

Отчет по лабораторной работе №14

Средства, применяемые при разработке программного обеспечения в ОС типа UNIX/Linux

Российский Университет Дружбы Народов

Факультет Физико-Математических и Естественных Наук

Дисциплина: Операционные системы

Студент: Алхатиб Осама

Группа: НПИбд-02-20

2021г.

1. Цель работы

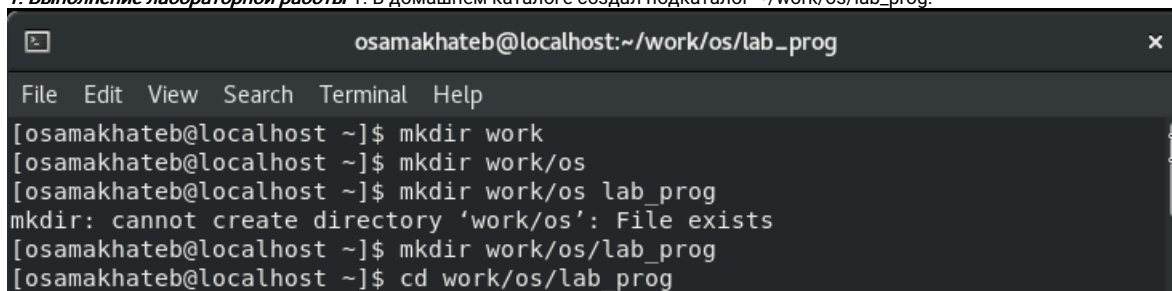
Приобрести простейшие навыки разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linux на примере создания на языке программирования C калькулятора с простейшими функциями

2 Задание 1. В домашнем каталоге создайте подкаталог ~/work/os/lab_prog. 2. Создайте в нём файлы: calculate.h, calculate.c, main.c. Это будет примитивнейший калькулятор, способный складывать, вычитать, умножать и делить, возводить число в степень, брать квадратный корень, вычислять sin, cos, tan. При запуске он будет запрашивать первое число, операцию, второе число. После этого программа выведет результат и остановится. Реализация функций калькулятора в файле calculate.h: // 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)); } 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); } } Интерфейсный файл calculate.h, описывающий формат вызова функции калькулятора: // calculate.h #ifndef CALCULATE_H_ #define CALCULATE_H_ float Calculate(float Numeral, char Operation[4]); #endif /CALCULATE_H_/ Основной файл main.c, реализующий интерфейс пользователя к калькулятору: // main.c 3. Выполните компиляцию программы посредством gcc: gcc -c calculate.c gcc -c main.c gcc calculate.o main.o -o calcul -lm 4. При необходимости исправьте синтаксические ошибки. 5. Создайте Makefile со следующим содержанием:

```
# 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 Поясните в отчёте его содержание. 6. С помощью gdb выполните отладку программы calcul (перед использованием gdb исправьте Makefile): – Запустите отладчик GDB, загрузив в него программу для отладки: gdb ./calcul – Для запуска программы внутри отладчика введите команду run: run – Для постраничного (по 9 строк) просмотра исходного кода используйте команду list: list – Для просмотра строк с 12 по 15 основного файла используйте list с параметрами: list 12,15 – Для просмотра определённых строк не основного файла используйте list с параметрами: list calculate.c:20,29 – Установите точку останова в файле calculate.c на строке номер 21: list calculate.c:20,27 break 21 – Выведите информацию об имеющихся в проекте точка останова: info breakpoints – Запустите программу внутри отладчика и убедитесь, что программа остановится в момент прохождения точки останова: run 5 • backtrace – Отладчик выдаст следующую информацию: #0 Calculate (Numeral=5, Operation=0x7fffffd280 "-") at calculate.c:21 #1 0x0000000000400b2b in main () at main.c:17 а команда backtrace покажет весь стек вызываемых функций от начала программы до текущего места. – Посмотрите, чему равно на этом этапе значение переменной Numeral, введя: print Numeral На экран должно быть выведено число 5. – Сравните с результатом вывода на экран после использования команды: display Numeral – Уберите точки останова: info breakpoints delete 1 7. С помощью утилиты splint попробуйте проанализировать коды файлов calculate.c и main.c

1. Выполнение лабораторной работы 1. В домашнем каталоге создал подкаталог ~/work/os/lab_prog.



```
osamakhteb@localhost:~/work/os/lab_prog
File Edit View Search Terminal Help
[osamakhteb@localhost ~]$ mkdir work
[osamakhteb@localhost ~]$ mkdir work/os
[osamakhteb@localhost ~]$ mkdir work/os lab_prog
mkdir: cannot create directory 'work/os': File exists
[osamakhteb@localhost ~]$ mkdir work/os/lab_prog
[osamakhteb@localhost ~]$ cd work/os/lab_prog
```

Figure 3.1: каталог 2. Создал в нём файлы: calculate.h, calculate.c, main.c.

