

Лабораторная работа №13

Дисциплина: Операционные системы

Кабанова Варвара Дмитриевна

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
3	Выводы	22

List of Figures

2.1	Создание подкаталога	6
2.2	Создание файлов	6
2.3	Программа в calculate.c	7
2.4	Программа в calculate.c	7
2.5	Программа в calculate.h	8
2.6	Программа в main.c	9
2.7	Компиляция программы	9
2.8	Программа в Makefile	10
2.9	Программа в Makefile	11
2.10	Удаление файлов	11
2.11	Компиляция файлов	12
2.12	Работа с gdb	12
2.13	Работа с gdb - run	12
2.14	Работа с gdb - list	13
2.15	Работа с gdb - list 12,15	13
2.16	Работа с gdb - list calculate.c:20,29	13
2.17	Работа с gdb - list calculate.c:20,27	14
2.18	Работа с gdb - info breakpoints	14
2.19	Работа с gdb - run	14
2.20	Работа с gdb - print Numeral	15
2.21	Работа с gdb - display Numeral	15
2.22	Работа с gdb - info breakpoints	15
2.23	Результат команды splint calculate.c	16
2.24	Результат конмады splint main.c	16

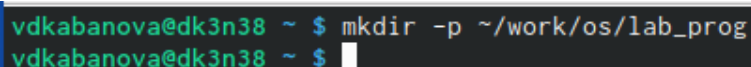
List of Tables

1 Цель работы

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

2 Выполнение лабораторной работы

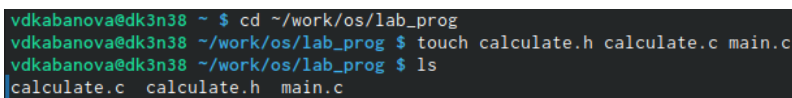
В домашнем каталоге создаю подкаталог `~/work/os/lab_prog` с помощью команды «`mkdir -p ~/work/os/lab_prog`» (рис.1). Вся необходимая информация про создания каталогов указана в следующем источнике: Программное обеспечение GNU/Linux. Лекция 9. Хранилище и дистрибутив (Г. Курячий, МГУ).



```
vdkabanova@dk3n38 ~ $ mkdir -p ~/work/os/lab_prog
vdkabanova@dk3n38 ~ $
```

Figure 2.1: Создание подкаталога

Создала в каталоге файлы: `calculate.h`, `calculate.c`, `main.c`, используя команды «`cd ~/work/os/lab_prog`» и «`touch calculate.h calculate.c main.c`» (рис.2).

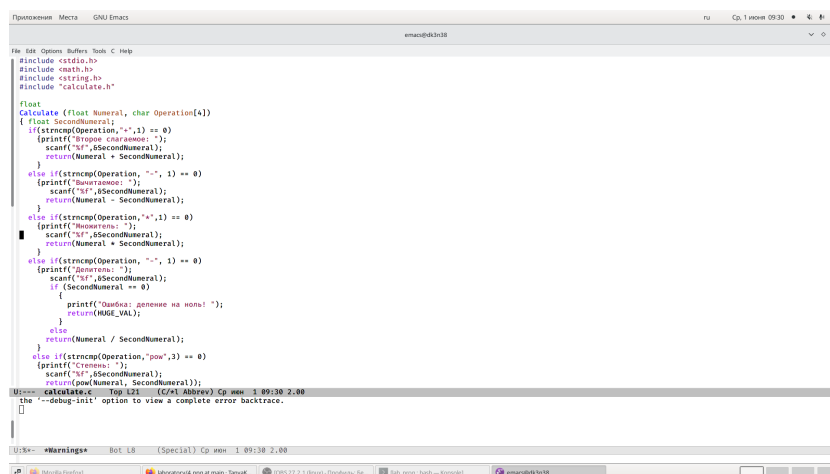


```
vdkabanova@dk3n38 ~ $ cd ~/work/os/lab_prog
vdkabanova@dk3n38 ~/work/os/lab_prog $ touch calculate.h calculate.c main.c
vdkabanova@dk3n38 ~/work/os/lab_prog $ ls
calculate.c calculate.h main.c
vdkabanova@dk3n38 ~/work/os/lab_prog $
```

