

Лабораторная работа №14 по предмету  
Операционные системы

Группа НПМбв-02-19

Нечаева Виктория Алексеевна

# Содержание

Цель работы	5
Задание	6
Выполнение лабораторной работы	9
Выводы	24
Контрольные вопросы	25

## Список таблиц

# Список иллюстраций

1	Рисунок 1 . . . . .	9
2	Рисунок 2 . . . . .	10
3	Рисунок 3 . . . . .	11
4	Рисунок 4 . . . . .	12
5	Рисунок 5 . . . . .	13
6	Рисунок 6 . . . . .	14
7	Рисунок 6.1 . . . . .	15
8	Рисунок 7 . . . . .	16
9	Рисунок 8 . . . . .	17
10	Рисунок 9 . . . . .	18
11	Рисунок 10 . . . . .	19
12	Рисунок 11 . . . . .	21
13	Рисунок 12 . . . . .	22
14	Рисунок 13 . . . . .	23

## 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

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`:  

```
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: 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 код используйте команду `list`:  

```
list
```
  - Для просмотра строк с 12 по 15 основного файла используйте `list` с парамет-

рами:

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=0x7ffffffd280 "-") 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

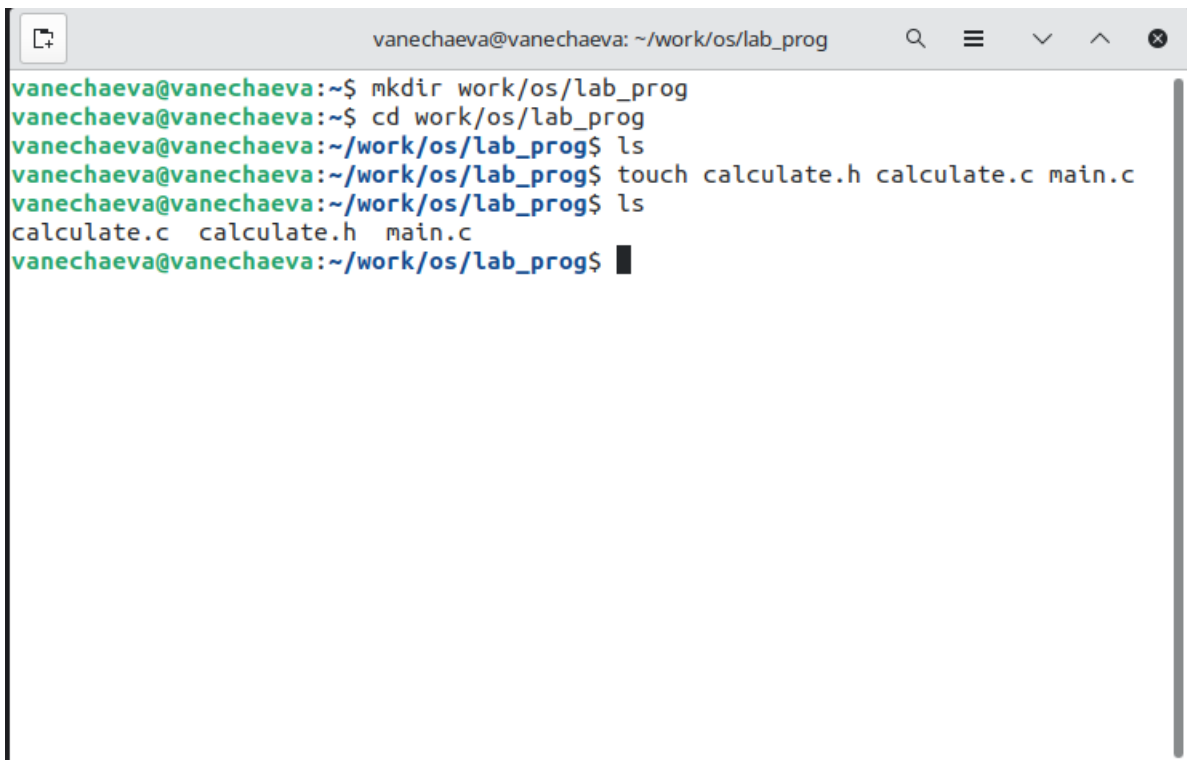
7. С помощью утилиты splint попробуйте проанализировать коды файлов

calculate.c и main.c.



# Выполнение лабораторной работы

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 (рис.1-3)