```
[osamakhatib@localhost lab_prog]$ touch calculate.h, calculate.c main.c
```

Figure 3.2: файлы:

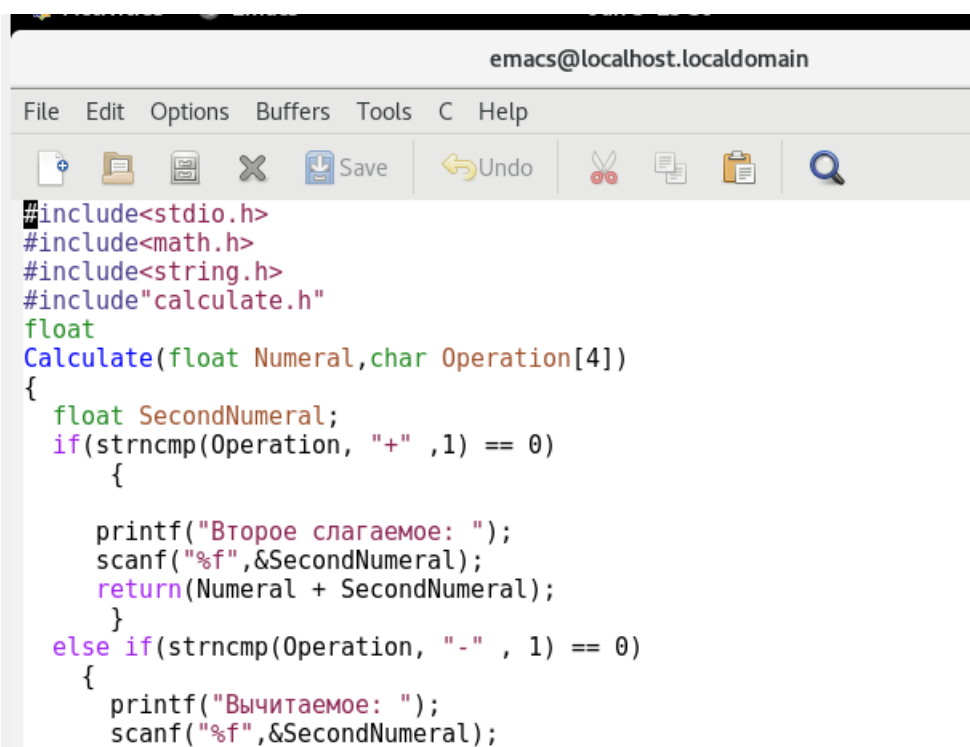
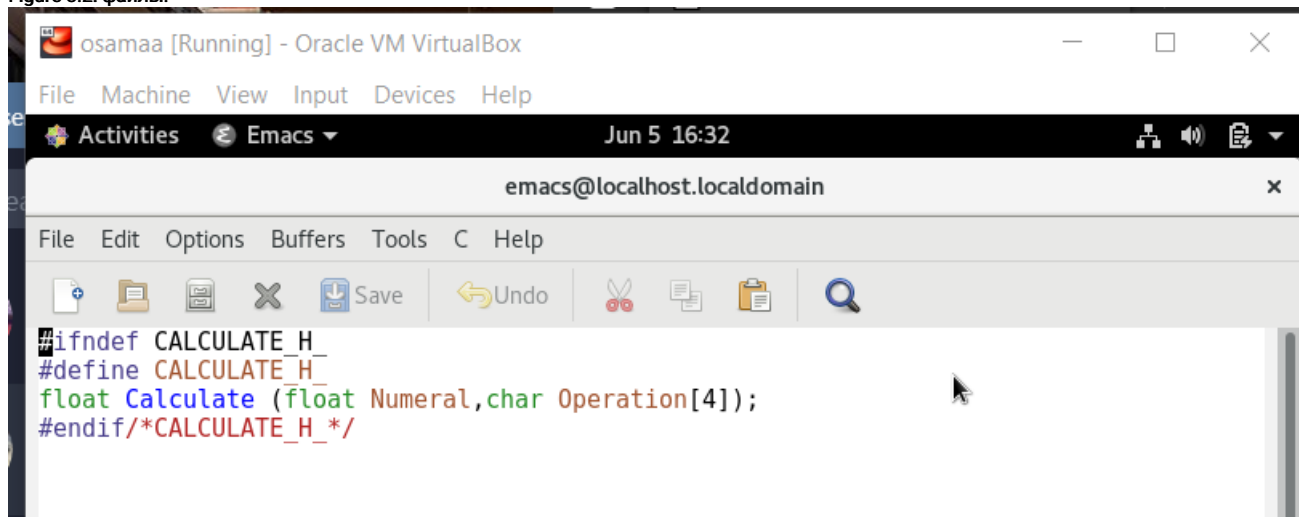
