

Отчёт по лабораторной работе № 14

Перелыгин Сергей Викторович

Содержание

1 Цель работы	5
2 Выполнение лабораторной работы	6
3 Ответы на контрольные вопросы	16
4 Выводы	22
5 Библиография	23

Список иллюстраций

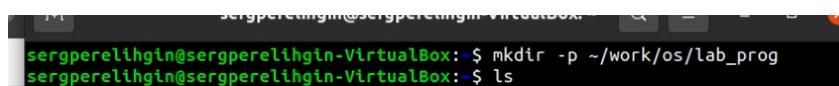
2.1	Создаем каталог	6
2.2	Создаем файлы в каталоге	6
2.3	calculate.c	7
2.4	calculate.c	8
2.5	calculate.h	8
2.6	main.c	9
2.7	Скомпилируем программы	9
2.8	Исправленный Makefile	10
2.9	make clean	10
2.10	Компиляция файлов	11
2.11	Отладка программы	11
2.12	Команда run	11
2.13	Команда list	12
2.14	Команда list 12,15	12
2.15	list calculate.c:20,29	12
2.16	Точка останова	13
2.17	Информация о точках останова	13
2.18	Запуск программы	13
2.19	Команда print Numeral	14
2.20	Команды display Numeral	14
2.21	Убираем точки останова	14
2.22	splint calculate.c	15

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 Выполнение лабораторной работы

1. В домашнем каталоге создаю подкаталог `~/work/os/lab_prog` с помощью команды `mkdir -p ~/work/os/lab_prog` (Рисунок 1).



```
sergperelihgin@sergperelihgin-VirtualBox:~$ mkdir -p ~/work/os/lab_prog
sergperelihgin@sergperelihgin-VirtualBox:~$ ls
```

Рис. 2.1: Создаем каталог

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



```
dash: cd: /work/os/lab_prog: No such file or directory
sergperelihgin@sergperelihgin-VirtualBox:~/work/os/lab_prog$ touch calculate.h calculate.c main.c
sergperelihgin@sergperelihgin-VirtualBox:~/work/os/lab_prog$ ls
calculate.c calculate.h main.c
sergperelihgin@sergperelihgin-VirtualBox:~/work/os/lab_prog$
```

Рис. 2.2: Создаем файлы в каталоге

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, приступил к редактированию созданных файлов. Реализация функций калькулятора в файле `calculate.c` (Рисунки 3, 4).

emacs@sergperelihgin-VirtualBox

File Edit Options Buffers Tools C Help

Save Undo

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

#include <stdio.h>
#include <math.h>
#include<string.h>
#include "calculate.h"
float
Calculate(float Numeral, char Operation[4])
{
    float SecondNumeral;
    if(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. 2.3: calculate.c

```

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

```

Рис. 2.4: calculate.c

Интерфейсный файл calculate.h, описывающий формат вызова функции калькулятора (Рисунок 5).

```

#ifndef CALCULATE_H_
#define CALCULATE_H_

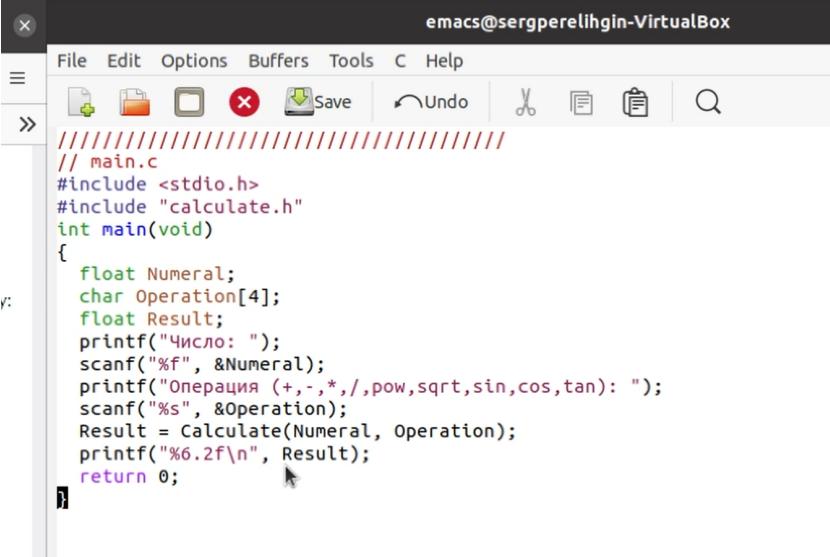
float Calculate(float Numerical, char Operation[4]);

#endif /*CALCULATE_H_*/

