

Лабораторная работа №13

Операционные системы

Шуваев Сергей Александрович.

11 июля 1985 года

Российский университет дружбы народов, Москва, Россия

Информация

- Шуваев Сергей Александрович.
- студент из группы НКАбд-05-22
- Факультет физико-математических и естественных наук
- Российский университет дружбы народов
- 1032224269@pfur.ru
- <https://github.com/Grinders060050/Grinders060050.github.io>



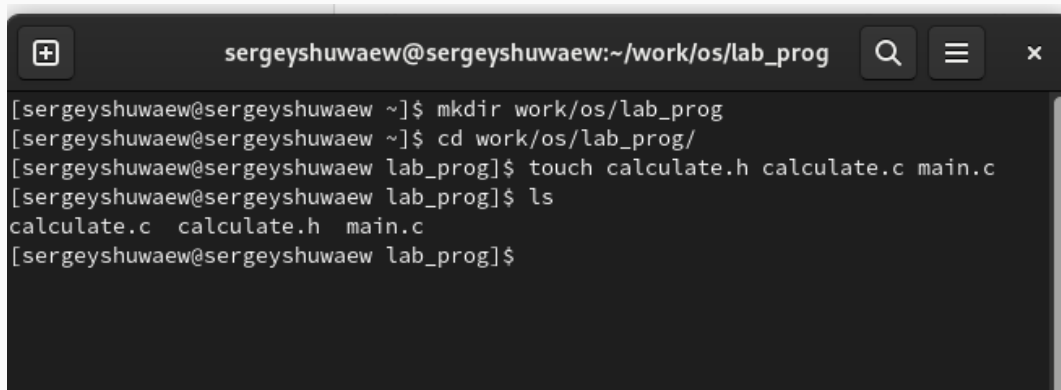
Цель работы

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

Выполнение лабораторной работы

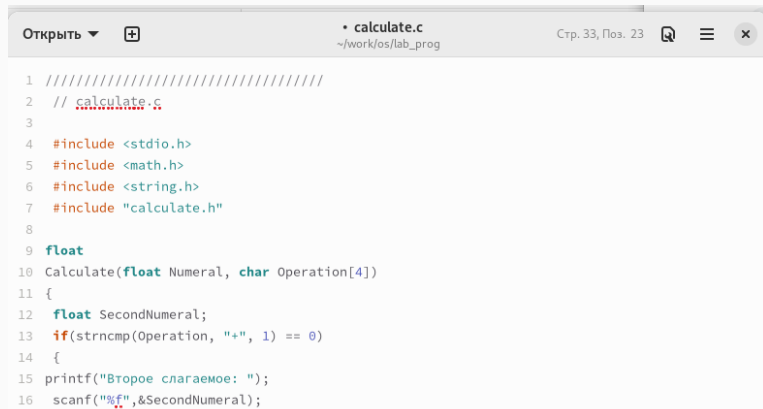
Создаю директорию `~/work/os/lab_prog`, перехожу в нее и в этой директории создаю файлы `calculate.h`, `calculate.c`, `main.c`

A terminal window with a dark background. The title bar shows the username 'sergeyshuwaew' and the current directory '~/work/os/lab_prog'. The terminal contains the following commands and output:

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

Выполнение лабораторной работы

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`



```
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);
```


Выполнение лабораторной работы

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



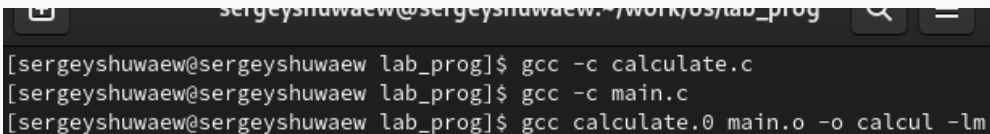
```
1  //////////////////////////////////////
2  // calculate.h
3
4  #ifndef CALCULATE_H_
5  #define CALCULATE_H_
6
7  float Calculate(float Numeral, char Operation[4]);
8
9  #endif /*CALCULATE_H_*/
```