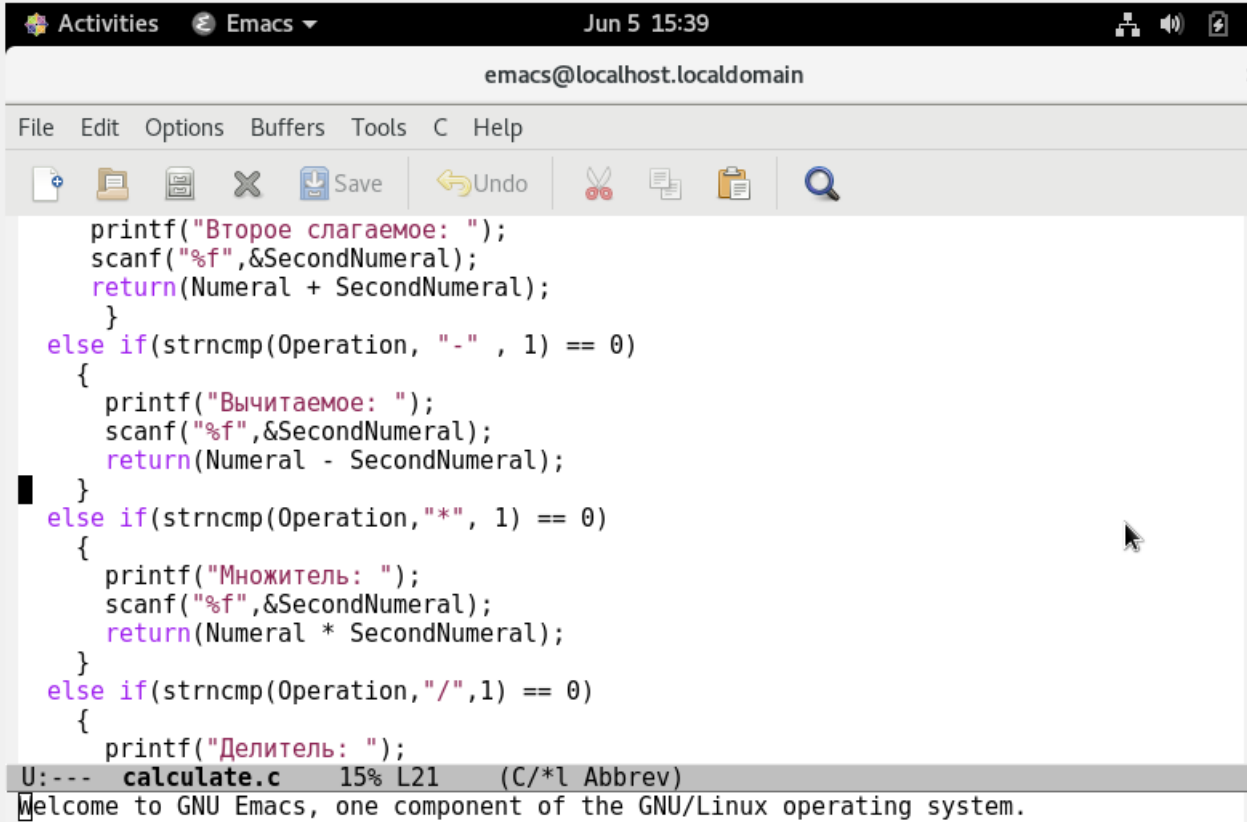
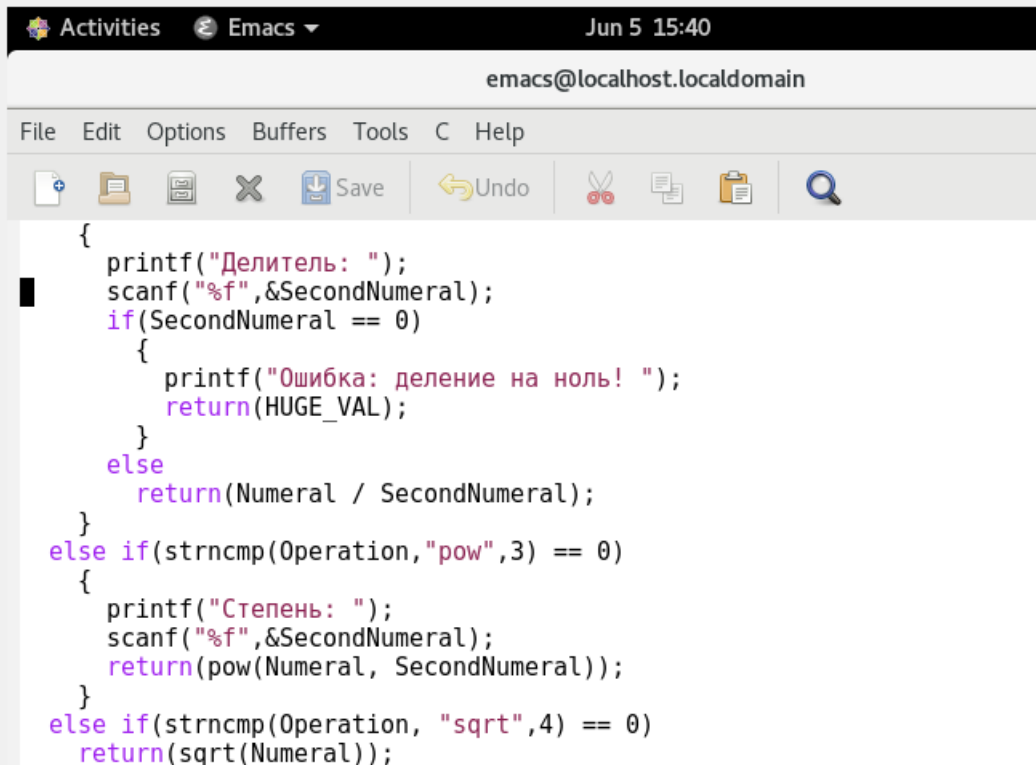


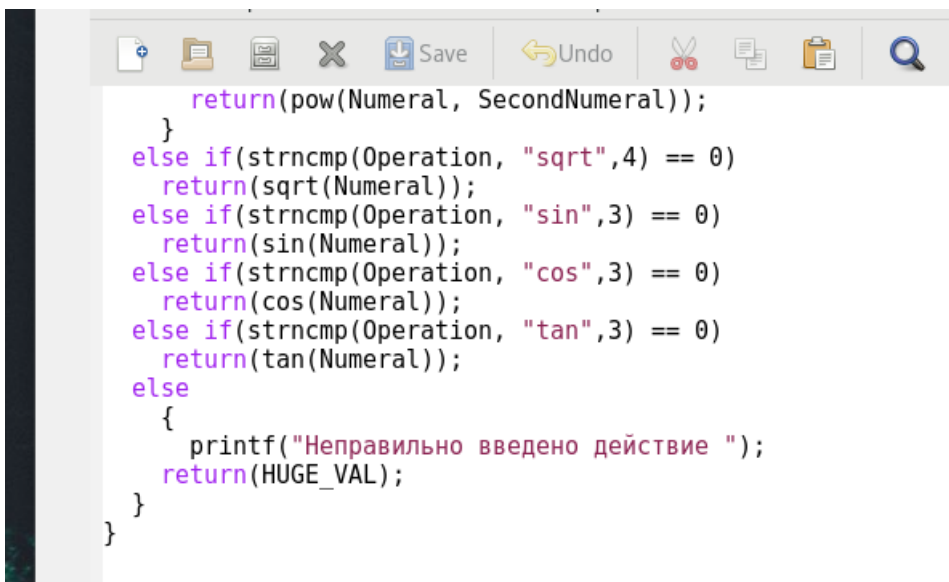
Figure 3.3: calculate.h файл



