

Лабораторная работа №14

Дисциплина: Операционные системы

Колчева Юлия Вячеславовна

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
3	Выводы	18
4	Контрольные вопросы	19
5	Библиография	23

List of Tables

List of Figures

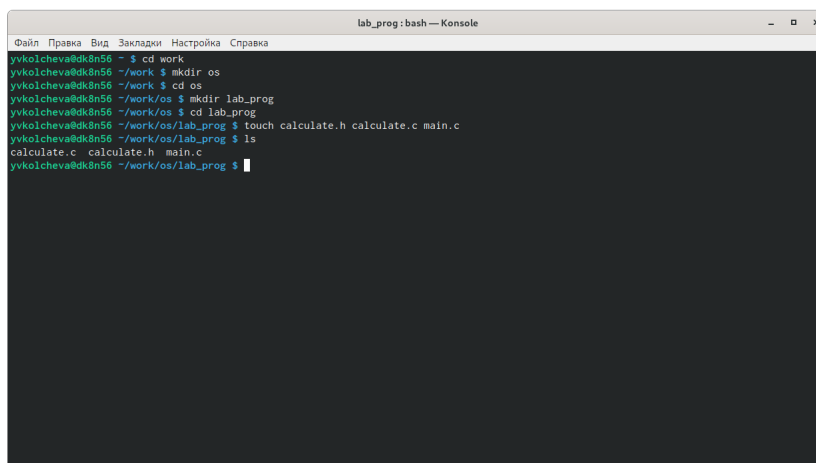
2.1	Создание	6
2.2	Первая часть	7
2.3	Вторая часть	8
2.4	Файл1	9
2.5	Файл2	10
2.6	Компиляция	10
2.7	Makefile	11
2.8	Makefile2	12
2.9	Работа в консоли	13
2.10	отладчик	13
2.11	Работа программы	14
2.12	Команда list	14
2.13	Команда list2	15
2.14	Команда list3	15
2.15	Точка останова	16
2.16	Запуск	16
2.17	splint calculate.c	17
2.18	splint main.c	17

1 Цель работы

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

2 Выполнение лабораторной работы

В домашнем каталоге создаю подкаталог `~/work/os/lab prog`. 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. 2.1)



```
lab_prog: bash — Konsole
Файл  Правка  Вид  Закладки  Настройка  Справка
yvkoicheva@dk8n56 ~ $ cd work
yvkoicheva@dk8n56 ~/work $ mkdir os
yvkoicheva@dk8n56 ~/work $ cd os
yvkoicheva@dk8n56 ~/work/os $ mkdir lab_prog
yvkoicheva@dk8n56 ~/work/os $ cd lab_prog
yvkoicheva@dk8n56 ~/work/os/lab_prog $ touch calculate.h calculate.c main.c
yvkoicheva@dk8n56 ~/work/os/lab_prog $ ls
calculate.c  calculate.h  main.c
yvkoicheva@dk8n56 ~/work/os/lab_prog $
```

Figure 2.1: Создание

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. 2.2) (рис. 2.3)

```
emac@dk8n56
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)
    {
        printf("Степень: ");
        scanf("%f",&SecondNumeral);
        return(pow(Numeral, SecondNumeral));
    }
}
U: ** - calculate.c Top L40 (C/*l Abbrev) Вт июн 1 09:15 1.12
cause of the error in your initialization file. Start Emacs with
the '--debug-init' option to view a complete error backtrace.
*Warnings* Bot L8 (Special) Вт июн 1 09:15 1.12
Beginning of buffer
```