Выполнение лабораторной работы

Основной файл main.c, реализующий интерфейс пользователя к калькулятору

```
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];
11     float Result;
12     printf("Число: ");
13     scanf("%f",&Numeral);
14     printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): ");
15     scanf("%s",&Operation);
16     Result = Calculate(Numeral, Operation);
17     printf("%6.2f\n",Result);
```

Далее выполняю компиляцию программ посредством gcc (

A screenshot of a terminal window with a dark background. The window title is 'sergeyshuwaew@sergeyshuwaew: ~/work/os/lab_prog'. It contains three lines of terminal output, each starting with a prompt '[sergeyshuwaew@sergeyshuwaew lab_prog]\$'. The commands are: 'gcc -c calculate.c', 'gcc -c main.c', and 'gcc calculate.o main.o -o calcul -lm'.

```
sergeyshuwaew@sergeyshuwaew: ~/work/os/lab_prog  
[sergeyshuwaew@sergeyshuwaew lab_prog]$ gcc -c calculate.c  
[sergeyshuwaew@sergeyshuwaew lab_prog]$ gcc -c main.c  
[sergeyshuwaew@sergeyshuwaew lab_prog]$ gcc calculate.o main.o -o calcul -lm
```

Выполнение лабораторной работы

Далее создаю Makefile с указанным в лабораторной работе содержанием

```
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
```

Выполнение лабораторной работы

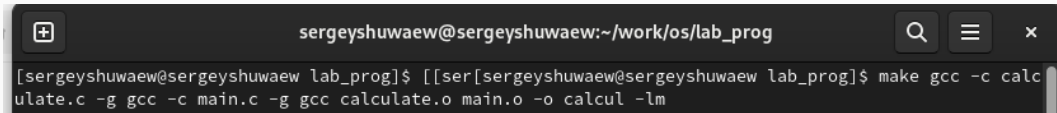
В переменную CFLAGS добавляю значение -g



The screenshot shows a code editor window titled "• Makefile" with the path "~/.work/os/lab_prog". The editor contains a Makefile with the following content:

```
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
22
```

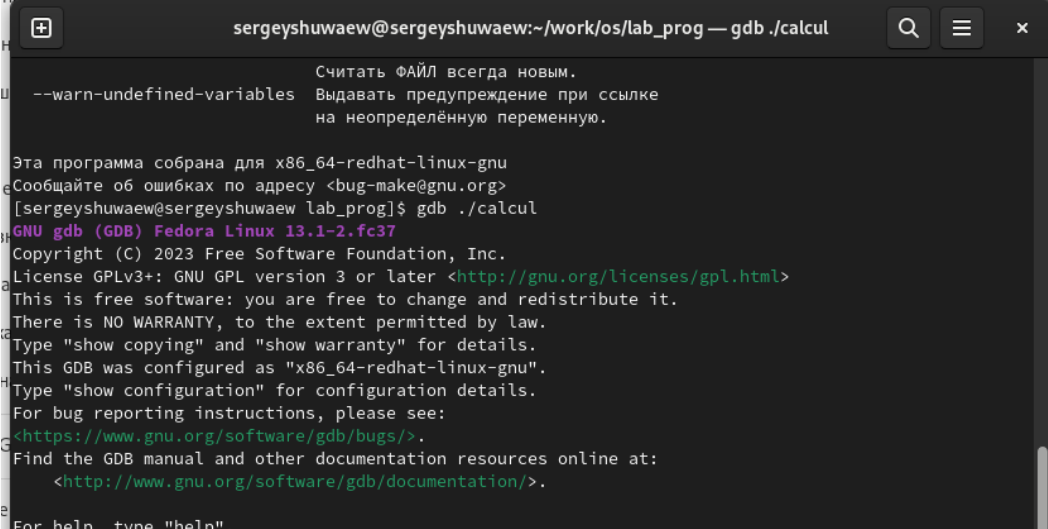
Далее использую `make clean`, чтобы удалить не совсем верно скомпилированные файлы, и использую `make`

