

# Лабораторная работа №13

Операционные системы

---

Дворкина Ева Владимировна

31 марта 2023

Российский университет дружбы народов, Москва, Россия

- Дворкина Ева Владимировна
- студентка из группы НКАбд-01-22
- Факультет физико-математических и естественных наук
- Российский университет дружбы народов
- 1132226447@rudn.ru
- <https://evdvorkina.github.io>



Цель данной лабораторной 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

## Выполнение лабораторной работы

---

Создаю директорию `~/work/os/lab_prog`, перехожу в нее и в этой директории создаю 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.

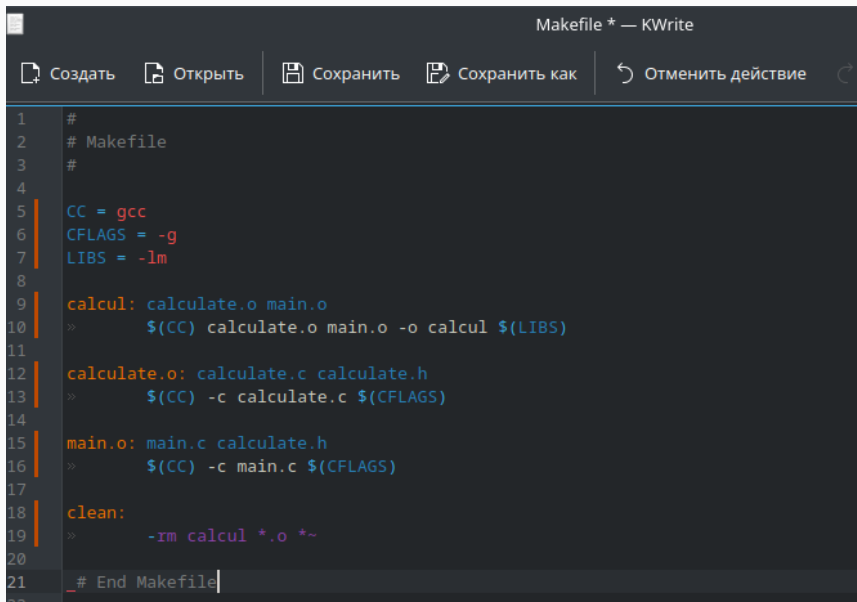
- Реализация функций калькулятора в файле `calculate.h`
- Интерфейсный файл `calculate.h`, описывающий формат вызова функции-калькулятора
- Основной файл `main.c`, реализующий интерфейс пользователя к калькулятору

Далее создаю Makefile. Исправляю Makefile:

Для использования GDB необходимо скомпилировать анализируемый код программы таким образом, чтобы отладочная информация содержалась в результирующем бинарном файле. Для этого следует воспользоваться опцией `-g` компилятора `gcc`

В переменную `CFLAGS` добавляю значение `-g`

## Выполнение лабораторной работы



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



Использую make

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

Рис. 2: Компиляция с помощью Makefile

## Выполнение лабораторной работы

Запускаю отладчик GDB, загрузив в него программу для отладки

```
[evdvorkina@evdvorkina 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...
```

## Выполнение лабораторной работы

Для запуска программы внутри отладчика ввожу команду run

```
(gdb) run
Starting program: /home/evdvorkina/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]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".
Число: 3
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): +
Второе слагаемое: 2
    5.00
[Inferior 1 (process 15004) exited normally]
```

Для постраничного (по 9 строк) просмотра исходного кода использую команду list

```
(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. 5: Постраничный просмотр кода

Для просмотра строк с 12 по 15 основного файла использую list с параметрами

```
(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. 6: Просмотр определенных строк кода

Устанавливаю точку остановки в файле calculate.c на строке номер 21 (break 21)

```
(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) █
```

Рис. 7: Установка точки остановки

Вывожу информацию об имеющихся в проекте точках остановки

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

Рис. 8: Информация об имеющихся точках остановки

## Выполнение лабораторной работы

Запускаю программу внутри отладчика и убеждаюсь, что программа останавливается в момент прохождения точки останова:

```
(gdb) run
Starting program: /home/evdvorkina/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=0x7fffffff934 "-") at calculate.c:21
21      printf("Вычитаемое: ");
(gdb) 
```

Рис. 9: Запуск программы



Команда `backtrace` покажет весь стек вызываемых функций от начала программы до текущего места

```
(gdb) backtrace
#0  Calculate (Numeral=5, Operation=0x7fffffff934 "-") at calculate.c:21
#1  0x000000004014eb in main () at main.c:16
(gdb) 
```

Рис. 10: Стек вызываемых функций

Смотрю, чему равно на этом этапе значение переменной Numeral, это можно сделать двумя способами. При первом я получу вывод значения переменной из bash-скрипта, второй способ более интуитивный, там значение соответствует переменной из кода на Си

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

Рис. 11: Просмотр промежуточного значения переменной Numeral

Удаляю точки останова

```
(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. 12: Удаление точек останова

С помощью утилиты splint пробую проанализировать код файла main.c.

```
[evdvorkina@evdvorkina 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
```

Рис. 13: Анализ кода файла main.c

При выполнении данной 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