

Лабораторная работа номер 13

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

Мальков Роман

Содержание

Цель работы	3
Задание	4
Ход работы	11
Выводы	23

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.

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.c`:

```
1 ///////////////////////////////////////////////////////////////////
2 // calculate.c
3
4 #include <stdio.h>
5 #include <math.h>
6 #include <string.h>
7 #include "calculate.h"
8
9 float
10 Calculate(float Numeral, char Operation[4])
11 {
12     float SecondNumeral;
13     if(strncmp(Operation, "+", 1) == 0)
14     {
15         printf("Второе слагаемое: ");
```

```
16 scanf("%f",&SecondNumeral);
17 return(Numeral + SecondNumeral);
18 }
19 else if(strncmp(Operation, "-", 1) == 0)
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);
30 }
31 else if(strncmp(Operation, "/", 1) == 0)
32 {
33 printf("Делитель: ");
34 scanf("%f",&SecondNumeral);
35 if(SecondNumeral == 0)
36 {
37 printf("Ошибка: деление на ноль! ");
38 return(HUGE_VAL);
39 }
40 else
41 return(Numeral / SecondNumeral);
42 }
43 else if(strncmp(Operation, "pow", 3) == 0)
44 {
```

```

45 printf("Степень: ");
46 scanf("%f",&SecondNumeral);
47 return(pow(Numeral, SecondNumeral));
48 }
49 else if(strncmp(Operation, "sqrt", 4) == 0)
50 return(sqrt(Numeral));
51 else if(strncmp(Operation, "sin", 3) == 0)
52 return(sin(Numeral));
53 else if(strncmp(Operation, "cos", 3) == 0)
54 return(cos(Numeral));
55 else if(strncmp(Operation, "tan", 3) == 0)
56 return(tan(Numeral));
57 else
58 {
59 printf("Неправильно введено действие ");
60 return(HUGE_VAL);
61 }
62 }
```

Интерфейсный файл calculate.h, описывающий формат вызова функции калькулятора:

```

1 ///////////////////////////////////////////////////
2 // calculate.h
3
4 #ifndef CALCULATE_H_
5 #define CALCULATE_H_
6
7 float Calculate(float Numeral, char Operation[4]);
8
9 #endif /*CALCULATE_H_*/
```

Основной файл main.c, реализующий интерфейс пользователя к калькулятору:

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

3. Выполните компиляцию программы посредством gcc:

```
1 gcc -c calculate.c
2 gcc -c main.c
3 gcc calculate.o main.o -o calcul -lm
```

4. При необходимости исправьте синтаксические ошибки.

5. Создайте Makefile со следующим содержанием:

```
1 #
2 # Makefile
3 #
4
5 CC = gcc
6 CFLAGS =
7 LIBS = -lm
8
9 calcul: calculate.o main.o
10 gcc calculate.o main.o -o calcul $(LIBS)
11
12 calculate.o: calculate.c calculate.h
13 gcc -c calculate.c $(CFLAGS)
14
15 main.o: main.c calculate.h
16 gcc -c main.c $(CFLAGS)
17
18 clean:
19 -rm calcul *.o *~
20
21 # 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 код 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 точки 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=0x7fffffff280 "-")
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:

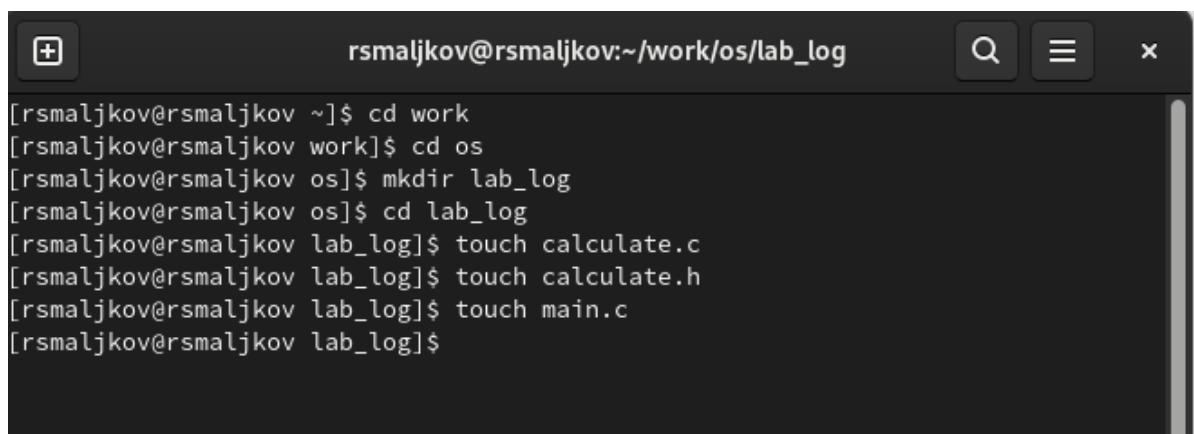
```
info breakpoints
```