```

Рис. 2.5: calculate.h

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

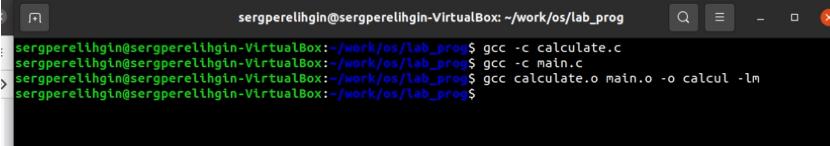


```
emacs@sergperelhgin-VirtualBox
```

```
File Edit Options Buffers Tools C Help
Save Undo Cut Copy Find Search
// 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;
}
```

Рис. 2.6: main.c

3. Выполнил 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» (Рисунок 7).



```
sergperelhgin@sergperelhgin-VirtualBox:~/work/os/lab_prog$ gcc -c calculate.c
sergperelhgin@sergperelhgin-VirtualBox:~/work/os/lab_prog$ gcc -c main.c
sergperelhgin@sergperelhgin-VirtualBox:~/work/os/lab_prog$ gcc calculate.o main.o -o calcul -lm
sergperelhgin@sergperelhgin-VirtualBox:~/work/os/lab_prog$
```

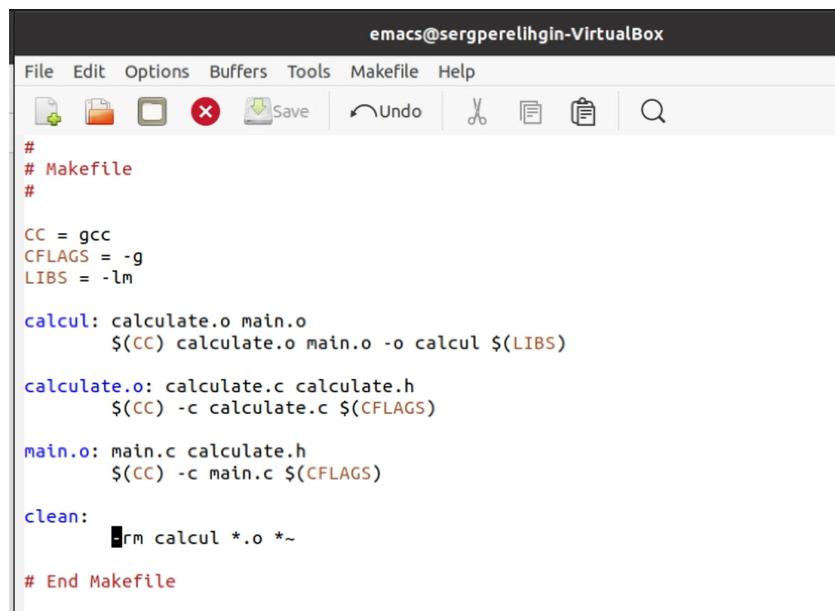
Рис. 2.7: Скомпилируем программы

4. Исправил синтаксические ошибки.
5. Создал Makefile с необходимым содержанием.

Данный файл необходим для автоматической компиляции файлов calculate.c (цель calculate.o), main.c (цель main.o), а также их объединения в один исполняемый файл calcul (цель calcul). Цель clean нужна для автоматического удаления файлов.

Переменная CC отвечает за утилиту для компиляции. Переменная CFLAGS отвечает за опции в данной утилите. Переменная LIBS отвечает за опции для объединения объектных файлов в один исполняемый файл.

6. Далее исправил Makefile (Рисунок 8).



