

Отчёт по лабораторной работе №14

Дисциплина: Операционные системы

Матвеева Анастасия Сергеевна

Содержание

1	Цель работы	4
2	Задачи	5
3	Выполнение лабораторной работы	6
4	Выводы	18
5	Контрольные вопросы	19

List of Figures

3.1	Создание подкаталога и файлов	6
3.2	Файл calculate.c	7
3.3	Файл calculate.c	8
3.4	Файл calculate.h	8
3.5	Файл main.c	9
3.6	Выполняем компиляцию посредством gcc	9
3.7	Исправленные ошибки	10
3.8	Makefile	10
3.9	Исправленный Makefile	11
3.10	Компиляция файлов	12
3.11	Запуск отладчика gdb и программы	12
3.12	Команда list	13
3.13	Команда list 12,15	13
3.14	list calculate.c:12,20	13
3.15	Установка точки останова	14
3.16	Информация о точках останова	14
3.17	Запуск программы	14
3.18	Значение переменной Numeral	15
3.19	Убираем точки останова	15
3.20	splint calculate.c	16
3.21	splint main.c	16

1 Цель работы

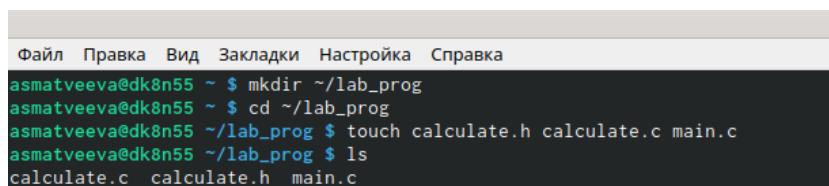
Приобрести простейшие навыки разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linux на примере создания на языке программирования С калькулятора с простейшими функциями.

2 Задачи

1. Познакомиться со стандартным средством для компиляции программ в ОС типа UNIX - GCC (GNU Compiler Collection).
2. Познакомиться с отладчиком GDB (GNU Debugger).
3. Познакомиться с утилитой splint.
4. В ходе работы проанализировать и выполнить данные программы.

3 Выполнение лабораторной работы

1. В домашнем каталоге создаю подкаталог `~/work/os/lab_prog` с помощью команды «`mkdir -p ~/work/os/lab_prog`». Далее 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. 3.1)



```
Файл  Правка  Вид  Закладки  Настройка  Справка
asmatveeva@dk8n55 ~ $ mkdir ~/lab_prog
asmatveeva@dk8n55 ~ $ cd ~/lab_prog
asmatveeva@dk8n55 ~/lab_prog $ touch calculate.h calculate.c main.c
asmatveeva@dk8n55 ~/lab_prog $ ls
calculate.c  calculate.h  main.c
```

Figure 3.1: Создание подкаталога и 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. 3.2, 3.3)

```
File Edit Options Buffers Tools C Help
////////////////////////////////////
// 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)
    {

```

Figure 3.2: Файл calculate.c

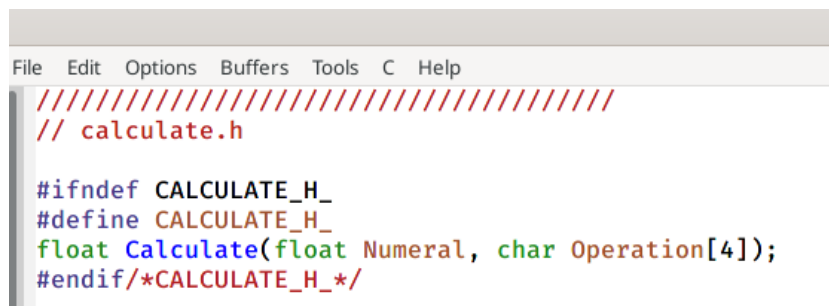
```

        printf("Степень: ");
        scanf("%f", &SecondNumeral);
        return(pow(Numeral, SecondNumeral));
    }
    else if(strncmp(Operation, "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);
    }
}