Figure 2.2: Создание файлов

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. Открыв редактор Emacs, приступила к редактированию созданных файлов. Реализация функций калькулятора в файле `calculate.c` (рис.3-4). Вся необходимая информация про написания программ указана в следующем источнике: Программное

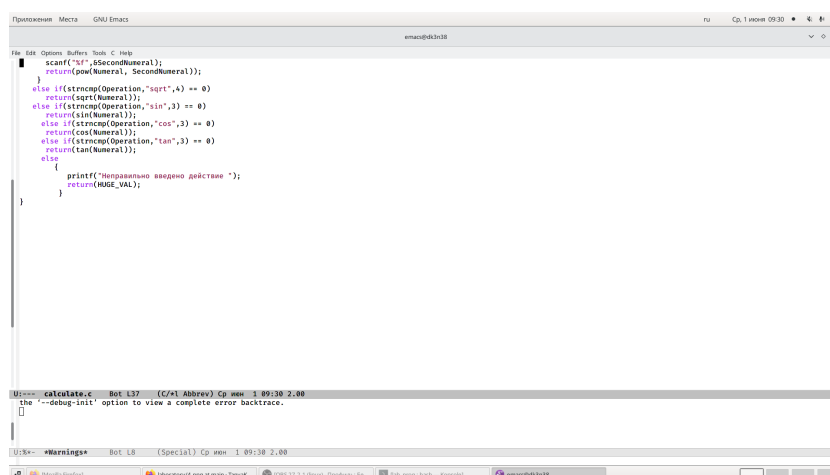
обеспечение GNU/Linux. Лекция 10. Минимальный набор знаний (Г. Курячий, МГУ).



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

float
Calculate (float Numeral, char Operation[4])
{
    float SecondNumeral;
    if (strcmp (Operation, "+") == 0)
    {
        printf ("Input: %s\n", Operation);
        scanf ("%f", &SecondNumeral);
        return (Numeral + SecondNumeral);
    }
    else if (strcmp (Operation, "-") == 0)
    {
        printf ("Input: %s\n", Operation);
        scanf ("%f", &SecondNumeral);
        return (Numeral - SecondNumeral);
    }
    else if (strcmp (Operation, "*") == 0)
    {
        printf ("Input: %s\n", Operation);
        scanf ("%f", &SecondNumeral);
        return (Numeral * SecondNumeral);
    }
    else if (strcmp (Operation, "/") == 0)
    {
        printf ("Input: %s\n", Operation);
        scanf ("%f", &SecondNumeral);
        if (SecondNumeral == 0)
        {
            printf ("Ошибка: деление на ноль\n");
            return (HUGE_VAL);
        }
        return (Numeral / SecondNumeral);
    }
    else if (strcmp (Operation, "pow", 3) == 0)
    {
        printf ("Input: %s\n", Operation);
        scanf ("%f", &SecondNumeral);
        return (pow (Numeral, SecondNumeral));
    }
    else
    {
        printf ("Ошибка: введенное действие\n");
        return (HUGE_VAL);
    }
}
```

Figure 2.3: Программа в calculate.c



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

Figure 2.4: Программа в calculate.c

Интерфейсный файл calculate.h, описывающий формат вызова функции калькулятора (рис.5). Подробная информация о написании программ в Linux указана в следующем источнике: Электронный ресурс: Электронный ресурс: <https://it.wikireading.ru/34160>

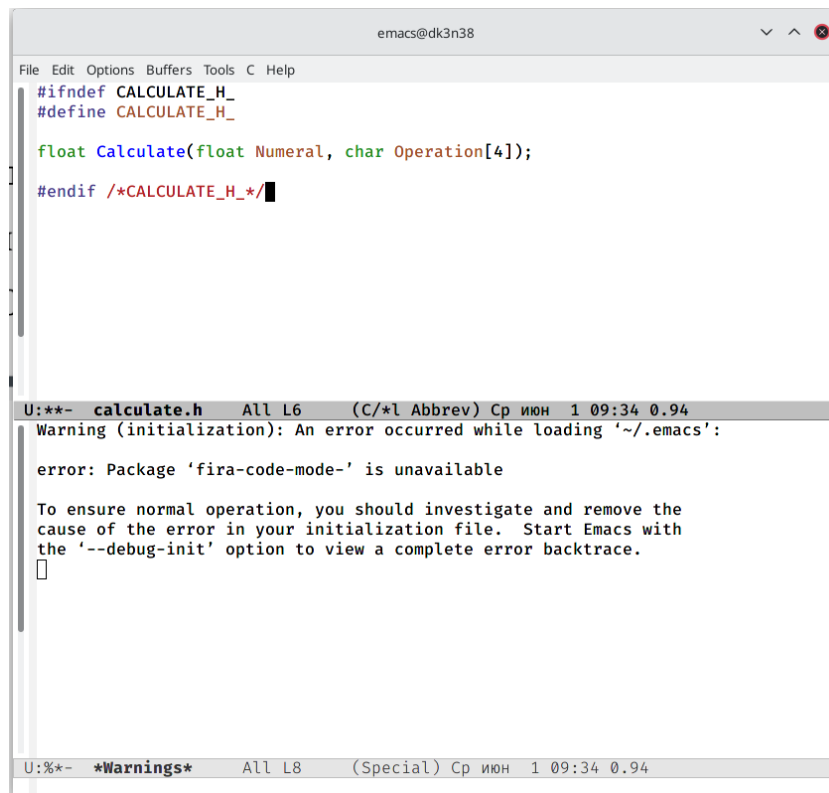


Figure 2.5: Программа в calculate.h

Основной файл main.c, реализующий интерфейс пользователя к калькулятору (рис.6).


