

# **ОТЧЕТ О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ №13**

*дисциплина: Операционные системы*

Лихтенштейн Алина 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>7</b>
<b>5</b>	<b>Выполнение лабораторной работы</b>	<b>8</b>
<b>6</b>	<b>Выводы</b>	<b>12</b>

## Список иллюстраций

5.1	Создание файлов . . . . .	8
5.2	Компиляция . . . . .	8
5.3	Makefile . . . . .	9
5.4	отладка программы calcul . . . . .	10
5.5	splint calculate.c . . . . .	11
5.6	splint main.c . . . . .	11

# 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` со следующим содержанием.
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

Процесс разработки программного обеспечения обычно разделяется на следующие этапы:

- планирование, включающее сбор и анализ требований к функционалу и другим характеристикам разрабатываемого приложения;
- проектирование, включающее в себя разработку базовых алгоритмов и спецификаций,
- определение языка программирования;
- непосредственная разработка приложения;
- кодирование — по сути создание исходного текста программы (возможно в нескольких вариантах);
- анализ разработанного кода;
- сборка, компиляция и разработка исполняемого модуля;
- тестирование и отладка, сохранение произведённых изменений;
- документирование.

## **4 Выполнение лабораторной работы**

## 5 Выполнение лабораторной работы

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 (рис. 5.1)



Рис. 5.1: Создание файлов

3. Выполните компиляцию программы посредством `gcc`.
4. При необходимости исправьте синтаксические ошибки (рис. 5.2)

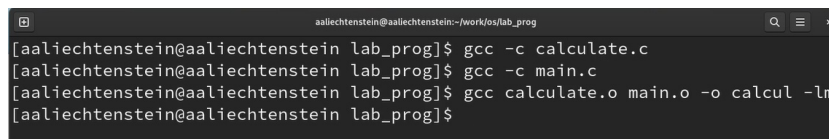



Рис. 5.2: Компиляция



## 5. Создайте Makefile (рис. 5.3)



```
#
# 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. 5.3: Makefile

### Объяснение файла:

#Объявление переменных

CC = gcc #компилятор

CFLAGS = -g #опция, которая отладочную информацию положит в результирующий бинарный файл

LIBS = -lm

#Создаем файл calcul из файлов calculate.o main.o

calcul: calculate.o main.o #ниже обращаемся к содержимому переменной

\$(CC) calculate.o main.o -o calcul \$(LIBS) #добавляем опцию

#Здесь отражена строка: gcc calculate.o main.o -o calcul -lm

#Создаем файл calculate.o

calculate.o: calculate.c calculate.h #

\$(CC) -c calculate.c \$(CFLAGS) #gcc -c calculate.c -g

#Создаем файл main.o

main.o: main.c calculate.h #gcc -c main.c -g

\$(CC) -c main.c \$(CFLAGS) #

#

clean: #при вызове make clean будем удалять все файлы с разрешением .o

-rm calcul \*.o \*~ #

6. С помощью gdb выполните отладку программы calcul (перед использованием gdb исправьте Makefile)(рис. 5.4)

```
24 }
25 else if(strncmp Operation, "*", 1) == 0)
26 {
27     printf("Множитель: ");
28     scanf("%f",&SecondNumeral);
29     return(Numeral * SecondNumeral);
(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) info breakpoints
Num Type Disp Enb Address What
1 breakpoint keep y 0x000000000040120f in Calculate at calculate.c:21
(gdb) run
Starting program: /home/dvshilonosov/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=0x7fffffffdf24 "-") at calculate.c:21
21     printf("Вычитаемое: ");
(gdb)
```

Рис. 5.4: отладка программы calcul

7. С помощью утилиты splint попробуйте проанализировать коды файлов calculate.c и main.c (рис. 5.5, 5.6)

```

aalechtenstein@aalechtenstein:~/work/os/lab_prog
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:35:11: Return value type double does not match declared type float:
(HUGE_VAL)
To allow all numeric types to match, use +relaxtypes.
calculate.c:43:5: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:44:11: Return value type double does not match declared type float:
(pow(Numeral, SecondNumeral))
calculate.c:47:11: Return value type double does not match declared type float:
(sqrt(Numeral))
calculate.c:49:11: Return value type double does not match declared type float:
(sin(Numeral))
calculate.c:51:11: Return value type double does not match declared type float:
(cos(Numeral))
calculate.c:53:11: Return value type double does not match declared type float:
(tan(Numeral))

```

Рис. 5.5: splint calculate.c

```

aalechtenstein@aalechtenstein:~/work/os/lab_prog -- less
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:13:1: 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:15:12: Format argument 1 to scanf (%s) expects char * gets char [4] *:
&Operation
Type of parameter is not consistent with corresponding code in format string.

```

Рис. 5.6: splint main.c

## 6 Выводы

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 С калькулятора с простейшими функциями.