```

Figure 3.3: Файл calculate.c

Интерфейсный файл calculate.h, описывающий формат вызова функции калькулятора. (рис. 3.4)



```

// calculate.h

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

```

Figure 3.4: Файл calculate.h

Основной файл main.c, реализующий интерфейс пользователя к калькулятору. (рис. 3.5)



```

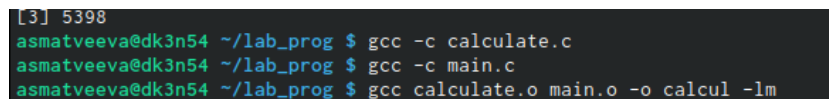
////////////////////////////////////
// main.c
#include<stdio.h>
#include"calculate.h"

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;
}

```

Figure 3.5: Файл main.c

2. 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. 3.6)



```

[3] 5398
asmatveeva@dk3n54 ~/lab_prog $ gcc -c calculate.c
asmatveeva@dk3n54 ~/lab_prog $ gcc -c main.c
asmatveeva@dk3n54 ~/lab_prog $ gcc calculate.o main.o -o calcul -lm

```

Figure 3.6: Выполняем компиляцию посредством gcc

3. Так как в ходе предыдущей компиляции программы выявились синтаксические ошибки, поэтому я их исправила и выполнила компиляцию заново. В строке “scanf(“%s”, &Operation);” нужно было убрать знак &, потому что имя массива символов уже является указателем на первый элемент этого массива.(рис. 3.7)

```

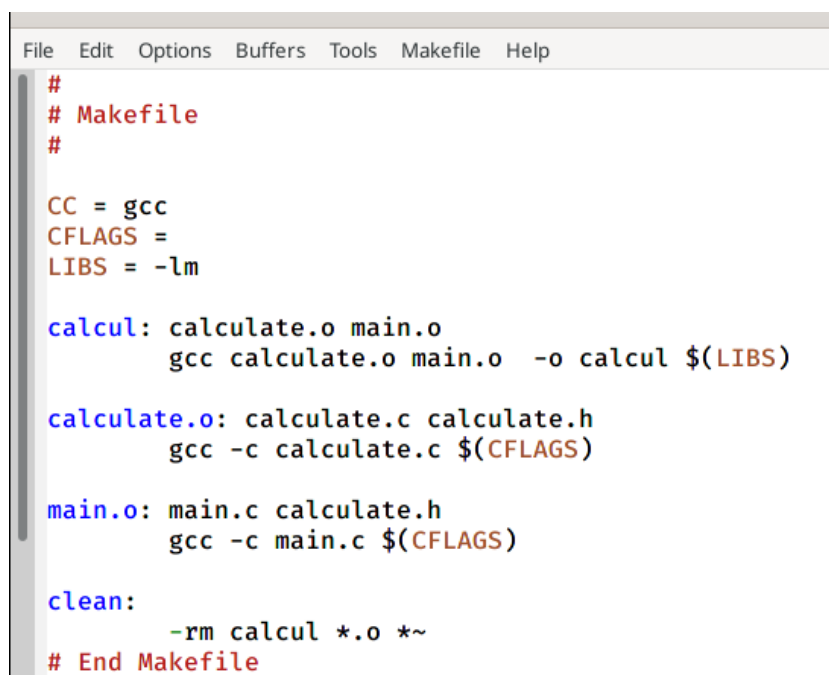
////////////////////////////////////
// main.c
#include<stdio.h>
#include"calculate.h"

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("%.2f\n",Result);
    return 0;
}