```
emacs@dk3n38
File Edit Options Buffers Tools C Help

#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;
}

U:--- main.c All L18 (C/*l Abbrev) Cp июн 1 09:38 1.27
Warning (initialization): An error occurred while loading '~/.emacs':

error: Package 'fira-code-mode-' is unavailable

To ensure normal operation, you should investigate and remove the
cause of the error in your initialization file. Start Emacs with
the '--debug-init' option to view a complete error backtrace.
[]

U:%*- *Warnings* All L8 (Special) Cp июн 1 09:38 1.27
```

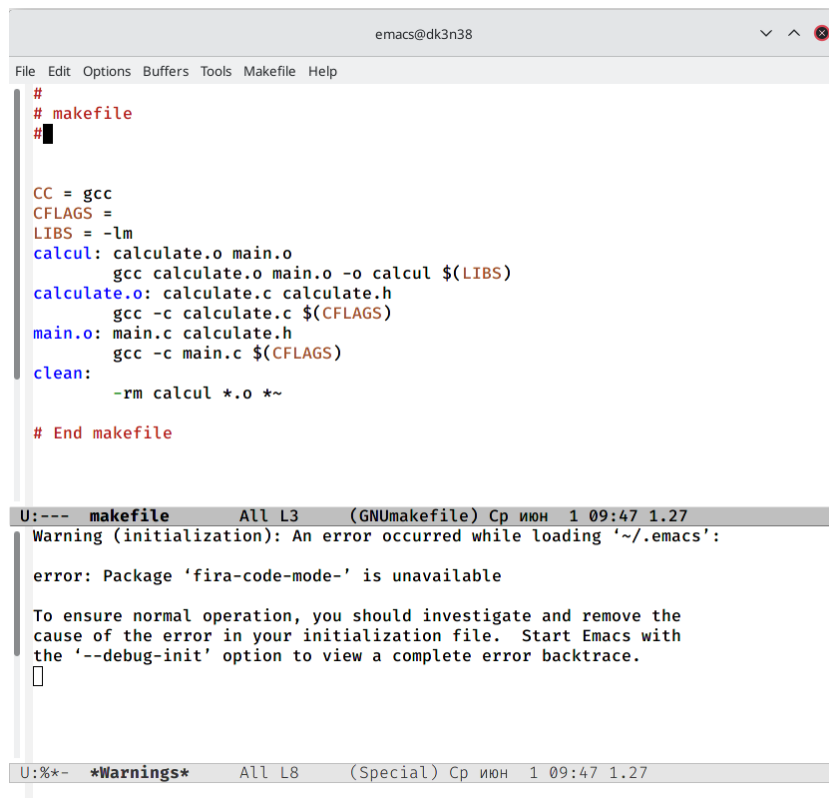
Figure 2.6: Программа в main.c

Выполнила компиляцию программы посредством gcc (версия компилятора :8.3.0-19), используя команды «gcc -c calculate.c», «gcc -c main.c» и «gcc calculate.o main.o -o calcul -lm» (рис.7).

