

Лабораторная работа №13

Операционные системы

Серёгина Ирина Андреевна

Содержание

1	Цель работы	3
2	Задание	4
3	Теоретическое введение	5
4	Выполнение лабораторной работы	6
5	Выводы	15
	Список литературы	16

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

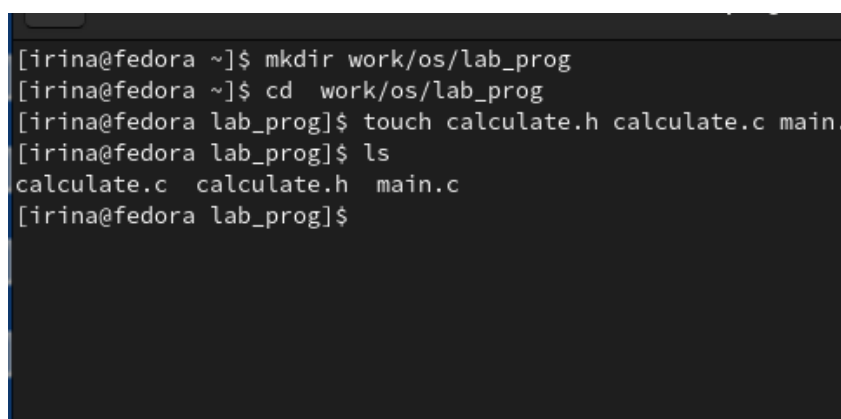
1. В домашнем каталоге создайте подкаталог `~/work/os/lab_prog`.
2. Создайте в нём файлы: `calculate.h`, `calculate.c`, `main.c`.
3. Выполните компиляцию программы посредством `gcc`.
4. При необходимости исправьте синтаксические ошибки.
5. Создайте `Makefile`.
6. С помощью `gdb` выполните отладку программы `calcul` (перед использованием `gdb` исправьте `Makefile`).
7. С помощью утилиты `splint` попробуйте проанализировать коды файлов `calculate.c` и `main.c`.

3 Теоретическое введение

Стандартным средством для компиляции программ в ОС типа UNIX является GCC (GNU Compiler Collection). Это набор компиляторов для разного рода языков программирования (C, C++, Java, Фортран и др.). Работа с GCC производится при помощи одноимённой управляющей программы gcc, которая интерпретирует аргументы командной строки, определяет и осуществляет запуск нужного компилятора для входного файла. Файлы с расширением (суффиксом) .c воспринимаются gcc как программы на языке C, файлы с расширением .cc или .C — как файлы на языке C++, а файлы с расширением .o считаются объектными. Во время работы над кодом программы программист неизбежно сталкивается с появлением ошибок в ней. Использование отладчика для поиска и устранения ошибок в программе существенно облегчает жизнь программиста. В комплект программ GNU для ОС типа UNIX входит отладчик GDB (GNU Debugger). Ещё одним средством проверки исходных кодов программ, написанных на языке C, является утилита splint. Эта утилита анализирует программный код, проверяет корректность задания аргументов использованных в программе функций и типов возвращаемых значений, обнаруживает синтаксические и семантические ошибки. В отличие от компилятора C анализатор splint генерирует комментарии с описанием разбора кода программы и осуществляет общий контроль, обнаруживая такие ошибки, как одинаковые объекты, определённые в разных файлах, или объекты, чьи значения не используются в работе программы, переменные с некорректно заданными значениями и типами и многое другое.

4 Выполнение лабораторной работы

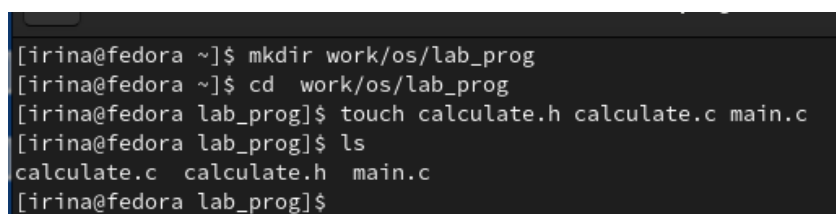
1. В домашнем каталоге создаю подкаталог ~/work/os/lab_prog (рис. fig. 4.1).



```
[irina@fedora ~]$ mkdir work/os/lab_prog
[irina@fedora ~]$ cd work/os/lab_prog
[irina@fedora lab_prog]$ touch calculate.h calculate.c main.c
[irina@fedora lab_prog]$ ls
calculate.c calculate.h main.c
[irina@fedora lab_prog]$
```

