

# **Лабораторная работа №13**

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

Демидова Екатерина Алексеевна

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>4</b>
<b>2</b>	<b>Задание</b>	<b>5</b>
<b>3</b>	<b>Теоретическое введение</b>	<b>6</b>
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>8</b>
<b>5</b>	<b>Выводы</b>	<b>16</b>
<b>6</b>	<b>Контрольные вопросы</b>	<b>17</b>
	<b>Список литературы</b>	<b>24</b>

## Список иллюстраций

4.1	Создание каталога и файлов . . . . .	8
4.2	Компиляция . . . . .	12
4.3	Запуск GDB . . . . .	13
4.4	Использование list . . . . .	14
4.5	Точка останова) . . . . .	14
4.6	Запуск программы с точкой останова . . . . .	14
4.7	Значение переменной Numeral . . . . .	14
4.8	Утилита splint . . . . .	15

# 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 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.
3. Выполните компиляцию программы посредством `gcc`.
4. При необходимости исправьте синтаксические ошибки.
5. Создайте `Makefile`.
6. С помощью `gdb` выполните отладку программы `calcul`.
7. С помощью утилиты `splint` попробуйте проанализировать коды файлов `calculate.c` и `main.c`.

### 3 Теоретическое введение

Отладчиком называется программа, которая выполняет внутри себя другую программу. Основное назначение отладчика - дать возможность пользователю в определенной степени осуществлять контроль за выполняемой программой, то есть определять, что происходит в процессе ее выполнения. Наиболее известным отладчиком для Linux является программа GNU GDB. GDB содержит множество полезных возможностей, но для простой отладки достаточно использовать лишь некоторые из них. Когда Вы запускаете программу, содержащую ошибки, обнаруживаемые лишь на стадии выполнения, есть несколько вопросов, на которые Вам нужно найти ответ.

- Какое выражение или оператор в программе вызывает ошибку?
- Если ошибка возникает в результате вызова функции, в каком месте программы происходит этот вызов?
- Какие значения содержат переменные и параметры программы в определенной точке ее выполнения?
- Что является результатом вычисления выражения в определенном месте программы?
- Каков действительный порядок выполнения операторов программы?

Эти действия требуют, чтобы пользователь отладчика был в состоянии

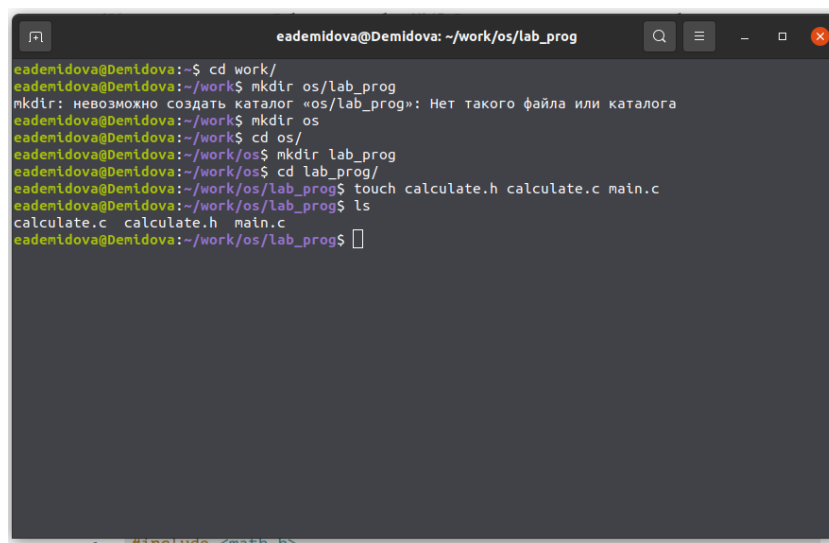
- проанализировать данные программы;
- получить трассу - список вызовов функций, которые были выполнены, с сортировкой, указывающей кто кого вызывал;

- установить точки останова, в которых выполнение программы приостанавливается, чтобы можно было проанализировать данные;
- выполнять программу по шагам, чтобы увидеть, что в действительности происходит.