```

Figure 3.7: Исправленные ошибки

4. Создала Makefile с необходимым содержанием (рис. 3.8)



```

File Edit Options Buffers Tools Makefile Help

#
# 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

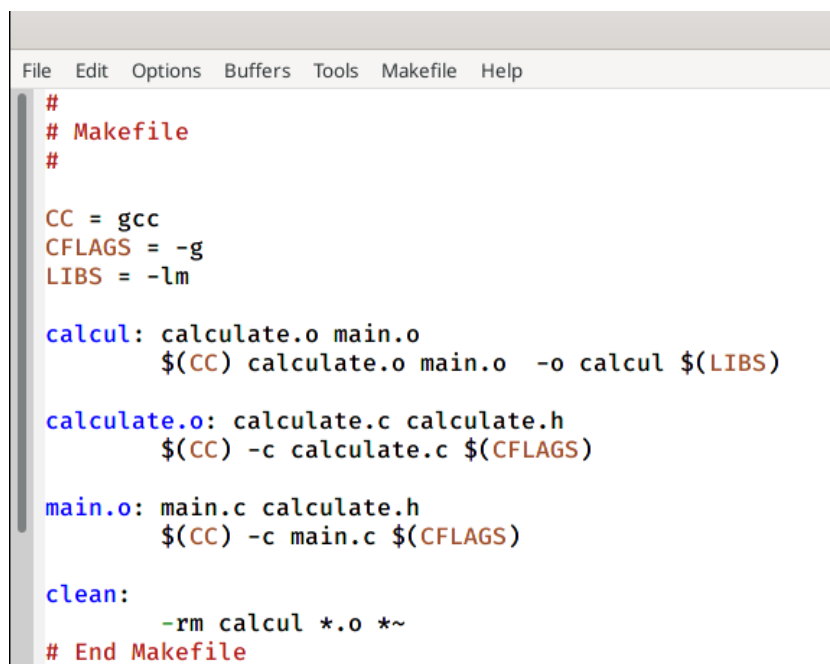
```

Figure 3.8: Makefile

Данный файл необходим для автоматической компиляции файлов calculate.c (цель calculate.o), main.c (цель main.o), а также их объединения в один испол-

няемый 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.

5. Далее исправила Makefile.(рис. 3.9)



```
#
# 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
```

Figure 3.9: Исправленный 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 clean`». Выполнила компиляцию файлов, используя команды «`make calculate.o`», «`make main.o`», «`make calcul`».(рис. 3.10)

```

asmatveeva@dk3n54 ~/lab_prog $ make clean
rm calcul *.o *~
asmatveeva@dk3n54 ~/lab_prog $ make main.o
gcc -c main.c
asmatveeva@dk3n54 ~/lab_prog $ make calculate.o
gcc -c calculate.c
asmatveeva@dk3n54 ~/lab_prog $ make calcul
gcc calculate.o main.o -o calcul -lm

```

Figure 3.10: Компиляция файлов

Далее с помощью gdb выполнила отладку программы calcul. Запустила отладчик GDB, загрузив в него программу для отладки, используя команду: «gdb ./calcul». Для запуска программы внутри отладчика ввела команду «run». (рис. 3.11)

```

asmatveeva@dk3n54 ~/lab_prog $ gdb ./calcul
GNU gdb (Gentoo 10.1 vanilla) 10.1
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-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<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...
(No debugging symbols found in ./calcul)
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/a/s/asmatveeva/lab_prog/calcul
Число: 6
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): *
Множитель: 3
18.00
[Inferior 1 (process 5862) exited normally]

```

Figure 3.11: Запуск отладчика gdb и программы

Для постраничного (по 10 строк) просмотра исходного кода использовала команду «list». { #fig:012 width=70% }

```

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

```

Figure 3.12: Команда list

Для просмотра строк с 12 по 15 основного файла использовала list с параметрами: “list 12,15”. (рис. 3.13)

```

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

```

Figure 3.13: Команда list 12,15

Для просмотра определённых строк не основного файла использовала list с параметрами: “list calculate.c:12,20”. (рис. 3.14)

```

(gdb) list calculate.c: 12,20
12     {
13         printf("Второе слагаемое: ");
14         scanf("%f", &SecondNumeral);
15         return(Numeral+SecondNumeral);
16     }
17     else if(strncmp(Operation, "-", 1) == 0)
18     {
19         printf("Вычитаемое: ");
20         scanf("%f", &SecondNumeral);

```

Figure 3.14: list calculate.c:12,20

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

”list calculate.c:20,27
break 21”. (рис. 3.15)

```
(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 0x97e: file calculate.c, line 21.
```