```
vdkabanova@dk3n38 ~/work/os/lab_prog $ gcc -c calculate.c
vdkabanova@dk3n38 ~/work/os/lab_prog $ gcc -c main.c
main.c: В функции «main»:
main.c:13:11: предупреждение: формат «%s» ожидает аргумент типа «char *», но аргумент 2 имеет тип «char (*)[4]» [-Wformat=]
13 | scanf("%s",&Operation);
    |         ^~
    |         |
    |         | char (*)[4]
    |         char *
vdkabanova@dk3n38 ~/work/os/lab_prog $ gcc calculate.o main.o -o calcul -lm
vdkabanova@dk3n38 ~/work/os/lab_prog $
```

Figure 2.7: Компиляция программы

В ходе компиляции программы никаких ошибок выявлено не было. Создала Makefile с необходимым содержанием (рис.8).



The screenshot shows an Emacs window titled 'emacs@dk3n38'. The main editor area contains a Makefile with the following content:

```
#
# 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
```

The bottom panel of the Emacs window shows the output of running 'make' in the directory 'makefile'. The output is as follows:

```
U:--- makefile All L3 (GNUmakefile) Cp июн 1 09:47 1.27
Warning (initialization): An error occurred while loading '~/.emacs':

error: Package 'fira-code-mode-' is unavailable

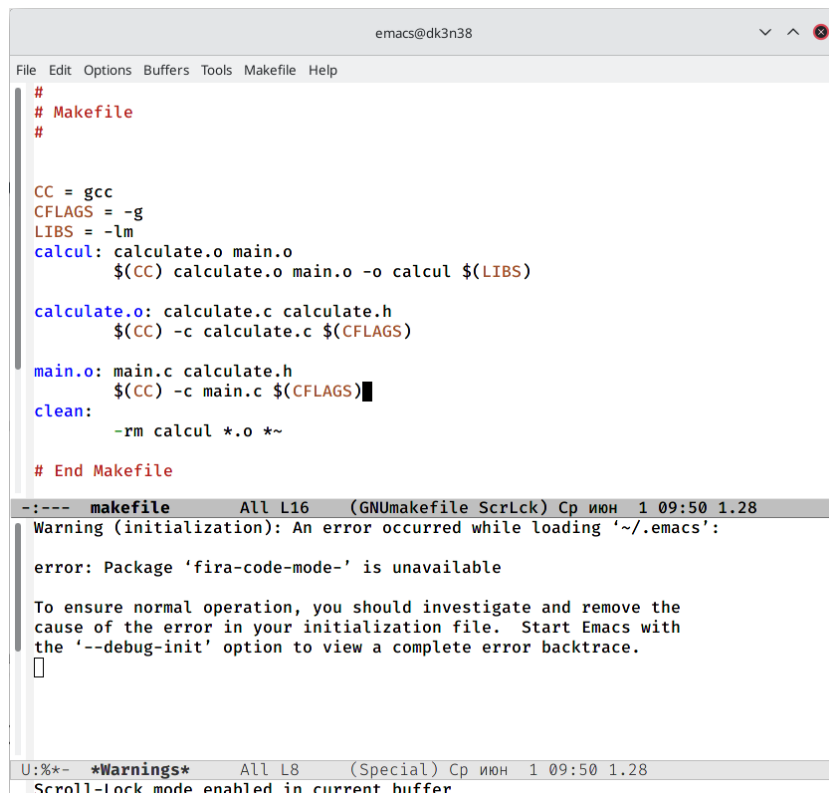
To ensure normal operation, you should investigate and remove the
cause of the error in your initialization file. Start Emacs with
the '--debug-init' option to view a complete error backtrace.
█