GDB предоставляет все перечисленные возможности. Он называется отладчиком на уровне исходного текста, создавая иллюзию, что Вы выполняете операторы C++ из Вашей программы, а не машинный код, в который они действительно транслируются.[1]

## 4 Выполнение лабораторной работы

В домашнем каталоге создадим подкаталог `~/work/os/lab_prog`. Создадим 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. (рис. 4.1)



```
eademidova@Demidova: ~/work/os/lab_prog
eademidova@Demidova:~$ cd work/
eademidova@Demidova:~/work$ mkdir os/lab_prog
mkdir: невозможно создать каталог «os/lab_prog»: Нет такого файла или каталога
eademidova@Demidova:~/work$ mkdir os
eademidova@Demidova:~/work$ cd os/
eademidova@Demidova:~/work/os$ mkdir lab_prog
eademidova@Demidova:~/work/os$ cd lab_prog/
eademidova@Demidova:~/work/os/lab_prog$ touch calculate.h calculate.c main.c
eademidova@Demidova:~/work/os/lab_prog$ ls
calculate.c calculate.h main.c
eademidova@Demidova:~/work/os/lab_prog$
```

Рис. 4.1: Создание каталога и файлов

Реализация функций калькулятора в файле `calculate.h`:

```
////////////////////////////////////
// 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);
}
else if(strncmp(Operation, "/", 1) == 0)
{

```

```

printf("Делитель: ");
scanf("%f",&SecondNumeral);
if(SecondNumeral == 0)
{
printf("Ошибка: деление на ноль! ");
return(HUGE_VAL);
}
else
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);
}
}

```

```
}
```

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

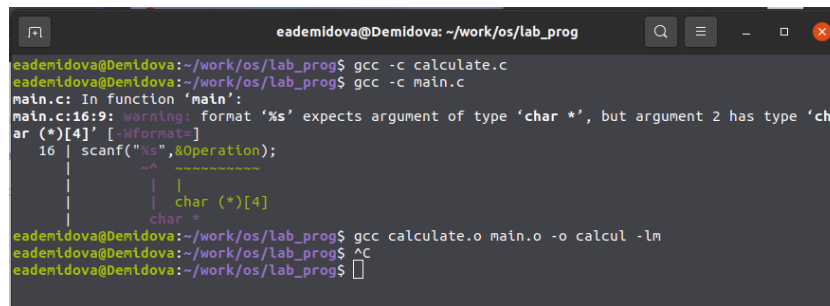
```
////////////////////////////////////  
// calculate.h  
  
#ifndef CALCULATE_H_  
#define CALCULATE_H_  
  
float Calculate(float Numeral, char Operation[4]);  
#endif /*CALCULATE_H_*/
```

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

```
////////////////////////////////////  
// 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;
}
```

Выполним компиляцию программы посредством gcc (рис. 4.2)



```
eademidova@Demidova: ~/work/os/lab_prog
eademidova@Demidova:~/work/os/lab_prog$ gcc -c calculate.c
eademidova@Demidova:~/work/os/lab_prog$ gcc -c main.c
main.c: In function 'main':
main.c:16:9: warning: format '%s' expects argument of type 'char *', but argument 2 has type 'char (*)[4]' [-Wformat=]
   16 | scanf("%s", &Operation);
      |          ^
      |          |
      |          | char (*)[4]
      |          |
      |          char *
eademidova@Demidova:~/work/os/lab_prog$ gcc calculate.o main.o -o calcul -lm
eademidova@Demidova:~/work/os/lab_prog$ ^C
eademidova@Demidova:~/work/os/lab_prog$
```

Рис. 4.2: Компиляция

Затем исправим ошибку в файле main.c, уберем амперсанта перед считыванием Operation.

