

# **Отчет по лабораторной работе №13**

*дисциплина: Операционные системы*

Егорова Юлия Владимировна

# Содержание

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

## Список иллюстраций

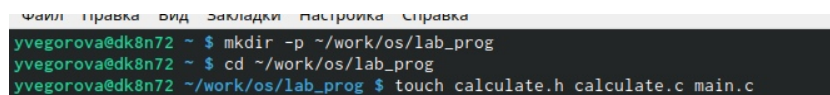
2.1	Создание подкаталога и файлов в нем. . . . .	5
2.2	Проверка содержимого и переход в Emacs. . . . .	5
2.3	Программа для calculate.h. . . . .	5
2.4	Программа для calculate.c. . . . .	6
2.5	Программа для calculate.c. . . . .	6
2.6	Программа для main.c. . . . .	7
2.7	Выполнение компиляции. . . . .	7
2.8	Вывод компиляции. . . . .	7
2.9	Вывод компиляции. . . . .	7
2.10	Вывод компиляции. . . . .	7
2.11	Создание файла. . . . .	8
2.12	Исправление файла. . . . .	8
2.13	Команда gdb ./calcul. . . . .	9
2.14	Проверка работы. . . . .	9
2.15	Команда list. . . . .	10
2.16	Команда list 12,15. . . . .	10
2.17	Команда list calculate.c:20,29. . . . .	10
2.18	Команды break 21, info breakpoints, backtrace. . . . .	11
2.19	Команда print Numeral, display Numeral, info breakpoints и delete 1. . . . .	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.

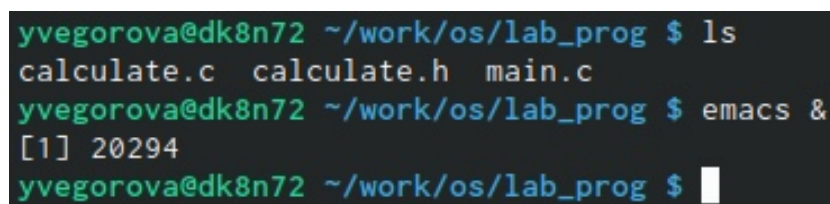
## 2 Выполнение лабораторной работы

1) В домашнем каталоге создала подкаталог `~/work/os/lab_prog` в нём файлы: `calculate.h`, `calculate.c`, `main.c`:



```
Файл  Правка  Вид  Закладки  Настройка  Справка
yvegorova@dk8n72 ~ $ mkdir -p ~/work/os/lab_prog
yvegorova@dk8n72 ~ $ cd ~/work/os/lab_prog
yvegorova@dk8n72 ~/work/os/lab_prog $ touch calculate.h calculate.c main.c
```

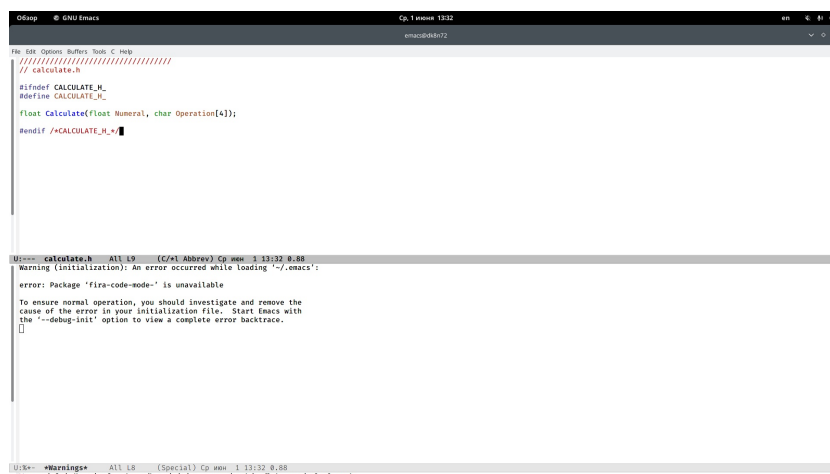
Рис. 2.1: Создание подкаталога и файлов в нём.



```
yvegorova@dk8n72 ~/work/os/lab_prog $ ls
calculate.c calculate.h main.c
yvegorova@dk8n72 ~/work/os/lab_prog $ emacs &
[1] 20294
yvegorova@dk8n72 ~/work/os/lab_prog $
```

Рис. 2.2: Проверка содержимого и переход в Emacs.

2) Написала программу для `calculate.h`:



```
GNU Emacs
Ср. 1 нояб. 19:32
File Edit Options Buffer Tools C Mode
// calculate.h
// =====
#ifndef CALCULATE_H
#define CALCULATE_H

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

#endif //CALCULATE_H
```

Warning (initialization): An error occurred while loading '~/.emacs':  
error: Package 'fira-code-mode' is unavailable  
To ensure normal operation, you should investigate and remove the cause of the error in your initialization file. Start Emacs with the '--debug-init' option to view a complete error backtrace.

