

# **Лабораторная работа №13**

**Операционные системы**

Сабралиева Марворид Нуралиевна

# 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>Выводы</b>	<b>10</b>
<b>4</b>	<b>Контрольные вопросы</b>	<b>11</b>
	<b>Список литературы</b>	<b>14</b>

## Список иллюстраций

2.1	Создание подкаталога . . . . .	6
2.2	Создание файлов . . . . .	6
2.3	Реализация функций калькулятора в файле calculate.h . . . . .	7
2.4	Интерфейсный файл calculate.h, описывающий формат вызова функции калькулятора . . . . .	7
2.5	Основной файл main.c, реализующий интерфейс пользователя к калькулятору . . . . .	8
2.6	компиляция программы . . . . .	8
2.7	запустим отладчик . . . . .	8
2.8	запуск программы . . . . .	9
2.9	утилита splint . . . . .	9

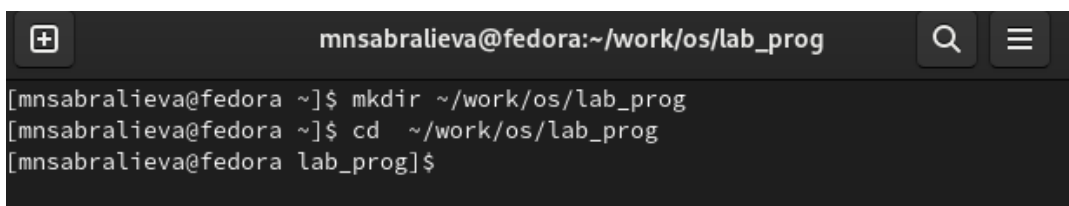
## Список таблиц

# 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

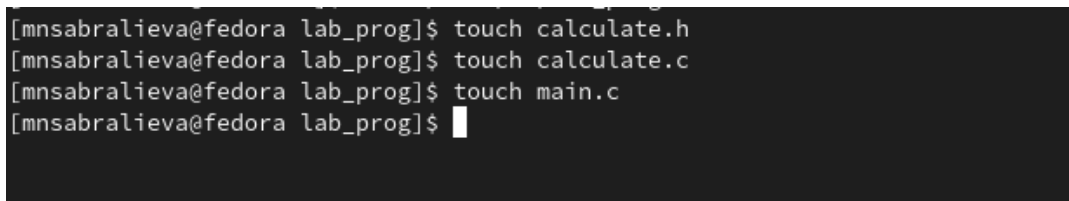
1. В домашнем каталоге создадим подкаталог `~/work/os/lab_prog`. (рис. 2.1).

A screenshot of a terminal window with a dark background. The title bar shows the user 'mnsabralieva' on a 'fedora' machine, with the current directory set to '~/.work/os/lab\_prog'. The terminal contains three lines of commands and their outputs: the first line shows 'mkdir ~/.work/os/lab\_prog' being executed; the second line shows 'cd ~/.work/os/lab\_prog' being executed; the third line shows the prompt changing to 'lab\_prog\$'.

```
mnsabralieva@fedora:~/work/os/lab_prog
[mnsabralieva@fedora ~]$ mkdir ~/.work/os/lab_prog
[mnsabralieva@fedora ~]$ cd ~/.work/os/lab_prog
[mnsabralieva@fedora lab_prog]$
```

