

# **Отчет по лабораторной работе №13**

**Дисциплина: операционные системы**

Лобанова Полина Иннокентьевна

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Выполнение лабораторной работы</b>	<b>6</b>
<b>3</b>	<b>Контрольные вопросы</b>	<b>12</b>
<b>4</b>	<b>Вывод</b>	<b>17</b>

## Список иллюстраций

2.1	Создание каталога. . . . .	6
2.2	Создание файлов. . . . .	6
2.3	Содержание файла <i>calculate.c</i> . . . . .	7
2.4	Содержание файла <i>calculate.h</i> . . . . .	7
2.5	Содержание файла <i>main.c</i> . . . . .	8
2.6	Компиляция программы. . . . .	8
2.7	Создание файла. . . . .	8
2.8	Содержание файла <i>Makefile</i> . . . . .	9
2.9	Вызов отладчика. . . . .	9
2.10	Запуск программы. . . . .	10
2.11	Просмотр исходного кода. . . . .	10
2.12	Анализ <i>calculate.c</i> . . . . .	11
2.13	Анализ <i>main.c</i> . . . . .	11

## **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 Выполнение лабораторной работы

1. В домашнем каталоге создадим подкаталог ~/work/os/lab\_prog.

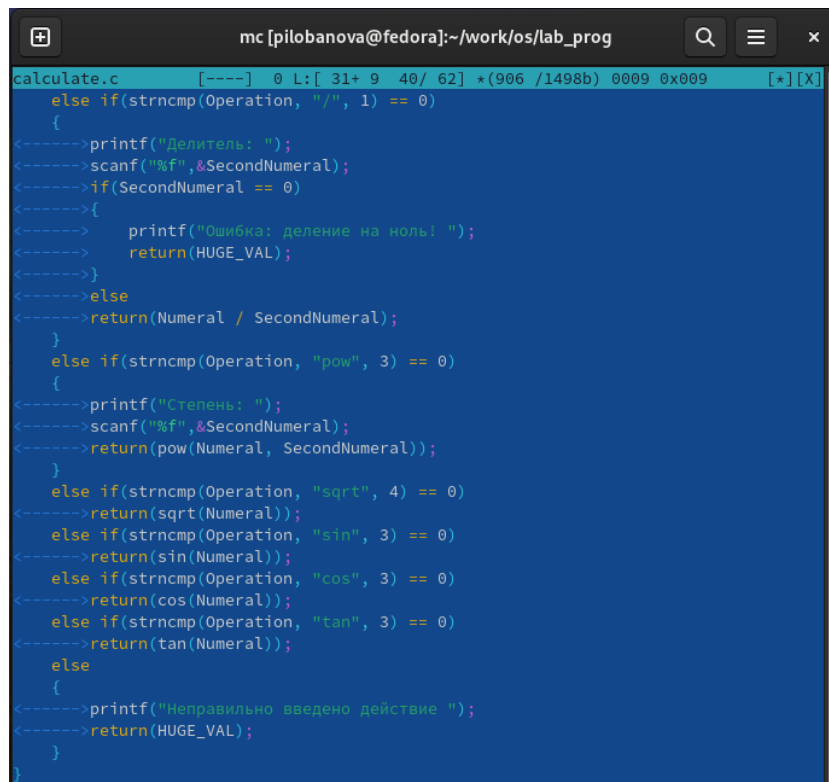
```
[pilobanova@fedora ~]$ cd ~/work/os/  
[pilobanova@fedora os]$ mkdir lab_prog  
[pilobanova@fedora os]$ cd lab_prog/
```

Рис. 2.1: Создание каталога.

2. 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.