Рис. 2.3: Программа для `calculate.h`.

Написала программу для calculate.c:

C:\Program Files\JetBrains\CLion 2024.2

cmake@kbln72

```
# Run Edit Options button from C Help  
// calculate.c  
#include<stdio.h>  
#include<math.h>  
#include<string.h>  
#include "calculate.h"  
  
float  
Calculate (float Numeral, char Operation[4])  
{ float SecondNumeral;  
    if(strlen(Operation) == 1) == 0)  
        {printf("Wrong character: ");  
         scanf("%f",&SecondNumeral);  
         return(Numeral + SecondNumeral);  
        }  
    else if(strlen(Operation) == 2) == 0)  
        {printf("Invent name:");  
         scanf("%f",&SecondNumeral);  
         return(Numeral * SecondNumeral);  
        }  
    else if(strlen(Operation) == 3) == 0)  
        {printf("Memory trace:");  
         scanf("%f",&SecondNumeral);  
         return(Numeral / SecondNumeral);  
        }  
    else if(strlen(Operation) == 4) == 0)  
        {printf("Inventory:");  
         scanf("%f",&SecondNumeral);  
         if(SecondNumeral == 0)  
             { printf("Dumbal! Na mo denzha nenas!");  
               return(HUGE_VAL);  
             }  
            else  
                return(Numeral/SecondNumeral);  
        }  
    else if(strlen(Operation, "pow", 3)) == 0)  
        {  
            printf("Cremona: ");  
            scanf("%f",&SecondNumeral);  
            return(pow(Numeral,SecondNumeral));  
        }  
}
```

User> calculate.c Top L45 (C/C++ Abbrv.) Cp wss 3 12:58 9.92  
cause of the error in your initialization file. Start Emacs with  
the "-debug-init" option to view a complete error backtrace.  
[]  
User> \*\*Warnings\*\* Bot L8 (Special) Cp wss 1 12:58 9.92

Рис. 2.4: Программа для calculate.c.

Odexp @ GNU Emacs

Cp. 1.0.0.0 12:58

emc@odex7

emc@odex7

```
#B: BRT Options Authors: John C. Web
{
    scan("%f", &SecondNumeral);
    return(pow(Numeral, SecondNumeral));
}
else if(strlen(Operation, "sqrt", 4) == 0)
    return(sqrt(Numeral));
else if(strlen(Operation, "sin", 3) == 0)
    return(sin(Numeral));
else if(strlen(Operation, "cos", 3) == 0)
    return(cos(Numeral));
else if(strlen(Operation, "tan", 3) == 0)
    return(tan(Numeral));
else
{
    printf("Неправильно введено действие.");
    return(MDGE_VAL);
}
}
```

U>= calculate.c Bot L41 [C/Al Abbrev] Cp. 0.0.0 12:58 0.92

cause of the error in your initialization file. Start Emacs with the "--debug-init" option to view a complete error backtrace.

U>= \*Warnings\* Bot L8 (Special) Cp. 0.0.0 12:58 0.92

Рис. 2.5: Программа для calculate.c.

Написала программу для main.c:

```

GNU Emacs
Cp. 1 error 1340
emacs@dk8n72

// main.c

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

int
main(void)
{
    float Numeral;
    char Operation[4];
    float Result;
    printf("Numeral: ");
    scanf("%f", &Numeral);
    printf("Operation (+, -, *, /, pow, sqrt, sin, cos, tan): ");
    scanf("%s", &Operation);
    Result = Calculate(Numeral, Operation);
    printf("Result: %f\n", Result);
    return 0;
}

Warning (initialization): An error occurred while loading `~/.emacs':
error: Package `fira-code-mode' is unavailable

To ensure normal operation, you should investigate and remove the
cause of the error in your initialization file. Start Emacs with
the `--debug-init' option to view a complete error backtrace.

```

Рис. 2.6: Программа для main.c.

3)Выполнила компиляцию программы посредством gcc:

```

yvegorova@dk8n72 ~/work/os/lab_prog $ gcc -c calculate.c
yvegorova@dk8n72 ~/work/os/lab_prog $ gcc -c main.c
yvegorova@dk8n72 ~/work/os/lab_prog $ gcc calculate.o main.o -o calcul -lm

```

Рис. 2.7: Выполнение компиляции.

```

Сообщайте об ошибках по адресу <bug.makexnu.org>
yvegorova@dk8n72 ~/work/os/lab_prog $ make calculate.o gcc -c calculate.c -g

```

Рис. 2.8: Вывод компиляции.

```

yvegorova@dk8n72 ~/work/os/lab_prog $ make main.o gcc -c main.c -g

```

Рис. 2.9: Вывод компиляции.

```

yvegorova@dk8n72 ~/work/os/lab_prog $ make calcul gcc calculate.o main.o -o calcul -lm

```

Рис. 2.10: Вывод компиляции.