```
vanechaeva@vanechaeva: ~/work/os/lab_prog
vanechaeva@vanechaeva:~$ mkdir work/os/lab_prog
vanechaeva@vanechaeva:~$ cd work/os/lab_prog
vanechaeva@vanechaeva:~/work/os/lab_prog$ ls
vanechaeva@vanechaeva:~/work/os/lab_prog$ touch calculate.h calculate.c main.c
vanechaeva@vanechaeva:~/work/os/lab_prog$ ls
calculate.c calculate.h main.c
vanechaeva@vanechaeva:~/work/os/lab_prog$
```

Рис. 1: Рисунок 1

```
GNU nano 6.2 calculate.c
////////////////////////////////////
// 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(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);
}
else if(strncmp(Operation, "pow", 3) == 0)
```

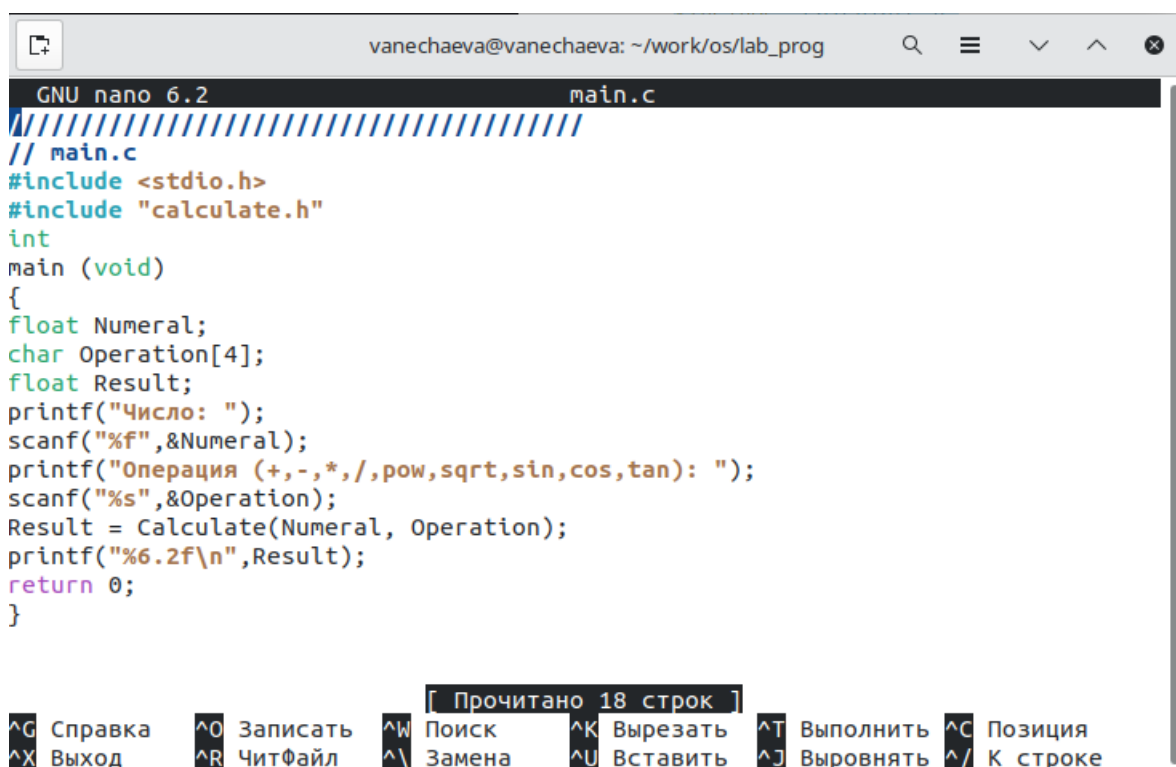
<b>^G</b> Справка	<b>^O</b> Записать	<b>^W</b> Поиск	<b>^K</b> Вырезать	<b>^T</b> Выполнить	<b>^C</b> Позиция
<b>^X</b> Выход	<b>^R</b> ЧитФайл	<b>^\\</b> Замена	<b>^U</b> Вставить	<b>^J</b> Выводить	<b>^/_</b> К строке

Рис. 2: Рисунок 2

```
GNU nano 6.2 calculate.h
////////////////////////////////////
// calculate.h
#ifndef CALCULATE_H_
#define CALCULATE_H_
float Calculate(float Numeral, char Operation[4]);
#endif /*CALCULATE_H_*/
```

^G Справка    ^O Записать    ^W Поиск    ^K Вырезать    ^T Выполнить    ^C Позиция  
^X Выход    ^R ЧитФайл    ^\ Замена    ^U Вставить    ^J Выводить    ^/ К строке

Рис. 3: Рисунок 3



