

Отчёт по лабораторной работе №13

Дисциплина: Операционные системы

Батова Ирина Сергеевна, НММбд-01-22

Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	8
4	Выводы	20
5	Контрольные вопросы	21

Список иллюстраций

3.1	Создание подкаталога	8
3.2	Создание файлов	8
3.3	Скрипт в calculate.c	9
3.4	Скрипт в calculate.c	10
3.5	Скрипт в calculate.h	10
3.6	Скрипт в main.c	11
3.7	Компиляция программы	11
3.8	Создание makefile	12
3.9	Редактирование makefile	13
3.10	Проверка makefile	13
3.11	Запуск отладчика	14
3.12	Запуск программы внутри отладчика	14
3.13	Постраничный просмотр исходного кода	15
3.14	Просмотр определенных строк основного файла	15
3.15	Просмотр определенных строк не основного файла	15
3.16	Установка точки останова	16
3.17	Информация о точках останова	16
3.18	Запуск программы с точкой останова	16
3.19	Проверка значения Numeral командой print	17
3.20	Проверка значения Numeral командой display	17
3.21	Удаление точек останова	17
3.22	Команда 'splint calculate.c'	18
3.23	Команда 'splint calculate.c'	18
3.24	Команда 'splint main.c'	19
5.1	Пример файла makefile	22

Список таблиц

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

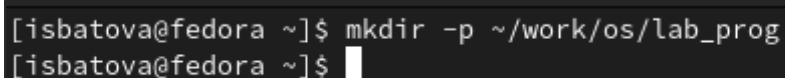
1. В домашнем каталоге создайте подкаталог `~/work/os/lab_prog`.
2. Создайте в нём файлы: `calculate.h`, `calculate.c`, `main.c`. В соответствие с лабораторной работы внести скрипт в файлы.
 - Реализация функций калькулятора в файле `calculate.h`
 - Интерфейсный файл `calculate.h`, описывающий формат вызова функции-калькулятора
 - Основной файл `main.c`, реализующий интерфейс пользователя к калькулятору
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`):
 - 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 точках останова – Запустите программу внутри отладчика и убедитесь, что программа остановится в момент прохождения точки останова – Посмотрите, чему равно на этом этапе значение переменной `Numeral` – Сравните с результатом вывода на экран после использования другой команды – Уберите точки останова

7. С помощью утилиты `splint` попробуйте проанализировать коды файлов `calculate.c` и `main.c`

3 Выполнение лабораторной работы

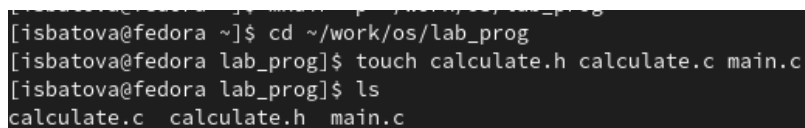
1. В домашнем каталоге создаем подкаталог `~/work/os/lab_prog` (рис. 3.1).



```
[isbatova@fedora ~]$ mkdir -p ~/work/os/lab_prog  
[isbatova@fedora ~]$
```

Рис. 3.1: Создание подкаталога

2. Создаем в нём файлы: `calculate.h`, `calculate.c`, `main.c` (рис. 3.2).



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

Рис. 3.2: Создание файлов

Далее вносим в файлы скрипты соответственно лабораторной работе для создания примитивного калькулятора.

Вводим скрипт для реализации функций калькулятора в файле `calculate.c` (рис. 3.3, 3.4).


```

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);
17         return(Numeral + SecondNumeral);
18     }
19     else if(strncmp(Operation, "-", 1) == 0)
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);
30     }
31     else if(strncmp(Operation, "/", 1) == 0)
32     {
33         printf("Делитель: ");
34         scanf("%f", &SecondNumeral);
35         if(SecondNumeral == 0)
36     {

```

Рис. 3.3: Скрипт в calculate.c

```

37 printf("Ошибка: деление на ноль! ");
38 return(HUGE_VAL);
39 }
40 else
41 return(Numeral / SecondNumeral);
42 }
43 else if(strncmp(Operation, "pow", 3) == 0)
44 {
45 printf("Степень: ");
46 scanf("%f",&SecondNumeral);
47 return(pow(Numeral, SecondNumeral));
48 }
49 else if(strncmp(Operation, "sqrt", 4) == 0)
50 return(sqrt(Numeral));
51 else if(strncmp(Operation, "sin", 3) == 0)
52 return(sin(Numeral));
53 else if(strncmp(Operation, "cos", 3) == 0)
54 return(cos(Numeral));
55 else if(strncmp(Operation, "tan", 3) == 0)
56 return(tan(Numeral));
57 else
58 {
59 printf("Неправильно введено действие ");
60 return(HUGE_VAL);
61 }
62 }

```

Рис. 3.4: Скрипт в calculate.c

Вводим скрипт в интерфейсный файл calculate.h, описывающий формат вызова функции-калькулятора (рис. 3.5).

```

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_*/

