

# **Лабораторная работа №13**

**Операционные системы**

Кирилюк С. А.

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Выполнение лабораторной работы</b>	<b>6</b>
<b>3</b>	<b>Выводы</b>	<b>14</b>

## Список иллюстраций

2.1	Создание файлов . . . . .	6
2.2	Скрипт для calculate.c . . . . .	6
2.3	Скрипт для calculate.h . . . . .	7
2.4	Скрипт для main.c . . . . .	7
2.5	Компиляция . . . . .	7
2.6	Создание файла Makefile . . . . .	8
2.7	Изменения в Makefile . . . . .	8
2.8	Проверка работы Makefile . . . . .	9
2.9	Запуск отладчика . . . . .	9
2.10	Запуск программы внутри отладчика . . . . .	10
2.11	Просмотр исходного кода . . . . .	10
2.12	Просмотр строк с 12 по 15 . . . . .	10
2.13	Просмотр определённых строк . . . . .	11
2.14	Установка точки останова (1) . . . . .	11
2.15	Установка точки останова (2) . . . . .	11
2.16	Вывод информации о точках останова . . . . .	11
2.17	Запуск программы внутри отладчика . . . . .	12
2.18	Значение переменной Numeral . . . . .	12
2.19	Сравнение значений переменной Numeral . . . . .	12
2.20	Убираем точки останова . . . . .	12
2.21	Анализ кода файла calculate.c . . . . .	13
2.22	Анализ кода файла main.c . . . . .	13

## Список таблиц

# 1 Цель работы

Приобрести простейшие навыки 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

Перед выполнением заданий в домашнем каталоге создала подкаталог ~/work/os/lab\_prog., а также файлы: calculate.h, calculate.c, main.c.(рис. 2.1).

```
[sakirilyuk@fedora lab_prog]$ touch calculate.h calculate.c main.c.  
[sakirilyuk@fedora lab_prog]$ ls  
calculate.c calculate.h main.c.  
[sakirilyuk@fedora lab_prog]$
```

Рис. 2.1: Создание файлов

При помощи редактора для данных файлов написала скрипты (рис. 2.2), (рис. 2.3), (рис. 2.4).

```
////////////////////////////////////  
// 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)
```

Рис. 2.2: Скрипт для calculate.c

```

// calculate.h

#ifndef CALCULATE_H_
#define CALCULATE_H_

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

#endif /*CALCULATE_H_*/

```

Рис. 2.3: Скрипт для calculate.h

```

// 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;
}

```

Рис. 2.4: Скрипт для main.c

После чего выполнила компиляцию программы посредством gcc (рис. 2.5).

```

[sakirilyuk@fedora lab_prog]$ emacs
[sakirilyuk@fedora lab_prog]$ gcc -c calculate.c
[sakirilyuk@fedora lab_prog]$ gcc -c main.c
[sakirilyuk@fedora lab_prog]$ gcc calculate.o main.o -o calcul -lm
[sakirilyuk@fedora lab_prog]$

```

Рис. 2.5: Компиляция

Затем создала Makefile и написала для него скрипт (рис. 2.6). Этот файл используется для автоматической компиляции main.c, calculate.c и создание из них исполняемого файла calcul. Помимо этого, в файле также есть функция 'clean', используемая для удаления всех файлов.

Затем я изменила его содержание (рис. 2.7) и проверила его работу (рис. 2.8).

```

#
# 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

```

Рис. 2.6: Создание файла Makefile

```

#
# 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

```

Рис. 2.7: Изменения в Makefile



```
[sakirilyuk@fedora lab_prog]$ make clean
rm calcul *.o *~
[sakirilyuk@fedora lab_prog]$ make calculate.o
gcc -c calculate.c -g
[sakirilyuk@fedora lab_prog]$ make main.o
gcc -c main.c -g
[sakirilyuk@fedora lab_prog]$ make calcul
gcc calculate.o main.o -o calcul -lm
[sakirilyuk@fedora lab_prog]$
```

Рис. 2.8: Проверка работы Makefile

С помощью gdb я начала выполнение отладки программы calcul. Сначала я запустила отладчик GDB, загрузив в него программу для отладки (рис. 2.9). Для запуска программы внутри отладчика ввела команду run (рис. 2.10).

```
[sakirilyuk@fedora lab_prog]$ gdb ./calcul
GNU gdb (GDB) Fedora Linux 12.1-7.fc37
Copyright (C) 2022 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".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
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)
```