```
delete 1
```

7. С помощью утилиты splint попробуйте проанализировать коды файлов calculate.c и main.c.8.

Ход работы

1. В домашнем каталоге создаем подкаталог `~/work/os/lab_prog` (Скриншот 1).



The screenshot shows a terminal window with the following session:

```
[rsmaljkov@rsmaljkov ~]$ cd work
[rsmaljkov@rsmaljkov work]$ cd os
[rsmaljkov@rsmaljkov os]$ mkdir lab_log
[rsmaljkov@rsmaljkov os]$ cd lab_log
[rsmaljkov@rsmaljkov lab_log]$ touch calculate.c
[rsmaljkov@rsmaljkov lab_log]$ touch calculate.h
[rsmaljkov@rsmaljkov lab_log]$ touch main.c
[rsmaljkov@rsmaljkov lab_log]$
```

Рис. 1: Screenshot_1

2. Создаем в нём файлы: `calculate.h`, `calculate.c`, `main.c` (Скриншоты 2 - 4).

The screenshot shows a code editor with two tabs: "calculate.h" and "calculate.c". The "calculate.h" tab is active, displaying the following C header file:

```
1 //////////////////////////////////////////////////////////////////
2 // calculate.h
3
4 #ifndef CALCULATE_H_
5 #define CALCULATE_H_
6
7 float Calculate(float Numeral, char Operation[4]);
8
9#endif /*CALCULATE_H_*/
10
```

Рис. 2: Screenshot_2

The screenshot shows a code editor with the "calculate.c" tab active, displaying the following C source code:

```
1 //////////////////////////////////////////////////////////////////
2 // calculate.c
3
4 #include <stdio.h>
5 #include <math.h>
6 #include <string.h>
7 #include "calculate.h"
8
9 float
10 Calculate(float Numeral, char Operation[4])
11 {
12     float SecondNumeral;
13     if(strncmp(Operation, "+", 1) == 0)
14     {
15         printf("Второе слагаемое: ");
16         scanf("%f", &SecondNumeral);
17         return(Numeral + SecondNumeral);
18     }
19     else if(strncmp(Operation, "-", 1) == 0)
20     {
```

Рис. 3: Screenshot_3

```
1 //////////////////////////////////////////////////////////////////
2 // main.c
3
4 #include <stdio.h>
5 #include "calculate.h"
6
7 int
8 main (void)
9 {
10 float Numeral;
11 char Operation[4];
12 float Result;
13 printf("Число: ");
14 scanf("%f",&Numeral);
15 printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): ");
16 scanf("%s",&Operation);
17 Result = Calculate(Numeral, Operation);
18 printf("%6.2f\n",Result);
19 return 0;
20 }
21 |
```

Рис. 4: Screenshot_4

3. Выполняем компиляцию программы посредством использования gcc:

```
[rsmaljkov@rsmaljkov lab_log]$ gcc -c calculate.c
[rsmaljkov@rsmaljkov lab_log]$ gcc -c main.c
[rsmaljkov@rsmaljkov lab_log]$ gcc calculate.o main.o -o calcul -lm
[rsmaljkov@rsmaljkov lab_log]$
```

Рис. 5: Screenshot_5

4. При необходимости исправляем синтаксические ошибки.

5. Создаем Makefile (Скриншоты 6 - 7)

```
[rsmaljkov@rsmaljkov lab_log]$ touch makefile
[rsmaljkov@rsmaljkov lab_log]$
```

Рис. 6: Screenshot_6

```
1 #
2 # Makefile
3 #
4
5 CC = gcc
6 CFLAGS =
7 LIBS = -lm
8
9 calcul: calculate.o main.o
10 gcc calculate.o main.o -o calcul $(LIBS)
11
12 calculate.o: calculate.c calculate.h
13 gcc -c calculate.c $(CFLAGS)
14
15 main.o: main.c calculate.h
16 gcc -c main.c $(CFLAGS)
17
18 clean:
19 -rm calcul *.o *~
20
21 # End Makefile
```

Рис. 7: Screenshot_7

Разберём этот файл, так в пятой строке мы видим переменную CC которой присвоена bash команда gcc, однако эта переменная не используется, впрочем в ней и не было никакой нужды, далее в строке 6 видим переменную CFLAGS которой будет присвоено значение -g, данная переменная используется в качестве флага в командах которые мы разберем ниже, LIBS также является флагом. В девятой строке calcul является целью, calculate.o и main.o — название файлов, который мы хотим скомпилировать; во второй строке, начиная с табуляции, задана команда

компиляции gcc с опциями. Строки ниже устроены аналогично. Стоит отметить что в примере кода приведенного на скриншоте есть несколько критических ошибок которые мы исправим далее.

6. С помощью gdb выполняем отладку программы calcul(при этом стоит исправить файл Makefile, удалить все файлы формата .o и calcul файл, затем запустить Makefile (Скриншоты 8 - 10)):

```
CFLAGS = -g
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. 8: Screenshot_8

```
[rsmaljkov@rsmaljkov lab_log]$ make
gcc calculate.o main.o -o calcul -lm
[rsmaljkov@rsmaljkov lab_log]$
```

Рис. 9: Screenshot_9

```
[rsmaljkov@rsmaljkov lab_log]$ ls
calcul calculate.h main.c Makefile
calculate.c calculate.o main.o Makefile~
[rsmaljkov@rsmaljkov lab_log]$ rm calcul
[rsmaljkov@rsmaljkov lab_log]$ rm calculate.o
[rsmaljkov@rsmaljkov lab_log]$ rm main.o
[rsmaljkov@rsmaljkov lab_log]$ ls
calculate.c calculate.h main.c Makefile Makefile~
[rsmaljkov@rsmaljkov lab_log]$ make
```

Рис. 10: Screenshot_10

– Запускаем отладчик GDB, загрузив в него программу для отладки (Скриншот 11):

```
gdb ./calcul
```

```
[rsmaljkov@rsmaljkov lab_log]$ gdb ./calcul
GNU gdb (GDB) Fedora 11.2-2.fc35
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:
<https://www.gnu.org/software/gdb/documentation/>
```

Рис. 11: Screenshot_11

– Для запуска программы внутри отладчика вводим команду run:

```
run
```

– Для постраничного (по 9 строк) просмотра исходного кода используем команду list (Скриншот 12):

```
list
```

```
Число: 4
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): +
Второе слагаемое: 5
9.00
[Inferior 1 (process 5190) exited normally]
(gdb) list
1      /////////////////
2      // main.c
3
4      #include <stdio.h>
5      #include "calculate.h"
6
7      int
8      main (void)
9      {
10      float Numeral;
(gdb)
```

Рис. 12: Screenshot_12

– Для просмотра строк с 12 по 15 основного файла используем list с параметрами (Скриншот 13):

```
list 12,15
```