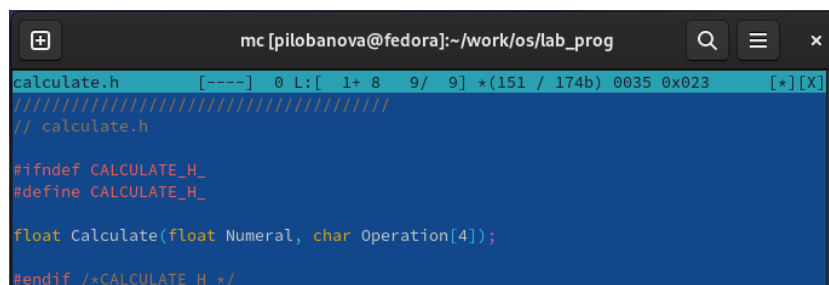
```
[pilobanova@fedora lab_prog]$ touch calculate.h  
[pilobanova@fedora lab_prog]$ touch calculate.c  
[pilobanova@fedora lab_prog]$ touch main.c
```

Рис. 2.2: Создание файлов.



```
mc [pilobanova@fedora]:~/work/os/lab_prog
calculate.c [----] 0 L: [ 31+ 9 40/ 62] *(906 /1498b) 0009 0x009 [*][X]
    else if(strcmp(Operation, "/" ) == 0)
    {
<----->printf("Делитель: ");
<----->scanf("%f",&SecondNumeral);
<----->if(SecondNumeral == 0)
<----->{
<----->    printf("Ошибка: деление на ноль! ");
<----->    return(HUGE_VAL);
<----->}
<----->else
<----->return(Numeral / SecondNumeral);
    }
    else if(strcmp(Operation, "pow", 3) == 0)
    {
<----->printf("Степень: ");
<----->scanf("%f",&SecondNumeral);
<----->return(pow(Numeral, SecondNumeral));
    }
    else if(strcmp(Operation, "sqrt", 4) == 0)
<----->return(sqrt(Numeral));
    else if(strcmp(Operation, "sin", 3) == 0)
<----->return(sin(Numeral));
    else if(strcmp(Operation, "cos", 3) == 0)
<----->return(cos(Numeral));
    else if(strcmp(Operation, "tan", 3) == 0)
<----->return(tan(Numeral));
    else
    {
<----->printf("Неправильно введено действие ");
<----->return(HUGE_VAL);
    }
}
```

Рис. 2.3: Содержание файла *calculate.c*.



```
mc [pilobanova@fedora]:~/work/os/lab_prog
calculate.h [----] 0 L: [ 1+ 8 9/ 9] *(151 / 174b) 0035 0x023 [*][X]
////////////////////////////////////
// calculate.h

#ifndef CALCULATE_H_
#define CALCULATE_H_

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

#endif /*CALCULATE_H_*/
```

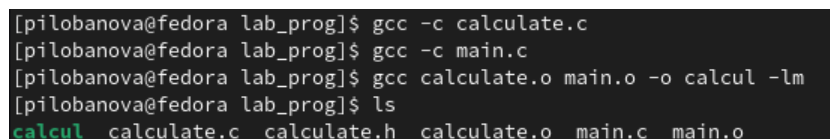
Рис. 2.4: Содержание файла *calculate.h*.

A screenshot of a code editor window titled 'mc [pilobanova@fedora]:~/work/os/lab\_prog'. The editor shows the content of a file named 'main.c'. The code includes a header file 'calculate.h', declares variables 'Numeral' (float), 'Operation' (char array), and 'Result' (float), and contains a 'main' function that prompts the user for a number and an operation, then calls a 'Calculate' function and prints the result.

```
main.c [----] 0 L: [ 1+18 19/ 19] *(407 / 408b) 0125 0x07D [*] [X]  
////////////////////////////////////  
// 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. 2.5: Содержание файла *main.c*.

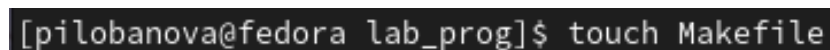
3. Выполним компиляцию программы посредством gcc.

A screenshot of a terminal window showing the compilation of the program using gcc. The commands compile 'calculate.c' and 'main.c' separately, then link them together into an executable named 'calcul' with the '-lm' flag. The 'ls' command is used to verify the files in the directory.

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

Рис. 2.6: Компиляция программы.

4. Создадим Makefile и заполним его.

A screenshot of a terminal window showing the command to create a new file named 'Makefile' using the 'touch' command.

```
[pilobanova@fedora lab_prog]$ touch Makefile
```

Рис. 2.7: Создание файла.





