Лабораторная работа №13

Лабораторная работа № 13. Средства, применяемые при разработке программного обеспечения в ОС типа UNIX/Linux

Старовойтов Егор Сергеевич

Содержание

[Цель работы 2](#_Toc104921110)

[Задание 2](#_Toc104921111)

[Теоретическое введение 3](#_Toc104921112)

[Этапы разработки приложений 3](#_Toc104921113)

[Ход работы 3](#_Toc104921114)

[Шаг 1 3](#_Toc104921115)

[Шаг 2 3](#_Toc104921116)

[Шаг 3 6](#_Toc104921117)

[Шаг 4 6](#_Toc104921118)

[Шаг 5 6](#_Toc104921119)

[Шаг 6 7](#_Toc104921120)

[Шаг 7 7](#_Toc104921121)

[Вывод 8](#_Toc104921122)

[Контрольные вопросы 8](#_Toc104921123)

[1. Как получить информацию о возможностях программ gcc, make, gdb и др.? 8](#_Toc104921124)

[2. Назовите и дайте краткую характеристику основным этапам разработки приложений в UNIX. 9](#_Toc104921125)

[3. Что такое суффикс в контексте языка программирования? Приведите примеры использования. 9](#_Toc104921126)

[4. Каково основное назначение компилятора языка С в UNIX? 9](#_Toc104921127)

[5. Для чего предназначена утилита make? 9](#_Toc104921128)

[6. Приведите пример структуры Makefile. Дайте характеристику основным элементам этого файла. 9](#_Toc104921129)

[7. Назовите основное свойство, присущее всем программам отладки. Что необходимо сделать, чтобы его можно было использовать? 10](#_Toc104921130)

[8. Назовите и дайте основную характеристику основным командам отладчика gdb. 10](#_Toc104921131)

[9. Опишите по шагам схему отладки программы, которую Вы использовали при выполнении лабораторной работы. 10](#_Toc104921132)

[10. Прокомментируйте реакцию компилятора на синтаксические ошибки в программе при его первом запуске. 10](#_Toc104921133)

[11. Назовите основные средства, повышающие понимание исходного кода программы. 10](#_Toc104921134)

[12. Каковы основные задачи, решаемые программой splint? 11](#_Toc104921135)

# 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.

# 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:

gcc -c calculate.c  
gcc -c main.c  
gcc calculate.o main.o -o calcul -lm

1. При необходимости исправьте синтаксические ошибки.
2. Создайте Makefile со следующим содержанием:

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 \*~

Поясните в отчёте его содержание. 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.

# Теоретическое введение

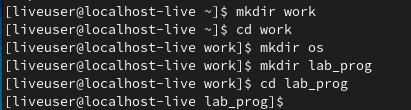
## Этапы разработки приложений

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.

# Ход работы

## Шаг 1

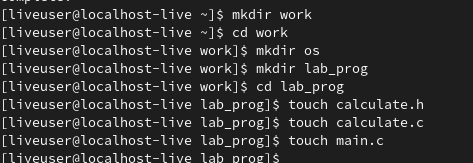
В домашнем каталоге создан подкаталог ~/work/os/lab\_prog



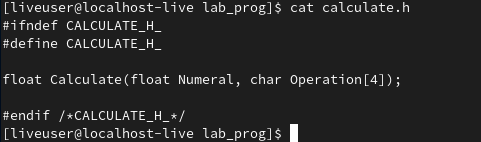
Создание подкаталога

## Шаг 2

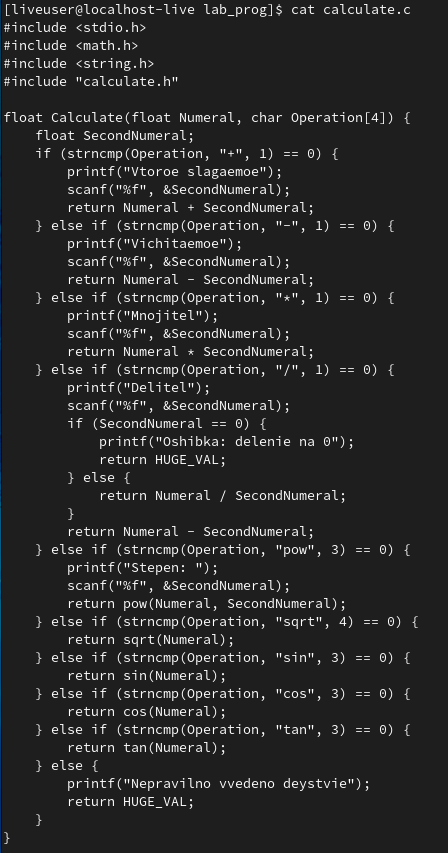
В каталоге ~/work/os/lab\_prog созданы файлы calculate.h, calculate.c, main.c.