```

Рис. 3.5: Скрипт в calculate.h

Вводим скрипт в основной файл main.c, реализующий интерфейс пользователя к калькулятору (рис. 3.6).

```

1 //////////////////////////////////////////////////
2 // main.c
3
4 #include <stdio.h>
5 #include "calculate.h"
6
7 int
8 main (void)
9 {
10 float Numeral;
11 char Operation[4];
12 float Result;
13 printf("Число: ");
14 scanf("%f",&Numeral);
15 printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): ");
16 scanf("%s",&Operation);
17 Result = Calculate(Numeral, Operation);
18 printf("%6.2f\n",Result);
19 return 0;
20 }

```

Рис. 3.6: Скрипт в main.c

3. Выполните компиляцию программы посредством gcc (рис. 3.7).

```

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

```

Рис. 3.7: Компиляция программы

4. Синтаксические ошибки не обнаружены.
5. Создаем файл с именем “makefile” и вводим в него скрипт соответственно лабораторной работе (рис. 3.8).

```

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

```

Рис. 3.8: Создание makefile

Файл используется для автоматической компиляции main.c, calculate.c и создание из них исполняемого файла calcul. Помимо этого, в файле также есть функция 'clean', используемая для удаления всех файлов. В начале скрипта также вводятся переменные: CC - команда для компиляции gcc, CFLAGS - опции к команде компиляции, LIBS - опции при создании исполняемого файла.

6. Немного редактируем makefile - заменяем в тексте скрипта команду gcc на соответствующую ей переменную CC, а также придаем переменной CFLAGS значение опции '-g', чтобы объектные файлы можно было использовать в отладчике.

```

#
# Makefile
#

CC = gcc
CFLAGS = -g
LIBS = -lm

calcul: calculate.o main.o
    $(CC) calculate.o main.o -o calcul $(LIBS)

calculate.o: calculate.c calculate.h
    $(CC) -c calculate.c $(CFLAGS)

main.o: main.c calculate.h
    $(CC) -c main.c $(CFLAGS)

clean:
    -rm calcul *.o *~

# End Makefile

```

Рис. 3.9: Редактирование makefile

Проверяем работу makefile - удаляем файлы, затем вновь их компилируем (рис. 3.10).

```

[isbatova@fedora lab_prog]$ make clean
rm calcul *.o *~
[isbatova@fedora lab_prog]$ make calculate.o
gcc -c calculate.c
[1]+  Завершён      emacs
[isbatova@fedora lab_prog]$ make main.o
gcc -c main.c
[isbatova@fedora lab_prog]$ make calcul
gcc calculate.o main.o -o calcul -lm

```

Рис. 3.10: Проверка makefile

Далее с помощью gdb выполняем отладку программы calcul.

Запускаем отладчик GDB, загрузив в него программу для отладки (рис. 3.11).

```
[isbatova@fedora lab_prog]$ gdb ./calcul
GNU gdb (GDB) Fedora 12.1-4.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...
```

Рис. 3.11: Запуск отладчика

Для запуска программы внутри отладчика вводим команду run (рис. 3.12).

```
(gdb) run
Starting program: /home/isbatova/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]) n
Debuginfod has been disabled.
To make this setting permanent, add 'set debuginfod enabled off' to .gdbinit.
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): +
Второе слагаемое: 3
      8.00
[Inferior 1 (process 4937) exited normally]
Missing separate debuginfos, use: dnf debuginfo-install glibc-2.36-7.fc37.x86_
(gdb)
```

Рис. 3.12: Запуск программы внутри отладчика

Для постраничного просмотра исходного код вводим команду list (рис. 3.13).

```

(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
11     char Operation[4];
12     float Result;
13     printf("Число: ");
14     scanf("%f",&Numeral);
15     printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): ");
16     scanf("%s",&Operation);
17     Result = Calculate(Numeral, Operation);
18     printf("%.2f\n",Result);
19     return 0;
20 }

```

Рис. 3.13: Постраничный просмотр исходного кода

Для просмотра строк с 12 по 15 основного файла вводим команду list с параметрами - номерами строк (рис. 3.14).

```

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

```

Рис. 3.14: Просмотр определенных строк основного файла

Для просмотра определённых строк не основного файла вводим команду list с параметрами - номерами строк и указанием файла (рис. 3.15).

```

(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: ");

```

Рис. 3.15: Просмотр определенных строк не основного файла

Устанавливаем точку останова в файле calculate.c на строке номер 21 командой

break (рис. 3.16).

```
(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.
```

Рис. 3.16: Установка точки останова

Выводим информацию об имеющихся в проекте точках останова командой info breakpoints (рис. 3.17).

