

Отчёт по лабораторной работе №14

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

Дисциплина: Операционные системы

Студент: Оразгелдиева Огулнур

Группа: НПИбд-02-20

Студ. номер: 1032205431

2021, Москва

Лабораторная работа №14

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

Цель:

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

Задачи:

1. Ознакомиться с теоретическим материалом
2. Создать в каталоге `~/work/os/lab_prog` файлы `calculate.h`, `calculate.c`, `main.c`
3. Написать программу калькулятора, способный складывать, вычитать, умножать и делить, возводить число в степень, брать квадратный корень, вычислять `sin`, `cos`, `tan`.
4. Выполнить компиляцию программы посредством `gcc`
5. Создать `Makefile`
6. Выполнить отладку программы `calcul` с помощью `gdb`
7. Проанализировать коды файлов `calculate.c` и `main.c`

Теоретические сведения

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.

Стандартным средством для компиляции программ в ОС типа UNIX является GCC (GNU Compiler Collection). Это набор компиляторов для разного рода языков программирования (C, C++, Java, Фортран и др.). Работа с GCC производится при помощи одноимённой управляющей программы gcc, которая интерпретирует аргументы командной строки, определяет и осуществляет запуск нужного компилятора для входного файла. 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. [1]

Отладчиком называется программа, которая выполняет внутри себя другую программу. Основное назначение отладчика - дать возможность пользователю в определенной степени осуществлять контроль за выполняемой программой, то есть определять, что происходит в процессе ее выполнения. Наиболее известным отладчиком для Linux является программа GNU GDB. GDB содержит множество полезных возможностей, но для простой отладки достаточно использовать лишь некоторые из них. [2]

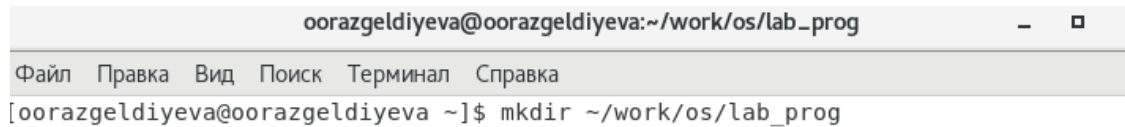
GDB — переносимый отладчик проекта GNU, который работает на многих UNIX-подобных системах и умеет производить отладку многих языков программирования, включая Си, C++, Free Pascal, FreeBASIC, Ada, Фортран, Python3, Swift, NASM и Rust.

Для того чтобы нам пройти по такому файлу нам нужно скомпилировать его с помощью G++ с использованием флага -g (это действительно важно, без этого флага, программа не будет корректно работать в GDB). [3]

Ход работы:

1. В домашнем каталоге создадим подкаталог `~/work/os/lab_prog`. (см. рис. 1)

Для этого используем команду создания каталогов `mkdir`

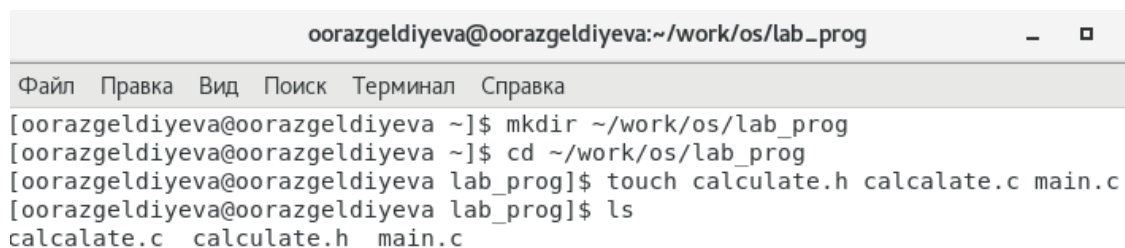


```
oorazgeldiyeva@oorazgeldiyeva:~/work/os/lab_prog
Файл  Правка  Вид  Поиск  Терминал  Справка
[oorazgeldiyeva@oorazgeldiyeva ~]$ mkdir ~/work/os/lab_prog
```

Рисунок 1. Создание каталога `~/work/os/lab_prog`

2. Перейдём в созданный подкаталог и создадим там файлы `calculate.h`, `calculate.c`, `main.c`. (см. рис. 2)