```
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: ");
U:--- calculate.c 15% L21 (C/*1 Abbrev)
Welcome to GNU Emacs, one component of the GNU/Linux operating system.
```

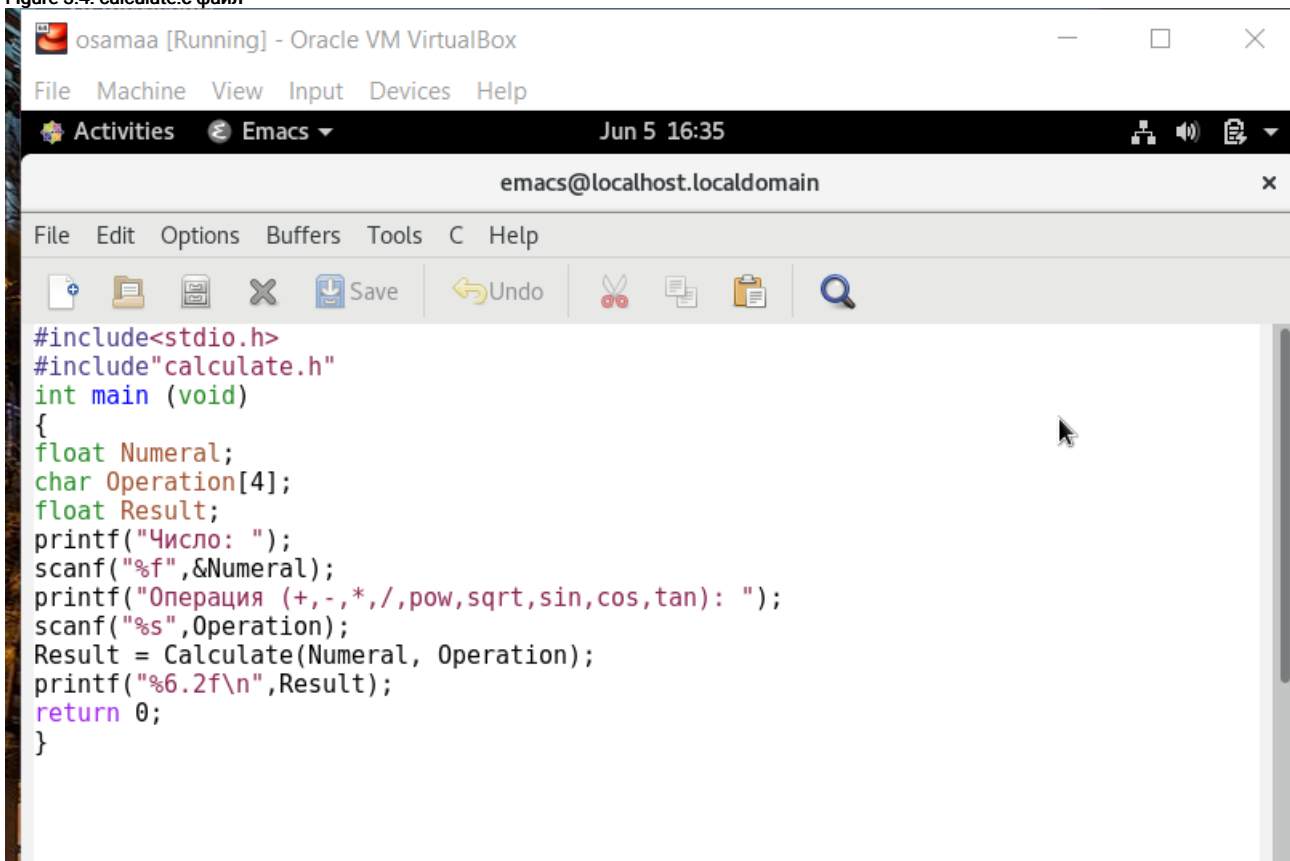