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

Рис. 13: Screenshot_13

– Для просмотра определённых строк не основного файла используем list с параметрами:

```
list calculate.c:20,29
```

– Устанавливаем точку останова в файле calculate.c на строке номер 21 (Скриншот 14):

```
list calculate.c:20,27
```

```
break 21
```

```
(gdb) list calculate.c:20,27
20      {
21      printf("Вычитаемое: ");
22      scanf("%f",&SecondNumeral);
23      return(Numerical - SecondNumeral);
24  }
25  else if(strcmp(Operation, "*", 1) == 0)
26  {
27      printf("Множитель: ");
(gdb) break 21
Breakpoint 1 at 0x40120f: file calculate.c, line 21.
(gdb)
```

Рис. 14: Screenshot_14

– Выводим информацию об имеющихся в проекте точка останова (Скриншот 15):

```
info breakpoints
```

```

(gdb) list calculate.c:20,27
20
21     printf("Вычитаемое: ");
22     scanf("%f",&SecondNumeral);
23     return(Numerical - SecondNumeral);
24 }
25 else if(strncmp(Operation, "*", 1) == 0)
26 {
27     printf("Множитель: ");
(gdb) break 21
Breakpoint 1 at 0x40120f: file calculate.c, line 21.
(gdb) info breakpoint
Num      Type            Disp Enb Address          What
1        breakpoint      keep y   0x000000000040120f in Calculate
                                         at calculate.c:21
(gdb) █

```

Рис. 15: Screenshot_15

– Запускаем программу внутри отладчика и убеждаемся, что программа останавливается в момент прохождения точки останова:

```

run
5
-
backtrace

```

– Отладчик выдаст следующую информацию (Скриншот 16):

```

#0 Calculate (Numerical=5, Operation=0x7fffffff280 "-")
at calculate.c:21
#1 0x0000000000400b2b in main () at main.c:17

```

```
(gdb) run
Starting program: /home/rsmaljkov/work/os/lab_log/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=0x7fffffffdf04 "-") at calculate.c:21
21      printf("Вычитаемое: ");
(gdb) backtrace
#0  Calculate (Numeral=5, Operation=0x7fffffffdf04 "-") at calculate.c:21
#1  0x00000000004014eb in main () at main.c:17
(gdb)
```

Рис. 16: Screenshot_16

а команда backtrace покажет весь стек вызываемых функций от начала программы до текущего места. – Посмотрим, чему равно на этом этапе значение переменной Numeral, введя:

```
print Numeral
```

На экран должно быть выведено число 5. – Сравниваем с результатом вывода на экран после использования команды:

```
display Numeral
```

– Убираем точки останова (Скриншот 17):

```
info breakpoints
```

```
delete 1
```

```
(gdb) print Numeral
$1 = 5
(gdb) displaey Numeral
Undefined command: "displaey". Try "help".
(gdb) display Numeral
1: Numeral = 5
(gdb) info Breakpoints
Undefined info command: "Breakpoints". Try "help info".
(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
Delete all breakpoints? (y or n) y
(gdb) █
```

Рис. 17: Screenshot_17

7. С помощью утилиты `splint` попробуем проанализировать коды файлов `calculate.c` и `main.c`. (Скриншоты 18 - 19).

```
[rsmaljkov@rsmaljkov lab_log]$ splint calculate.c
Splint 3.1.2 --- 23 Jul 2021

calculate.h:7:37: 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:10:31: Function parameter Operation declared as manifest array
                           (size constant is meaningless)
calculate.c: (in function Calculate)
calculate.c:16:1: 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:22:1: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:28:1: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:34:1: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:35:4: 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
```

Рис. 18: Screenshot_18

```
[rsmaljkov@rsmaljkov lab_log]$ splint main.c
Splint 3.1.2 --- 23 Jul 2021

calculate.h:7:37: 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:14:1: 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:16:12: 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:16:9: Corresponding format code
main.c:16:1: Return value (type int) ignored: scanf("%s", &Ope...
```

Рис. 19: Screenshot_19

Выводы

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