U:%*- *Warnings* All L8 (Special) Cp июн 1 09:47 1.27
```

Figure 2.8: Программа в Makefile

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.

Далее исправил Makefile (рис.9). Подробная информация о написании программ в Linux указана в следующем источнике: Электронный ресурс: <https://vunivere.ru/work23597>



The screenshot shows the Emacs editor window titled 'emacs@dk3n38'. The main text area contains a Makefile with the following content:

```
#
# 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
```

Below the Makefile, a status bar shows: '-:--- makefile All L16 (GNUmakefile ScrLck) Cp июн 1 09:50 1.28'. The message area displays the following warning and error:

```
Warning (initialization): An error occurred while loading '~/.emacs':

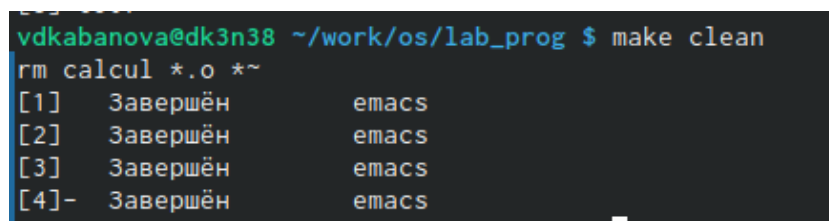
error: Package 'fira-code-mode-' is unavailable

To ensure normal operation, you should investigate and remove the
cause of the error in your initialization file. Start Emacs with
the '--debug-init' option to view a complete error backtrace.
[]
```

At the bottom, another status bar shows: 'U:%*- *Warnings* All L8 (Special) Cp июн 1 09:50 1.28' and 'Scroll-Lock mode enabled in current buffer'.

Figure 2.9: Программа в 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. После этого я удалила исполняемые и объектные файлы из каталога с помощью команды «make clear» (рис.10). Выполнила компиляцию файлов, используя команды «make calculate.o», «make main.o», «make calcul» (рис.11).



The screenshot shows a terminal window with the prompt 'vdkabanova@dk3n38 ~/work/os/lab_prog \$'. The command 'make clean' has been entered, and the output is:

```
rm calcul *.o *~
[1]   Завершён      emacs
[2]   Завершён      emacs
[3]   Завершён      emacs
[4]-  Завершён      emacs
```

Figure 2.10: Удаление файлов

```

vdkabanova@dk3n38 ~/work/os/lab_prog $ make calculate.o
gcc -c calculate.c -g
vdkabanova@dk3n38 ~/work/os/lab_prog $ make main.o
gcc -c main.c -g
main.c: В функции «main»:
main.c:13:11: предупреждение: формат «%s» ожидает аргумент типа «char *», но аргумент 2 имеет тип «char (*)[4]» [-Wformat=]
   13 |     scanf("%s", &operation);
      |           ^
      |           |
      |           | char (*)[4]
      |           +-----+
      |           |
      |           | char *
      |           +-----+
vdkabanova@dk3n38 ~/work/os/lab_prog $ make calcul
gcc calculate.o main.o -o calcul -lm
vdkabanova@dk3n38 ~/work/os/lab_prog $

```

Figure 2.11: Компиляция файлов

Далее с помощью gdb выполнила отладку программы calcul. Запустила отладчик GDB, загрузив в него программу для отладки, используя команду: «gdb./calcul» (рис.12).

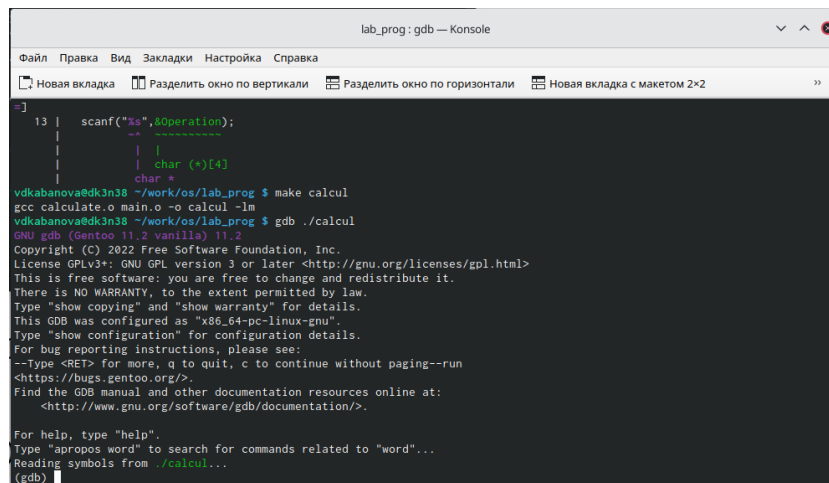
```