Рис. 4.1: создаю подкаталог

2. Создаю в нём файлы: calculate.h, calculate.c, main.c. (рис. fig. 4.2).



```
[irina@fedora ~]$ mkdir work/os/lab_prog
[irina@fedora ~]$ cd work/os/lab_prog
[irina@fedora lab_prog]$ touch calculate.h calculate.c main.c
[irina@fedora lab_prog]$ ls
calculate.c calculate.h main.c
[irina@fedora lab_prog]$
```

Рис. 4.2: создаю нужные файлы

3. Реализация функций калькулятора в файле calculate.h (рис. fig. 4.3).



```

Открыть ▾ ⊕ calculate.c
~/work/os/lab_prog

////////////////////////////////////
// calculate.c

#include <stdio.h>
#include <math.h>
#include <string.h>
#include "calculate.h"

float
Calculate(float Numeral, char Operation[4])
{
    float SecondNumeral;
    if(strcmp(Operation, "+", 1) == 0)
    {
        printf("Второе слагаемое: ");
        scanf("%f",&SecondNumeral);
        return(Numeral + SecondNumeral);
    }
    else if(strcmp(Operation, "-", 1) == 0)
    {

```

Рис. 4.3: реализация функций

4. Интерфейсный файл calculate.h, описывающий формат вызова функции калькулятора (рис. fig. 4.4).



```

Открыть ▾ ⊕ • calculate.h
~/work/os/lab_prog

calculate.c calculate.h

////////////////////////////////////
// calculate.h

#ifndef CALCULATE_H_
#define CALCULATE_H_

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

#endif /*CALCULATE_H_*/

```

Рис. 4.4: интерфейсный файл

5. Основной файл main.c, реализующий интерфейс пользователя к калькулятору (рис. fig. 4.5).



```
main.c
~/work/os/lab_prog

calculate.c | calculate.h | main.c

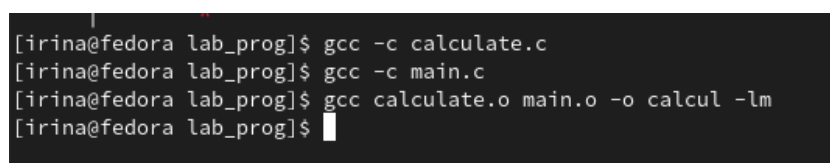
////////////////////////////////////
// main.c

#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("%6.2f\n",Result);
    return 0;
}
```

Рис. 4.5: основной файл

6. Выполняю компиляцию программы посредством gcc (рис. fig. 4.6).



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

Рис. 4.6: компиляция программы

7. Создаю Makefile, исправляю ошибки (рис. fig. 4.7).

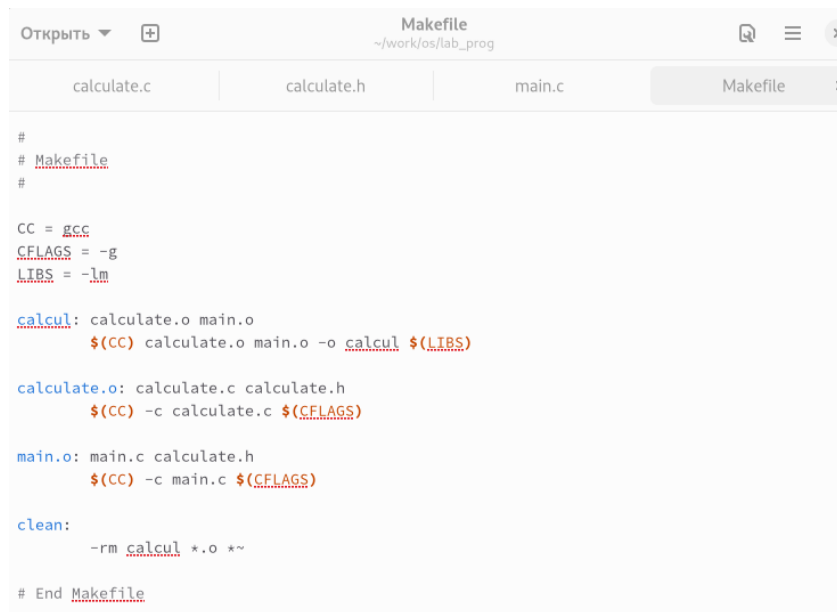


Рис. 4.7: Создаю Makefile

8. Выполняю команду make (рис. fig. 4.8).

