Лабораторная работа №13

Гэинэ Андрей

Содержание

# 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:
4. При необходимости исправьте синтаксические ошибки.
5. Создайте Makefile со следующим содержанием.
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.

# Выполнение лабораторной работы

Создаем файл calculate.c.

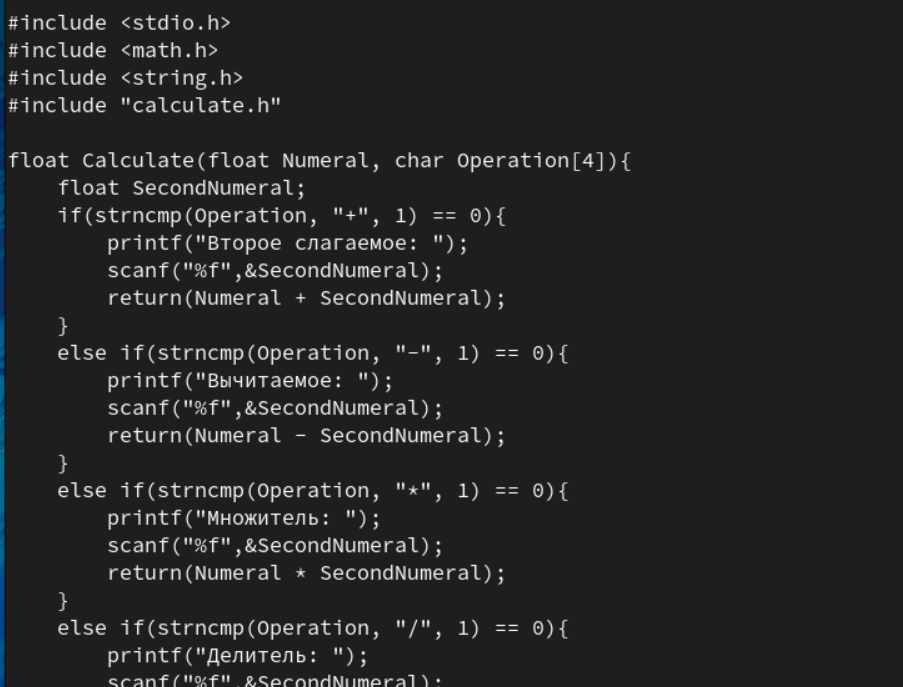


Рис.1

Создаем файл calculate.h.

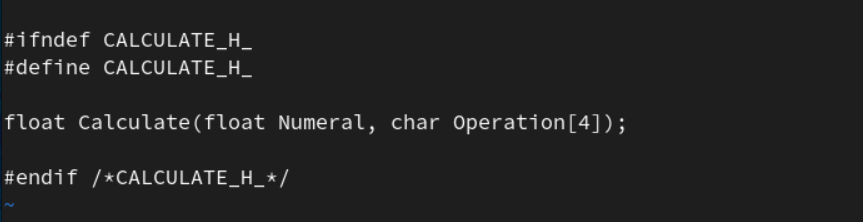


Рис.2

Создаем файл main.c.

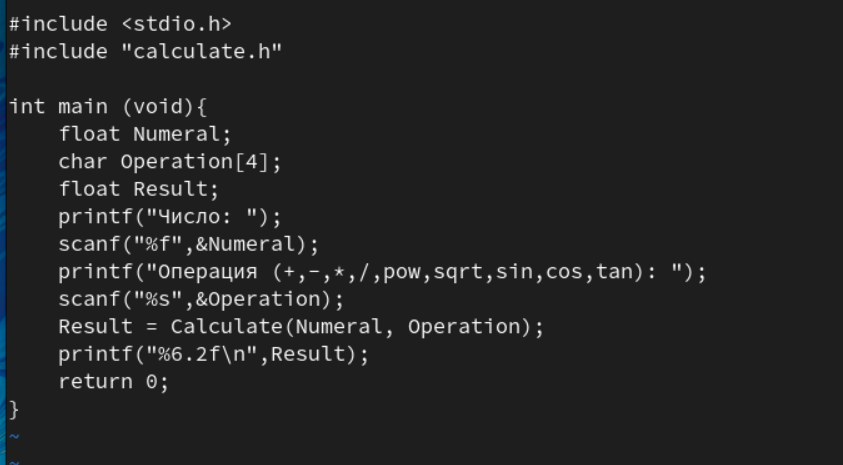


Рис.3

Выполняем компиляцию.

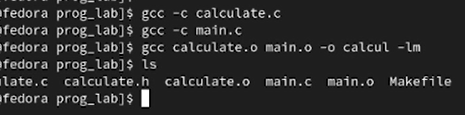


Рис.4

Создаем Makefile.