Figure 3.15: Установка точки останова

Вывела информацию об имеющихся в проекте точках останова: “info breakpoints”. (рис. 3.16)

```
(gdb) info breakpoints
Num   Type             Disp Enb Address            What
1     breakpoint      keep y   0x00000000000097e in Calculate at calculate.c:21
```

Figure 3.16: Информация о точках останова

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

”run

5

•

backtrace”. (рис. 3.17)

```
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/a/s/asmatveeva/lab_prog/calcul
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): -
Вычитаемое: 2
Breakpoint 1, Calculate (Numeral=5, Operation=0x7fffffffce34 "-") at calculate.c:21
21         return(Numeral-SecondNumeral);
(gdb) backtrace
#0 Calculate (Numeral=5, Operation=0x7fffffffce34 "-") at calculate.c:21
#1 0x000055555400c31 in main () at main.c:16
```

Figure 3.17: Запуск программы

Посмотрела, чему равно на этом этапе значение переменной Numeral, введя команду «print Numeral». Сравнила с результатом вывода на экран после использования команды «display Numeral». Значения совпадают. (рис. 3.18)

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

Figure 3.18: Значение переменной Numeral

Убрала точки останова с помощью команд «info breakpoints» и «delete 1». (рис. 3.19)

```
(gdb) info breakpoints
Num   Type             Disp Enb Address            What
1     breakpoint        keep y   0x00005555540097e in calculate at calculate.c:21
      breakpoint already hit 1 time
(gdb) delete 1
```

Figure 3.19: Убираем точки останова

С помощью утилиты splint проанализировала коды файлов calculate.c и main.c. Я воспользовалась командами «splint calculate.c» и «splint main.c». (рис. 3.20, 3.21)

```

asmatveeva@dk3n54 ~/lab_prog $ splint calculate.c
Splint 3.1.2 --- 13 Jan 2021

calculate.h:6:37: 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:8:32: Function parameter Operation declared as manifest array (size
        constant is meaningless)
calculate.c: (in function Calculate)
calculate.c:14:7: 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:20:7: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:26:7: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:32:5: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:33:8: 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:36:8: Return value type double does not match declared type float:
        (HUGE_VAL)
    To allow all numeric types to match, use +relaxtypes.
calculate.c:43:7: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:44:13: Return value type double does not match declared type float:
        (pow(Numeral, SecondNumeral))
calculate.c:47:11: Return value type double does not match declared type float:
        (sqrt(Numeral))
calculate.c:49:11: Return value type double does not match declared type float:
        (sin(Numeral))
calculate.c:51:11: Return value type double does not match declared type float:
        (cos(Numeral))
calculate.c:53:11: Return value type double does not match declared type float:
        (tan(Numeral))
calculate.c:57:13: Return value type double does not match declared type float:
        (HUGE_VAL)

Finished checking --- 15 code warnings

```

Figure 3.20: splint calculate.c

```

asmatveeva@dk3n54 ~/lab_prog $ splint main.c
Splint 3.1.2 --- 13 Jan 2021

calculate.h:6:37: 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:13:3: 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:15:3: Return value (type int) ignored: scanf("%s", Oper...

Finished checking --- 3 code warnings

```

Figure 3.21: splint main.c

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.

4 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.

5 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...]:[:] [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 = gcc
```

```
CFLAGS =
```

```
# Compile abcd.c normaly
```

```
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 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 (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 и типами и многое другое. # Библиография

1. https://esystem.rudn.ru/pluginfile.php/1142099/mod_resource/content/2/011-lab_prog.pdf
2. 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>.
3. Робачевский А.М. Операционная система UNIX [текст] : Учебное пособие / А.М. Робачевский, С.А. Немнюгин, О.Л. Стесик. — 2-е изд., перераб. и доп. — СПб. : БХВ-Петербург, 2005, 2010. — 656 с. : ил. — ISBN 5-94157-538-6 : 164.56. (ЕТ 60)

4. 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. (ЕТ 50)