Создадим Makefile со следующим содержанием:

```
#
# 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
```

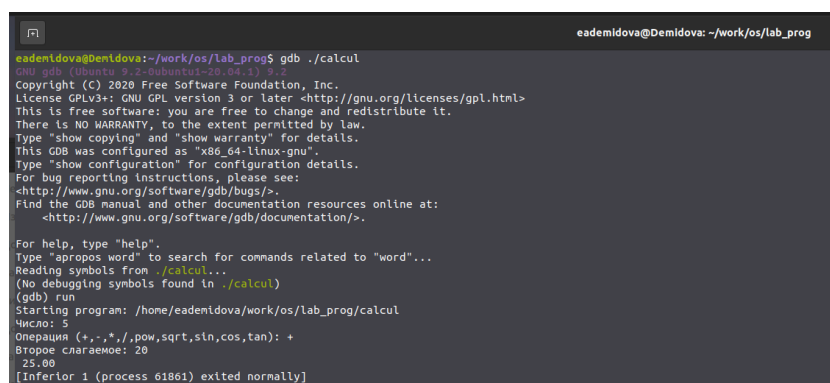
```
gcc -c main.c $(CFLAGS)
```

```
clean:
```

```
-rm calcul *.o *~
```

```
# End Makefile
```

С помощью gdb выполним отладку программы calcul. Запустим отладчик GDB, загрузив в него программу для отладки с помощью команды `gdb ./calcul`. Для запуска программы внутри отладчика введём команду `run`. (рис. 4.3)



```
eademidova@Demidova: ~/work/os/lab_prog$ gdb ./calcul
GNU gdb (Ubuntu 7.12-2015.08-2ubuntu1) 7.12.1
Copyright (C) 2015 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-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://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/eademidova/work/os/lab_prog/calcul
Число: 5
Операция (+, -, *, /, pow, sqrt, sin, cos, tan): +
Второе слагаемое: 20
25.00
[Inferior 1 (process 61861) exited normally]
```

Рис. 4.3: Запуск GDB

Для постраничного (по 9 строк) просмотра исходного код используем команду `list`. Для просмотра строк с 12 по 15 основного файла используем `list` с параметрами. Для просмотра определённых строк не основного файла используем `list` с параметрами. Установим точку останова в файле `calculate.c` на строке номер 12. Выведем информацию об имеющихся в проекте точка останова. (рис. 4.4, 4.5))

```

Reading symbols from ./calcul...
(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) list 12,15
12     float Result;
13     printf("Число: ");
14     scanf("%f",&Numeral);
15     printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): ");
(gdb) list calculate.c:20-23

```

Рис. 4.4: Использование list

```

(gdb) list main.c:4,7
4     #include <stdio.h>
5     #include "calculate.h"
6
7     int
(gdb) list main.c:4,7
4     #include <stdio.h>
5     #include "calculate.h"
6
7     int
(gdb) break 12
Breakpoint 1 at 0x1558: file main.c, line 13.
(gdb) info breakpoints
Num Type             Disp Enb Address          What
1 breakpoint         keep y  0x0000000000001558 in main at main.c:13
(gdb)

```

Рис. 4.5: Точка останова)

Запустим программу внутри отладчика и убедимся, что программа остановится в момент прохождения точки останова. (рис. 4.6)

```

1 breakpoint keep y 0x0000000000001558 in main at main.c:13
(gdb) run
Starting program: /home/eadenidova/work/os/lab_prog/calcul
Breakpoint 1, main () at main.c:13
13     printf("Число: ");
(gdb) backtrace
#0 main () at main.c:13
(gdb)

```

Рис. 4.6: Запуск программы с точкой останова

Посмотрим, чему равно на этом этапе значение переменной Numeral. Сравним с результатом вывода на экран, они равны. Уберём точки останова. (рис. 4.7)

```

#0 main () at main.c:13
(gdb) print Numeral
$2 = 3.0611365e-41
(gdb) display Numeral
1: Numeral = 3.0611365e-41
(gdb) info breakpoints
Num Type             Disp Enb Address          What
1 breakpoint         keep y  0x0000555555555558 in main at main.c:13
breakpoint already hit 1 time
(gdb) delete 1
(gdb)

```

Рис. 4.7: Значение переменной Numeral

С помощью утилиты splint попробуем проанализировать коды файлов calculate.c и main.c.(рис. 4.8)



## 5 Выводы

В результате выполнения лабораторной работы 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.



