Отчет по лабораторной работе №14

Дисциплина: Операционные системы

Чекалова Лилия Руслановна, ст.б. 1032201654

Содержание

List of Figures

1 Цель работы

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

2 Выполнение лабораторной работы

Создаю в домашнем каталоге подкаталог work/os/lab_prog командой mkdir -p и перехожу в этот каталог командой cd (рис. 2.1)

```
lrchekalova@dk8n72 ~ $ mkdir -p ~/work/os/lab_prog
lrchekalova@dk8n72 ~ $ cd work/os/lab_prog
lrchekalova@dk8n72 ~/work/os/lab_prog $
```

Figure 2.1: Создание подкаталога

Создаю файлы calculate.c, calculate.h и main.c командой touch (рис. 2.2)

```
lrchekalova@dk8n72 ~/work/os/lab_prog $ touch calculate.h
lrchekalova@dk8n72 ~/work/os/lab_prog $ touch calculate.c
lrchekalova@dk8n72 ~/work/os/lab_prog $ touch main.c
lrchekalova@dk8n72 ~/work/os/lab_prog $
```

Figure 2.2: Создание файлов

Описываю в файле calculate.h формат вызова функции-калькулятора (рис. 2.3)

```
#ifndef CALCULATE_H_
#define CALCULATE_H_

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

#endif /*CALCULATE_H_*/
```

Figure 2.3: Описание формата вызова функции

Реализую в файле calculate.c функции калькулятора (рис. 2.4) (рис. 2.5)

```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
```

Figure 2.4: Реализация функций калькулятора (1)

```
                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);
    }
}
```

Figure 2.5: Реализация функций калькулятора (2)