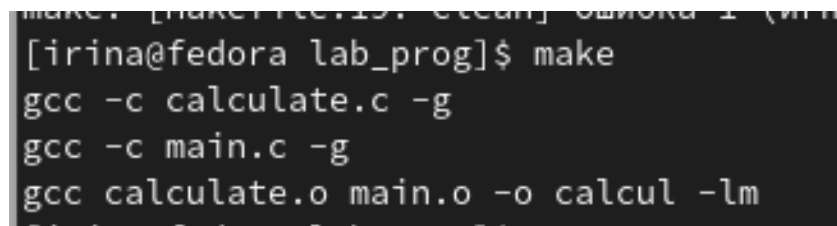


Рис. 4.8: команда make

9. Запускаю отладчик GDB, загрузив в него программу для отладки (рис. fig. 4.9).

```

[irina@fedora lab_prog]$ gdb ./calcul
GNU gdb (GDB) Fedora 12.1-2.fc36
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...

```

Рис. 4.9: Запускаю отладчик GDB

10. Для запуска программы внутри отладчика ввожу команду run (рис. fig. 4.10).

```

(gdb) run
Starting program: /home/irina/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.

Downloading 0.01 MB separate debug info for system-supplied DSO at 0x7ffff7fc4000
0
Downloading 2.25 MB separate debug info for /lib64/libm.so.6
Downloading 7.42 MB 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: 4
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): -
Вычитаемое: 3
1.00
[Inferior 1 (process 4258) exited normally]
(gdb)

```

Рис. 4.10: команда run

11. Для постраничного (по 9 строк) просмотра исходного код использую команду list (рис. fig. 4.11).

```

[Inferior 1 (process 4258) 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. 4.11: использую команду list

12. Для просмотра строк с 12 по 15 основного файла использую команду list(рис. fig. 4.12).

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

Рис. 4.12: использую команду list с параметрами

13. Для просмотра определённых строк не основного файла использую list с параметрами (рис. fig. 4.13).

```
(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. 4.13: использую list с параметрами

14. Устанавливаю точку останова в файле calculate.c на строке номер 21 (рис. fig. 4.14).

```
(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. 4.14: Устанавливаю точку останова

15. Вывожу информацию об имеющихся в проекте точках останова (рис. fig. 4.15).

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

Рис. 4.15: вывод информации точек останова

16. Запускаю программу внутри отладчика и убеждаюсь, что программа останавливается в момент прохождения точки останова (рис. fig. 4.16).

```
(gdb) run
Starting program: /home/irina/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=0x7fffffffdfb4 "-") at calculate.c:21
21      printf("Вычитаемое: ");
(gdb) backtrace
#0 Calculate (Numeral=5, Operation=0x7fffffffdfb4 "-") at calculate.c:21
#1 0x00000000004014eb in main () at main.c:17
(gdb)
```

Рис. 4.16: запуск программы

17. Смотрю, чему равно на этом этапе значение переменной Numeral, сравниваю с результатом вывода на экран после использования другой команды (рис. fig. 4.17).

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

Рис. 4.17: сравниваю значения переменной

18. Убираю точки останова (рис. fig. 4.18).

```
(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. 4.18: Убираю точки останова

19. С помощью утилиты splint анализирую коды файла calculate.c (рис. fig. 4.19).

```
[irina@fedora lab_prog]$ splint calculate.c
Splint 3.1.2 --- 22 Jan 2022

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:16:7: 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:22:7: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:28:8: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:34:8: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:35:11: 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. 4.19: анализирую коды файла calculate.c

20. С помощью утилиты splint попробуйте анализирую коды файла main.c. (рис. fig. 4.20).

```

[irina@fedora lab_prog]$ splint main.c
Splint 3.1.2 --- 22 Jan 2022

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:14:3: 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:16:14: 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:16:11: Corresponding format code
main.c:16:3: Return value (type int) ignored: scanf("%s", &Ope...

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

```

Рис. 4.20: анализирую коды файла main.c.

5 Выводы

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.

Список литературы