```
Makefile [----] 0 L: [ 1+20 21/ 22] *(276 / 291b) 0035 0x023 [*][X]
#
# 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)

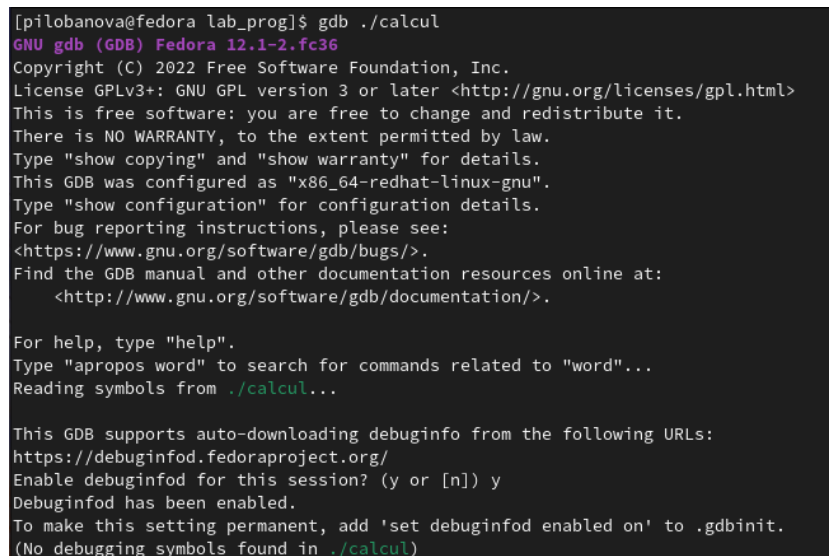
main.o: main.c calculate.h
    gcc -c main.c $(CFLAGS)

clean:
    -rm calcul *.o *~

End Makefile
```

Рис. 2.8: Содержание файла Makefile.

5. С помощью gdb выполним отладку программы calcul.



```
[pilobanova@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...

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.
(No debugging symbols found in ./calcul)
```

Рис. 2.9: Вызов отладчика.

```
(gdb) run
Starting program: /home/pilobanova/work/os/lab_prog/calcul
Downloading 0.01 MB separate debug info for system-supplied DSO at 0x7ffff7fc4000
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: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): +
Второе слагаемое: 6
11.00
[Inferior 1 (process 3051) exited normally]
```

Рис. 2.10: *Запуск программы.*

```
(gdb) list
1      /* Terminate the frame unwind info section with a 4byte 0 as a sentinel;
2         this would be the 'length' field in a real FDE.  */
3
4      typedef unsigned int ui32 __attribute__((mode(SI)));
5      static const ui32 __FRAME_END__[1]
6          __attribute__((used, section(".eh_frame")))
7          = { 0 };
```

Рис. 2.11: *Просмотр исходного кода.*

6. С помощью утилиты splint попробуем проанализировать коды файлов calculate.c и main.c.

```
[pilobanova@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:16:2: 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:38:12: 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:
        (tan(Numeral))
calculate.c:60:8: Return value type double does not match declared type float:
        (HUGE_VAL)

Finished checking --- 15 code warnings
```

Рис. 2.12: Анализ *calculate.c*.

```
[pilobanova@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:13:5: 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:15:16: 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:15:13: Corresponding format code
main.c:15:5: Return value (type int) ignored: scanf("%s", &Ope...

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

Рис. 2.13: Анализ *main.c*.

### 3 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.

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. компиляция 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 использования.

Использование суффикса “.c” 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 .c компилятор распознает, что файл `abcd.c` должен компилироваться, а по суффиксу

.o, 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 `-p` 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`. Второй способ 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;

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

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; slist – выводит исходный код; в качестве параметра передаются название файла исходного кода, затем, через двоеточие, номер начальной и конечной строки; – next – 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.

Выполнили компиляцию программы Увидели ошибки в программе Открыли редактор и исправили программу Загрузили программу в отладчик gdb run — отладчик выполнил программу, мы ввели требуемые значения. 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.



## 4 Вывод

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.