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

Рис. 3.17: Информация о точках останова

Чтобы убедиться, что программа остановится в момент прохождения точки останова, запускаем программу внутри отладчика (рис. 3.18).

```
(gdb) run
Starting program: /home/isbatova/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=0x7fffffffdf04 "-") at calculate.c:21
21      printf("Вычитаемое: ");
(gdb) backtrace
#0 Calculate (Numeral=5, Operation=0x7fffffffdf04 "-") at calculate.c:21
#1 0x00000000004014eb in main () at main.c:17
```

Рис. 3.18: Запуск программы с точкой останова

Проверяем, чему равно на этом этапе значение переменной Numeral командой print (рис. 3.19).


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

Рис. 3.19: Проверка значения Numeral командой print

Проверяем, чему равно на этом этапе значение переменной Numeral командой display (рис. 3.20).

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

Рис. 3.20: Проверка значения Numeral командой display

Убираем точки останова командой delete (рис. 3.21).

```
(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
```

Рис. 3.21: Удаление точек останова

7. Вводим команды 'splint calculate.c' (рис. 3.22, 3.23) и 'splint main.c' (рис. 3.24) для анализа кодов файлов.

```
[isbatova@fedora lab_prog]$ splint calculate.c
Splint 3.1.2 --- 23 Jul 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:16:1: 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)
```

Рис. 3.22: Команда 'splint calculate.c'

```
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
```

Рис. 3.23: Команда 'splint calculate.c'

```

[isbatova@fedora lab_prog]$ splint main.c
Splint 3.1.2 --- 23 Jul 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:14:1: 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:16:12: 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:16:9: Corresponding format code
main.c:16:1: Return value (type int) ignored: scanf("%s", &Ope...

Finished checking --- 4 code warnings

```

Рис. 3.24: Команда 'splint main.c'

С помощью данной команды мы узнали, что значения типа double в функциях pow, sin, cos, tan, sqrt записываются в переменную float, а значит, есть потеря данных. Также в обоих файлах есть функция scanf, которая возвращает целое значение, нигде не сохраняющиеся и не использующееся дальше в скрипте.

4 Выводы

В ходе данной лабораторной работы мной были приобретены простейшие навыки разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linux на примере создания на языке программирования С калькулятора с простейшими функциями.

5 Контрольные вопросы

1. Для получения информации о возможностях программ `gcc`, `make`, `gdb` и других можно воспользоваться командой `'man'`.
2. Основные этапы разработки приложений в UNIX:
 - Планирование: сбор и анализ требований к функционалу и другим характеристикам разрабатываемого приложения
 - Проектирование: разработка базовых алгоритмов и спецификаций, определение языка программирования
 - Кодирование: создание исходного текста программы
 - Анализ разработанного кода
 - Сборка, компиляция, разработка исполняемого модуля
 - Тестирование и отладка, сохранение изменений
 - Документирование
3. Суффикс определяет какая компиляция требуется для имени входного файла и указывают на тип объекта. Например, в команде `'gcc -c calculate.c'` по суффиксу `.c` распознается тип файла как файл на языке Си и формируется объектный файл с суффиксом `.o`.
4. Компилятор языка Си в UNIX используется для компиляции всей программы и получения исполняемого файла.

5. Утилита make предназначена для автоматизирования процесса преобразования файлов программы из одного формата в другой.

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

Рис. 5.1: Пример файла makefile

6.

Файл используется для автоматической компиляции main.c, calculate.c и создание из них исполняемого файла calcul. Помимо этого, в файле также есть функция 'clean', используемая для удаления всех файлов. В начале скрипта также вводятся переменные: CC - команда для компиляции gcc, CFLAGS - опции к команде компиляции, LIBS - опции при создании исполняемого файла.

7. Для того, чтобы можно было использовать программы отладки, необходимо скомпилировать анализируемый код программы так, чтобы отладочная

информация содержалась в результирующем бинарном файле (реализуется опцией -g компилятора gcc).

8. Основные команды отладчика gdb:

- backtrace - вывод на экран пути к текущей точке останова
- break - установить точку останова
- clear - удалить все точки останова в функции
- continue - продолжить выполнение программы
- delete - удалить точку останова
- display - добавить выражение в список выражений, значения которых отображаются при достижении точки останова программы
- finish - выполнить программу до момента выхода из функции
- info breakpoints - вывести на экран список используемых точек останова
- info watchpoints - вывести на экран список используемых контрольных выражений
- list - вывести на экран исходный код
- next - выполнить программу пошагово, но без выполнения вызываемых в программе функций
- print - вывести значение указываемого в качестве параметра выражения
- run - запуск программы на выполнение
- set - установить новое значение переменной
- step - пошаговое выполнение программы
- watch - установить контрольное выражение, при изменении значения которого программа будет остановлена

9. Схема отладки программы пошагово описана в шестом пункте лабораторной работы.

10. Синтаксических ошибок в программе при первом запуске обнаружено не было.

11. Основные средства, повышающие понимание исходного кода программы - cscope (исследование функций программы) и lint (проверка программ языка Си)
12. Программа splint анализирует программный код и выполняет проверку корректности всех аргументов, функций, значений, синтаксиса программы. Помимо этого, программа выдает комментарии с разбором кода программы.