```
GNU nano 6.2 main.c
////////////////////////////////////
// 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;
}
```

[ Прочитано 18 строк ]

<b>^G</b> Справка	<b>^O</b> Записать	<b>^W</b> Поиск	<b>^K</b> Вырезать	<b>^T</b> Выполнить	<b>^C</b> Позиция
<b>^X</b> Выход	<b>^R</b> ЧитФайл	<b>^_</b> Замена	<b>^U</b> Вставить	<b>^J</b> Выводить	<b>^/_</b> К строке

Рис. 4: Рисунок 4

Выполните компиляцию программы посредством gcc:

gcc -c calculate.c

gcc -c main.c

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

```
vanechaeva@vanechaeva: ~/work/os/lab_prog
Настраивается пакет binutils-x86-64-linux-gnu (2.38-4ubuntu2.1) ...
Настраивается пакет binutils (2.38-4ubuntu2.1) ...
Настраивается пакет gcc-11 (11.3.0-1ubuntu1~22.04) ...
Настраивается пакет gcc (4:11.2.0-1ubuntu1) ...
Обрабатываются триггеры для man-db (2.10.2-1) ...
Обрабатываются триггеры для libc-bin (2.35-0ubuntu3.1) ...
vanechaeva@vanechaeva:~/work/os/lab_prog$ gcc -c calculate.c
vanechaeva@vanechaeva:~/work/os/lab_prog$ gcc -c main.c
main.c: In function 'main':
main.c:14:9: warning: format '%s' expects argument of type 'char *', but argumen
t 2 has type 'char (*)[4]' [-Wformat=]
   14 | scanf("%s",&Operation);
      |         ~^ ~~~~~
      |         | |
      |         | char (*)[4]
      |         char *
```

Рис. 5: Рисунок 5

4. При необходимости исправьте синтаксические ошибки. Убираю на (рис.6) знак доллара у Operation.