Чтобы перейти в каталог используем команду `cd`, и при помощи команды создания файлов `touch` создадим указанные файлы.



```
oorazgeldiyeva@oorazgeldiyeva:~/work/os/lab_prog
Файл  Правка  Вид  Поиск  Терминал  Справка
[oorazgeldiyeva@oorazgeldiyeva ~]$ mkdir ~/work/os/lab_prog
[oorazgeldiyeva@oorazgeldiyeva ~]$ cd ~/work/os/lab_prog
[oorazgeldiyeva@oorazgeldiyeva lab_prog]$ touch calculate.h calcalate.c main.c
[oorazgeldiyeva@oorazgeldiyeva lab_prog]$ ls
calcalate.c  calculate.h  main.c
```

Рисунок 2. Создание файлов

3. В файле `calculate.c` пишем следующую программу: (см. рис. 3-5)

```
oorazgeldiyeva@oorazgeldiyeva:~/work/os/lab_prog
Файл Правка Вид Поиск Терминал Справка
#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);
    }
}
-- RCTARKA --
```

1 1 HARENXV

Рисунок 3. *calculate.c*

```
oorazgeldiyeva@oorazgeldiyeva:~/work/os/lab_prog
Файл  Правка  Вид  Поиск  Терминал  Справка

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));
}
-- ВСТАВКА --
```

49,1

69%

Рисунок 4. *calculate.c*

```
oorazgeldiyeva@oorazgeldiyeva:~/work/os/lab_prog
Файл  Правка  Вид  Поиск  Терминал  Справка
        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);
}
}

ВСТАВКА
```

Рисунок 5. *calculate.c*

Пояснения: задается функция Calculate типа вещественный с вещественным аргументом Numeral и символьный аргумент размером 4 Operation. Эта функция работает следующим образом: при вводе операции + выводится запрос на ввод второго слагаемого, и функция выдаёт сумму введенных чисел. Аналогично с операциями -, *, /, и тд.

Внутри файла calculate.h пишем следующий текст: (см. рис. 6)

```
oorazgeldiyeva@oorazgeldiyeva:~/work/os/lab_prog
Файл  Правка  Вид  Поиск  Терминал  Справка
#ifndef CALCULATE_H_
#define CALCULATE_H_
float Calculate(float Numeral, char Operation[4]);
#endif /*CALCULATE_H_*/
~
~
~
```

Рисунок 6. *calculate.h*

В main.c запишем основную функцию программы, (см. рис. 7) которая выводит сообщение о вводе числа и операции, читает введенные переменные и выполняет функцию, находящуюся в файле calculate.c.

```
oorazgeldiyeva@oorazgeldiyeva:~/work/os/lab_prog
Файл  Правка  Вид  Поиск  Терминал  Справка
#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;
}
```

Рисунок 7. main.c

4. Выполнила компиляцию программы посредством *gcc*: (см. рис. 8)

```
[oorazgeldiyeva@oorazgeldiyeva lab_prog]$ ls
calcalate.c  calculate.h  main.c
[oorazgeldiyeva@oorazgeldiyeva lab_prog]$ vi calculate.c
[oorazgeldiyeva@oorazgeldiyeva lab_prog]$ vi calculate.h
[oorazgeldiyeva@oorazgeldiyeva lab_prog]$ vi main.c
[oorazgeldiyeva@oorazgeldiyeva lab_prog]$ gcc -c calculate.c
calculate.c:3:20: фатальная ошибка: strinh.h: Нет такого файла или каталога
#include <strinh.h>
      ^
компиляция прервана.
[oorazgeldiyeva@oorazgeldiyeva lab_prog]$ vi calculate.c
[oorazgeldiyeva@oorazgeldiyeva lab_prog]$ gcc -c calculate.c
[oorazgeldiyeva@oorazgeldiyeva lab_prog]$ gcc -c main.c
[oorazgeldiyeva@oorazgeldiyeva lab_prog]$ gcc calculate.o main.o -o calcul -lm
```

Рисунок 8. Компиляция программы

5. При помощи редактора *vi* создала и открыла файл *Makefile*. Записала в него следующий текст по образцу. (см. рис. 9)

```
oorazgeldiyeva@oorazgeldiyeva:~/work/os/lab_prog
Файл  Правка  Вид  Поиск  Терминал  Справка
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 *~
~
--
```

Рисунок 9. Makefile

Makefile содержит переменные CC, CFLAGS и LIBS, а также цели calcul, calculate.o, main.o и clean, которые в свою очередь содержат команды компиляции и удаления файлов.