4)Создание и заполнение содержимого Makefile:

```
#
#makefile
#
CC=gcc
FLAGS=-lm
calcul: calculate.o main.o
gcc calculate.o main.o -o calcul $(FLAGS)
calculate.o: calculate.c calculate.o
gcc -c calculate.c $(FLAGS)
main.o: main.c calculate.h
gcc -c main.c $(FLAGS)
clean:
rm calcul *.o
#End makefile
```

Warning (Initialization): An error occurred while loading '~/.emacs':  
error: Package 'fira-code-mode' is unavailable  
To ensure normal operation, you should investigate and remove the cause of the error in your initialization file. Start Emacs with the '--debug-init' option to view a complete error backtrace.

Wrote /afs/ds.sci.pfu.edu.ru/home/j/y/yegegorova/work/os/lab\_prog/makefile

Рис. 2.11: Создание файла.

## 5)Исправила этот файл:

```
#
#makefile
#
CC=gcc
FLAGS=-g
LFLAGS=-lm
calcul: calculate.o main.o
gcc calculate.o main.o -o calcul $(LFLAGS)
calculate.o: calculate.c calculate.o
gcc -c calculate.c $(FLAGS)
main.o: main.c calculate.h
gcc -c main.c $(FLAGS)
clean:
rm calcul *.o
#End makefile
```

Warning (Initialization): An error occurred while loading '~/.emacs':  
error: Package 'fira-code-mode' is unavailable  
To ensure normal operation, you should investigate and remove the cause of the error in your initialization file. Start Emacs with the '--debug-init' option to view a complete error backtrace.

Find file ~/work/os/lab\_prog

Рис. 2.12: Исправление файла.

6)С помощью gdb выполнила отладку программы calcul (перед использованием gdb исправила Makefile), а затем запустила отладчик GDB, загрузив в него программу для отладки:



```

yvegorova@dk8n72 ~/work/os/lab_prog $ gdb ./calcul
GNU gdb (Gentoo 10.2 vanilla) 10.2
Copyright (C) 2021 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-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://bugs.gentoo.org/>.
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...
(gdb)

```

Рис. 2.13: Команда gdb ./calcul.

Нажала run, ввела число 7, выбрала операцию сложения, прибавила 7 и получила верный ответ 14:

```

yvegorova@dk8n72 ~/work/os/lab_prog $ gdb ./calcul
GNU gdb (Gentoo 10.2 vanilla) 10.2
Copyright (C) 2021 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-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://bugs.gentoo.org/>.
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...
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/y/v/yvegorova/work/os/lab_prog/calcul
Число: 7
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): +
Второе слагаемое: 7
14.00
[Inferior 1 (process 33672) exited normally]
(gdb)

```

Рис. 2.14: Проверка работы.

Затем последовательно ввела команды list, list 12,15, list calculate.c:20,29, list calculate.c:20,27, break 21, info breakpoints, backtrace, print Numeral, display Numeral, info breakpoints и delete 1:

```

[Inferior 1 (process 33672) 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. 2.15: Команда list.

```

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

```

Рис. 2.16: Команда list 12,15.