Figure 2.2: Первая часть

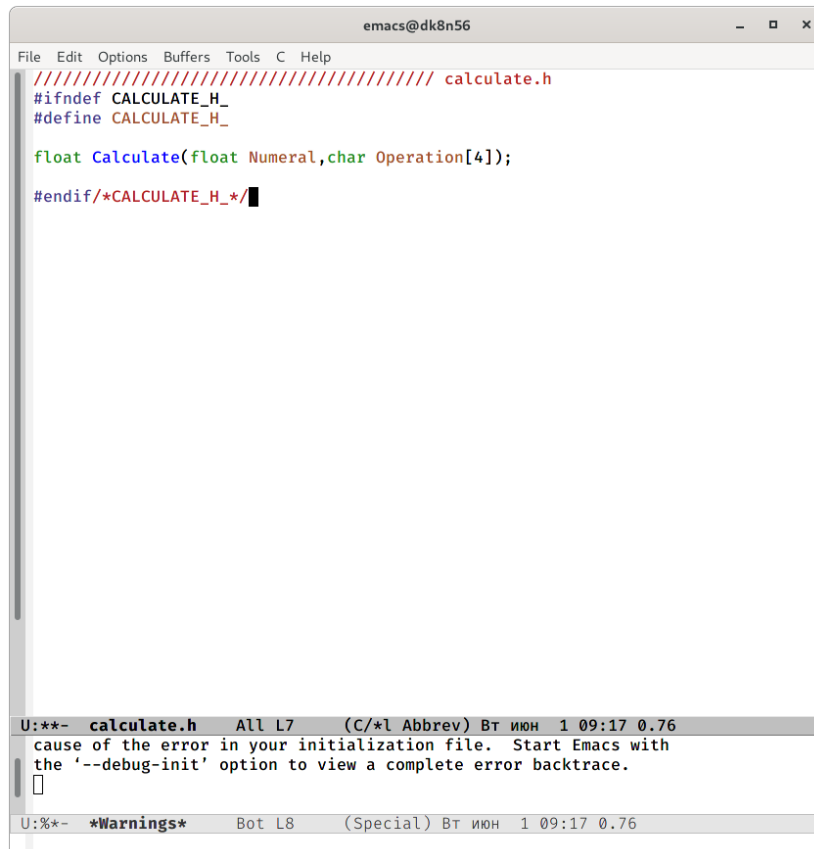
```
emacs@dk8n56
File Edit Options Buffers Tools C Help
scanf("%f",&SecondNumeral);
return(pow(Numeral, SecondNumeral));
}
else if(strncmp(Operation,"sqrt",4)==0)
return(sqrt(Numeral));
else
if(strncmp(Operation,"sin",3)==0)
return(sin(Numeral));
else if(strncmp(Operation,"cos",3)==0)
return(cos(Numeral));
else if(strncmp(Operation,"tan",3)==0)
return(tan(Numeral));
else{
printf("Неправильно введено действие ");return(HUGE_VAL);
}
}
```

U:*** calculate.c Bot L40 (C/*l Abbrev) Вт июн 1 09:15 1.12
cause of the error in your initialization file. Start Emacs with
the '--debug-init' option to view a complete error backtrace.
□

U:*** *Warnings* Bot L8 (Special) Вт июн 1 09:15 1.12

Figure 2.3: Вторая часть

Интерфейсный файл `calculate.h`, описывающий формат вызова функции калькулятора(рис. 2.4)



```
emacs@dk8n56
File Edit Options Buffers Tools C Help
//////////////////////////////////// calculate.h
#ifndef CALCULATE_H_
#define CALCULATE_H_

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

#endif /*CALCULATE_H_*/

U:**- calculate.h All L7 (C/*l Abbrev) Вт июн 1 09:17 0.76
cause of the error in your initialization file. Start Emacs with
the '--debug-init' option to view a complete error backtrace.
U:%*- *Warnings* Bot L8 (Special) Вт июн 1 09:17 0.76
```

Figure 2.4: Файл1

Основной файл main.c, реализующий интерфейс пользователя калькулято-
ру(рис. 2.5)

```

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

```

U:--- main.c All L16 (C/*l Abbrev) Вт июн 1 09:19 1.18
cause of the error in your initialization file. Start Emacs with
the '--debug-init' option to view a complete error backtrace.

U:%*- *Warnings* Bot L8 (Special) Вт июн 1 09:19 1.18
Wrote /afs/.dk.sci.pfu.edu.ru/home/y/v/yvkolcheva/work/os/lab_prog/main.c

Figure 2.5: Файл2

Выполнила компиляцию программы посредством gcc ,используя команды «gcc-ccalculate.c», «gcc -c main.c» и « gcc calculate.o main.o -o calcul-lm» (рис. 2.6)