## 6 Контрольные вопросы

1. Как получить информацию о возможностях программ gcc, make, gdb и др.?

Дополнительную информацию о этих программах можно получить с помощью функций info и man.

2. Назовите и дайте краткую характеристику основным этапам разработки приложений в UNIX.

Unix поддерживает следующие основные этапы разработки приложений:

- создание исходного кода программы;
  - представляется в виде файла;
  - сохранение различных вариантов исходного текста;
  - анализ исходного текста; Необходимо отслеживать изменения исходного кода, а также при работе более двух программистов над проектом программы нужно, чтобы они не делали изменений кода в одно время.
- к омпияция исходного текста и построение исполняемого модуля;
- тестирование и отладка;
  - проверка кода на наличие ошибок
  - сохранение всех изменений, выполняемых при тестировании и отладке.

3. Что такое суффикс в контексте языка программирования? Приведите примеры ис- пользования.

Использование суффикса “.с” для имени файла с программой на языке Си отражает удобное и полезное соглашение, принятое в ОС UNIX. Для любого имени входного файла суффикс определяет какая компиляция требуется. Суффиксы и префиксы указывают тип объекта. Одно из полезных свойств компилятора Си — его способность по суффиксам определять типы файлов. По суффиксу .с компилятор распознает, что файл abcd.c должен компилироваться, а по суффиксу .о, что файл abcd.o является объектным модулем и для получения исполняемой программы необходимо выполнить редактирование связей. Простейший пример командной строки для компиляции программы abcd.c и построения исполняемого модуля abcd имеет вид: `gcc -o abcd abcd.c`. Некоторые проекты предпочитают показывать префиксы в начале текста изменений для старых (old) и новых (new) файлов. Опция `-prefix` может быть использована для установки такого префикса. Плюс к этому команда `bzr diff -p1` выводит префиксы в форме которая подходит для команды `patch -p1`.

4. Каково основное назначение компилятора языка C в UNIX?

Основное назначение компилятора с языка Си заключается в компиляции всей программы в целом и получении исполняемого модуля.

5. Для чего предназначена утилита make?

При разработке большой программы, состоящей из нескольких исходных файлов заголовков, приходится постоянно следить за файлами, которые требуют перекомпиляции после внесения изменений. Программа make освобождает пользователя от такой рутинной работы и служит для документирования взаимосвязей между файлами. Описание взаимосвязей и соответствующих действий хранится в так называемом make-файле, который по умолчанию имеет имя `makefile` или `Makefile`.

6. Приведите пример структуры Makefile. Дайте характеристику основным элементам этого файла.

makefile для программы abcd.c мог бы иметь вид:

```
#
```

```
#
```

```
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
```

В общем случае make-файл содержит последовательность записей (строк), определяющих зависимости между файлами. Первая строка записи представляет собой список целевых (зависимых) файлов, разделенных пробелами, за которыми следует двоеточие и список файлов, от которых зависят целевые. Текст, следующий за точкой с запятой, и все последующие строки, начинающиеся с литеры табуляции, являются командами ОС UNIX, которые необходимо выполнить для обновления целевого файла. Таким образом, спецификация взаимосвязей имеет формат: `target1 [ target2...]: [:] [dependment1...] [(tab)commands] [#commentary] [(tab)commands] [#commentary]`, где `#` — специфицирует начало комментария, так как содержимое строки, начиная с `#` и до конца строки, не будет обрабатываться командой `make`; `:` — последовательность команд ОС UNIX должна содержаться в одной строке make-файла (файла описаний), есть возможность переноса команд `()`, но она считается как одна строка; `::` — последовательность команд ОС UNIX может содержаться в нескольких последовательных строках файла описаний. Приведённый выше make-файл для программы `abcd.c` включает два способа компиляции и построения исполняемого модуля. Первый способ предусматривает обычную компиляцию с построением исполняемого модуля с именем `abcd`. Вторым способом позволяет включать в исполняемый модуль `testabcd` возможность выполнить процесс отладки на уровне исходного текста.

7. Назовите основное свойство, присущее всем программам отладки. Что необходимо сделать, чтобы его можно было использовать?