Реализую в файле main.c интерфейс пользователя к калькулятору (рис. 2.6)

```
#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 2.6: Реализация пользовательского интерфейса

Произвожу компиляцию файлов с помощью gcc и создаю Makefile командой touch (рис. 2.7)

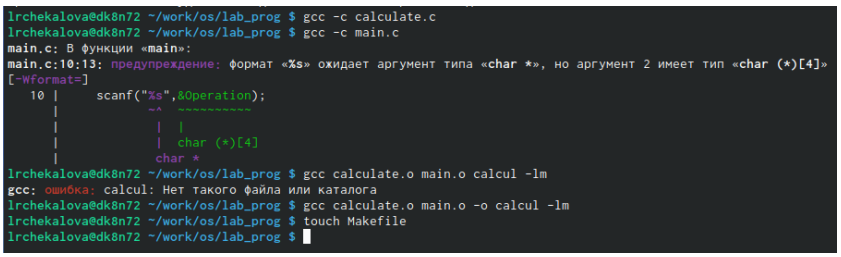

Figure 2.7: Компиляция программы

Исправляю в Makefile синтаксические ошибки: задаю значение CFLAGS и заменяю “gcc” на объявленную, но не используемую переменную CC. Makefile проверяет наличие файлов calculate.o и main.o и компилирует их под названием calcul. Если эти файлы не были найдены, он проверяет наличие файлов calculate.h, calculate.c и main.c, при их обнаружении он делает из них необходимые исполняемые файлы, в противном случае выдает сообщение об ошибке. После завершения компиляции он удаляет уже не нужные исполняемые файлы (файлы с расширением .o) (рис. 2.8)

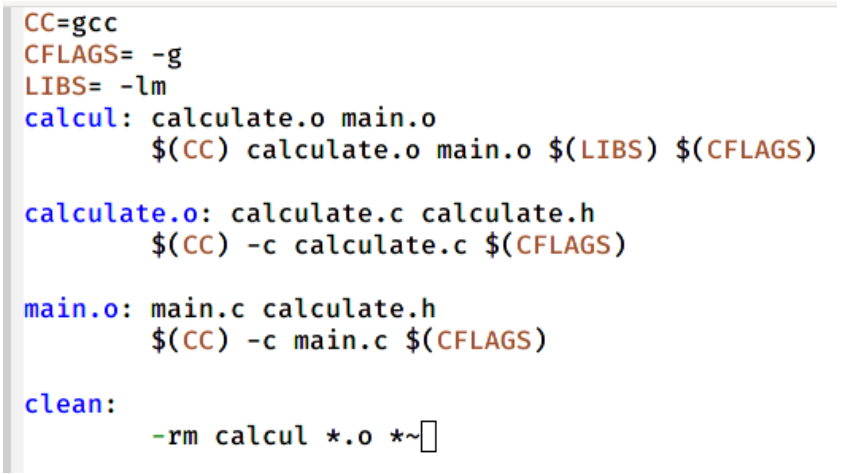

Figure 2.8: Makefile

Загружаю программу в отладчик gdb (рис. 2.9)

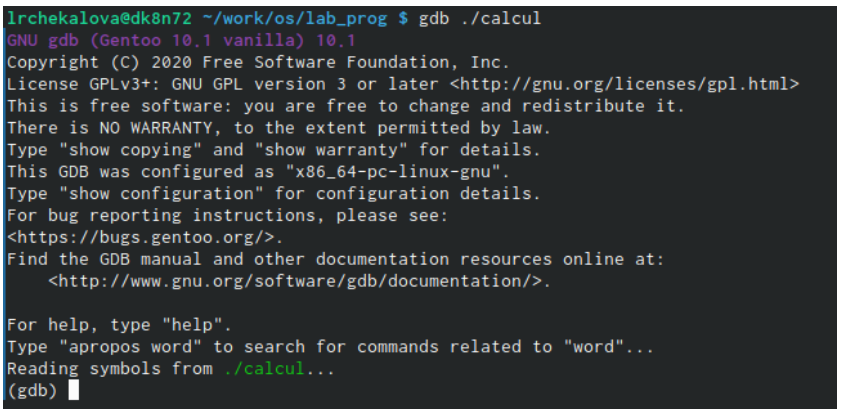

Figure 2.9: Запуск отладчика gdb

Запускаю программу внутри отладчика командой run (рис. 2.10)

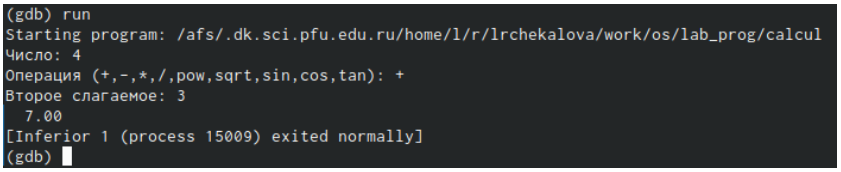

Figure 2.10: Запуск программы в отладчике

Просматриваю исходный код (первые 9 строк) с помощью команды list (рис. 2.11)

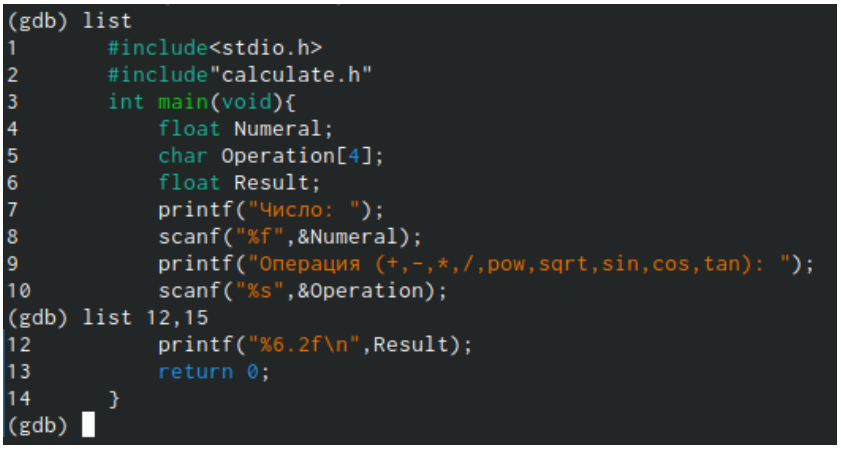

Figure 2.11: Просмотр исходного кода (1)