```
GNU nano 6.2 main.c
////////////////////////////////////
// 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("%.6f\n",Result);
return 0;
}
```

\$

[ Прочитано 18 строк ]

<b>^G</b> Справка	<b>^O</b> Записать	<b>^W</b> Поиск	<b>^K</b> Вырезать	<b>^T</b> Выполнить	<b>^C</b> Позиция
<b>^X</b> Выход	<b>^R</b> ЧитФайл	<b>^\\</b> Замена	<b>^U</b> Вставить	<b>^J</b> Выровнять	<b>^/_</b> К строке

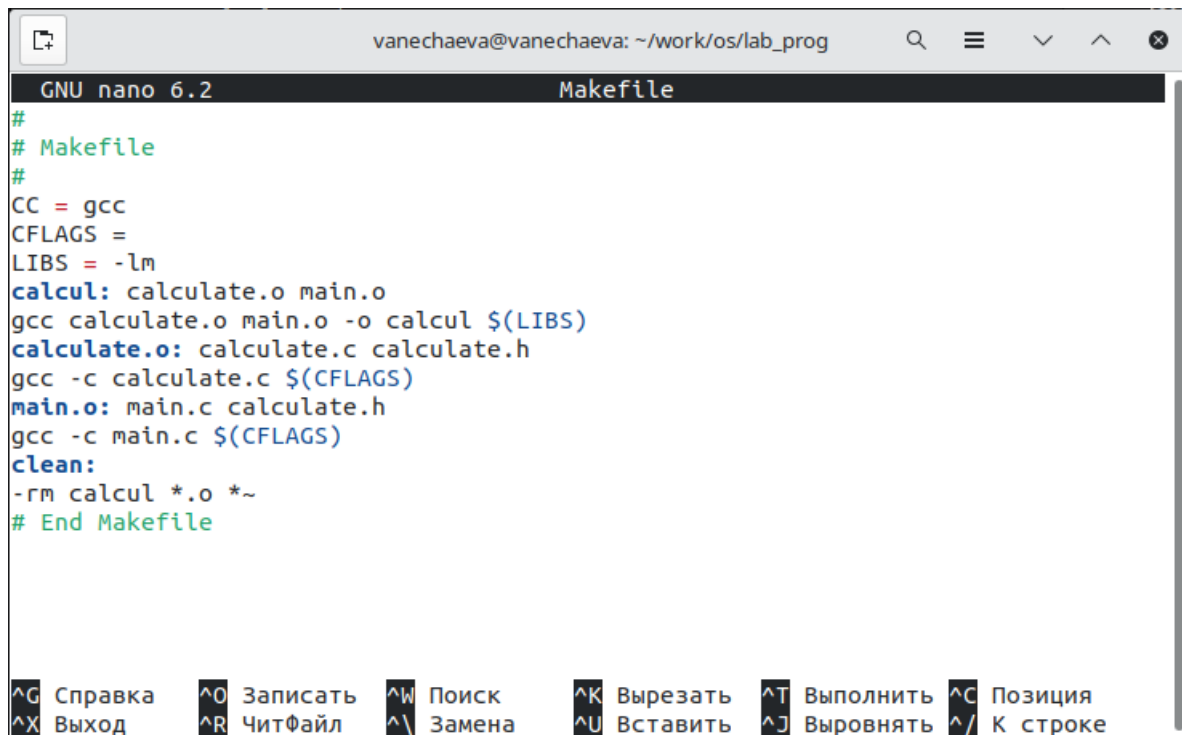
Рис. 6: Рисунок 6

5. Создайте Makefile. Поясните в отчёте его содержание.

В начале файла определяются переменные для компилятора СС (gcc в данном случае), флагов компиляции CFLAGS (которые не заданы в данном случае) и библиотеки LIBS (-lm, которая используется для математических операций).

Далее описываются правила для компиляции и линковки объектных файлов. В данном случае описывается, что для создания исполняемого файла “calcul” необходимо скомпилировать файлы calculate.c и main.c в объектные файлы calculate.o и main.o соответственно, а затем выполнить линковку этих объектных файлов с помощью команды gcc.

Также заданы правила для компиляции каждого из исходных файлов (calculate.c и main.c) в соответствующие объектные файлы.

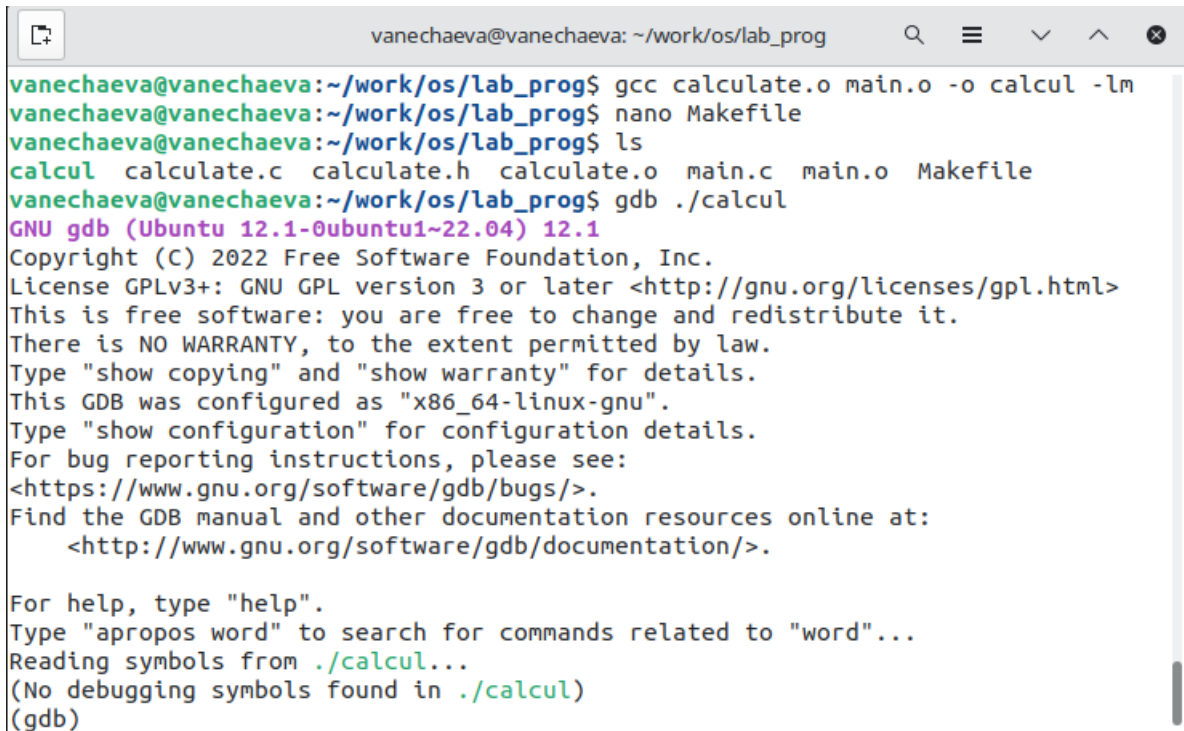