vdkabanova@dk3n38 ~/work/os/lab_prog $ gdb ./calcul
GNU gdb (Gentoo 11.2 vanilla) 11.2
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-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
--Type <RET> for more, q to quit, c to continue without paging--

```

Figure 2.12: Работа с gdb

Для запуска программы внутри отладчика ввела команду «run» (рис.13).



```

lab_prog: gdb --- Konsole
Файл  Правка  Вид  Закладки  Настройка  Справка
[+] Новая вкладка  [ ] Разделить окно по вертикали  [ ] Разделить окно по горизонтали  [ ] Новая вкладка с макетом 2x2
13 |     scanf("%s", &operation);
    |           ^
    |           |
    |           | char (*)[4]
    |           +-----+
    |           |
    |           | char *
    |           +-----+
vdkabanova@dk3n38 ~/work/os/lab_prog $ make calcul
gcc calculate.o main.o -o calcul -lm
vdkabanova@dk3n38 ~/work/os/lab_prog $ gdb ./calcul
GNU gdb (Gentoo 11.2 vanilla) 11.2
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-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
--Type <RET> for more, q to quit, c to continue without paging--run
<https://bugs.gentoo.org/>.
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...
(gdb)

```

Figure 2.13: Работа с gdb - run

Для постраничного (по 10 строк) просмотра исходного кода использовала команду «list» (рис.14).

```
(gdb) 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) █
```

Figure 2.14: Работа с gdb - list

Для просмотра строк с 12 по 15 основного файла использовала команду «list 12,15» (рис.15).

```
(gdb) list 12,15
12     printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): ");
13     scanf("%s",&Operation);
14     Result = Calculate(Numeral, Operation);
15     printf("6.2f\n",Result);
(gdb) █
```

Figure 2.15: Работа с gdb - list 12,15

Для просмотра определённых строк не основного файла использовала команду «list calculate.c:20,29» (рис.16).

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

Figure 2.16: Работа с gdb - list calculate.c:20,29

Установила точку останова в файле calculate.c на строке номер 21, используя команды «list calculate.c:20,27» и «break 21» (рис.17).

```

(gdb) list calculate.c:20,27
20         {printf("Множитель: ");
21         scanf("%f",&SecondNumeral);
22         return(Numeral * SecondNumeral);
23     }
24     else if(strncmp(Operation, "-", 1) == 0)
25     {printf("Делитель: ");
26     scanf("%f",&SecondNumeral);
27     if (SecondNumeral == 0)
(gdb) break 21
Breakpoint 1 at 0x12bb: file calculate.c, line 21.
(gdb)

```

Figure 2.17: Работа с gdb - list calculate.c:20,27

Вывела информацию об имеющихся в проекте точках останова с помощью команды «info breakpoints» (рис.18).

```

(gdb) info breakpoints
Num  Type      Disp Enb Address          What
1    breakpoint keep y   0x00000000000012bb in Calculate at calculate.c:21
(gdb)

```

Figure 2.18: Работа с gdb - info breakpoints

Запустила программу внутри отладчика и убедилась, что программа остановилась в момент прохождения точки останова. Использовала команды «run», «5», «*» и «backtrace» (рис.19).

