Лабораторная работа №13

1032224521

Атанесов Александр Николаевич

Содержание

# 1 Цель работы

Приобрести простейшие навыки разработки, анализа, тестирования и отладки при- ложений в ОС типа UNIX/Linux на примере создания на языке программирования С калькулятора с простейшими функциями.

# 2 Задание

1. Создать простейший калькулятор в UNIX;

# 3 Выполнение лабораторной работы

1. Создаю папку lab\_prog. (рис. [??])

Использую команду mkdir

Использую команду mkdir

## 3.1

1. Создаю файлы calculate.c calculate.h main.c. (рис. [??])

Использую команду touch

Использую команду touch

## 3.2

1. Делаем эти файлы исполняемыми. (рис. [??])

Использую команду chmod

Использую команду chmod

## 3.3

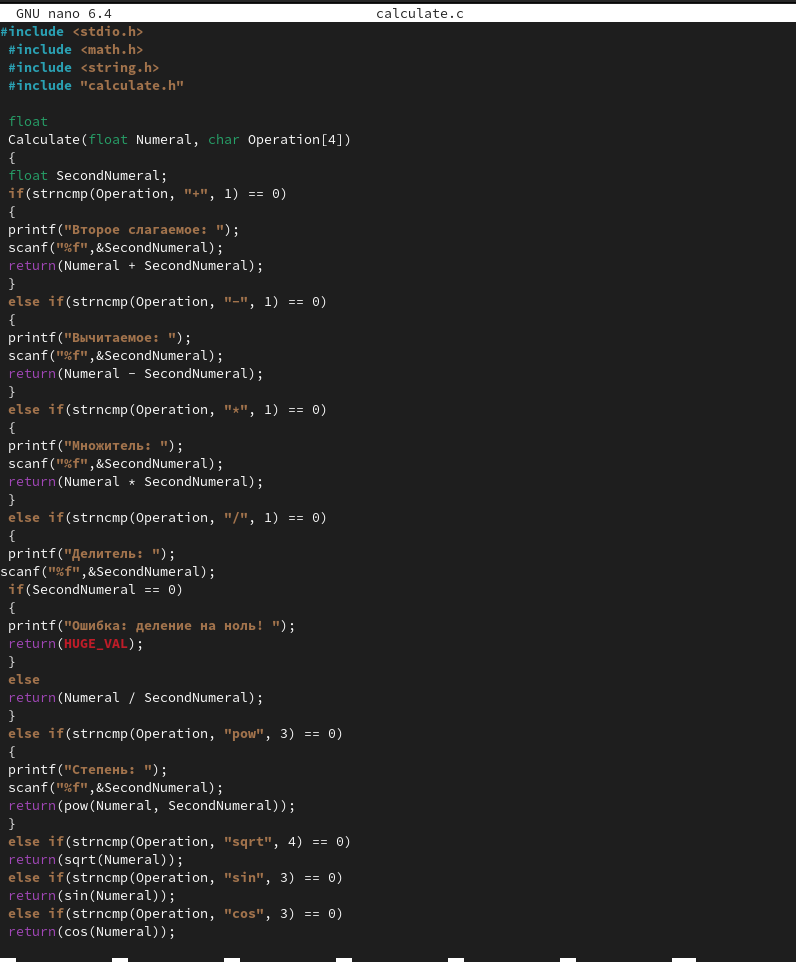
1. Открываю файл calculate.c через nano. (рис. [??])

Использую команду nano

Использую команду nano

## 3.4

1. Пишу код для будущего калькулятора. (рис. [??])



Использую редактор nano

## 3.5

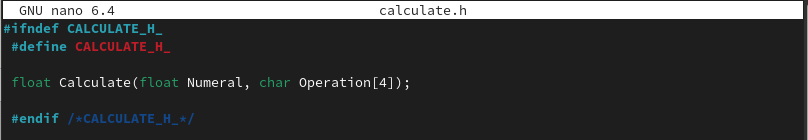
1. Открываю файл calculate.h через nano. (рис. [??])

Использую команду nano

Использую команду nano

## 3.6

1. Пишу необходимый код для calculate.h . (рис. [??])



Использую редактор nano

## 3.7

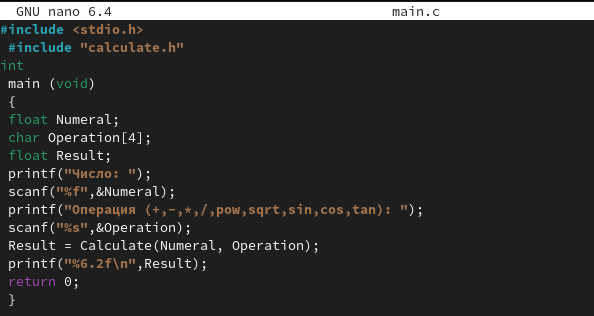
1. Открываю файл main.c через nano. (рис. [??])

