Отчёт по лабораторной работе №13

Средства для создания приложений в ОС UNIX

Лапасов Зелимхан Абитханович НБИ-БД-01-20

Содержание

[1 Цель работы 1](#_Toc146536303)

[2 Выполнение лабораторной работы 1](#_Toc146536304)

[3 Вывод 7](#_Toc146536305)

[4 Контрольные вопросы 7](#_Toc146536306)

# 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. Создали подкаталог для файлов лаб работы
2. Создал 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.

Код файла calculate.с (реализует функции калькулятора)

////////////////////////////////////  
// calculate.c  
  
#include <stdio.h>  
#include <math.h>  
#include <string.h>  
#include "calculate.h"  
  
Float Calculate(float Numeral, char Operation[4])  
{  
 float SecondNumeral;  
 if(strncmp(Operation, "+", 1) == 0)  
 {  
 printf("Второе слагаемое: ");  
 scanf("%f",&SecondNumeral);  
 return(Numeral + SecondNumeral);  
 }  
 else if(strncmp(Operation, "-", 1) == 0)  
 {  
 printf("Вычитаемое: ");  
 scanf("%f",&SecondNumeral);  
 return(Numeral - SecondNumeral);  
 }  
 else if(strncmp(Operation, "\*", 1) == 0)  
 {  
 printf("Множитель: ");  
 scanf("%f",&SecondNumeral);  
 return(Numeral \* SecondNumeral);  
 }  
 else if(strncmp(Operation, "/", 1) == 0)  
 {  
 printf("Делитель: ");  
 scanf("%f",&SecondNumeral);  
 if(SecondNumeral == 0)  
 {  
 printf("Ошибка: деление на ноль! ");  
 return(HUGE\_VAL);  
 }  
 else  
 return(Numeral / SecondNumeral);  
 }  
 else if(strncmp(Operation, "pow", 3) == 0)  
 {  
 printf("Степень: ");  
 scanf("%f",&SecondNumeral);  
 return(pow(Numeral, SecondNumeral));  
 }  
 else if(strncmp(Operation, "sqrt", 4) == 0)  
 return(sqrt(Numeral));  
 else if(strncmp(Operation, "sin", 3) == 0)  
 return(sin(Numeral));  
 else if(strncmp(Operation, "cos", 3) == 0)  
 return(cos(Numeral));  
 else if(strncmp(Operation, "tan", 3) == 0)  
 return(tan(Numeral));  
 else  
 {  
 printf("Неправильно введено действие ");  
 return(HUGE\_VAL);  
 }  
}

Код файла calculate.h (описывает формат вызова функции калькулятора)

///////////////////////////////////////  
// calculate.h  
#ifndef CALCULATE\_H\_  
#define CALCULATE\_H\_  
float Calculate(float Numeral, char Operation[4]);  
#endif /\*CALCULATE\_H\_\*/  
  
Код файла main.c (реализует интерфейс пользователя к калькулятору)  
////////////////////////////////////////  
// main.c  
  
#include <stdio.h>  
#include "calculate.h"  
  
Int main (void)  
{  
 float Numeral;  
 char Operation[4];  
 float Result;  
 printf("Число: ");  
 scanf("%f",&Numeral);  
 printf("Операция (+,-,\*,/,pow,sqrt,sin,cos,tan): ");  
 scanf("%s",&Operation);  
 Result = Calculate(Numeral, Operation);  
 printf("%6.2f\n",Result);  
 return 0;  
}

1. Выполнили компиляцию программы посредством gcc :

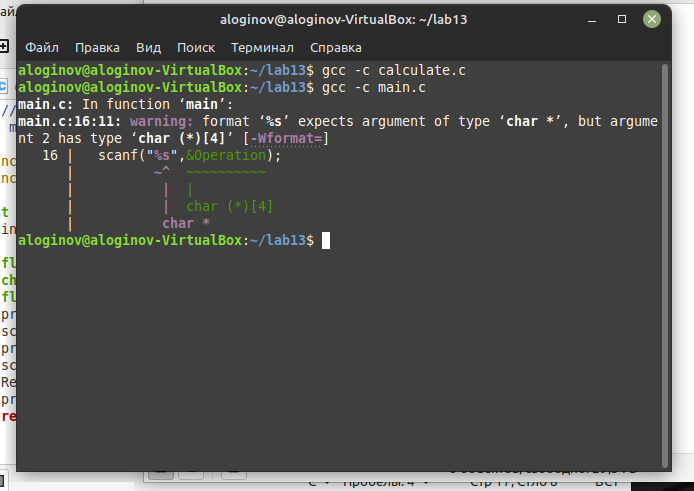


Figure 1: Компиляция

1. При необходимости исправили синтаксические ошибки.
2. Создали Makefile со следующим содержанием:

#  
# 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

С помощью программы make получаем различные варианты построения исполняемого модуля.