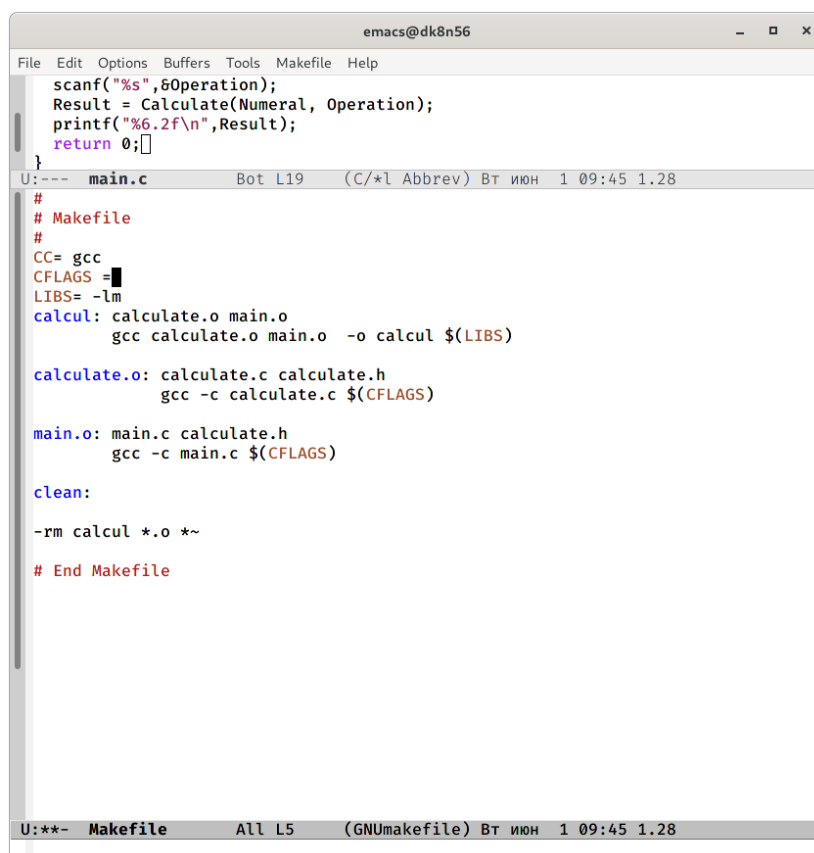
```

lab_prog: bash — Konsole
Файл Правка Вид Закладки Настройка Справка
yvkolcheva@dk8n56 ~/work/os/lab_prog $ gcc -c main.c
main.c: В функции «main»:
main.c:16:11: предупреждение: формат «%s» ожидает аргумент типа «char *», но аргумент 2 имеет тип «char (*)[4]» [-Wformat=]
16 | scanf("%s",&operation);
    |          ^
    |          |
    |          | char (*)[4]
    |          | char *
yvkolcheva@dk8n56 ~/work/os/lab_prog $ gcc -c main.c
main.c: В функции «main»:
main.c:16:11: предупреждение: формат «%s» ожидает аргумент типа «char *», но аргумент 2 имеет тип «char (*)[4]» [-Wformat=]
16 | scanf("%s",&operation);
    |          ^
    |          |
    |          | char (*)[4]
    |          | char *
yvkolcheva@dk8n56 ~/work/os/lab_prog $ gcc -c calculate.c
main.c: В функции «main»:
main.c:16:11: предупреждение: формат «%s» ожидает аргумент типа «char *», но аргумент 2 имеет тип «char (*)[4]» [-Wformat=]
16 | scanf("%s",&operation);
    |          ^
    |          |
    |          | char (*)[4]
    |          | char *
yvkolcheva@dk8n56 ~/work/os/lab_prog $ gcc calculate.o main.o -o calcul -lm
yvkolcheva@dk8n56 ~/work/os/lab_prog $ touch Makefile
yvkolcheva@dk8n56 ~/work/os/lab_prog $ make clean
Makefile:8: *** пропущен разделитель. Останов.
yvkolcheva@dk8n56 ~/work/os/lab_prog $ make clean
rm calcul *.o *~
yvkolcheva@dk8n56 ~/work/os/lab_prog $

```

Figure 2.6: Компиляция

В ходе компиляции программы никаких ошибок не было, только предупреждение, не влияющее на работу программы. Создала Makefile (рис. 2.7)



The screenshot shows an Emacs editor window titled 'emacs@dk8n56'. The top menu bar includes 'File', 'Edit', 'Options', 'Buffers', 'Tools', 'Makefile', and 'Help'. The main text area is divided into two sections. The top section shows the end of a C program in 'main.c':

```
}  
scanf("%s", &operation);  
Result = Calculate(Numeral, Operation);  
printf("%.2f\n", Result);  
return 0;  
}
```