Использую команду nano

Использую команду nano

## 3.8

1. Пишу необходимый код, представленный в выполнение лабораторной работы №13 . (рис. [??])



Использую редактор nano

## 3.9

1. Создаю файл Makefile. (рис. [??])

Использую команду chmod

Использую команду chmod

## 3.10

1. Открываю файл Makefile через nano. (рис. [??])

Использую команду nano

Использую команду nano

## 3.11

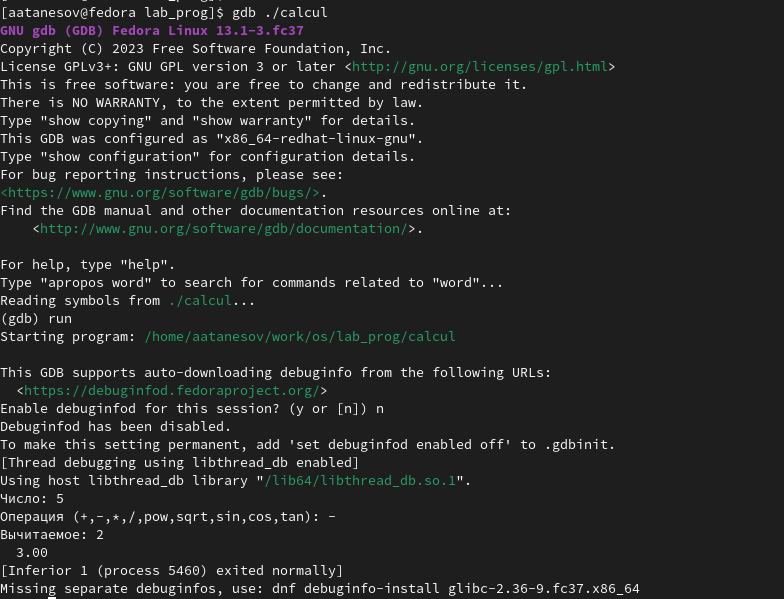
1. Пишу необходимый код. (рис. [??])



Использую редактор nano

## 3.12

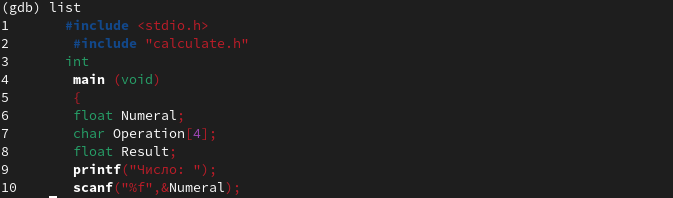
1. Запускаю файл калькулятор. (рис. [??])



Использую команду gdb./calcul, run

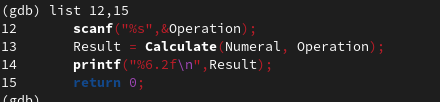
## 3.13

1. Вывожу 10 строк кода . (рис. [??])

(image/13.png){#fig:014 width=90%}

## 3.14

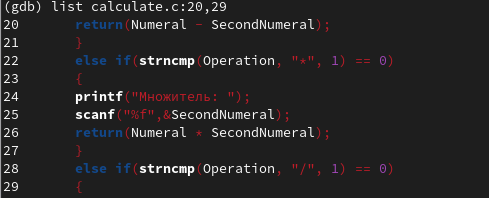
1. Вывожу с 12 по 15 строку кода нашей программы. (рис. [??])



Использую команду list 12,15

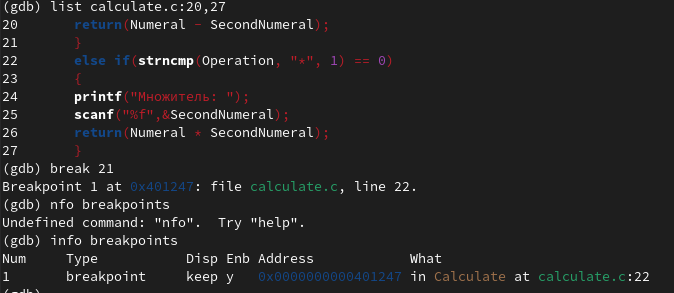
## 3.15

1. Вывожу с 20 по 29 строку кода программы. (рис. [??])

(image/18.png){#fig:018 width=90%}

## 3.16

1. Вывожу с 20 по 27 и ставлю точку остановки . (рис. [??])



Использую команду break

## 3.17

1. Вывожу автоматический анализ кода программ main.c и calculate.c . (рис. [??])



Использую команду splint

# 4 Выводы

* Я научился создавать простые приложения и открывать их через терминал .