Просматриваю исходный код с 12 по 15 строку командой List 12,15 и определенные строки не основного файла командой list с параметрами название_фай-

ла:начальная_строка,конечная_строка (рис. 2.12)

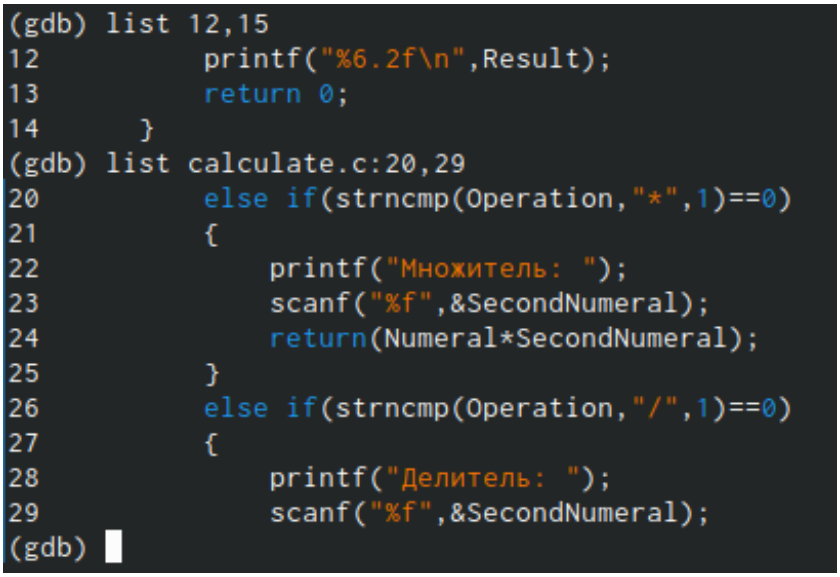

Figure 2.12: Просмотр исходного кода (2)

Устанавливаю точку останова на 21 строке файла calculate.c с помощью команды break (рис. 2.13)

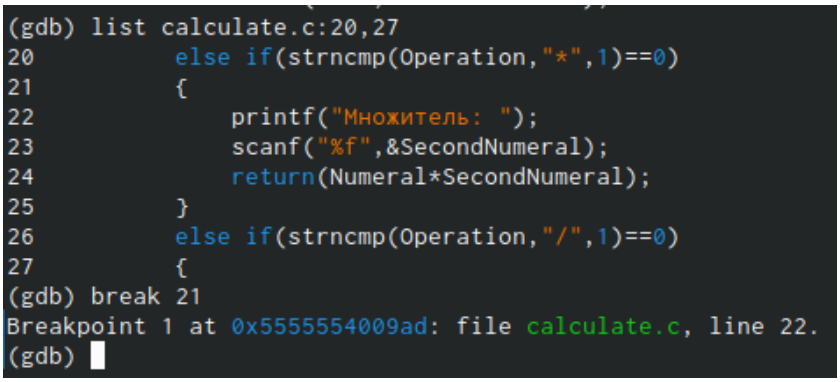

Figure 2.13: Установка точки останова

Вывожу информацию об имеющихся в проекте точках останова командой info breakpoints (рис. 2.14)

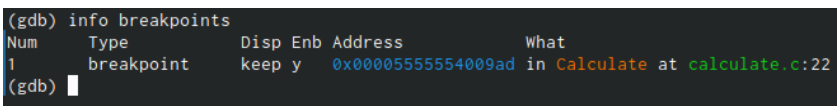

Figure 2.14: Вывод информации о точках останова

Запускаю программу внутри отладчика, чтобы убедиться, что программа останавливается в точке останова (рис. 2.15)

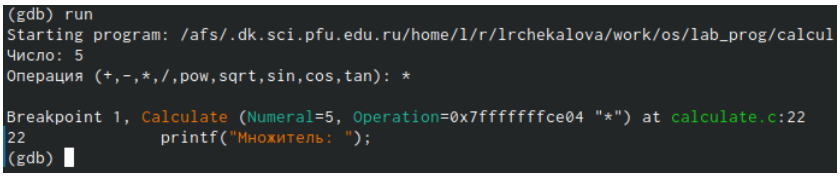

Figure 2.15: Проверка работы точки останова

Просматриваю стек вызываемых функций от начала программы до точки останова командой backtrace (рис. 2.16)

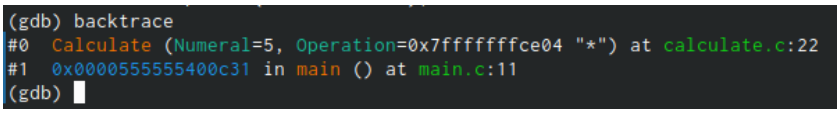

Figure 2.16: Работа команды backtrace

Смотрю, чему равно значение переменной Numeral на данном этапе с помощью команд print и display. Print просто выводит значение переменой, а display выводит название переменной и ее значение. В обоих случаях ее значение равно пяти (рис. 2.17)

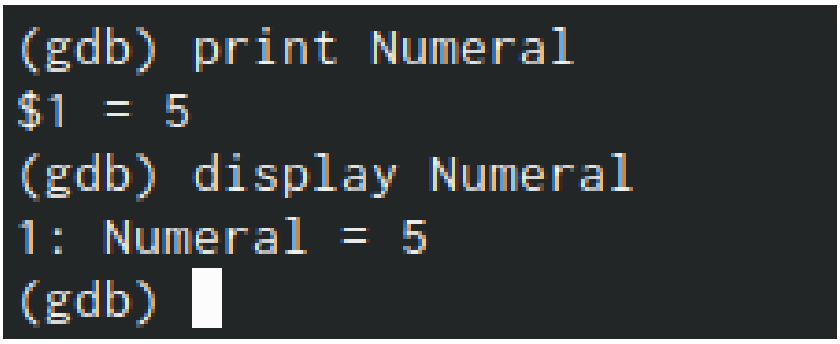

Figure 2.17: Просмотр значения переменной

Удаляю точку останова командой delete 1 и проверяю успешность удаления с помощью info breakpoints (рис. 2.18)