```

(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/v/d/vdkabanova/work/os/lab_prog/calcul
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): *
Breakpoint 1, Calculate (Numeral=5, Operation=0x7fffffffcd34 "*") at calculate.c:21
21     scanf("%f",&SecondNumeral);
(gdb) backtrace
#0 Calculate (Numeral=5, Operation=0x7fffffffcd34 "*") at calculate.c:21
#1 0x00005555555555a9 in main () at main.c:14
(gdb)

```

Figure 2.19: Работа с gdb - run

Посмотрела, чему равно на этом этапе значение переменной Numeral, введя команду «print Numeral» (рис.20).

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

Figure 2.20: Работа с gdb - print Numeral

Сравнила с результатом вывода на экран после использования команды «display Numeral». Значения совпадают (рис.21).

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

Figure 2.21: Работа с gdb - display Numeral

Убрала точки останова с помощью команд «info breakpoints» и «delete 1» (рис.22).

```
(gdb) info breakpoints
Num   Type      Disp Enb Address          What
1     breakpoint keep y  0x0000555555552bb in calculate at calculate.c:21
      breakpoint already hit 1 time
(gdb) delete 1
(gdb) 
```

Figure 2.22: Работа с gdb - info breakpoints

Далее воспользовалась командами «splint calculate.c» и «splint main.c» (рис.23-24). С помощью утилиты splint выяснилось, что в файлах calculate.c и main.c присутствует функция чтения scanf, возвращающая целое число (тип int), 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.

```

vdkabanova@dk3n38 ~/work/os/lab_prog $ splint calculate.c
Splint 3.1.2 --- 13 Jan 2021

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:7:32: Function parameter Operation declared as manifest array (size
        constant is meaningless)
calculate.c: (in function Calculate)
calculate.c:11:7: 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:16:8: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:21:7: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:26:8: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:27:12: 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:30:11: Return value type double does not match declared type float:
    (HUGE_VAL)
    To allow all numeric types to match, use +relaxtypes.
calculate.c:37:7: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:37:13: Return value type double does not match declared type float:
    (pow(Numeral, SecondNumeral))
calculate.c:41:13: Return value type double does not match declared type float:
    (sqrt(Numeral))
calculate.c:43:13: Return value type double does not match declared type float:
    (sin(Numeral))
calculate.c:45:13: Return value type double does not match declared type float:
    (cos(Numeral))
calculate.c:47:13: Return value type double does not match declared type float:
    (tan(Numeral))
calculate.c:51:11: Return value type double does not match declared type float:
    (HUGE_VAL)
Finished checking --- 15 code warnings

```

Figure 2.23: Результат команды splint calculate.c

```

vdkabanova@dk3n38 ~/work/os/lab_prog $ splint main.c
Splint 3.1.2 --- 13 Jan 2021

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:11:3: 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:13:14: 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:13:11: Corresponding format code
main.c:13:3: Return value (type int) ignored: scanf("%s", &Op...
main.c:15:3: Format string for printf has 0 args, given 1
    Types are incompatible. (Use -type to inhibit warning)
Finished checking --- 5 code warnings

```

Figure 2.24: Результат команды splint main.c

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: окодирование – по сути создание исходного текста программы (возможно в нескольких вариантах); – анализ разработанного кода; сборка, компиляция и разработка исполняемого модуля; отестирование 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) .swp воспринимаются gcc как программы на языке C, файлы с расширением .c – как файлы на языке C++, а файлы с расширением .o – как файлы на языке C. Например, в команде «gcc -o 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 в качестве параметра задать имя создаваемого файла: «gcc -o hello main.c». В ходе выполнения данной лабораторной работы я приобрела простейшие навыки разработки, анализа, тестирования и отладки приложений в ОС

типа UNIX/Linuxна примере создания на языке программирования С калькулятора с простейшими функциями.п.с».

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 makefileили 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 Makefileможет 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 Makefileимеет 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 = gccCFLAGS =# Compile abcd.c normallyabcd:
abcd.c$(CC) -o abcd $(CFLAGS) abcd.cclean:-rm abcd.o ~# EndMakefileforabcd.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 (вВ 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. 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-hi` и `mangdb`.

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: `xcscope` – исследование функций, содержащихся в программе, `xlint` – 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 Санализатор `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.

3 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.