A terminal window with a dark background. The title bar shows a plus icon, the username and host 'sergeyshuwaew@sergeyshuwaew', and the current directory '~/work/os/lab_prog'. On the right are search, menu, and close icons. The terminal text shows the prompt '[sergeyshuwaew@sergeyshuwaew lab_prog]\$' followed by the command 'make gcc -c calculate.c -g gcc -c main.c -g gcc calculate.o main.o -o calcul -lm'.

```
sergeyshuwaew@sergeyshuwaew:~/work/os/lab_prog  
[sergeyshuwaew@sergeyshuwaew lab_prog]$ make gcc -c calculate.c -g gcc -c main.c -g gcc calculate.o main.o -o calcul -lm
```

Выполнение лабораторной работы

Запускаю отладчик GDB, загрузив в него программу для отладки

A screenshot of a terminal window with a dark background. The title bar at the top shows the username 'sergeyshuwaew' and the current directory and command '~/work/os/lab_prog — gdb ./calcul'. The terminal output displays the GDB startup sequence, including the 'GNU gdb (GDB) Fedora Linux 13.1-2.fc37' banner, copyright information for the Free Software Foundation, and various license details. The output is in Russian, mentioning file handling and warning options. The prompt '[sergeyshuwaew@sergeyshuwaew lab_prog]\$' is visible before the GDB banner.

```
sergeyshuwaew@sergeyshuwaew:~/work/os/lab_prog — gdb ./calcul

Считать ФАЙЛ всегда новым.
--warn-undefined-variables Выдавать предупреждение при ссылке
на неопределённую переменную.

Эта программа собрана для x86_64-redhat-linux-gnu
Сообщайте об ошибках по адресу <bug-make@gnu.org>
[sergeyshuwaew@sergeyshuwaew lab_prog]$ gdb ./calcul
GNU gdb (GDB) Fedora Linux 13.1-2.fc37
Copyright (C) 2023 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"
```

Для запуска программы внутри отладчика ввожу команду run

```
<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"...
./calcul: Нет такого файла или каталога.
(gdb) run
Starting program:
No executable file specified.
Use the "file" or "exec-file" command.
(gdb) █
```


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) █
```

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) █
```

Для просмотра определённых строк не основного файла использую 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) █
```

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) 
```

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) {#
```

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

```
[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) |
```

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)
```

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) 
```


Удаляю точки останова

```
(gdb) info breakpoints
Num      Type           Disp Enb Address            What
1        breakpoint     keep y   0x0000000000040120f in Calculate at calculate.c:21
          breakpoint already hit 1 time
(gdb) delete 1
(gdb) info breakpoints
No breakpoints or watchpoints.
(gdb) 
```

Выполнение лабораторной работы

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

```
[evdvorkina@evdvorkina lab_prog]$ splint calculate.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)
calculate.c:10:31: Function parameter Operation declared as manifest array
        (size constant is meaningless)
calculate.c: (in function Calculate)
calculate.c:16:2: 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:22:2: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:28:2: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:34:2: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:35:5: 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:38:8: Return value type double does not match declared type float:
        (HUGE_VAL)
    To allow all numeric types to match, use +relaxtypes.
calculate.c:46:2: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:47:8: Return value type double does not match declared type float:
        (pow(Numeral, SecondNumeral))
calculate.c:50:8: Return value type double does not match declared type float:
        (sqrt(Numeral))
calculate.c:52:8: Return value type double does not match declared type float:
        (sin(Numeral))
calculate.c:54:8: Return value type double does not match declared type float:
        (cos(Numeral))
calculate.c:56:8: Return value type double does not match declared type float:
```

Выполнение лабораторной работы

С помощью утилиты splint пробую проанализировать код файла 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
```

Выводы

При выполнении данной лабораторной работы я изучил основы программирования в оболочке ОС UNIX, научился писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

Спасибо за внимание