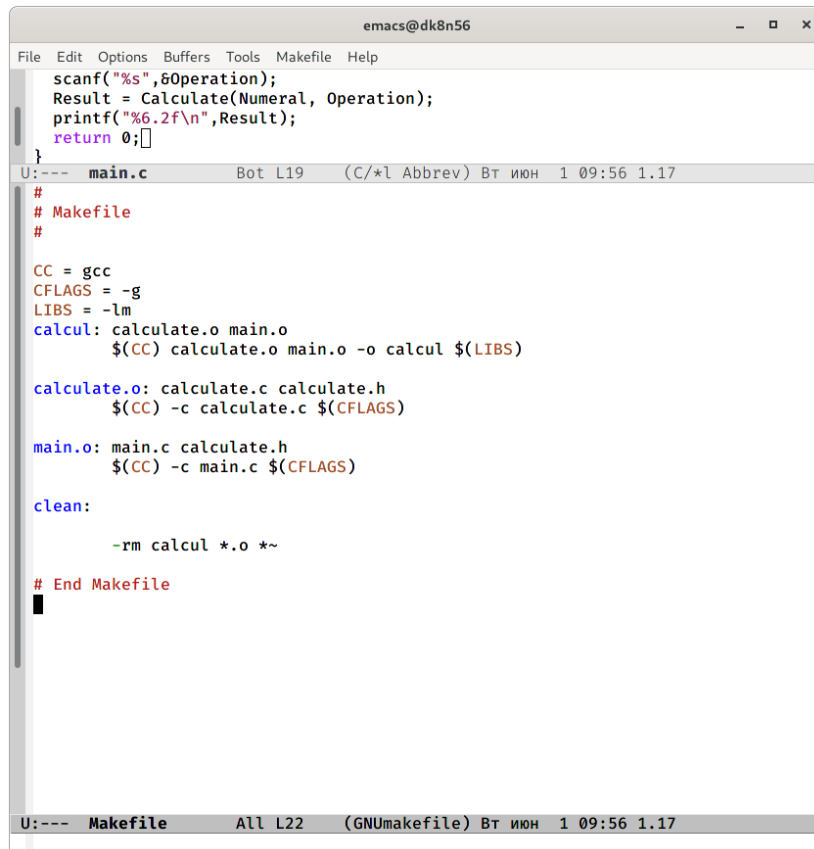

The status bar below this section reads: 'U:--- main.c Bot L19 (C/*l Abbrev) Вт июн 1 09:45 1.28'. The bottom section shows the content of a 'Makefile':

```
#  
# 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
```


The status bar at the bottom of the window reads: 'U:**-- Makefile All L5 (GNUmakefile) Вт июн 1 09:45 1.28'.

Figure 2.7: 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 исправила Makefile (рис. 2.8)



The screenshot shows an Emacs editor window titled 'emacs@dk8n56'. The top menu bar includes 'File', 'Edit', 'Options', 'Buffers', 'Tools', 'Makefile', and 'Help'. The main text area is divided into two sections. The top section shows a C program snippet with the following code:

```
scanf("%s", &operation);
Result = Calculate(Numeral, Operation);
printf("%.2f\n", Result);
return 0;
}
```

 Below this, a status line indicates the current file is 'main.c' at line 19, with a timestamp of '1 09:56 1.17'. The bottom section shows 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
```

 At the bottom, another status line indicates the current file is 'Makefile' at line 22, with the same timestamp. The Emacs interface includes a vertical scrollbar on the left and a status bar at the bottom.

Figure 2.8: Makefile2

В переменную CFLAGS добавила опцию -g,необходимую для ккомпиляции 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».Выполнила компиляцию файлов,используя команды «make calculate.o»,«make main.o»,«make calcul» (рис. 2.9)