Пошаговая отладка программ заключается в том, что выполняется один оператор программы и, затем контролируются те переменные, на которые должен

был воздействовать данный оператор. Если в программе имеются уже отлаженные подпрограммы, то подпрограмму можно рассматривать, как один оператор программы и воспользоваться вторым способом отладки программ. Если в программе существует достаточно большой участок программы, уже отлаженный ранее, то его можно выполнить, не контролируя переменные, на которые он воздействует. Использование точек останова позволяет пропускать уже отлаженную часть программы. Точка останова устанавливается в местах, где необходимо проверить содержимое переменных или просто проконтролировать, передаётся ли управление данному оператору. Практически во всех отладчиках поддерживается это свойство (а также выполнение программы до курсора и выход из подпрограммы). Затем отладка программы продолжается в пошаговом режиме с контролем локальных и глобальных переменных, а также внутренних регистров микроконтроллера и напряжений на выводах этой микросхемы.

8. Назовите и дайте основную характеристику основным командам отладчика gdb.

- `backtrace` – выводит весь путь к текущей точке останова, то есть названия всех функций, начиная от `main()`; иными словами, выводит весь стек функций;
- `break` – устанавливает точку останова; параметром может быть номер строки или название функции;
- `clear` – удаляет все точки останова на текущем уровне стека (то есть в текущей функции);
- `continue` – продолжает выполнение программы от текущей точки до конца;
- `delete` – удаляет точку останова или контрольное выражение;
- `display` – добавляет выражение в список выражений, значения которых отображаются каждый раз при остановке программы;

- `finish` – выполняет программу до выхода из текущей функции; отображает возвращаемое значение, если такое имеется;
  - `info breakpoints` – выводит список всех имеющихся точек останова;
  - `info watchpoints` – выводит список всех имеющихся контрольных выражений;
  - `splist` – выводит исходный код; в качестве параметра передаются название файла исходного кода, затем, через двоеточие, номер начальной и конечной строки;
  - `nex` – пошаговое выполнение программы, но, в отличие от команды `step`, не выполняет пошагово вызываемые функции;
  - `print` – выводит значение какого-либо выражения (выражение передаётся в качестве параметра);
  - `run` – запускает программу на выполнение;
  - `set` – устанавливает новое значение переменной
  - `step` – пошаговое выполнение программы;
  - `watch` – устанавливает контрольное выражение, программа остановится, как только значение контрольного выражения изменится;
9. Опишите по шагам схему отладки программы, которую Вы использовали при выполнении лабораторной работы.

- 1) Выполняли компиляцию программы 2) Увидели ошибки в программе
- 2) Открыли редактор и исправили программу
- 3) Загрузили программу в отладчик `gdb`
- 4) `run` — отладчик выполнил программу, мы ввели требуемые значения.
- 5) программа завершена, `gdb` не видит ошибок.

10. Прокомментируйте реакцию компилятора на синтаксические ошибки в программе при его первом запуске.

Отладчику не понравился формат %s для &Operation, т.к %s — символьный формат, а значит необходим только Operation.

11. Назовите основные средства, повышающие понимание исходного кода программы.

Если вы работаете с исходным кодом, который не вами разрабатывался, то назначение различных конструкций может быть не совсем понятным. Система разработки приложений UNIX предоставляет различные средства, повышающие понимание исходного кода. К ним относятся:

- cscope - исследование функций, содержащихся в программе;
- splint — критическая проверка программ, написанных на языке Си.

12. Каковы основные задачи, решаемые программой splint?

13. Проверка корректности задания аргументов всех исполняемых функций , а также типов возвращаемых ими значений;

14. Поиск фрагментов исходного текста, корректных с точки зрения синтаксиса языка Си, но малоэффективных с точки зрения их реализации или содержащих в себе семантические ошибки;

15. Общая оценка мобильности пользовательской программы.

## Список литературы

1. Отладчик GDB [Электронный ресурс]. Maxim Chirkov, 2002. URL: [https://www.opennet.ru/docs/RUS/linux\\_base/node199.html](https://www.opennet.ru/docs/RUS/linux_base/node199.html).