```
GNU nano 6.2 Makefile
#
# Makefile
#
CC = gcc
CFLAGS =
LIBS = -lm
calcul: calculate.o main.o
gcc calculate.o main.o -o calcul $(LIBS)
calculate.o: calculate.c calculate.h
gcc -c calculate.c $(CFLAGS)
main.o: main.c calculate.h
gcc -c main.c $(CFLAGS)
clean:
-rm calcul *.o *~
# End Makefile
```

^G Справка   ^O Записать   ^W Поиск   ^K Вырезать   ^T Выполнить   ^C Позиция  
^X Выход   ^R ЧитФайл   ^\ Замена   ^U Вставить   ^J Выводить   ^\_ К строке

Рис. 7: Рисунок 6.1

6. С помощью gdb выполните отладку программы calcul (перед использованием gdb исправьте Makefile):
  - Запустите отладчик GDB, загрузив в него программу для отладки: `gdb ./calcul`



```
vanechaeva@vanechaeva: ~/work/os/lab_prog
vanechaeva@vanechaeva:~/work/os/lab_prog$ gcc calculate.o main.o -o calcul -lm
vanechaeva@vanechaeva:~/work/os/lab_prog$ nano Makefile
vanechaeva@vanechaeva:~/work/os/lab_prog$ ls
calcul calculate.c calculate.h calculate.o main.c main.o Makefile
vanechaeva@vanechaeva:~/work/os/lab_prog$ gdb ./calcul
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
Copyright (C) 2022 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-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

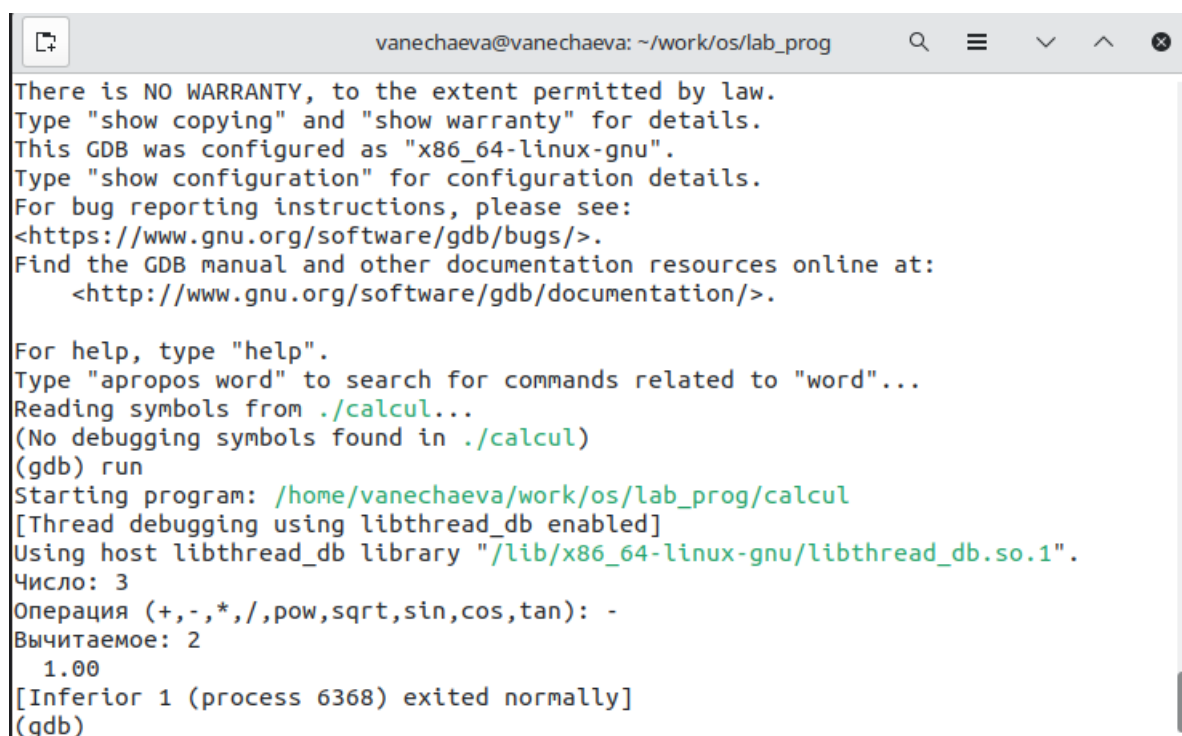
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./calcul...
(No debugging symbols found in ./calcul)
(gdb)
```

Рис. 8: Рисунок 7

- Для запуска программы внутри отладчика введите команду `run`:

`run`





```
vanechaeva@vanechaeva: ~/work/os/lab_prog
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-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
  <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./calcul...
(No debugging symbols found in ./calcul)
(gdb) run
Starting program: /home/vanechaeva/work/os/lab_prog/calcul
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Число: 3
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): -
Вычитаемое: 2
  1.00
[Inferior 1 (process 6368) exited normally]
(gdb)
```

Рис. 9: Рисунок 8

– Для постраничного (по 9 строк) просмотра исходного код используйте команду

list:

list