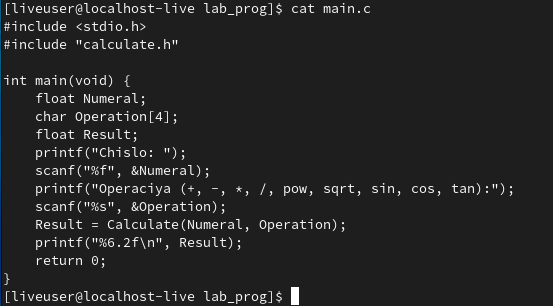
Создание файлов



calculate.h



calculate.c



main.c

## Шаг 3

Я выполнил компиляцию программы.

Компиляция программы

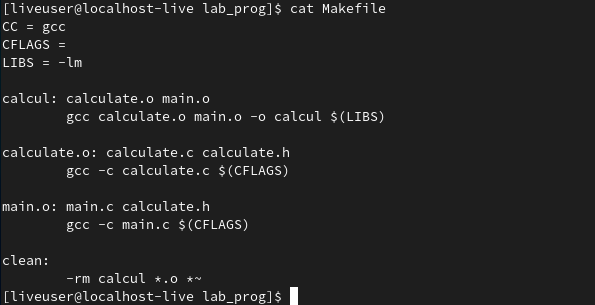
Компиляция программы

## Шаг 4

Исправил синтаксические ошибки (опечатки)