Рис. 2.9: Запуск отладчика

```

(gdb) run
Starting program: /home/sakirilyuk/work/os/lab_prog/calcul

This GDB supports auto-downloading debuginfo from the following URLs:
https://debuginfod.fedoraproject.org/
Enable debuginfod for this session? (y or [n]) n
Debuginfod has been disabled.
To make this setting permanent, add 'set debuginfod enabled off' to .gdbinit.
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): +
Второе слагаемое: 9
14.00
[Inferior 1 (process 4003) exited normally]
Missing separate debuginfos, use: dnf debuginfo-install glibc-2.36-9.fc37.x86_64
(gdb)

```

Рис. 2.10: Запуск программы внутри отладчика

Для постраничного (по 9 строк) просмотра исходного код использовала команду `list` (рис. 2.11), для просмотра строк с 12 по 15 основного файла - `list` с параметрами (рис. 2.12). Для просмотра определённых строк не основного файла так же использовала `list` с параметрами (рис. 2.13).

```

(gdb) list
1  //////////////////////////////////////////////////
2  // main.c
3
4  #include <stdio.h>
5  #include "calculate.h"
6  int
7  main (void)
8  {
9  float Numeral;
10 char Operation[4];
(gdb)

```

Рис. 2.11: Просмотр исходного кода

```

(gdb) list 12,15
12 printf("Число: ");
13 scanf("%f",&Numeral);
14 printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): ");
15 scanf("%s",&Operation);
(gdb)

```

Рис. 2.12: Просмотр строк с 12 по 15

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

Рис. 2.13: Просмотр определённых строк

Затем установила точку останова в файле calculate.c на строке номер 21 (рис. 2.14), (рис. 2.15). После чего вывела информацию об имеющихся в проекте точках останова (рис. 2.16). Я запустила программу внутри отладчика и убедилась, что программа остановилась в момент прохождения точки останова (рис. 2.17).

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

Рис. 2.14: Установка точки останова (1)

```
(gdb) break 21
Breakpoint 1 at 0x40120f: file calculate.c, line 21.
(gdb)
```

Рис. 2.15: Установка точки останова (2)

```
(gdb) info breakpoints
Num   Type             Disp Enb Address            What
1      breakpoint      keep y   0x000000000040120f in Calculate
                                at calculate.c:21
(gdb)
```

Рис. 2.16: Вывод информации о точках останова

```

(gdb) run
Starting program: /home/sakirilyuk/work/os/lab_prog/calcul
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): -

Breakpoint 1, calculate (Numeral=5, Operation=0x7fffffffdf14 "-") at calculate.c:21
21      printf("Вычитаемое: ");
(gdb) backtrace
#0  calculate (Numeral=5, Operation=0x7fffffffdf14 "-") at calculate.c:21
#1  0x00000000004014eb in main () at main.c:16
(gdb)

```

Рис. 2.17: Запуск программы внутри отладчика

Также я посмотрела, чему равно на этом этапе значение переменной Numeral (рис. 2.18). На экран было выведено число 5. Сравнила его с результатом вывода на экран после использования другой команды (рис. 2.19). После чего убрала точки останова (рис. 2.20).

```

(gdb) print Numeral
$1 = 5
(gdb)

```

Рис. 2.18: Значение переменной Numeral

```

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

```

Рис. 2.19: Сравнение значений переменной Numeral

```

(gdb) info breakpoints
Num   Type      Disp Enb Address          What
1     breakpoint keep y  0x000000000040120f in calculate
                                           at calculate.c:21
breakpoint already hit 1 time
(gdb) delete 1
(gdb)

```

Рис. 2.20: Убираем точки останова

В заключение, с помощью утилиты `split` я попробовала проанализировать коды файлов `calculate.c` (рис. 2.21) и `main.c` (рис. 2.22).

```
[sakirilyuk@fedora lab_prog]$ splint calculate.c
Splint 3.1.2 --- 23 Jul 2022

calculate.h:7:38: 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:10:32: Function parameter Operation declared as manifest array
(size constant is meaningless)
calculate.c: (in function Calculate)
calculate.c:16:2: 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:22:2: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:28:2: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:34:2: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:35:5: 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
```

Рис. 2.21: Анализ кода файла calculate.c

```
[sakirilyuk@fedora lab_prog]$ splint main.c
Splint 3.1.2 --- 23 Jul 2022

calculate.h:7:38: 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:13:2: 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:15:13: 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:15:10: Corresponding format code
main.c:15:2: Return value (type int) ignored: scanf("%s", &Ope...

Finished checking --- 4 code warnings
[sakirilyuk@fedora lab_prog]$
```

Рис. 2.22: Анализ кода файла main.c

## 3 Выводы

В ходе 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.