```
lab_prog: bash — Konsole
Файл Правка Вид Закладки Настройка Справка
| | | char (*)[4]
| | | char *
yvkolcheva@dk8n56 ~/work/os/lab_prog $ gcc -c main.c
main.c: В функции «main»:
main.c:16:11: предупреждение: формат «%s» ожидает аргумент типа «char *», но аргумент 2 имеет тип «char (*)[4]» [-Wformat=]
16 |     scanf("%s", &operation);
    |     ~~~~~^
    |     |
    |     | | char (*)[4]
    |     | | char *
yvkolcheva@dk8n56 ~/work/os/lab_prog $ gcc calculate.o main.o -o calcul -lm
yvkolcheva@dk8n56 ~/work/os/lab_prog $ touch Makefile
yvkolcheva@dk8n56 ~/work/os/lab_prog $ make clean
Makefile:8: *** пропущен разделитель. Останов.
yvkolcheva@dk8n56 ~/work/os/lab_prog $ make clean
rm calcul *.o *~
yvkolcheva@dk8n56 ~/work/os/lab_prog $ make calculate.o
gcc -c calculate.c -g
yvkolcheva@dk8n56 ~/work/os/lab_prog $ make main.o
gcc -c main.c -g
main.c: В функции «main»:
main.c:16:11: предупреждение: формат «%s» ожидает аргумент типа «char *», но аргумент 2 имеет тип «char (*)[4]» [-Wformat=]
16 |     scanf("%s", &operation);
    |     ~~~~~^
    |     |
    |     | | char (*)[4]
    |     | | char *
yvkolcheva@dk8n56 ~/work/os/lab_prog $ make calcul
gcc calculate.o main.o -o calcul -lm
yvkolcheva@dk8n56 ~/work/os/lab_prog $ gdb ./calcul
```

Figure 2.9: Работа в консоли

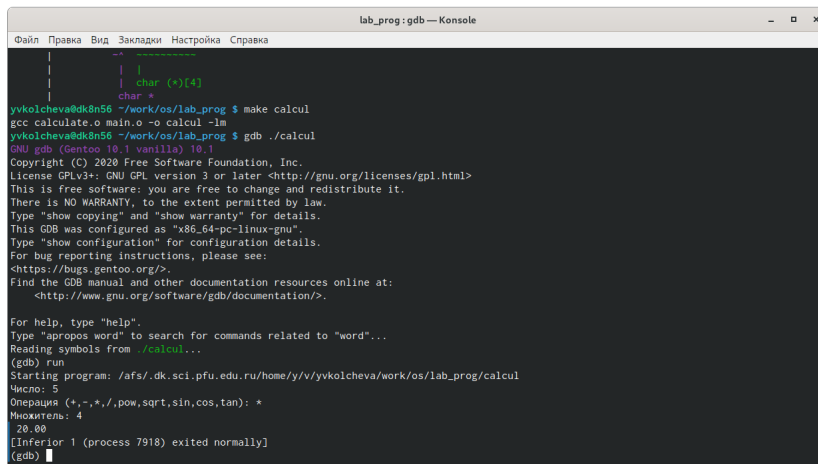
Далее с помощью gdb выполнила отладку программы calcul. Запустила отладчик GDB, загрузив в него программу для отладки, используя команду: «gdb./calcul».(рис. 2.10)

```
lab_prog: gdb — Konsole
Файл Правка Вид Закладки Настройка Справка
yvkolcheva@dk8n56 ~/work/os/lab_prog $ make calculate.o
gcc -c calculate.c -g
yvkolcheva@dk8n56 ~/work/os/lab_prog $ make main.o
gcc -c main.c -g
main.c: В функции «main»:
main.c:16:11: предупреждение: формат «%s» ожидает аргумент типа «char *», но аргумент 2 имеет тип «char (*)[4]» [-Wformat=]
16 |     scanf("%s", &operation);
    |     ~~~~~^
    |     |
    |     | | char (*)[4]
    |     | | char *
yvkolcheva@dk8n56 ~/work/os/lab_prog $ make calcul
gcc calculate.o main.o -o calcul -lm
yvkolcheva@dk8n56 ~/work/os/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-pe-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...
(gdb)
```

Figure 2.10: отладчик

Для запуска программы внутри отладчика ввела команду «run».(рис. 2.11)

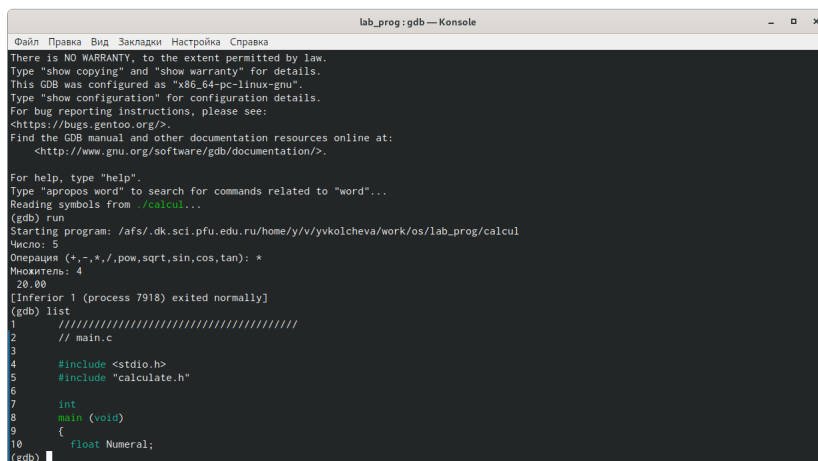