```
{
printf("Делитель: ");
scanf("%f",&SecondNumeral);
if(SecondNumeral == 0)
{
printf("Ошибка: деление на ноль! ");
return(HUGE_VAL);
}
else
return(Numeral / SecondNumeral);
}
else if(strncmp(Operation, "pow", 3) == 0)
{
printf("Степень: ");
scanf("%f",&SecondNumeral);
return(pow(Numeral, SecondNumeral));
}
else if(strncmp(Operation, "sqrt", 4) == 0)
return(sqrt(Numeral));
}
```



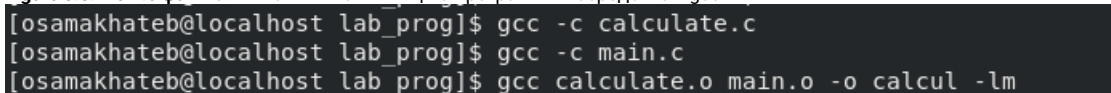
```
return(pow(Numeral, SecondNumer));
}
else if(strncmp(Operation, "sqrt",4) == 0)
return(sqrt(Numeral));
else if(strncmp(Operation, "sin",3) == 0)
return(sin(Numeral));
else if(strncmp(Operation, "cos",3) == 0)
return(cos(Numeral));
else if(strncmp(Operation, "tan",3) == 0)
return(tan(Numeral));
else
{
printf("Неправильно введено действие ");
return(HUGE_VAL);
}
}
```

Figure 3.4: calculate.c файл