The screenshot shows an Emacs window titled "emacs@sergperelihgin-VirtualBox". The buffer 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
```

Рис. 2.8: Исправленный Makefile

В переменную CFLAGS добавил опцию -g, необходимую для компиляции объектных файлов и их использования в программе отладчика GDB. Сделал так, что утилита компиляции выбирается с помощью переменной CC. После этого я удалил исполняемые и объектные файлы из каталога с помощью команды «make clean» (Рисунок 9). Выполнил компиляцию файлов, используя команды «make calculate.o», «make main.o», «make calcul» (Рисунок 10).



The screenshot shows a terminal window with the following command and output:

```
sergperelihgin@sergperelihgin-VirtualBox:~/work/os/lab_prog$ make clean  
rm calcul *.o *~  
sergperelihgin@sergperelihgin-VirtualBox:~/work/os/lab_prog$
```

Рис. 2.9: make clean

```

sergperelhgin@sergperelhgin-VirtualBox:~/work/os/lab_prog$ make calculate.o
gcc -c calculate.c -g
sergperelhgin@sergperelhgin-VirtualBox:~/work/os/lab_prog$ make main.o
gcc -c main.c -g
sergperelhgin@sergperelhgin-VirtualBox:~/work/os/lab_prog$ make calcul
gcc calculate.o main.o -o calcul -lm
sergperelhgin@sergperelhgin-VirtualBox:~/work/os/lab_prog$ 

```

Рис. 2.10: Компиляция файлов

Далее с помощью gdb выполнил отладку программы calcul. Запустил отладчик GDB, загрузив в него программу для отладки, используя команду: «gdb ./calcul» (Рисунок 11).

```

sergperelhgin@sergperelhgin-VirtualBox:~/work/os/lab_prog$ gdb ./calcul
GNU gdb (Ubuntu 9.2-0ubuntu1-20.04) 9.2
Copyright (C) 2020 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:
<http://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...
(gdb) 

```

Рис. 2.11: Отладка программы

Для запуска программы внутри отладчика ввел команду «run» (Рисунок 12).

```

(gdb) run
Starting program: /home/sergperelhgin/work/os/lab_prog/calcul
Число: 1
Операция (+,-,*,/ ,pow,sqrt,sin,cos,tan): +
Второе слагаемое: 2
      3.00
[Inferior 1 (process 25187) exited normally]
(gdb) 

```

Рис. 2.12: Команда run

Для постраничного (по 10 строк) просмотра исходного кода использовал команду «list» (Рисунок 13).

```
(gdb) list
1      /////////////////////////////////
2      // main.c
3
4      #include <stdio.h>
5      #include "calculate.h"
6
7      int
8      main (void)
9      {
10         float Numeral;
(gdb) list
11         char Operation[4];
12         float Result;
13         printf("Число: ");
14         scanf("%f", &Numeral);
15         printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): ");
16         scanf("%s", Operation);
17         Result = Calculate(Numeral, Operation);
18         printf("%6.2f\n", Result);
19         return 0;
20     }
```

Рис. 2.13: Команда list

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

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

Рис. 2.14: Команда list 12,15

Для просмотра определённых строк не основного файла использовал команду «list calculate.c:20,29» (Рисунок 15).

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

Рис. 2.15: list calculate.c:20,29

Установил точку останова в файле calculate.c на строке номер 21, используя

команды «list calculate.c:20,27» и «break 21» (Рисунок 16).

```
(gdb) list calculate.c:20,27
20      {
21          printf("Вычитаемое: ");
22          scanf("%f", &SecondNumeral);
23          return(Numeral - SecondNumeral);
24      }
25      else if(strcmp(Operation, "*", 1) == 0)
26      {
27          printf("Множитель: ");
(gdb) break 21
Breakpoint 1 at 0x55555555552dd: file calculate.c, line 21.
(gdb)
```

Рис. 2.16: Точка останова

Вывел информацию об имеющихся в проекте точках останова с помощью команды «info breakpoints» (Рисунок 17).

```
breakpoint 1 at 0x55555555552dd: file calculate.c, line 21.
(gdb) info breakpoints
Num  Type    Disp Enb Address      What
1   breakpoint  keep y  0x0000555555552dd in Calculate at calculate.c:21
(gdb) run
```

Рис. 2.17: Информация о точках останова

Запустил программу внутри отладчика и убедился, что программа остановилась в момент прохождения точки останова. Использовал команды «run», «5», «-» и «backtrace» (Рисунок 18).