```
vanechaeva@vanechaeva: ~/work/os/lab_prog
Число: 3
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): *
Множитель: 3
9.00
[Inferior 1 (process 7070) exited normally]
(gdb) list
1  //////////////////////////////////////
2  // main.c
3  #include <stdio.h>
4  #include "calculate.h"
5  int
6  main (void)
7  {
8  float Numeral;
9  char Operation[4];
10 float Result;
(gdb) list 12,15
12 scanf("%f",&Numeral);
13 printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): ");
14 scanf("%s",Operation);
15 Result = Calculate(Numeral, Operation);
(gdb) list calculate.c:20,27
20 scanf("%f",&SecondNumeral);
21 return(Numeral - SecondNumeral);
22 }
23 else if(strncmp(Operation, "*", 1) == 0)
24 {
25 printf("Множитель: ");
26 scanf("%f",&SecondNumeral);
27 return(Numeral * SecondNumeral);
(gdb) list calculate.c:20,29
20 scanf("%f",&SecondNumeral);
21 return(Numeral - SecondNumeral);
22 }
23 else if(strncmp(Operation, "*", 1) == 0)
24 {
25 printf("Множитель: ");
26 scanf("%f",&SecondNumeral);
27 return(Numeral * SecondNumeral);
28 }
29 else if(strncmp(Operation, "/", 1) == 0)
(gdb) list calculate.c:20,29
```

Рис. 10: Рисунок 9

- 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 пара-

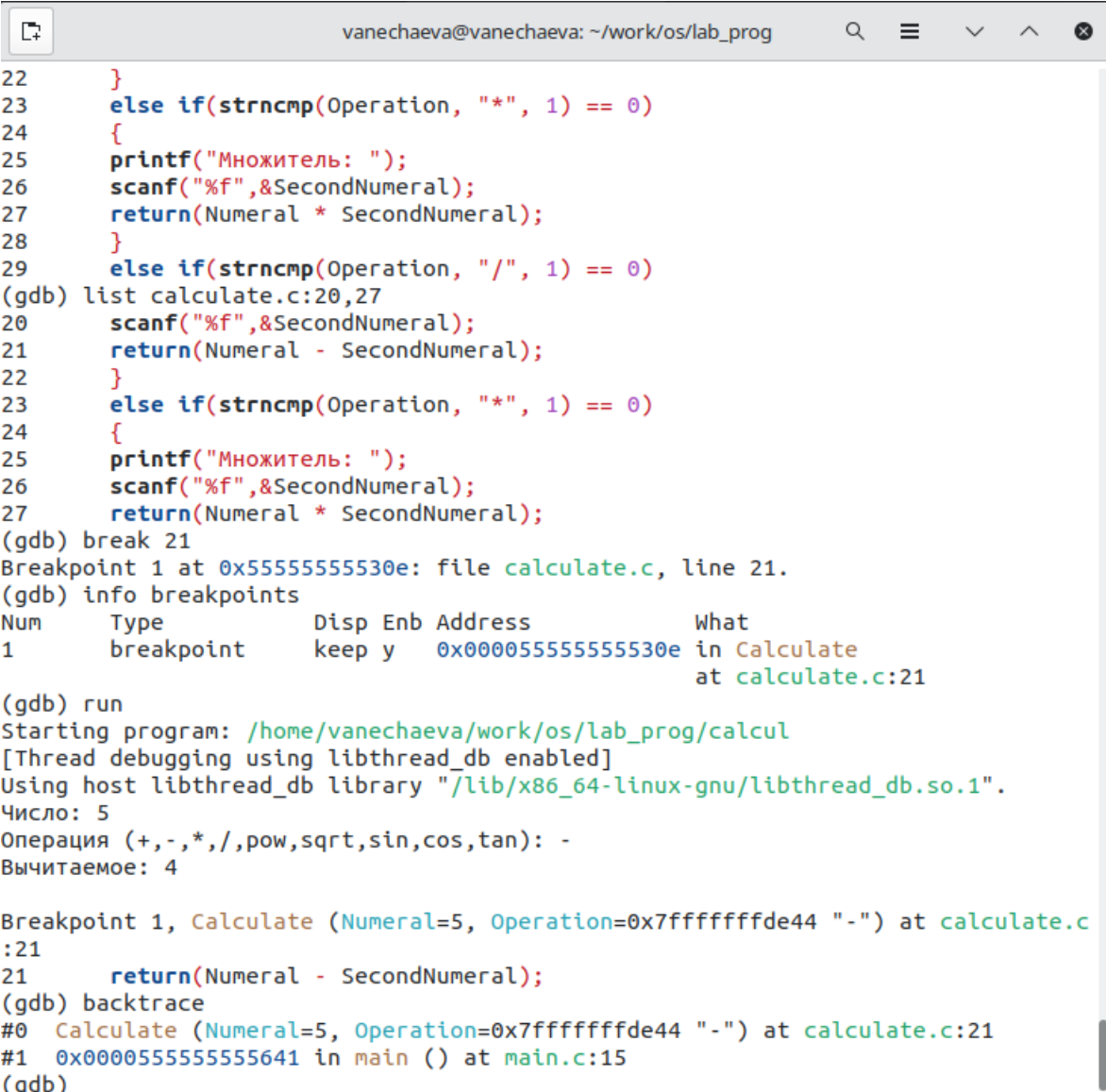
метрами:

list calculate.c:20,29

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

list calculate.c:20,27

break 21