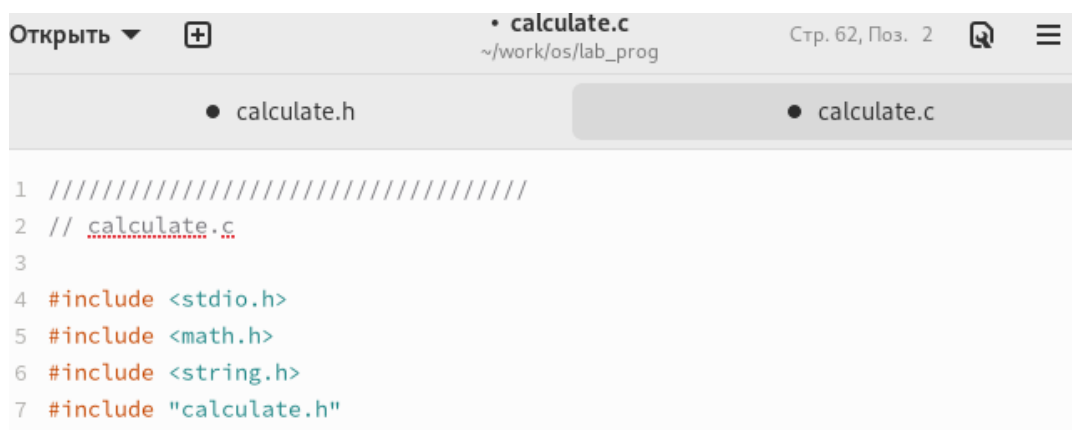
Рис. 2.1: Создание подкаталога

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. (рис. 2.2).

A screenshot of a terminal window with a dark background. The title bar shows the user 'mnsabralieva' on a 'fedora' machine, with the current directory set to 'lab\_prog'. The terminal contains four lines of commands and their outputs: the first line shows 'touch calculate.h' being executed; the second line shows 'touch calculate.c' being executed; the third line shows 'touch main.c' being executed; the fourth line shows the prompt 'lab\_prog\$' with a cursor. The first line of the screenshot is partially cut off.

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

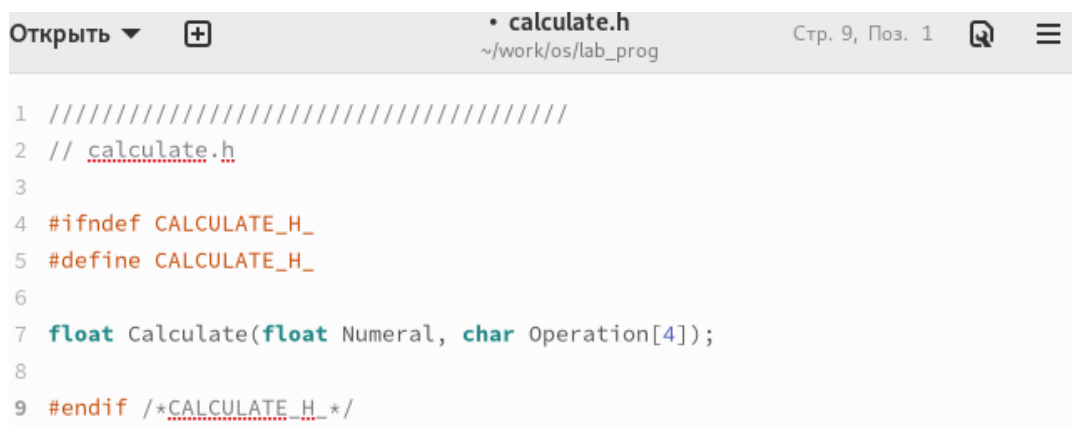
Рис. 2.2: Создание файлов



The screenshot shows a code editor window with the title bar "calculate.c" and the path "~/work/os/lab\_prog". The editor displays the implementation of calculator functions in the file calculate.c. The code includes standard headers and the calculator.h header.

```
1 ///////////////////////////////////////////////////  
2 // calculate.c  
3  
4 #include <stdio.h>  
5 #include <math.h>  
6 #include <string.h>  
7 #include "calculate.h"
```

Рис. 2.3: Реализация функций калькулятора в файле calculate.h



The screenshot shows a code editor window with the title bar "calculate.h" and the path "~/work/os/lab\_prog". The editor displays the interface file calculate.h, which defines the function prototype for the calculator.

```
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. 2.4: Интерфейсный файл calculate.h, описывающий формат вызова функции калькулятора

Рис. 2.5: Основной файл main.c, реализующий интерфейс пользователя к калькулятору

3. Выполните компиляцию программы посредством gcc (рис. 2.6).

```
mnsabralieva@fedora lab_prog]$ gcc -c calculate.c
mnsabralieva@fedora lab_prog]$ gcc -c main.c
mnsabralieva@fedora lab_prog]$ gcc calculate.o main.o -o calcul -lm
mnsabralieva@fedora lab_prog]$
```

Рис. 2.6: компиляция программы

4. Создадим Makefile и с помощью gdb выполним отладку программы calcul (рис. 2.7).