```
* breakpoint 1 keep y
(gdb) run
Starting program: /home/sergperelihgin/work/os/lab_prog/calcul
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): -
Breakpoint 1, Calculate (Numeral=5, Operation=0x7fffffffde54 "-") at calculate.c:21
21      printf("Вычитаемое: ");
(gdb) backtrace
#0  Calculate (Numeral=5, Operation=0x7fffffffde54 "-") at calculate.c:21
#1  0x00005555555555bd in main () at main.c:17
(gdb) ■
```

Рис. 2.18: Запуск программы

Посмотрел, чему равно на этом этапе значение переменной Numeral, введя команду «print Numeral» (Рисунок 19).

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

Рис. 2.19: Команда print Numeral

Сравнил с результатом вывода на экран после использования команды «display Numeral». Значения совпадают (Рисунок 20).

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

Рис. 2.20: Команды display Numeral

Убрал точки останова с помощью команд «info breakpoints» и «delete 1» (Рисунок 21).

```
(gdb) info breakpoints
Num      Type            Disp Enb Address          What
1       breakpoint      keep y 0x0000555555552dd in Calculate at calculate.c:21
breakpoint already hit 1 time
(gdb) delete 1
(gdb) █
```

Рис. 2.21: Убираем точки останова

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

Предварительно я установил данную утилиту с помощью команд «sudo apt update» и «sudo apt install splint».

Далее воспользовался командами «splint calculate.c» и «splint main.c» (Рисунки 22, 23). 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 потери данных.

```
sergperelihgin@sergperelihgin-VirtualBox:~/work/os/lab_prog$ splint calculate.c
Splint 3.1.2 --- 20 Feb 2018

calculate.h:7:37: Function parameter Operation declared as manifest array (size
    constant is meaningless)
    A formal parameter is declared as an array with size. The size of the array
    is ignored in this context, since the array formal parameter is treated as a
    pointer. (Use -fixedformalarray to inhibit warning)
calculate.c:10:31: Function parameter Operation declared as manifest array
    (size constant is meaningless)
calculate.c: (in function Calculate)
calculate.c:16:7: 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:22:7: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:28:7: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:34:7: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:35:10: Dangerous equality comparison involving float types:
    SecondNumeral == 0
    Two real (float, double, or long double) values are compared directly using
    == or != primitive. This may produce unexpected results since floating point
    representations are inexact. Instead, compare the difference to FLT_EPSILON
    or DBL_EPSILON. (Use -realcompare to inhibit warning)
calculate.c:38:10: Return value type double does not match declared type float:
    (HUGE_VAL)
    To allow all numeric types to match, use +relaxtypes.
calculate.c:45:7: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:46:13: Return value type double does not match declared type float:
    (pow(Numerical, SecondNumeral))
calculate.c:49:11: Return value type double does not match declared type float:
    (sqrt(Numerical))
calculate.c:51:11: Return value type double does not match declared type float:
    (sin(Numerical))
calculate.c:53:11: Return value type double does not match declared type float:
    (cos(Numerical))
calculate.c:55:11: Return value type double does not match declared type float:
    (tan(Numerical))
calculate.c:59:13: Return value type double does not match declared type float:
    (HUGE_VAL)

Finished checking --- 15 code warnings
```

Рис. 2.22: splint calculate.c

```
sergperelihgin@sergperelihgin-VirtualBox:~/work/os/lab_prog$ splint main.c
Splint 3.1.2 --- 20 Feb 2018

calculate.h:7:37: Function parameter Operation declared as manifest array (size
    constant is meaningless)
    A formal parameter is declared as an array with size. The size of the array
    is ignored in this context, since the array formal parameter is treated as a
    pointer. (Use -fixedformalarray to inhibit warning)
main.c: (in function main)
main.c:14:3: Return value (type int) ignored: scanf("%f", &Num...
    Result returned by function call is not used. If this is intended, can cast
    result to (void) to eliminate message. (Use -retvalint to inhibit warning)
main.c:16:3: Return value (type int) ignored: scanf("%s", Oper...

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

Рис. 2.23: splint main.c

3 Ответы на 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:
 1. кодирование - по сути создание исходного текста программы (возможно в нескольких вариантах); - анализ разработанного кода;
 2. сборка, компиляция и разработка исполняемого модуля;
 3. тестирование и отладка, сохранение произведённых изменений;
 - документирование.

Для создания исходного текста программы разработчик может воспользоваться любым удобным для него редактором текста: vi, vim, mceditor, emacs, geany и др.

После завершения написания исходного кода программы (возможно состоящей из нескольких файлов), необходимо её скомпилировать и получить исполняемый модуль.