```
vanechaeva@vanechaeva: ~/work/os/lab_prog
22     }
23     else if(strncmp(Operation, "*", 1) == 0)
24     {
25         printf("Множитель: ");
26         scanf("%f",&SecondNumeral);
27         return(Numeral * SecondNumeral);
28     }
29     else if(strncmp(Operation, "/", 1) == 0)
(gdb) list calculate.c:20,27
20         scanf("%f",&SecondNumeral);
21         return(Numeral - SecondNumeral);
22     }
23     else if(strncmp(Operation, "*", 1) == 0)
24     {
25         printf("Множитель: ");
26         scanf("%f",&SecondNumeral);
27         return(Numeral * SecondNumeral);
(gdb) break 21
Breakpoint 1 at 0x55555555530e: file calculate.c, line 21.
(gdb) info breakpoints
Num   Type             Disp Enb Address            What
1      breakpoint      keep y   0x000055555555530e  in Calculate
                                           at calculate.c:21

(gdb) run
Starting program: /home/vanechaeva/work/os/lab_prog/calcul
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Число: 5
Операция (+, -, *, /, pow, sqrt, sin, cos, tan): -
Вычитаемое: 4

Breakpoint 1, Calculate (Numeral=5, Operation=0x7fffffffde44 "-") at calculate.c:21
21         return(Numeral - SecondNumeral);
(gdb) backtrace
#0  Calculate (Numeral=5, Operation=0x7fffffffde44 "-") at calculate.c:21
#1  0x0000555555555641 in main () at main.c:15
(gdb)
```

Рис. 11: Рисунок 10

– 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=0x7ffffffd280 "-") 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

```
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): -
Вычитаемое: 4

Breakpoint 1, Calculate (Numeral=5, Operation=0x7fffffffde44 "-") at calculate.c:21
21      return(Numeral - SecondNumeral);
(gdb) backtrace
#0 Calculate (Numeral=5, Operation=0x7fffffffde44 "-") at calculate.c:21
#1 0x0000555555555641 in main () at main.c:15
(gdb) print Numeral
$1 = 5
(gdb) display Numeral
1: Numeral = 5
(gdb) info breakpoints
Num      Type             Disp Enb Address                      What
1        breakpoint       keep y  0x000055555555530e in Calculate
                                                at calculate.c:21
        breakpoint already hit 1 time
(gdb) delete 1
(gdb)
```

Рис. 12: Рисунок 11

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

Анализ splint рассказывает о том, где необоснованно обозначаются переменные и впоследствии не используются, где неверные типы, которые могут привести к неожиданным результатам. На рис.12 рассказывается, что в calculate.c игнорируются опции scanf, используется не совсем корректное сравнение с вещественным числом и используются с вещественными числами функции, которые возвращают неверный тип данных.

```

vanechaeva@vanechaeva:~/work/os/lab_prog$ splint calculate.c
Splint 3.1.2 --- 21 Feb 2021

calculate.h:5:37: 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:8:31: Function parameter Operation declared as manifest array (size
        constant is meaningless)
calculate.c: (in function calculate)
calculate.c:14:1: 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:20:1: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:26:1: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:32:1: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:33:4: 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:36:7: Return value type double does not match declared type float:
        (HUGE_VAL)
    To allow all numeric types to match, use +relaxtypes.
calculate.c:44:1: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:45:7: Return value type double does not match declared type float:
        (pow(Numeral, SecondNumeral))
calculate.c:48:7: Return value type double does not match declared type float:
        (sqrt(Numeral))
calculate.c:50:7: Return value type double does not match declared type float:
        (sin(Numeral))
calculate.c:52:7: Return value type double does not match declared type float:
        (cos(Numeral))
calculate.c:54:7: Return value type double does not match declared type float:
        (tan(Numeral))
calculate.c:58:7: Return value type double does not match declared type float:
        (HUGE_VAL)

```

Рис. 13: Рисунок 12

В main.c говорится, что в коде main.c бессмысленно использовать задачу размера Operation и что игнорируется тип возврата функции scanf.