```
#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: main.c файл 3. Выполнил компиляцию программы посредством gcc



```
[osamakhateb@localhost lab_prog]$ gcc -c calculate.c
[osamakhateb@localhost lab_prog]$ gcc -c main.c
[osamakhateb@localhost lab_prog]$ gcc calculate.o main.o -o calcul -lm
```

Figure 3.6: компиляция gcc 4.

Исправил синтаксические ошибки в файле main.c (удалил & перед operator в линии scanf("%s",&Operation);) 5. Создал Makefile со следующим содержанием

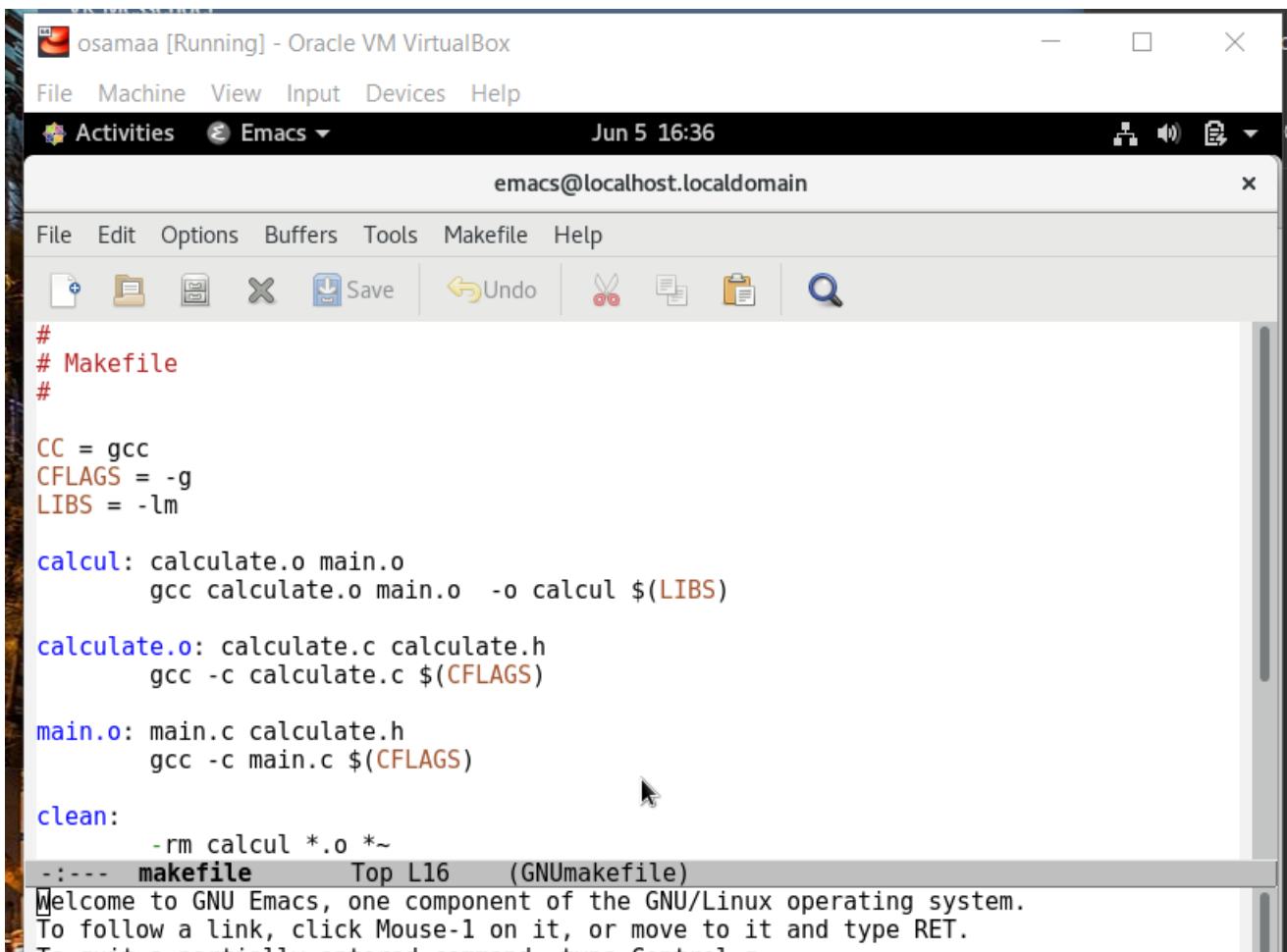


Figure 3.7: Makefile 1. С помощью gdb выполнил отладку программы calcul (перед использованием gdb исправьте Makefile) • Запустил отладчик GDB, загрузил в него программу для отладки: `gdb ./calcul`