Исправим содержимое Makefile. (см. рис. 10)

```
oorazgeldiyeva@oorazgeldiyeva:~/work/os/lab_prog
Файл  Правка  Вид  Поиск  Терминал  Справка
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 *~
~
~
--
```

Рисунок 10. Makefile

Так как задана переменная CC, которая содержит gcc, можем поменять gcc на эту переменную. Чтобы отладчик работал исправно в переменную CFLAGS вносим -g.

С помощью команды make можем выполнить цели из Makefile. (см. рис. 11)


```
[oorazgeldiyeva@oorazgeldiyeva lab_prog]$ vi Makefile
[oorazgeldiyeva@oorazgeldiyeva lab_prog]$ make clean
rm calcul *.o *~
rm: невозможно удалить «*~»: Нет такого файла или каталога
make: [clean] Ошибка 1 (игнорирована)
[oorazgeldiyeva@oorazgeldiyeva lab_prog]$ make calculate.o
gcc -c calculate.c -g
[oorazgeldiyeva@oorazgeldiyeva lab_prog]$ make main.o
gcc -c main.c -g
[oorazgeldiyeva@oorazgeldiyeva lab_prog]$ make calcul
gcc calculate.o main.o -o calcul -lm
```

Рисунок 11. Выполнение Makefile

Таким образом, скомпилировали файлы.

6. С помощью gdb выполните отладку программы calcul.

- Для этого в командной строке вводим *gdb ./calcul*. (см. рис. 12)

```
[oorazgeldiyeva@oorazgeldiyeva lab_prog]$ gdb ./calcul
GNU gdb (GDB) Red Hat Enterprise Linux 7.6.1-120.el7
Copyright (C) 2013 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-redhat-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/oorazgeldiyeva/work/os/lab_prog/calcul...done.
(gdb)
```

Рисунок 12. Отладчик gdb

В результате этого вывелось сообщение с информацией об отладчике.

- Для запуска программы внутри отладчика вводим команду *run*. (см. рис. 13)

```
Reading symbols from /home/oorazgeldiyeva/work/os/lab_prog/calcul...done.
(gdb) run
Starting program: /home/oorazgeldiyeva/work/os/lab_prog/./calcul
Число: 8
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): *
Множитель: 2
16.00
[Inferior 1 (process 8273) exited normally]
Missing separate debuginfos, use: debuginfo-install glibc-2.17-324.el7_9.x86_64
(gdb) █
```

Рисунок 13. Запуск программы

Вводим для примера 8, *, 2. В результате выполнилось умножение 8 на 2, ответ: 16.

- Для постраничного (по 9 строк) просмотра исходного кода используем команду *list*: (см. рис. 14)

```

16.00
[Inferior 1 (process 7049) exited normally]
Missing separate debuginfos, use: debuginfo-install glibc-2.17-324.el7_9.x86_64
(gdb) list
warning: Source file is more recent than executable.
1      #include <stdio.h>
2      #include "calculate.h"
3      int
4      main (void)
5      {
6      float Numeral;
7      char Operation[4];
8      float Result;
9      printf("Число: ");
10     scanf("%f",&Numeral);
(gdb) █

```

Рисунок 14. Постраничный просмотр кода

- Для просмотра строк с 12 по 15 основного файла используем list с параметрами: (см. рис. 15)

```

-      .-----,
9      printf("Число: ");
10     scanf("%f",&Numeral);
(gdb) list 12,15
12     scanf("%s",&Operation);
13     Result = Calculate(Numeral, Operation);
14     printf("%6.2f\n",Result);
15     return 0;
(gdb) █