## Шаг 5

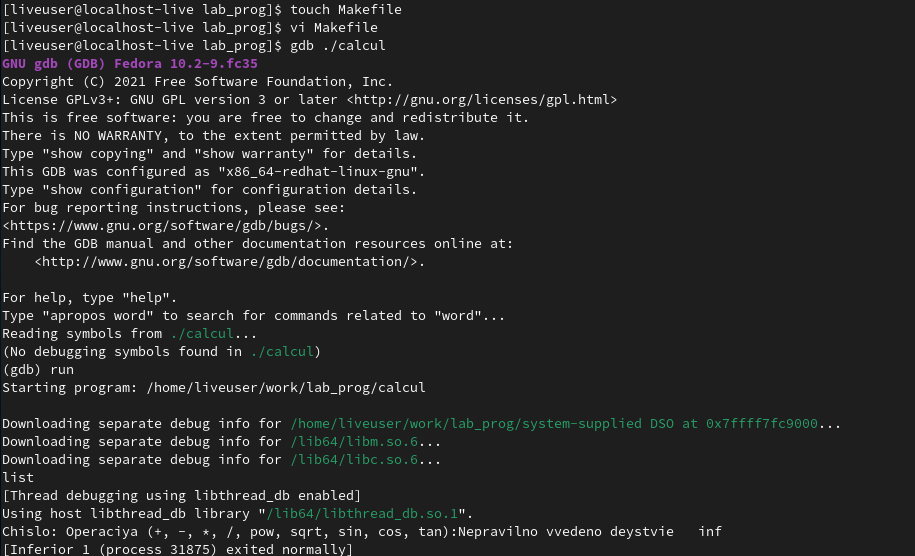
Создал Makefile



Makefile

## Шаг 6

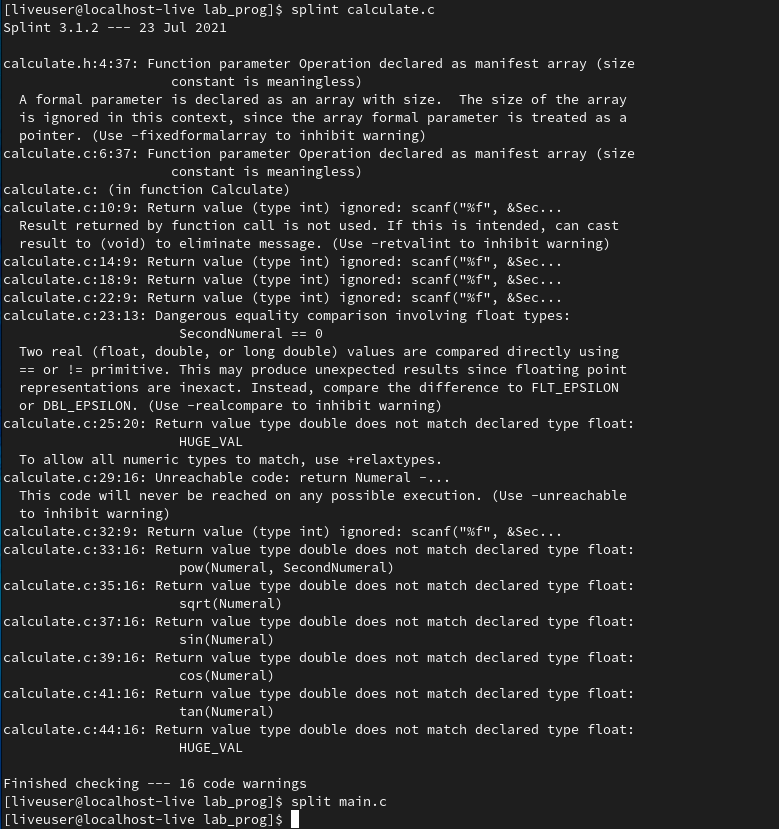
С помощью gdb выполнил отладку.



Отладка

## Шаг 7

С помощью утилиты splint проанализировал исходный код файлов calculate.c и main.c, увидел 16 предупреждений связанных с преобразованием типов в calculate.c.



Анализ исходного кода

# Вывод

я приобрел простейшие навыки разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linux на примере создания наязыке программирования C калькулятора с простейшими функциями.

# Контрольные вопросы

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

Спросить в интернете или использовать утилиту man, также можно использовать опцию -h у gcc для получения дополнительной информации.

## 2. Назовите и дайте краткую характеристику основным этапам разработки приложений в UNIX.

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. Что такое суффикс в контексте языка программирования? Приведите примеры использования.

Суффикс - это составная часть имени файла, например его расширение.

## 4. Каково основное назначение компилятора языка С в UNIX?

Преобразование исходного кода программ в объектные файлы.

## 5. Для чего предназначена утилита make?

Утилита make позволяет автоматизировать процесс преобразования файлов программы из одной формы в другую, отслеживает взаимосвязи между файлами.

## 6. Приведите пример структуры Makefile. Дайте характеристику основным элементам этого файла.

В самом простом случае Makefile имеет следующий синтаксис:

<цель\_1> <цель\_2> ... : <зависимость\_1> <зависимость\_2> ...  
<команда 1>  
...  
<команда n>

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. Рассмотрим пример Makefile для написанной выше простейшей программы, выводящей на экран приветствие ’Hello World!’:

hello: main.c  
gcc -o hello main.c

Здесь в первой строке hello — цель, main.c — название файла, который мы хотим скомпилировать; во второй строке, начиная с табуляции, задана команда компиляции gcc с опциями. Для запуска программы необходимо в командной строке набрать команду make:

make

Общий синтаксис Makefile имеет вид:  
1 target1 [target2...]:[:] [dependment1...]  
2 [(tab)commands] [#commentary]  
3 [(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 должна 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

## 7. Назовите основное свойство, присущее всем программам отладки. Что необходимо сделать, чтобы его можно было использовать?

Возможность останавливать выполнение программы на определенных строчках кода. Для этого нужно установить так называемые брейкпоинты.

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

Установка брейкопинтов, пошаговое выполнение отлаживаемой программы, вывод исходного кода постранично или построчно, возможность узнать значение переменных.

## 9. Опишите по шагам схему отладки программы, которую Вы использовали при выполнении лабораторной работы.

1. Установка брейкпоинтов
2. Пошаговое исполнение программы и вывод значения переменных

## 10. Прокомментируйте реакцию компилятора на синтаксические ошибки в программе при его первом запуске.

Процесс компиляции аварийно завершается, указывая на ошибки.

## 11. Назовите основные средства, повышающие понимание исходного кода программы.

Комментарии, единый стиль кода, линтеры.

## 12. Каковы основные задачи, решаемые программой splint?

Увидеть ошибки и предупреждения, указывающие на различные проблемы в коде программы.