```

(gdb) list calculate.c:20,29
20         return(Numeral-SecondNumeral);
21     }
22     else if(strncmp(Operation,"*",1) == 0)
23     {printf("Множитель: ");
24       scanf("%f",&SecondNumeral);
25       return(Numeral*SecondNumeral);
26     }
27     else if(strncmp(Operation,"/",1) == 0)
28     {printf("Делитель: ");
29       scanf("%f",&SecondNumeral);
(gdb) █

```

Рис. 2.17: Команда list calculate.c:20,29.



### 3 Контрольные вопросы

Контрольные вопросы:

1). Чтобы получить информацию о возможностях программ gcc, make, gdb и др. нужно воспользоваться командой man или опцией -help(-h) для каждой команды.

2). 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. Для создания исходного текста программы разработчик может воспользоваться любым удобным для него редактором текста: vi, vim, mceditor, emacs, geany и др. После завершения написания исходного кода программы (возможно состоящей из нескольких файлов), необходимо её скомпилировать и получить исполняемый модуль. 3). Для имени входного файла суффикс определяет какая компиляция требуется. Суффиксы указывают на тип объекта. Файлы с расширением (суффиксом) .c воспринимаются gcc как программы на языке C, файлы с расширением .cpp – как файлы на языке C++, а файлы с расширением .o считаются объектными. Например, в команде «gcc -o main.o main.c»: gcc по расширению

(суффиксу) .o распознает тип файла для компиляции и формирует объектный модуль – файл с расширением .o. Если требуется получить исполняемый файл с определённым именем (например, hello), то требуется воспользоваться опцией -o в качестве параметра задать имя создаваемого файла: «gcc -o hello main.c». В ходе выполнения данной лабораторной работы я приобрела простейшие навыки разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linux на примере создания на языке программирования C калькулятора с простейшими функциями.

4). Основное назначение компилятора языка Си в UNIX заключается в компиляции всей программы и получении исполняемого файла/модуля.

5). Для сборки разрабатываемого приложения и собственно компиляции полезно воспользоваться утилитой make. Она позволяет автоматизировать процесс преобразования файлов программы из одной формы в другую, отслеживает взаимосвязи между файлами.

6). Для работы с утилитой make необходимо в корне рабочего каталога с Вашим проектом создать файл с названием makefile или Makefile, в котором будут описаны правила обработки файлов Вашего программного комплекса. В самом простом случае Makefile имеет следующий синтаксис: ... : ... <команда 1> ... Сначала задаётся список целей, разделённых пробелами, за которым идёт двоеточие и список зависимостей. Затем в следующих строках указываются команды. Строки с командами обязательно должны начинаться с табуляции. В качестве цели в Makefile может выступать имя файла или название какого-то действия. Зависимость задаёт исходные параметры (условия) для достижения указанной цели. Зависимость также может быть названием какого-то действия. Команды – собственно действия, которые необходимо выполнить для достижения цели. Общий синтаксис Makefile имеет вид: target1 [target2...]:[:] [dependment1...][ (tab) commands] [#commentary][ (tab) commands] [#commentary]. Здесь знак # определяет начало комментария (содержимое от знака # и до конца строки не будет обрабатываться. Одинарное двоеточие указывает на то, что последовательность команд долж-

на содержаться в одной строке. Для переноса можно в длинной строке команд можно использовать обратный слэш (). Двойное двоеточие указывает на то, что последовательность команд может содержаться в нескольких последовательных строках. Пример более сложного синтаксиса Makefile: ## Makefile for abcd.c #CC = gcc #CFLAGS = # Compile abcd.c normally abcd: abcd.c(CFLAGS) abcd.o cclean: -rm abcd.o ~# End Makefile for abcd.c. В этом примере в начале файла заданы три переменные: CC и CFLAGS. Затем указаны цели, их зависимости и соответствующие команды. В командах происходит обращение к значениям переменных. Цель с именем clean производит очистку каталога от файлов, полученных в результате компиляции. Для её описания использованы регулярные выражения.

7). 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). Для использования GDB необходимо скомпилировать анализируемый код программы таким образом, чтобы отладочная информация содержалась в результирующем бинарном файле. Для этого следует воспользоваться опцией -g компилятора gcc: gcc -c file.c -g. После этого для начала работы с gdb необходимо в командной строке ввести одноимённую команду, указав в качестве аргумента анализируемый бинарный файл: gdb file.o

8). Основные команды отладчика gdb:

backtrace – вывод на экран пути к текущей точке останова (по сути вывод – названий всех функций); break – установить точку останова (в качестве параметра может быть указан номер строки или название функции); 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 – вы-

вести на экран список используемых контрольных выражений; `list` – вывести на экран исходный код (вВ ходе выполнения данной лабораторной работы я приобрела простейшие навыки разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linux на примере создания на языке программирования C калькулятора с простейшими функциями. качестве параметра может быть указано название файла и через двоеточие номера начальной и конечной строк); `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. Для выхода из `gdb` можно воспользоваться командой `quit` (или её сокращённым вариантом `q`) или комбинацией клавиш `Ctrl-d`. Более подробную информацию по работе с `gdb` можно получить с помощью команд `gdb-hi` и `mangdb`. 9). Схема отладки программы показана в 6 пункте лабораторной работы.

10). При первом запуске компилятор не выдал никаких ошибок, но в коде программы `main.c` допущена ошибка, которую компилятор мог пропустить (возможно, из-за версии 8.3.0-19): в строке `scanf("%s", &Operation);` нужно убрать знак `&`, потому что имя массива символов уже является указателем на первый элемент этого массива.

11). Система разработки приложений UNIX предоставляет различные средства, повышающие понимание исходного кода. К ним относятся: `xcscope` – исследование функций, содержащихся в программе, `xlint` – критическая проверка программ, написанных на языке Си.

12). Утилита `splint` анализирует программный код, проверяет корректность задания аргументов использованных в программе функций и типов возвращаемых значений, обнаруживает синтаксические и семантические ошибки. В отличие от компилятора Санализатор `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 Выводы

В ходе выполнения данной лабораторной работы я приобрела простейшие навыки разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linux на примере создания на языке программирования C калькулятора с простейшими функциями.