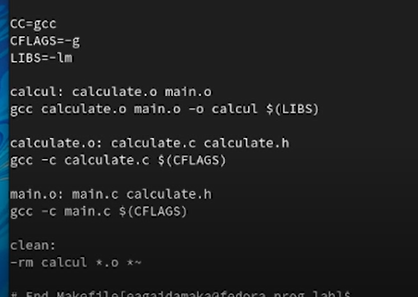


Рис.5

Проверяем работу программы с помощью gdb.

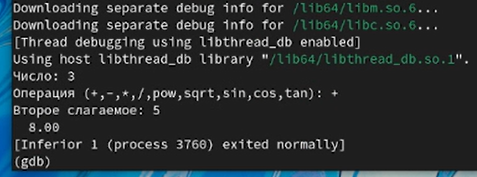


Рис.6

Смотрим информацию о файле calculate.c с помощью splint.

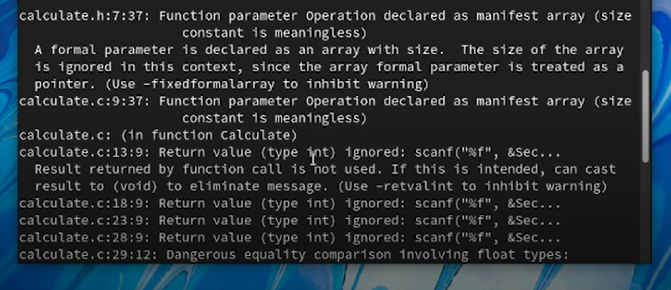


Рис.8

Смотрим информацию о файле main.c с помощью splint.

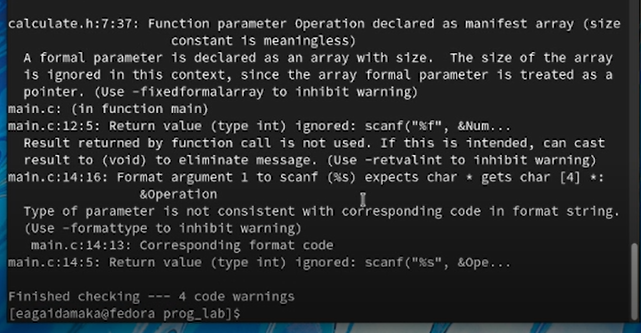


Рис.9

# Контрольные вопросы

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

man

1. 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.

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

Суффикс это составная часть имени файла. Система сборки каких-либо программ (например язык java) требует, чтобы имена файлов исходного кода заканчивались на .java.

1. Каково основное назначение компилятора языка С в UNIX?

Компилировать файлы c.

1. Для чего предназначена утилита make?

make — утилита предназначенная для автоматизации преобразования файлов из одной формы в другую.

1. Приведите пример структуры Makefile. Дайте характеристику основным элементам этого файла.

Пример Makefile:

#  
# Makefile  
#  
  
CC=gcc  
CFLAGS=-g  
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

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

Возможность останавливать выполнение программы и выводить информацию о переменных. В gdb можно ставить точки останова с помощью break.

1. 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;
* 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 кото- рых отображаются каждый раз при остановке программы;
* finish – выполняет программу до выхода из текущей функции; отоб- ражает 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 пере- даётся 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;

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

* Установите точку останова в файле calculate.c на строке номер 21:

list calculate.c:20,27  
break 21

* Выведите информацию об имеющихся в проекте точка останова:

info breakpoints

* Запустите программу внутри отладчика и убедитесь, что программа остановится в момент прохождения точки останова:

run  
5  
-  
backtrace

– Отладчик выдаст следующую информацию:

#0 Calculate (Numeral=5, Operation=0x7fffffffd280 "-")  
at calculate.c:21  
#1 0x0000000000400b2b in main () at main.c:17

а команда backtrace покажет весь стек вызываемых функций от начала программы до текущего места. - Посмотрите, чему равно на этом этапе значение переменной Numeral, введя:

print Numeral

На экран должно быть выведено число 5. - Сравните с результатом вывода на экран после использования команды:

display Numeral

* Уберите точки останова:

info breakpoints  
delete 1

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

У меня не было ошибок, так как я изначально исправила их в файлах.

1. Назовите основные средства, повышающие понимание исходного кода программы.

Для статической проверки программ на языке C на наличие уязвимостей в системе безопасности и типичных ошибок программирования можно использовать splint.

1. Каковы основные задачи, решаемые программой splint?

Splint — это инструмент для статической проверки программ на языке C на наличие уязвимостей в системе безопасности и типичных ошибок программирования. С минимальными усилиями Splint можно использовать в качестве лучшего lint(1). Если приложить дополнительные усилия для добавления аннотаций к программам, Splint может выполнять более строгие проверки, чем любой стандартный lint. Полную документацию см. на http://www.splint.org.

# Выводы

Благодаря данной работе мы приобрели простейшие навыки разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linux на примере создания на языке программирования С калькулятора с простейшими функциями.