```

Рисунок 15. Просмотр строк с 12 по 15

- Для просмотра определённых строк не основного файла используем list с параметрами: (см. рис. 16)

```
oorazgeldiyeva@oorazgeldiyeva:~/work/os/lab_prog
Файл  Правка  Вид  Поиск  Терминал  Справка
2      #include "calculate.h"
3      int
4      main (void)
5      {
6      float Numeral;
7      char Operation[4];
8      float Result;
9      printf("Число: ");
10     scanf("%f",&Numeral);
(gdb) list 12,15
12     scanf("%s",&Operation);
13     Result = Calculate(Numeral, Operation);
14     printf("%6.2f\n",Result);
15     return 0;
(gdb) list calculate.c:20,29
warning: Source file is more recent than executable.
20     }
21     else if(strncmp(Operation, "*", 1) == 0)
22     {
23         printf("Множитель: ");
24         scanf("%f",&SecondNumeral);
25         return(Numeral * SecondNumeral);
26     }
27     else if(strncmp(Operation, "/", 1) == 0)
28     {
29         printf("Делитель: ");
(gdb)
```

Рисунок 16. Просмотр определённых строк не основного файла

Таким образом, просмотрели строки с 20 по 29 файла calculate.c.

- Установим точку останова в файле calculate.c на строке номер 21.

Для этого используем команды list и break. (см. рис. 17)

```
oorazgeldiyeva@oorazgeldiyeva:~/work/os/lab_prog
Файл  Правка  Вид  Поиск  Терминал  Справка
13      Result = Calculate(Numeral, Operation);
14      printf("%6.2f\n",Result);
15      return 0;
(gdb) list calculate.c:20,29
warning: Source file is more recent than executable.
20      }
21      else if(strncmp(Operation, "*", 1) == 0)
22      {
23          printf("Множитель: ");
24          scanf("%f",&SecondNumeral);
25          return(Numeral * SecondNumeral);
26      }
27      else if(strncmp(Operation, "/", 1) == 0)
28      {
29          printf("Делитель: ");
(gdb) list calculate.c:20,27
20      }
21      else if(strncmp(Operation, "*", 1) == 0)
22      {
23          printf("Множитель: ");
24          scanf("%f",&SecondNumeral);
25          return(Numeral * SecondNumeral);
26      }
27      else if(strncmp(Operation, "/", 1) == 0)
(gdb) break 21
Breakpoint 1 at 0x400810: file calculate.c, line 21.
(gdb) █
```

Рисунок 17. Установление точки останова

Как видим, 21 строка - это строка с операцией умножения.

- Выведем информацию об имеющихся в проекте точках останова, используя *info breakpoints*: (см. рис. 18)

```
27      else if(strncmp(Operation, "/", 1) == 0)
(gdb) break 21
Breakpoint 1 at 0x400810: file calculate.c, line 21.
(gdb) info breakpoints
Num      Type          Disp Enb Address              What
1        breakpoint    keep y   0x0000000000400810  in Calculate
                                                at calculate.c:21
(gdb)
```

Рисунок 18. Точки останова

Как видно из рис. 18, точка останова на 21 строке файла calculate.c функции Calculate.

- Запустим программу внутри отладчика. используя команду *run*, и убедимся, что программа остановится в момент прохождения точки останова.

Чтобы сделать это, вводим, например. число 5 и операцию умножения, так как точка останова именно на этой строке.

Вывелось сообщение о том, что установлена точка останова. (см. рис. 19)

```
(gdb) break 21
Breakpoint 1 at 0x400810: file calculate.c, line 21.
(gdb) info breakpoints
Num     Type             Disp Enb Address                  What
1       breakpoint      keep y   0x0000000000400810 in Calculate
                                                at calculate.c:21

(gdb) run
Starting program: /home/oorazgeldiyeva/work/os/lab_prog/./calcul
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): *

Breakpoint 1, Calculate (Numeral=5, Operation=0x7fffffffdd60 "**")
  at calculate.c:21
21          else if(strncmp(Operation, "**", 1) == 0)
(gdb)
```

Рисунок 19. Запуск программы

Вводим `backtrace`, отладчик выводит следующее сообщение: (см. рис. 20)

```
Breakpoint 1, Calculate (Numeral=5, Operation=0x7fffffffdd60 "**")
  at calculate.c:21
21          else if(strncmp(Operation, "**", 1) == 0)
(gdb) backtrace
#0 Calculate (Numeral=5, Operation=0x7fffffffdd60 "**") at calculate.c:21
#1 0x0000000000400a90 in main () at main.c:13
(gdb) █
```

Рисунок 20. `backtrace`

Команда `backtrace` показывает весь стек вызываемых функций от начала программы до текущего места.

