

# Лабораторная работа 13

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

---

Арина Олеговна Аристова

2022, 3 June

RUDN University, Moscow, Russian Federation

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.

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.
3. Выполните компиляцию программы посредством `gcc`.
4. При необходимости исправьте синтаксические ошибки.
5. Создайте `Makefile` со заданным содержанием. 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:
  - Для запуска программы внутри отладчика введите команду `run`.
  - Для постраничного (по 9 строк) просмотра исходного код используйте команду `list`.
  - Для просмотра строк с 12 по 15 основного файла используйте `list` с параметрами.

- Для просмотра определённых строк не основного файла используйте `list` с параметрами.
- Установите точку останова в файле `calculate.c` на строке номер 21.
- Выведите информацию об имеющихся в проекте точка останова.
- Запустите программу внутри отладчика и убедитесь, что программа остановится в момент прохождения точки останова.

- Посмотрите, чему равно на этом этапе значение переменной `Numeral`, введя `print Numeral`.
  - Сравните с результатом вывода на экран после использования команды: `display Numeral`.
  - Уберите точки останова.
7. С помощью утилиты `splint` попробуйте проанализировать коды файлов `calculate.c` и `main.c`.

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.



Отладка — этап разработки компьютерной программы, на котором обнаруживают, локализуют и устраняют ошибки.

Чтобы понять, где возникла ошибка, приходится:

- узнавать текущие значения переменных;
- выяснять, по какому пути выполнялась программа.

1. Создаю поддиректорию `~/work/os/lab_prog`:

```
[aoaristova@fedora ~]$ cd work/os  
[aoaristova@fedora os]$ mkdir lab_prog  
[aoaristova@fedora os]$ cd lab_prog  
[aoaristova@fedora lab_prog]$
```

Рис. 1: Создание поддиректории `~/work/os/lab_prog`.

2. Создаю файлы `calculate.h`, `calculate.c`, `main.c` и заполняю их, согласно описанию лабораторной работы:

```
[aoaristova@fedora lab_prog]$ vi calculate.h
[aoaristova@fedora lab_prog]$ vi calculate.h
[aoaristova@fedora lab_prog]$ mv calculate.h calculate.c
[aoaristova@fedora lab_prog]$ ls
calculate.c
[aoaristova@fedora lab_prog]$ vi calculate.h
[aoaristova@fedora lab_prog]$ vi main.c
[aoaristova@fedora lab_prog]$ vi
```

Рис. 2: Создание и заполнение файлов.

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.

# Выполнение лабораторной работы

```
///////////////////////////////////////////////////////////////////
// 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)
    {
        printf("Вычитаемое: ");
        scanf("%f",&SecondNumeral);
        return(Numeral - SecondNumeral);
    }
    else if(strncmp(Operation, "*", 1) == 0)
    {
        printf("Множитель: ");
        scanf("%f",&SecondNumeral);
        return(Numeral * SecondNumeral);
    }
}
```

Рис. 3: Содержимое файла calculate.c.

```
////////////////////////////////////  
// calculate.h  
  
#ifndef CALCULATE_H_  
#define CALCULATE_H_  
  
float Calculate(float Numeral, char Operation[4]);  
  
#endif /*CALCULATE_H_*/
```

Рис. 4: Содержимое файла 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. 5: Содержимое файла main.c.

### 3. Выполняю компиляцию программы посредством gcc:

```
[aoaristova@fedora lab_prog]$ gcc -c calculate.c -g  
[aoaristova@fedora lab_prog]$ gcc -c main.c -g  
[aoaristova@fedora lab_prog]$ gcc calculate.o main.o -o calcul -lm
```

Рис. 6: Компиляция программы.



4. Создаю. Makefile с содержанием, согласно описанию лабораторной работы, при этом немного изменяю его:

```
#  
# Makefile  
#  
  
CC=gcc  
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. 7: Содержимое Makefile.

В Makefile указываю компилятор gcc, флаг -g и дополнительные библиотеки -lm. Описываю, какие команды необходимо запустить, чтобы получить файлы calcul, calculate.o и main.o, подключив дополнительные библиотеки и флаги. А в поле clean описывается удаление файлов calcul и файлов, оканчивающихся на “.o”.

5. С помощью gdb выполняю отладку программы calcul:

Запускаю отладчик GDB, загрузив в него программу для отладки.

Запускаю программу внутри отладчика, введя команду run:

```
[aoristova@fedora lab_prog]$ gdb ./calcul
GNU gdb (GDB) Fedora 10.2-9.fc35
Copyright (C) 2021 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) run
Starting program: /home/aoristova/work/os/lab_prog/calcul
Downloading separate debug info for /home/aoristova/work/os/lab_prog/system-su
plied DSO at 0x1ffff7fc9000...
Downloading separate debug info for /lib64/libm.so.6...
Downloading separate debug info for /lib64/libc.so.6...
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".
Число: 2
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): pow
Степень: 2
4.00
[Inferior 1 (process 36848) exited normally]
(gdb) □
```

# Выполнение лабораторной работы

Постранично просматриваю исходный код с помощью команды `list`.

Просматриваю строки с 12 по 15 основного файла с помощью команды `list` с параметрами:

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

Рис. 9: Просмотр исходного кода постранично и указанных строк.

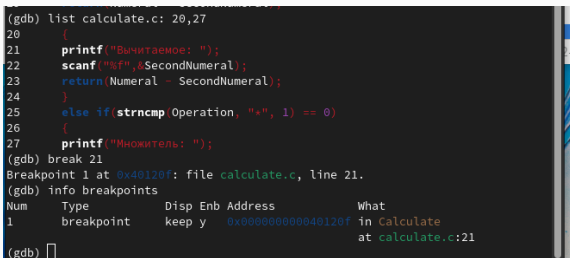
Просматриваю определённые строки не основного файла с помощью команды `list` с параметрами:

```
(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. 10: Просмотр определенных строк не основного файла.

# Выполнение лабораторной работы

Устанавливаю точку останова в файле `calculate.c` на строке номер 21 и вывожу информацию об имеющихся в проекте точках останова:

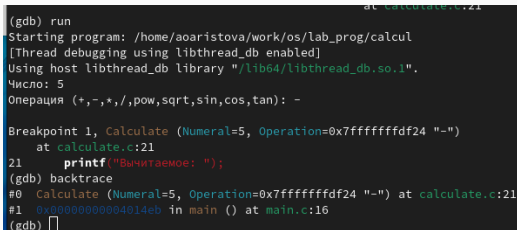


```
(gdb) list calculate.c: 20,27
20      {
21      printf("Вычитаемое: ");
22      scanf("%f",&SecondNumeral);
23      return(Numeral - SecondNumeral);
24      }
25      else if(strncmp(Operation, "+", 1) == 0)
26      {
27      printf("Множитель: ");
(gdb) break 21
Breakpoint 1 at 0x40120f: file calculate.c, line 21.
(gdb) info breakpoints
Num   Type             Disp Enb Address            What
1      breakpoint      keep y   0x0000000000040120f in calculate
                                           at calculate.c:21
(gdb) 
```

Рис. 11: Установка точки останова на строке 21 файла `calculate.c` и просмотр информации.

# Выполнение лабораторной работы

Запускаю программу внутри отладчика и убеждаюсь, что программа остановилась в момент прохождения точки останова, затем с помощью команды `backtrace` просматриваю весь стек вызываемых функций от начала программы до текущего места:



```
(gdb) run
Starting program: /home/aoaristova/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=0x7fffffffdf24 "-")
  at calculate.c:21
21  printf("Вычитаемое: ");
(gdb) backtrace
#0  Calculate (Numeral=5, Operation=0x7fffffffdf24 "-") at calculate.c:21
#1  0x00000000004014eb in main () at main.c:16
(gdb) █
```

Рис. 12: Проверка точки останова, просмотр стека вызываемых функций.

# Выполнение лабораторной работы

Просматриваю, чему равно значение переменной Numeral, введя сначала `print Numeral`, она равна 5, а затем сравниваю с выводом команды `display Numeral`. Можем заметить, что выводы разные, но значение одно - 5. Затем удаляю точку останова:

```
(gdb) print Numeral
$1 = 5
(gdb) display Numeral
1: Numeral = 5
(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) info breakpoints
No breakpoints or watchpoints.
(gdb) □
```

Рис. 13: Просмотр значения переменной Numeral и удаление точки останова.



6. С помощью заранее установленной утилиты splint анализирую коды файла `calculate.c` (вижу 15 предупреждений) и файла `main.c` (вижу 4 предупреждения)

# Выполнение лабораторной работы

```
[aoristova@fedora lab_prog]$ 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
    representations are inexact. Instead, compare the difference to FLT_EPSILON
    or DBL_EPSILON. (Use -realcompare to inhibit warning)
calculate.c:38:7: Return value type double does not match declared type float:
(HUGE_VAL)
    To allow all numeric types to match, use +relaxtypes.
calculate.c:46:1: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:47:7: Return value type double does not match declared type float:
(pow(Numeral, SecondNumeral))
calculate.c:50:7: Return value type double does not match declared type float:
(sqrt(Numeral))
calculate.c:52:7: Return value type double does not match declared type float:
(sin(Numeral))
calculate.c:54:7: Return value type double does not match declared type float:
(cos(Numeral))
calculate.c:56:7: Return value type double does not match declared type float:
(tan(Numeral))
calculate.c:60:7: Return value type double does not match declared type float:
(HUGE_VAL)

Finished checking --- 15 code warnings
[aoristova@fedora lab_prog]$
```

Рис. 14: Выполнение команды splint calculate.c.

# Выполнение лабораторной работы

```
[aoaristova@fedora lab_prog]$ 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:13:1: 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:15:12: 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:15:9: Corresponding format code
main.c:15:1: Return value (type int) ignored: scanf("%s", &Ope...

Finished checking --- 4 code warnings
```

Рис. 15: Выполнение команды splint main.c.

В ходе выполнения лабораторной работы я 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.