# 5 Пояснение содержимого файла Makefile

* Данный Makefile описывает процесс сборки программы “calcul” из исходных файлов “calculate.c” и “main.c”, а также заголовочного файла “calculate.h”.

Переменные CC, CFLAGS и LIBS содержат информацию о компиляторе, флагах компиляции и необходимых для линковки библиотеках соответственно.

Цель “calcul” (строка 9) зависит от объектных файлов “calculate.o” и “main.o”, и собирается командой “gcc calculate.o main.o -o calcul $(LIBS)”.

Цели “calculate.o” и “main.o” (строки 12-16) компилируют соответствующие исходные файлы в объектные файлы.

Цель “clean” (строка 18) удаляет собранные объектные файлы и исполняемый файл.

Комментарии на каждой строке поясняют назначение каждой переменной или команды.

# 6 Ответы на контрольные вопросы

* 1. Для получения информации о возможностях программ gcc, make, gdb и др. можно обратиться к их официальной документации, доступной в сети Интернет, а также использовать команду man в терминале UNIX.
  2. Основными этапами разработки приложений в UNIX являются: проектирование, написание исходного кода, компиляция, отладка, тестирование, установка и настройка приложения.
  3. Суффикс в контексте языка программирования - это часть названия файла, указывающая на его тип и формат. Например, файл со суффиксом “.c” обозначает исходный код на языке С, а файл со суффиксом “.o” - скомпилированный объектный файл.
  4. Основное назначение компилятора языка С в UNIX - это компиляция исходного кода на этом языке в машинный код, который может быть выполнен на компьютере.
  5. Утилита make предназначена для автоматизации процесса сборки приложения из исходного кода и скомпилированных объектных файлов.
  6. Пример структуры Makefile:

TARGET = my\_program  
CC = gcc  
CFLAGS = -Wall -O2  
  
$(TARGET): main.o functions.o  
 $(CC) -o $(TARGET) main.o functions.o  
  
main.o: main.c functions.h  
 $(CC) $(CFLAGS) -c main.c  
  
functions.o: functions.c functions.h  
 $(CC) $(CFLAGS) -c functions.c

* Основные элементы Makefile:
* TARGET - имя целевого файла приложения;
* CC - имя компилятора;
* CFLAGS - опции компилятора;
* $(TARGET), $(CC), $(CFLAGS) - переменные, значения которых используются в правилах Makefile;
* $(TARGET): main.o functions.o - правило сборки, которое указывает, что для создания целевого файла $(TARGET) необходимы объектные файлы main.o и functions.o;
* main.o, functions.o - зависимости, т.е. файлы, которые необходимы для создания других файлов;
* main.c, functions.c, functions.h - исходные файлы приложения;
* $(CC) -o $(TARGET) main.o functions.o - команда сборки, которая говорит компилятору собрать файл $(TARGET) из - - объектных файлов main.o и functions.o;
* $(CC) $(CFLAGS) -c main.c - команда компиляции исходного файла main.c в объектный файл main.o с опциями $(CFLAGS).
  1. Основное свойство, присущее всем программам отладки - это возможность управления выполнением программы, т.е. - пошаговой отладки. Для использования этого свойства необходимо установить точки останова в программе и запустить - ее в режиме отладки.
  2. Основные команды отладчика gdb:
* run - запустить программу;
* break - установить точку останова в коде программы;
* next - выполнить следующую строку кода и остановиться;
* step - выполнить следующую строку кода и зайти в подпрограмму, если она вызывается в этой строке;
* print - вывести значение переменной;
* watch - установить точку останова на изменение значения переменной.
  1. Схема отладки программы:
  2. Компилирование программы с опцией -g, которая добавляет отладочную информацию к скомпилированному файлу.
  3. Запуск gdb с указанием имени скомпилированного файла.
  4. Установка точек останова в нужных местах с помощью команды break.
  5. Запуск программы с помощью команды run.
  6. Пошаговое выполнение программы с помощью команд next или step.
  7. Использование команды print для вывода значений переменных или выражений.
  8. Остановка выполнения программы с помощью команды break, если необходимо.
  9. Выход из gdb с помощью команды quit.
  10. Компилятор при первом запуске анализирует синтаксическую корректность исходного кода программы и выдает сообщения об ошибках, если они есть. Эти сообщения содержат информацию о месте ошибки и ее характере. Размер и сложность программы также могут влиять на время компиляции.
  11. Основные средства, повышающие понимание исходного кода программы - это комментарии в коде, описания функций и переменных, документация к проекту, а также использование среды разработки, которая облегчает чтение и изменение кода.
  12. Основные задачи, решаемые программой splint - это статический анализ исходного кода на языке С с целью выявления потенциальных ошибок, утечки памяти и других проблем, связанных с безопасностью, надежностью и качеством кода.

# Список литературы