```
lab_prog: gdb -- Konsole
Файл Правка Вид Закладки Настройка Справка
|
| char (*)[4]
| char *
yvkolcheva@dk8n56 ~/work/os/lab_prog $ make calcul
gcc calculate.o main.o -o calcul -lm
yvkolcheva@dk8n56 ~/work/os/lab_prog $ gdb ./calcul
GNU gdb (Gentoo 10.1 vanilla) 10.1
Copyright (C) 2020 Free Software 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...
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/y/v/yvkolcheva/work/os/lab_prog/calcul
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): *
Множитель: 4
20.00
[Inferior 1 (process 7918) exited normally]
(gdb)
```

Figure 2.11: Работа программы

Для постраничного (по 10 строк) просмотра исходного кода использовала команду «list» (рис. 2.12)



```
lab_prog: gdb -- Konsole
Файл Правка Вид Закладки Настройка Справка
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...
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/y/v/yvkolcheva/work/os/lab_prog/calcul
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): *
Множитель: 4
20.00
[Inferior 1 (process 7918) exited normally]
(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)
```

Figure 2.12: Команда list

Для просмотра строк с 13 по 15 основного файла использовала команду «list 13,15» (рис. 2.13)

```
lab_prog:gdb -- Konsole
Файл Правка Вид Закладки Настройка Справка
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...
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/y/v/yvkoicheva/work/os/lab_prog/calcul
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): *
Выходные: 4
20.00
[Inferior 1 (process 7918) exited normally]
(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 13,15
13     printf("Число: ");
14     scanf("%f",&Numeral);
15     printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): ");
(gdb)
```

Figure 2.13: Команда list2

Для просмотра определённых строк неосновного файла использовала команду «listcalculate.c:2,6» (рис. 2.14)

```
lab_prog:gdb -- Konsole
Файл Правка Вид Закладки Настройка Справка
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./calcul...
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/y/v/yvkoicheva/work/os/lab_prog/calcul
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): *
Выходные: 4
20.00
[Inferior 1 (process 7918) exited normally]
(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 13,15
13     printf("Число: ");
14     scanf("%f",&Numeral);
15     printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): ");
(gdb) list calculate.c:2,6
2  #include <stdio.h>
3  #include <math.h>
4  #include <string.h>
5  #include "calculate.h"
6  float calculate (float Numeral, char Operation[4])
(gdb)
```

Figure 2.14: Команда list3

Установила точку останова в файле calculate.c на строке номер 8, (рис. 2.15)

```

lab_prog: gdb -- Konsole
Файл  Правка  Вид  Закладки  Настройка  Справка
9      {
10     float Numeral;
(gdb) list 13,15
13     printf("%d\n", Numeral);
14     scanf("%f", &Numeral);
15     printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): ");
(gdb) list calculate.c:2,6
2      #include <stdio.h>
3      #include <math.h>
4      #include <string.h>
5      #include "calculate.h"
6      float Calculate (float Numeral, char Operation[4])
(gdb) list calculate.c:2,9
2      #include <stdio.h>
3      #include <math.h>
4      #include <string.h>
5      #include "calculate.h"
6      float Calculate (float Numeral, char Operation[4])
7      {
8          float SecondNumeral;
9          if (strcmp(Operation, "+") == 0)
(gdb) break 8
Breakpoint 1 at 0x5555554008e5: file calculate.c, line 9.
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/y/v/yvkolcheva/work/os/lab_prog/calcul
Число: 4
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): +
Breakpoint 1, Calculate (Numeral=4, Operation=0x7fffffffc14 "+") at calculate.c:9
9      if (strcmp(Operation, "+") == 0)
(gdb)

```

Figure 2.15: Точка останова

Запустила программу внутри отладчика и убедилась, что программа остановилась в момент прохождения точки останова. 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. Убрала точки останова с помощью команд «info breakpoints» и «delete 1» (рис. 2.16)

