

Отчет по лабораторной работе №14

Дисциплина: Операционные системы

Чекалова Лилия Руслановна, ст.б. 1032201654

Содержание

1	Цель работы	4
2	Выполнение лабораторной работы	5
3	Выводы	14
4	Библиография	15

List of Figures

2.1	Создание подкаталога	5
2.2	Создание файлов	5
2.3	Описание формата вызова функции	5
2.4	Реализация функций калькулятора (1)	6
2.5	Реализация функций калькулятора (2)	7
2.6	Реализация пользовательского интерфейса	7
2.7	Компиляция программы	8
2.8	Makefile	8
2.9	Запуск отладчика gdb	9
2.10	Запуск программы в отладчике	9
2.11	Просмотр исходного кода (1)	9
2.12	Просмотр исходного кода (2)	10
2.13	Установка точки останова	10
2.14	Вывод информации о точках останова	10
2.15	Проверка работы точки останова	11
2.16	Работа команды backtrace	11
2.17	Просмотр значения переменной	11
2.18	Удаление точки останова	12
2.19	Анализ кода calculate.c	12
2.20	Анализ кода main.c	13

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)

```

#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)

```

lrchekalova@dkn72 ~/work/os/lab_prog $ gcc -c calculate.c
lrchekalova@dkn72 ~/work/os/lab_prog $ gcc -c main.c
main.c: 8 функции «main»:
main.c:10:13: предупреждение: формат «%s» ожидает аргумент типа «char *», но аргумент 2 имеет тип «char (*)[4]»
[-Wformat=]
10 |         scanf("%s", &operation);
    |         ^~~~~~
    |         |
    |         | char (*)[4]
    |         char *
lrchekalova@dkn72 ~/work/os/lab_prog $ gcc calculate.o main.o calcul -lm
gcc: ошибка: calcul: Нет такого файла или каталога
lrchekalova@dkn72 ~/work/os/lab_prog $ gcc calculate.o main.o -o calcul -lm
lrchekalova@dkn72 ~/work/os/lab_prog $ touch Makefile
lrchekalova@dkn72 ~/work/os/lab_prog $

```

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)

```

CC=gcc
CFLAGS= -g
LIBS= -lm
calcul: calculate.o main.o
    $(CC) calculate.o main.o $(LIBS) $(CFLAGS)

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 *~

```

Figure 2.8: Makefile

Загружаю программу в отладчик gdb (рис. 2.9)


```
lrchekalova@dk8n72 ~/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) █
```

Figure 2.9: Запуск отладчика gdb

Запускаю программу внутри отладчика командой run (рис. 2.10)

```
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/l/r/lrchekalova/work/os/lab_prog/calcul
Число: 4
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): +
Второе слагаемое: 3
7.00
[Inferior 1 (process 15009) exited normally]
(gdb) █
```

Figure 2.10: Запуск программы в отладчике

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

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

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)

```
(gdb) list 12,15
12         printf("%6.2f\n",Result);
13         return 0;
14     }
(gdb) list calculate.c:20,29
20         else if(strncmp(Operation,"*",1)==0)
21         {
22             printf("Множитель: ");
23             scanf("%f",&SecondNumeral);
24             return(Numeral*SecondNumeral);
25         }
26         else if(strncmp(Operation,"/",1)==0)
27         {
28             printf("Делитель: ");
29             scanf("%f",&SecondNumeral);
(gdb) █
```

Figure 2.12: Просмотр исходного кода (2)

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

```
(gdb) list calculate.c:20,27
20         else if(strncmp(Operation,"*",1)==0)
21         {
22             printf("Множитель: ");
23             scanf("%f",&SecondNumeral);
24             return(Numeral*SecondNumeral);
25         }
26         else if(strncmp(Operation,"/",1)==0)
27         {
(gdb) break 21
Breakpoint 1 at 0x5555554009ad: file calculate.c, line 22.
(gdb) █
```

Figure 2.13: Установка точки останова

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

```
(gdb) info breakpoints
Num   Type             Disp Enb Address              What
1     breakpoint      keep y   0x00005555554009ad in calculate at calculate.c:22
(gdb) █
```

Figure 2.14: Вывод информации о точках останова

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

```
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/l/r/lrchekalova/work/os/lab_prog/calcul
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): *

Breakpoint 1, Calculate (Numeral=5, Operation=0x7fffffffce04 "*") at calculate.c:22
22      printf("Множитель: ");
(gdb) █
```

Figure 2.15: Проверка работы точки останова

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

```
(gdb) backtrace
#0 Calculate (Numeral=5, Operation=0x7fffffffce04 "*") at calculate.c:22
#1 0x000055555400c31 in main () at main.c:11
(gdb) █
```

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 переменной, а display выводит название переменной и ее значение. В обоих случаях ее значение равно пяти (рис. 2.17)

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

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   0x0000555554009ad 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)

```

calculate.c:48:15: Return value type double does not match declared type float:
      (cos(Numeral))
calculate.c:50:15: Return value type double does not match declared type float:
      (tan(Numeral))
calculate.c:54:15: Return value type double does not match declared type float:
      (HUGE_VAL)

Finished checking --- 15 code warnings
lrchekalova@dk8n72 ~/work/os/lab_prog $ splint main.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)
main.c: (in function main)
main.c:8:5: 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:10:16: 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:10:13: Corresponding format code
main.c:10:5: Return value (type int) ignored: scanf("%s", &ope...

Finished checking --- 4 code warnings
lrchekalova@dk8n72 ~/work/os/lab_prog $ █

```

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/1111111/mod_resource/content/1/lab_prog.pdf