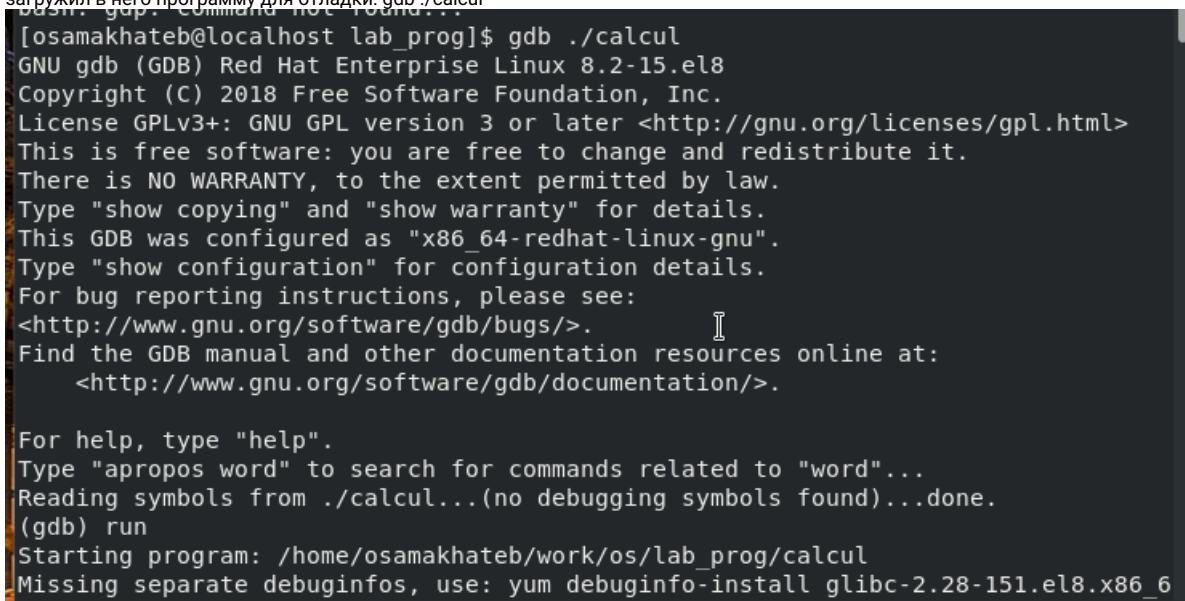


Figure 3.8: gdb • Для запуска программы внутри отладчика ввел команду `run: run`

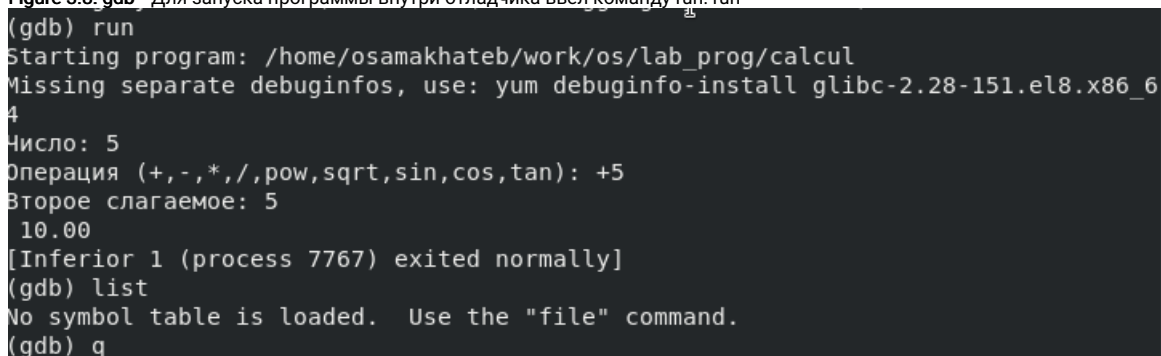


Figure 3.9: run • Для постраничного (по 9 строк) просмотра исходного код использовал команду `list: list`

```

(gdb) list
1      #include<stdio.h>
2      #include"calculate.h"
3      int main (void)
4      {
5          float Numeral;
6          char Operation[4];
7          float Result;
8          printf("Число: ");
9          scanf("%f",&Numeral);
10         printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): ");
(gdb) list 12,15
12         Result = Calculate(Numeral, Operation);
13         printf("%6.2f\n",Result);
14         return 0;
15     }
(gdb) list calculate.c:20,29
20         return(Numeral - SecondNumeral);

```

Figure 3.10: list • Для просмотра строк с 12 по 15 основного файла использовал list с параметрами: list 12,15

```

(gdb) list 12,15
12         Result = Calculate(Numeral, Operation);
13         printf("%6.2f\n",Result);
14         return 0;
15     }

```

Figure 3.11: list • Для просмотра определённых строк не основного файла использовал list с параметрами: list calculate.c:20,29

```

15     }
(gdb) list calculate.c:20,29
20         return(Numeral - SecondNumeral);
21     }
22     else if(strncmp(Operation,"*", 1) == 0)
23     {
24         printf("Множитель: ");
25         scanf("%f",&SecondNumeral);
26         return(Numeral * SecondNumeral);
27     }
28     else if(strncmp(Operation,"/",1) == 0)
29     {

```

Figure 3.12: list • Установил точку останова в файле calculate.c на строке номер 21: list calculate.c:20,27 break 21

```

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

```

Figure 3.13: list • Вывел информацию об имеющихся в проекте точка останова: info breakpoints

```

(gdb) break 21
Breakpoint 1 at 0x40087b: file calculate.c, line 22.
(gdb) info breakpoints
Num   Type             Disp Enb Address            What
1     breakpoint       keep y   0x000000000040087b  in Calculate
                                           at calculate.c:22
(gdb) run

```

Figure 3.14: breakpoints • Запустил программу внутри отладчика и убедитесь, что программа остановится в момент прохождения точки останова: run 5 •


```

Undefined command: 5. Try help.
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/osamakhatib/work/os/lab_prog/calcul
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): *

Breakpoint 1, Calculate (Numeral=5, Operation=0x7fffffffdf54 "**")
    at calculate.c:22
22     else if(strncmp(Operation,"*", 1) == 0)
(gdb) backtrace
#0  Calculate (Numeral=5, Operation=0x7fffffffdf54 "**") at calculate.c:22
#1  0x0000000000400ad4 in main () at main.c:12
(gdb) print Numeral
$1 = 5

