: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:25:2: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:31:2: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:32:6: 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:35:12: Return value type double does not match declared type float:
                    (HUGE_VAL)
    To allow all numeric types to match, use +relaxtypes.
calculate.c:43:2: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:44:8: Return value type double does not match declared type float:
                    (pow(Numeral, SecondNumeral))
calculate.c:47:8: Return value type double does not match declared type float:
                    (sqrt(Numeral))
calculate.c:49:8: Return value type double does not match declared type float:
                    (sin(Numeral))
calculate.c:51:8: Return value type double does not match declared type float:
                    (cos(Numeral))
calculate.c:53:8: Return value type double does not match declared type float:
                    (tan(Numeral))
calculate.c:57:8: Return value type double does not match declared type float:
                    (HUGE_VAL)

Finished checking --- 15 code warnings
[ibgoloshchapowa@ibgoloshchapowa lab_prog]$
```

*Рис.19 "splint calculate.c"*

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

calculate.h:4:37: 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:11:5: 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:13:17: 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:13:13: Corresponding format code
main.c:13:5: Return value (type int) ignored: scanf("%s", &Ope...

Finished checking --- 4 code warnings
[ibgoloshchapowa@ibgoloshchapowa lab_prog]$ █
```

---

*Рис.20 "splint main.c"*

## Выводы

В ходе лабораторной работы я приобрела простейшие навыки разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linux на примере создания на языке программирования C калькулятора с простейшими функциями.

## Контрольные вопросы

- Информацию об этих программах можно получить с помощью функций `info` и `man`.
- Unix поддерживает следующие основные этапы разработки приложений:
  - создание исходного кода программы; - представляется в виде файла
  - сохранение различных вариантов исходного текста;
  - анализ исходного текста; необходимо отслеживать изменения исходного кода, а также при работе более двух программистов над проектом программы нужно, чтобы они не делали изменений кода в одно время.
  - компиляция исходного текста и построение исполняемого модуля;
  - тестирование и отладка; - проверка кода на наличие ошибок
  - сохранение всех изменений, выполняемых при тестировании и отладке.
- Использование суффикса ".c" для имени файла с программой на языке Си отражает удобное и полезное соглашение, принятое в ОС UNIX. Для любого имени входного

файла суффикс определяет какая компиляция требуется. Суффиксы и префиксы указывают тип объекта. Одно из полезных свойств компилятора Си — его способность по суффиксам определять типы файлов. По суффиксу .c компилятор распознает, что файл abcd.c должен компилироваться, а по суффиксу .o, что файл abcd.o является объектным модулем и для получения исполняемой программы необходимо выполнить редактирование связей. Простейший пример командной строки для компиляции программы abcd.c и построения исполняемого модуля abcd имеет вид: gcc -o abcd abcd.c. Некоторые проекты предпочитают показывать префиксы в начале текста изменений для старых (old) и новых (new) файлов. Опция - prefix может быть использована для установки такого префикса. Плюс к этому команда bzr diff -p1 выводит префиксы в форме которая подходит для команды patch -p1.

2. Основное назначение компилятора с языка Си заключается в компиляции всей программы в целом и получении исполняемого модуля.
3. При разработке большой программы, состоящей из нескольких исходных файлов заголовков, приходится постоянно следить за файлами, которые требуют перекомпиляции после внесения изменений. Программа make освобождает пользователя от такой рутинной работы и служит для документирования взаимосвязей между файлами. Описание взаимосвязей и соответствующих действий хранится в так называемом make-файле, который по умолчанию имеет имя makefile или Makefile.
4. В общем случае make-файл содержит последовательность записей (строк), определяющих зависимости между файлами. Первая строка записи представляет собой список целевых (зависимых) файлов, разделенных пробелами, за которыми следует двоеточие и список файлов, от которых зависят целевые. Текст, следующий за точкой с запятой, и все последующие строки, начинающиеся с литеры табуляции, являются командами ОС UNIX, которые необходимо выполнить для обновления целевого файла. Таким образом, спецификация взаимосвязей имеет формат: target1 [ target2...]: [:] [dependment1...] [(tab)commands] [#commentary] [(tab)commands] [#commentary], где # — специфицирует начало комментария, так как содержимое строки, начиная с # и до конца строки, не будет обрабатываться командой make; : — последовательность команд ОС UNIX должна содержаться в одной строке make-файла (файла описаний), есть возможность переноса команд (), но она считается как одна строка; :: — последовательность команд ОС UNIX может содержаться в нескольких последовательных строках файла описаний. Приведённый выше make-файл для программы abcd.c включает два способа компиляции и построения исполняемого модуля. Первый способ предусматривает обычную компиляцию с построением исполняемого модуля с именем abcd. Второй способ позволяет включать в исполняемый модуль testabcd возможность выполнить процесс отладки на уровне исходного текста. Пример можно найти в задании 5.
5. 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 отлаженную 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.

1. `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

1. 1) Выполнила компиляцию программы 2) Увидела ошибки в программе 3) Открыла редактор и исправила программу 4) Загрузила программу в отладчик gdb 5) run — отладчик выполнил программу, ввела требуемые значения. 6) Использовала другие команды отладчика и проверила работу программы
2. Отладчику не понравился формат %s для &Operation, т.к %s — символьный формат, а значит необходим только Operation.
3. Если вы работаете с исходным кодом, который не вами разрабатывался, то назначение различных конструкций может быть не совсем понятным. Система

разработки приложений UNIX предоставляет различные средства, повышающие понимание исходного кода. К ним относятся:

– cscope - исследование функций, содержащихся в программе;

– splint — критическая проверка программ, написанных на языке Си.

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