- 3) Для имени входного файла суффикс определяет какая компиляция требуется. Суффиксы указывают на тип объекта. Файлы с расширением (суффиксом) .c воспринимаются gcc как программы на языке C, файлы с расширением .cc или .C - как файлы на языке C++, а файлы с расширением .o считаются объектными. Например, в команде «gcc-c main.c»: gcc по расширению (суффиксу) .c распознает тип файла для компиляции и формирует объектный модуль - файл с расширением .o. Если требуется получить исполняемый файл с определённым именем (например, hello), то требуется воспользоваться опцией -o и в качестве параметра задать имя создаваемого файла: «gcc -o hello main.c».
- 4) Основное назначение компилятора языка Си в UNIX заключается в компиляции всей программы и получении исполняемого файла/модуля.
- 5) Для сборки разрабатываемого приложения и собственно компиляции полезно воспользоваться утилитой make. Она позволяет автоматизировать процесс преобразования файлов программы из одной формы в другую, отслеживает взаимосвязи между файлами.
- 6) Для работы с утилитой make необходимо в корне рабочего каталога с Вашим проектом создать файл с названием makefile или Makefile, в котором будут описаны правила обработки файлов Вашего программного комплекса.

В самом простом случае Makefile имеет следующий синтаксис:

... : ... <команда 1>

...

Сначала задаётся список целей, разделённых пробелами, за которым идёт двоеточие и список зависимостей. Затем в следующих строках указываются команды. Строки с командами обязательно должны начинаться с табуляции.

В качестве цели в Makefile может выступать имя файла или название какого-то действия. Зависимость задаёт исходные параметры (условия) для достижения указанной цели. Зависимость также может быть названием какого-то действия. Команды - собственно действия, которые необходимо выполнить для достижения цели.

Общий синтаксис Makefile имеет вид:

```
target1 [target2... ]:: [dependments1... ]
[(tab)commands] [#commentary]
[(tab)commands] [#commentary]
```

Здесь знак `#` определяет начало комментария (содержимое от знака `#` и до конца строки не будет обрабатываться). Одинарное двоеточие указывает на то, что последовательность команд должна содержаться в одной строке. Для переноса можно в длинной строке команд можно использовать обратный слэш `\`. Двойное двоеточие указывает на то, что последовательность команд может содержаться в нескольких последовательных строках.

Пример более сложного синтаксиса Makefile:

```
#
# Makefile for abcd.c
#
CC = gcc
CFLAGS =
# Compile abcd.c normally
abcd: abcd.c
$(CC) -o abcd $(CFLAGS) abcd.c
```

clean:

```
-rm abcd *.o *~
# End Make file for abcd.c
```

В этом примере в начале файла заданы три переменные: CC и CFLAGS. Затем указаны цели, их зависимости и соответствующие команды. В командах происходит обращение к значениям переменных. Цель с именем clean производит очистку каталога от файлов, полученных в результате компиляции. Для её описания использованы регулярные выражения.

7) Во время работы над кодом программы программист неизбежно сталкивается с появлением ошибок в ней. Использование отладчика для поиска и устранения ошибок в программе существенно облегчает жизнь программиста. В комплекте программ GNU для ОС типа UNIX входит отладчик GDB (GNUDebugger).

Для использования GDB необходимо скомпилировать анализируемый код программы таким образом, чтобы отладочная информация содержалась в результирующем бинарном файле. Для этого следует воспользоваться опцией -gкомпилятора gcc:

```
gcc -c file.c -g
```

После этого для начала работы с gdb необходимо в командной строке ввести одноимённую команду, указав в качестве аргумента анализируемый бинарный файл:

```
gdb file.o
```

8) Основные команды отладчика gdb:

- backtrace - вывод на экран пути к текущей точке останова (по сути вывод - названий всех функций)
- break - установить точку останова (в качестве параметра может быть указан номер строки или название функции)
- clear - удалить все точки останова в функции
- continue - продолжить выполнение программы
- delete - удалить точку останова
- display - добавить выражение в список выражений, значения которых отображаются при достижении точки останова программы

- finish - выполнить программу до момента выхода из функции
- info breakpoints - вывести на экран список используемых точек останова
- info watchpoints - вывести на экран список используемых контрольных выражений
- list - вывести на экран исходный код (в качестве параметра может быть указано название файла и через двоеточие номера начальной и конечной строк)
- next - выполнить программу пошагово, но без выполнения вызываемых в программе функций
- print - вывести значение указываемого в качестве параметра выражения
- run - запуск программы на выполнение
- set - установить новое значение переменной
- step - пошаговое выполнение программы
- watch - установить контрольное выражение, при изменении значения которого программа будет остановлена

Для выхода из gdb можно воспользоваться командой quit (или её сокращённым вариантом q) или комбинацией клавиш Ctrl-d. Более подробную информацию по работе с gdb можно получить с помощью команд gdb -h и man gdb.