```
[mnsabralieva@fedora lab_prog]$ gdb ./calcul
GNU gdb (GDB) Fedora Linux 12.1-7.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.
```

Рис. 2.7: запустим отладчик



```
Debuginfod has been enabled.  
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.  
(No debugging symbols found in ./calcul)  
(gdb) run  
Starting program: /home/mnsabralieva/work/os/lab_prog/calcul
```

Рис. 2.8: запуск программы

5. С помощью утилиты splint попробуйте проанализировать коды файлов calculate.c и main.c. (рис. 2.9).

```
[mnsabralieva@fedora lab_prog]$ splint  
bash: splint: команда не найдена...  
Установить пакет «splint», предоставляющий команду «splint»? [N/y] y  
  
* Ожидание в очереди...  
Следующие пакеты должны быть установлены:  
splint-3.1.2-29.fc37.x86_64    An implementation of the lint program  
Продолжить с этими изменениями? [N/y] y
```

Рис. 2.9: утилита splint

## 3 Выводы

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

## 4 Контрольные вопросы

1. Как получить информацию о возможностях программ gcc, make, gdb и др.?

Ответ: Для этого есть команда man и предлагающиеся к ней файлы.

2. Назовите и дайте краткую характеристику основным этапам разработки приложений в UNIX. Ответ: Кодировка, Компиляция, Тест.

3. Что такое суффикс в контексте языка программирования? Приведите примеры использования. Ответ: Это расширения файлов.

4. Каково основное назначение компилятора языка C в UNIX? Ответ: Программа gcc, которая интерпретирует к определенному языку программирования аргументы командной строки и определяет запуск нужного компилятора для нужного файла.

5. Для чего предназначена утилита make? Ответ: Для компиляции группы файлов. Собрания из них программы, и последующего удаления.

6. Приведите пример структуры Makefile. Дайте характеристику основным элементам этого файла. Ответ: `program: main.o lib.o cc -o program main.o lib.o`  
`main.o lib.o: defines.h` В имени второй цели указаны два файла и для этой же цели не указана команда компиляции. Кроме того, нигде явно не указана зависимость объектных файлов от «\*.c»-файлов. Дело в том, что программа make имеет предопределённые правила для получения файлов с определёнными расширениями. Так, для цели-объектного файла (расширение «.o») при обнаружении соответствующего файла с расширением «.c» будет вызван компилятор «cc -c» с указанием в параметрах этого «.c»-файла и всех файлов-зависимостей.

7. Назовите основное свойство, присущее всем программам отладки. Что необходимо сделать, чтобы его можно было использовать? Ответ: Программы для отладки нужны для нахождения ошибок в программе. Для их использования надо скомпилировать программу таким образом, чтобы отладочная информация содержалась в конечном бинарном файле.
8. Назовите и дайте основную характеристику основным командам отладчика gdb. Ответ: `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; `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; `list` – 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. Ответ:

10. `gdb –silent ./calcul`
11. `run`
12. `list`
13. `backtrace`
14. `breakpoints`
15. `print Numeral`
16. Splint (Не использовался по причине отсутствия команды в консоли).
17. Прокомментируйте реакцию компилятора на синтаксические ошибки в программе при его первом запуске. Ответ: Консоль выводит ошибку с номером строки и ошибочным сегментом, но при этом есть возможность выполнить программу сразу.
18. Назовите основные средства, повышающие понимание исходного кода программы. Ответ:
  - a) Правильный синтаксис
  - b) Наличие комментариев
  - c) Разбиение большой сложной программы на несколько сегментов попроще.
12. Каковы основные задачи, решаемые программой splint? Ответ: split – разбиение файла на меньшие, определённого размера. Может разбивать текстовые файлы по строкам и любые – по байтам. По умолчанию читает со стандартного ввода и создает файлы с именами вида хаа, хаb и т.д. По умолчанию разбиение идёт по 1000 строк в файле.

## **Список литературы**