Посмотрим, чему равно на этом этапе значение переменной `Numeral`, введя: `print Numeral`. (см. рис. 21)

```
(gdb) backtrace
#0 Calculate (Numeral=5, Operation=0x7fffffffdd60 "**") at calculate.c:21
#1 0x0000000000400a90 in main () at main.c:13
(gdb) print Numeral
$1 = 5
(gdb)
```

Рисунок 21. Значение переменной `Numeral`

Сравним с результатом вывода на экран после использования команды: `display Numeral`. (см. рис. 22)

```
#1 0x0000000000400a90 in main () at main.c:13
(gdb) print Numeral
$1 = 5
(gdb) display Numeral
1: Numeral = 5
(gdb)
```

Рисунок 22. Значение переменной `Numeral`

В первом случае вывелось только значение переменной, а во втором случае можно увидеть и название самой переменной.

- Уберём точки останова, используя команду `delete`. (см. рис. 23)

```
oorazgeldiyeva@oorazgeldiyeva:~/work/os/lab_prog
Файл  Правка  Вид  Поиск  Терминал  Справка
Num    Type      Disp Enb Address      What
1      breakpoint keep y  0x0000000000400810 in Calculate
                                           at calculate.c:21

(gdb) run
Starting program: /home/oorazgeldiyeva/work/os/lab_prog/./calcul
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): *

Breakpoint 1, Calculate (Numeral=5, Operation=0x7fffffffdd60 "")
  at calculate.c:21
21      else if(strncmp(Operation, "*", 1) == 0)
(gdb) backtrace
#0  Calculate (Numeral=5, Operation=0x7fffffffdd60 "") at calculate.c:21
#1  0x0000000000400a90 in main () at main.c:13
(gdb) print Numeral
$1 = 5
(gdb) display Numeral
1: Numeral = 5
(gdb) info breakpoints
Num    Type      Disp Enb Address      What
1      breakpoint keep y  0x0000000000400810 in Calculate
                                           at calculate.c:21

      breakpoint already hit 1 time
(gdb) delete 1
(gdb) info breakpoints
No breakpoints or watchpoints.
(gdb)
```

Рисунок 23. Удаление точек останова

Когда еще раз ввели команду *info breakpoints*, вывелось сообщение о том, что точек останова нет.

7. С помощью утилиты *splint* попробуем проанализировать коды файлов *calculate.c*. (см. рис. 24-25)

```
oorazgeldiyeva@oorazgeldiyeva:~/work/os/lab_prog
Файл Правка Вид Поиск Терминал Справка
Transaction test succeeded
Running transaction
  Установка      : splint-3.1.2-15.el7.x86_64
  Проверка       : splint-3.1.2-15.el7.x86_64

Установлено:
  splint.x86_64 0:3.1.2-15.el7

Выполнено!
[oorazgeldiyeva@oorazgeldiyeva lab_prog]$ splint calculate.c
Splint 3.1.2 --- 11 Oct 2015

calculate.h:3:37: 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:12:1: 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:18:1: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:24:1: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:30:1: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:31:4: Dangerous equality comparison involving float types:
      SecondNumeral == 0
  Two real (float, double, or long double) values are compared directly using
```