```

lab_prog: gdb -- Konsole
Файл  Правка  Вид  Закладки  Настройка  Справка
6      float Calculate (float Numeral, char Operation[4])
(gdb) list calculate.c:2,9
2      #include <stdio.h>
3      #include <math.h>
4      #include <string.h>
5      #include "calculate.h"
6      float Calculate (float Numeral, char Operation[4])
7      {
8          float SecondNumeral;
9          if (strcmp(Operation, "+") == 0)
(gdb) break 8
Breakpoint 1 at 0x5555554008e5: file calculate.c, line 9.
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/y/v/yvkolcheva/work/os/lab_prog/calcul
Число: 4
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): +
Breakpoint 1, Calculate (Numeral=4, Operation=0x7fffffffc14 "+") at calculate.c:9
9      if (strcmp(Operation, "+") == 0)
(gdb) print Numeral
$1 = 4
(gdb) display Numeral
1: Numeral = 4
(gdb) info breakpoints
Num  Type             Disp  Enb  Address             What
1     breakpoint       keep  y   0x5555554008e5      in calculate at calculate.c:9
      breakpoint already hit 1 time
(gdb) delete 1
Undefined command: "delite". Try "help".
(gdb) delete 1
(gdb)

```

Figure 2.16: Запуск

Далее воспользовалась командами «splint calculate.c» и «splint main.c» (рис. 2.17) (рис. 2.18)


```

lab_prog: bash — Konsole
Файл Правка Вид Закладки Настройка Справка
yukoicheva@dk8n56 ~/work $ cd os
yukoicheva@dk8n56 ~/work/os $ cd lab_prog
yukoicheva@dk8n56 ~/work/os/lab_prog $ splint calculate.c
Splint 3.1.2 --- 13 Jan 2021

calculate.h:6:36: 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:6:37: Function parameter Operation declared as manifest array (size
constant is meaningless)
calculate.c: (in function Calculate)
calculate.c:12:7: 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:18:5: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:24:7: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:30:7: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:31:10: 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:34:10: Return value type double does not match declared type float:
(HUGE_VAL)
To allow all numeric types to match, use +relaxtypes.
calculate.c:40:7: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:41:13: Return value type double does not match declared type float:
(pow(Numeral, SecondNumeral))

```

Figure 2.17: splint calculate.c

```

lab_prog: bash — Konsole
Файл Правка Вид Закладки Настройка Справка
calculate.c:47:13: Return value type double does not match declared type float:
(sin(Numeral))
calculate.c:49:13: Return value type double does not match declared type float:
(cos(Numeral))
calculate.c:51:13: Return value type double does not match declared type float:
(tan(Numeral))
calculate.c:53:79: Return value type double does not match declared type float:
(HUGE_VAL)

Finished checking --- 15 code warnings
yukoicheva@dk8n56 ~/work/os/lab_prog $ splint main.c
Splint 3.1.2 --- 13 Jan 2021

calculate.h:6:36: 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:16:14: 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:16:11: Corresponding format code
main.c:16:13: Return value (type int) ignored: scanf("%s", &ope...

Finished checking --- 4 code warnings
yukoicheva@dk8n56 ~/work/os/lab_prog $

```

Figure 2.18: splint main.c

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.

4 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: кодирование – по сути создание исходного текста программы (возможно в нескольких вариантах); – анализ разработанного кода; сборка, компиляция и разработка исполняемого модуля; 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 `.cpp` или `.cc` – как файлы на языке C++, а файлы с расширением `.o` считаются объектными. 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

-ои в качестве параметра задать имя создаваемого файла: «gcc-ohellomain.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 normally abcd: abcd.c \$(CC) -o abcd \$(CFLAGS) abcd.c clean: -rm abcd.o ~ # End Makefile 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 со-

кращённым вариантом q) или комбинацией клавиш Ctrl-d. Более подробную информацию по работе с gdb можно получить с помощью команд gdb-hi 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: `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.

5 Библиография

Лекция Кудрявцева <https://esystem.rudn.ru/mod/url/view.php?id=718563>

Лабораторная работа №14 <https://esystem.rudn.ru/mod/resource/view.php?id=718613>