```
vanechaeva@vanechaeva:~/work/os/lab_prog$ splint main.c
Splint 3.1.2 --- 21 Feb 2021

calculate.h:5:37: 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:12:1: 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:14:1: Return value (type int) ignored: scanf("%s", Oper...

Finished checking --- 3 code warnings
vanechaeva@vanechaeva:~/work/os/lab_prog$
```

Рис. 14: Рисунок 13

## Выводы

В ходе лабораторной работы мною были приобретены простейшие навыки разработки, анализа, тестирования и отладки приложений в ОС UNIX на примере калькулятора, написанного на C.



# Контрольные вопросы

1. Как получить информацию о возможностях программ `gcc`, `make`, `gdb` и др.?

Информацию о возможностях программ `gcc`, `make`, `gdb` и др. можно получить, прочитав документацию по этим программам, которая обычно поставляется вместе с операционной системой или доступна в Интернете на официальных сайтах производителей. Также можно использовать команды `man` и `info` в терминале UNIX для получения справочной информации о программе.

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

Написание кода на языке программирования;

Компиляция исходного кода в исполняемый файл с помощью компилятора;

Отладка программы с помощью отладчика;

Создание файла `Makefile` и использование утилиты `make` для автоматизации процесса сборки приложения;

Тестирование и доработка приложения.

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

Суффикс - это часть имени файла, которая обозначает тип файла или его формат. Например, в названии файла `source.c` суффикс `“.c”` указывает, что это файл исходного кода на языке программирования C.

4. Каково основное назначение компилятора языка C в UNIX?

Основное назначение компилятора языка C в UNIX - это компиляция исходного кода на языке C в исполняемый файл, который можно запустить в операционной системе UNIX.

5. Для чего предназначена утилита make?

Утилита make предназначена для автоматизации процесса сборки приложения в UNIX. Она использует файл Makefile, который содержит инструкции по сборке приложения, и автоматически выполняет необходимые действия для компиляции исходного кода и создания исполняемого файла.

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

```
CC = gcc
```

```
CFLAGS = -Wall -g
```

```
app: main.o utils.o
```

```
$(CC) $(CFLAGS) -o app main.o utils.o
```

```
main.o: main.c
```

```
$(CC) $(CFLAGS) -c main.c
```

```
utils.o: utils.c
```

```
$(CC) $(CFLAGS) -c utils.c
```

Основными элементами этого файла являются переменные (например, CC и CFLAGS), цели (например, app) и правила сборки (например, app: main.o utils.o).

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

Основное свойство, присущее всем программам отладки - это возможность управления выполнением программы, остановки ее на определенном месте, просмотра значений переменных и выполнения команд в контексте отладки. Чтобы использовать эту функциональность, необходимо иметь доступ к исходному коду программы.

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

Основные команды отладчика gdb:

break: установка точки останова на определенной строке или функции  
run: запуск программы

next: выполнение текущей строки кода без захода внутрь функции

step: выполнение текущей строки кода с заходом внутрь функции

print: вывод значения переменной

watch: установка точки останова на изменении значения переменной

backtrace: вывод стека вызовов функций

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

Скомпилировать программу с опцией -с.

Создать Makefile.

Запустить отладчик gdb, указав имя скомпилированного исполняемого файла

Запустить программу run

Посмотреть исходный код list

Установить точки останова на нужных строках кода с помощью команды break

Запустить программу с помощью команды run

После нахождения ошибки исправить код программы и повторить процесс отладки

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

Компилятор при первом запуске анализирует код программы на предмет синтаксических ошибок. Если он находит ошибку, он выводит сообщение об ошибке, которое может включать номер строки и тип ошибки. Это помогает разработчику быстро находить и исправлять ошибки в коде программы.

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

Комментарии в коде, которые объясняют, что делает код и как он работает

Отладочные сообщения, которые выводят информацию о состоянии программы в различных точках ее выполнения

Документация к программе, которая описывает ее функциональность и интерфейс

12. Каковы основные задачи, решаемые программой splint?

Поиск ошибок в коде программы, таких как неинициализированные переменные, использование нулевых указателей, переполнение буфера и других

Анализ стиля кодирования, включая соответствие стандартам программирования и принятым практикам

Поиск потенциальных уязвимостей в безопасности программы, таких как уязвимости, связанные с памятью и неправильным использованием строковых функций.