```

Figure 3.15: run • Посмотрел, чему равно на этом этапе значение переменной Numeral, введя: print Numeral

```

(gdb) print Numeral
$1 = 5

```

Figure 3.16: print • Сравнил с результатом вывода на экран после использования команды: display Numeral

```

(gdb) display Numeral
1: Numeral = 5
(gdb) info breakpoints

```

Figure 3.17: display • Убрал точки останова: info breakpoints delete 1

```

(gdb) info breakpoints
Num      Type           Disp Enb Address              What
1        breakpoint     keep y   0x000000000040087b in Calculate
                                at calculate.c:22

breakpoint already hit 1 time
(gdb) delete 1

```

Figure 3.18: delete 7. С помощью утилиты splint попробовал проанализировать коды файлов calculate.c и main.c.

```

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:6:31: Function parameter Operation declared as manifest array (size
        constant is meaningless)
calculate.c: (in function Calculate)
calculate.c:12:3: 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:18:3: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:24:3: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:30:3: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:31:6: 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:

```

Figure 3.19: 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)
main.c: (in function main)
main.c:10:2: 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:12:2: Return value (type int) ignored: scanf("%s", Oper...
Finished checking --- 3 code warnings
```

```
main.c
```

Figure 3.20: splint

4. Выводы В результате работы , я приобрёл простейшие навыки разработки, анализа, тестирования и отладки приложений в Линук

5 Контрольные вопросы 1. Дополнительную информацию о этих программах можно получить с помощью функций info и man. 2. Unix поддерживает следующие основные этапы разработки приложений: -создание исходного кода программы; • представляется в виде файла; -сохранение различных вариантов исходного текста; -анализ исходного текста; Необходимо отслеживать изменения исходного кода, а также при работе более двух программистов над проектом программы нужно, чтобы они не делали изменений кода в одно время. -компиляция исходного текста и построение исполняемого модуля; -тестирование и отладка; -проверка кода на наличие ошибок -сохранение всех изменений, выполняемых при тестировании и отладке. 3. Использование суффикса ".c" для имени файла с программой на языке Си отражает удобное и полезное соглашение, принятое в ОС UNIX. Для любого имени входного файла суффикс определяет какая компиляция требуется. Суффиксы и префиксы указывают тип объекта. Одно из полезных свойств компилятора Си — его способность по суффиксам определять типы файлов. По суффиксу .c компилятор распознает, что файл abcd.c должен компилироваться, а по суффиксу .o, что файл abcd.o является объектным модулем и для получения исполняемой программы необходимо выполнить редактирование связей. Простейший пример командной строки для компиляции программы abcd.c и построения исполняемого модуля abcd имеет вид: gcc -o abcd abcd.c. Некоторые проекты предпочитают показывать префиксы в начале текста изменений для старых (old) и новых (new) файлов. Опция -prefix может быть использована для установки такого префикса. Плюс к этому команда bzr diff -p1 выводит префиксы в форме которая подходит для команды patch -p1. 4. Основное назначение компилятора с языка Си заключается в компиляции всей программы в целом и получении исполняемого модуля. 5. При разработке большой программы, состоящей из нескольких исходных файлов заголовков, приходится постоянно следить за файлами, которые требуют перекомпиляции после внесения изменений. Программа make освобождает пользователя от такой рутинной работы и служит для документирования взаимосвязей между файлами. Описание взаимосвязей и соответствующих действий хранится в так называемом make-файле, который по умолчанию имеет имя makefile или Makefile. 6. makefile для программы abcd.c мог бы иметь вид: 9. 1) Выполнили компиляцию программы 2)Увидели ошибки в программе 7) Открыли редактор и исправили программу 4) Загрузили программу в отладчик gdb 5) run — отладчик выполнил программу, мы ввели требуемые значения. 6) программа завершена, gdb не видит ошибок. 10. 1 и 2.) Мы действительно забыли закрыть комментарии; 3.) отладчику не понравился формат %s для &Operation, т.к %s — символьный формат, а значит необходим только Operation. 11. Если вы работаете с исходным кодом, который не вами разрабатывался, то назначение различных конструкций может быть не совсем понятным. Система разработки приложений UNIX предоставляет различные средства, повышающие понимание исходного кода. К ним относятся: — cscope - исследование функций, содержащихся в программе; — splint — критическая проверка программ, написанных на языке Си. 12. 1. Проверка корректности задания аргументов всех исп функций, а также типов возвращаемых ими значений; 2. Поиск фрагментов исходного текста, корректных с точки зрения синтаксиса языка Си, но малоэффективных с точки зрения их реализации или содержащих в себе семантические ошибки; 3. Общая оценка мобильности пользовательской программы.