9) Схема отладки программы показана в 6 пункте лабораторной работы.

10) При запуске компилятор выдал ошибку в строке:

```
scanf("%s", &Operation);
```

нужно убрать знак &, потому что имя массива символов уже является указателем на первый элемент этого массива.

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

- cscope - исследование функций, содержащихся в программе,
- lint - критическая проверка программ, написанных на языке Си.

12) Утилита `splint` анализирует программный код, проверяет корректность задания аргументов использованных в программе функций и типов возвращаемых значений, обнаруживает синтаксические и семантические ошибки.

В отличие от компилятора С анализатор `splint` генерирует комментарии с описанием разбора кода программы и осуществляет общий контроль, обнаруживая такие ошибки, как одинаковые объекты, определённые в разных файлах, или объекты, чьи значения не используются в работе программы, переменные с некорректно заданными значениями типами и многое другое.

4 Выводы

В ходе выполнения данной лабораторной работы я приобрел 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 Библиография

- Кулябов Д.С. Операционные системы: лабораторные работы: учебное пособие / Д.С. Кулябов, М.Н. Геворкян, А.В. Королькова, А.В. Демидова. — М. : Изд-во РУДН, 2016. — 117 с. — ISBN 978-5-209-07626-1 : 139.13; То же [Электронный ресурс]. — URL: <http://lib.rudn.ru/MegaPro2/Download/MObject/6118>.
- Робачевский А.М. Операционная система UNIX [текст] : Учебное пособие / А.М. Робачевский, С.А. Немлюгин, О.Л. Стесик. — 2-е изд., перераб. и доп. — СПб. : БХВ-Петербург, 2005, 2010. — 656 с. : ил. — ISBN 5-94157-538-6 : 164.56. (ET 60)
- Таненбаум Эндрю. Современные операционные системы [Текст] / Э. Таненбаум. — 2-е изд. — СПб. : Питер, 2006. — 1038 с. : ил. — (Классика Computer Science). — ISBN 5-318-00299-4 : 446.05. (ET 50)
- Ван Стеен М., Эндрю Таненбаум Распределенные системы. Принципы и парадигмы [Текст] / Э. Таненбаум, в.М. Стеен. — СПб. : Питер, 2003. — 877 с. : ил. — (Классика Computer science). — ISBN 5-272-00053-6 : 377.52. (ET 50)
- Сафонов, В.О. Основы современных операционных систем : учебное пособие / В.О. Сафонов. — Москва : Интернет-Университет Информационных Технологий, 2011. — 584 с. — (Основы информационных технологий). — ISBN 978-5-9963-0495-0 ; То же [Электронный ресурс]. — URL: <http://biblioclub.ru/index.php?page=book&id=233210>.
- Немет Эви. UNIX — руководство системного администратора [Текст] / Э. Немет, Г. Снайдер, С. Сибасс; Э.Немет, Г.Снайдер, С.Сибасс, Х.Р.Трент. — 3-е изд. — СПб. : Питер, 2004. — 925 с. : ил. — (Для профессионалов). — ISBN

0-13-020601-6. — ISBN 5-318-00754-6 : 280.00. (ET 30)

- Бек Л. Введение в системное программирование [Текст] / Л. Бек; Пер. с англ. Н.А.Богомолова, В.М.Вязовского и С.Е.Морковина; Под ред. Л.Н.Королева. — М. : Мир, 1988. — 448 с. : ил. — ISBN 5-03-000011-9 : 2.60. (ET 3)
- Дьяконов Владимир Юрьевич. Системное программирование [Текст] : Учебное пособие для втузов / В.Ю. Дьяконов, В.А. Китов, И.А. Калинчев; Под ред. А.Л.Горелика. — М. : Высшая школа, 1990. — 221 с. : ил. — ISBN 5-06-000732-4 : 0.55.