```
(gdb) info breakpoints
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x00005555554009ad in Calculate at calculate.c:22
        breakpoint already hit 1 time
(gdb) delete 1
(gdb) info breakpoints
No breakpoints or watchpoints.
(gdb)
```

Figure 2.18: Удаление точки останова

Анализирую код файла calculate.c с помощью утилиты splint. Она вывела 15 предупреждений о неточностях и несовпадениях в коде (рис. 2.19)

```
lrchekalova@dk8n72 ~/work/os/lab_prog $ splint calculate.c
Splint 3.1.2 --- 13 Jan 2021

calculate.h:4:36: 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:5:36: Function parameter Operation declared as manifest array (size
                    constant is meaningless)
calculate.c: (in function Calculate)
calculate.c:11:9: 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:17:9: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:23:9: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:29:9: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:30:16: 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:32:23: Return value type double does not match declared type float:
                     (HUGE_VAL)
  To allow all numeric types to match, use +relaxtypes.
calculate.c:40:9: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:41:15: Return value type double does not match declared type float:
                     (pow(Numeral, SecondNumeral))
```

Figure 2.19: Анализ кода calculate.c

Анализирую код файла main.c с помощью той же утилиты. Она вывела 4 предупреждения о неточностях в коде (рис. 2.20)

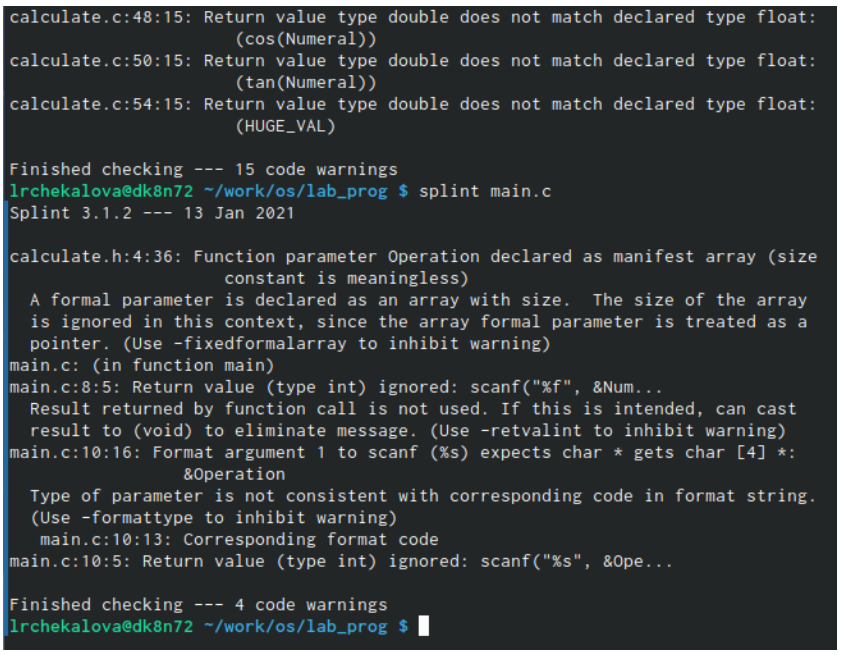

Figure 2.20: Анализ кода main.c

3 Выводы

После выполнения данной лабораторной работы я научилась реализовывать простейший калькулятор, компилировать программы, производить отладку программы в GDB и анализировать программу с помощью утилиты splint.

4 Библиография

1. Теоретические материалы к лабораторной работе: https://esystem.rudn.ru/pluginfile.php/11
lab_prog.pdf