Рисунок 24. Анализ кода файла calculate.c

```
oorazgeldiyeva@oorazgeldiyeva:~/work/os/l
Файл Правка Вид Поиск Терминал Справка
calculate.c:24:1: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:30:1: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:31:4: 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:34:7: Return value type double does not match declared type float:
      (HUGE_VAL)
  To allow all numeric types to match, use +relaxtypes.
calculate.c:42:1: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:43:7: Return value type double does not match declared type float:
      (pow(Numeral, SecondNumeral))
calculate.c:46:7: Return value type double does not match declared type float:
      (sqrt(Numeral))
calculate.c:48:7: Return value type double does not match declared type float:
      (sin(Numeral))
calculate.c:50:7: Return value type double does not match declared type float:
      (cos(Numeral))
calculate.c:52:7: Return value type double does not match declared type float:
      (tan(Numeral))
calculate.c:56:7: Return value type double does not match declared type float:
      (HUGE_VAL)

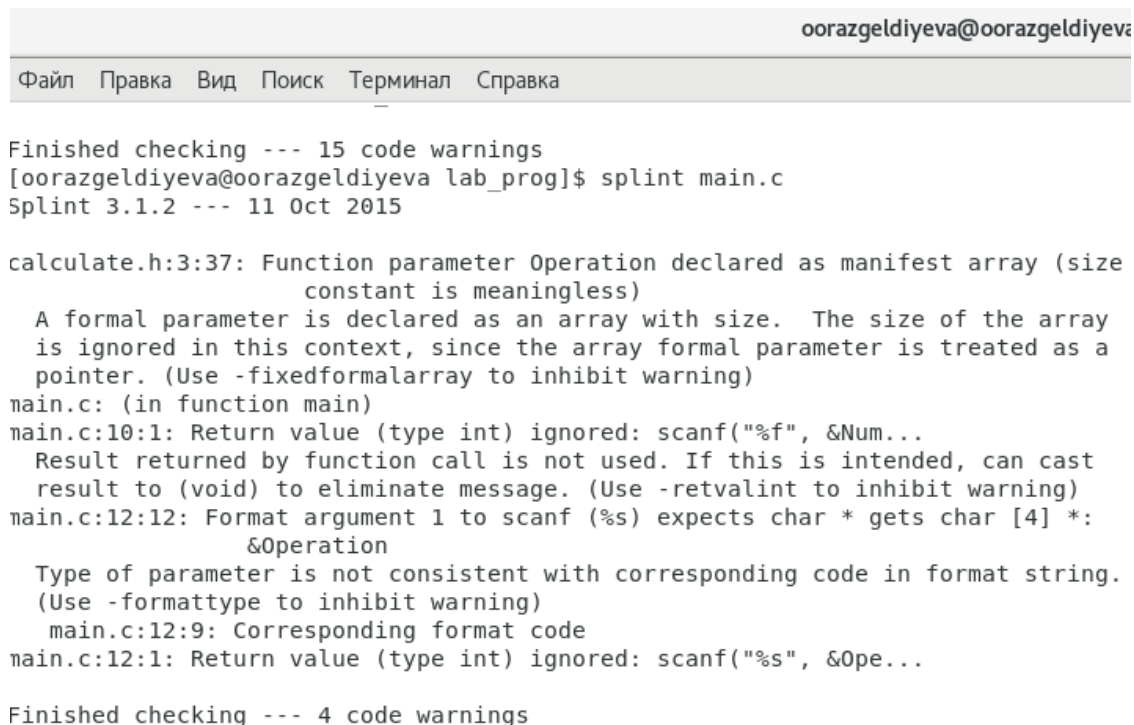
Finished checking --- 15 code warnings
```

Рисунок 25. Анализ кода файла calculate.c

Как видим из рис. 24-25, в программе файла calculate.c 15 предупреждений: первые два предупреждения для параметра Operation - можно было не указывать размер

аргумента; дальше 4 предупреждения для scanf; затем идет предупреждение для равенства двух вещественных чисел и тд.

Попробуем проанализировать main.c. (см. рис. 26)



```
oorazgeldiyeva@oorazgeldiyeva
Файл  Правка  Вид  Поиск  Терминал  Справка

Finished checking --- 15 code warnings
[oorazgeldiyeva@oorazgeldiyeva lab_prog]$ splint main.c
Splint 3.1.2 --- 11 Oct 2015

calculate.h:3:37: 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:10:1: 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:12:12: 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:12:9: Corresponding format code
main.c:12:1: Return value (type int) ignored: scanf("%s", &Ope...

Finished checking --- 4 code warnings
```

Рисунок 26. Анализ кода файла main.c

В этом файле всего 4 предупреждения: первое предупреждение как в предыдущем файле для параметра Operation; остальные для scanf.

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

Библиография

[1] - РУДН, Операционные системы, "Средства, применяемые при разработке программного обеспечения в ОС типа UNIX/Linux"

[2] - Отладчик GDB

[3] - Краткая инструкция по использованию GDB