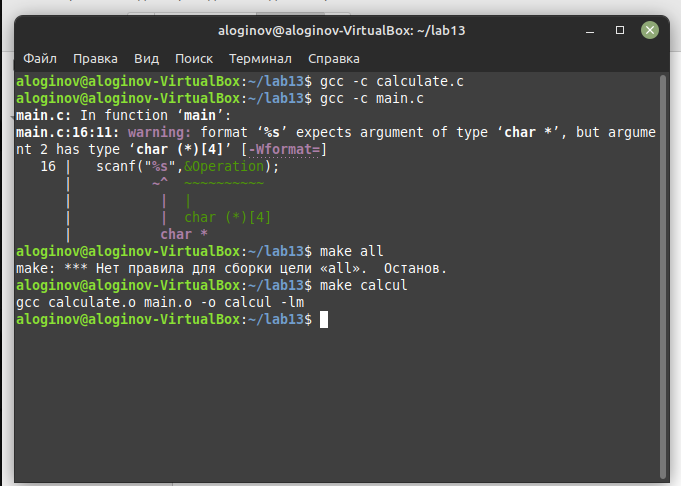


Figure 2: Использование make

1. С помощью gdb выполнил отладку программы calcul

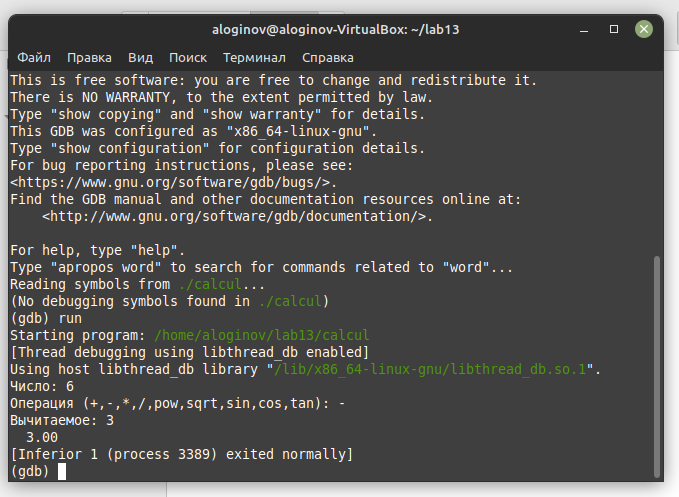


Figure 3: Использование отладчика

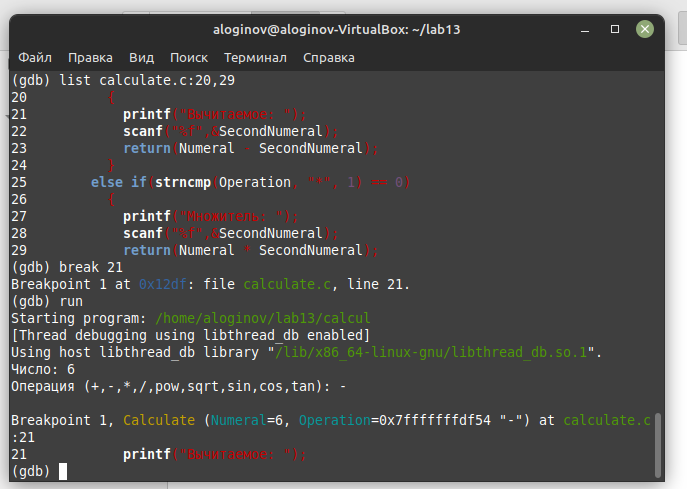


Figure 4: Использование отладчика

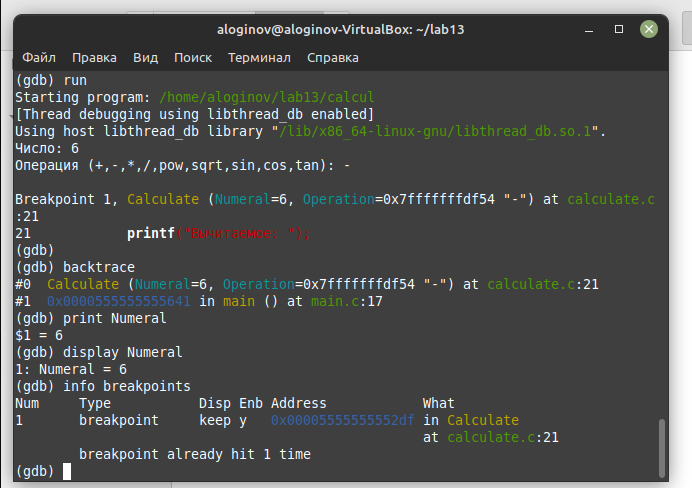


Figure 5: Использование отладчика

1. С помощью утилиты splint попробовали проанализировать коды файлов

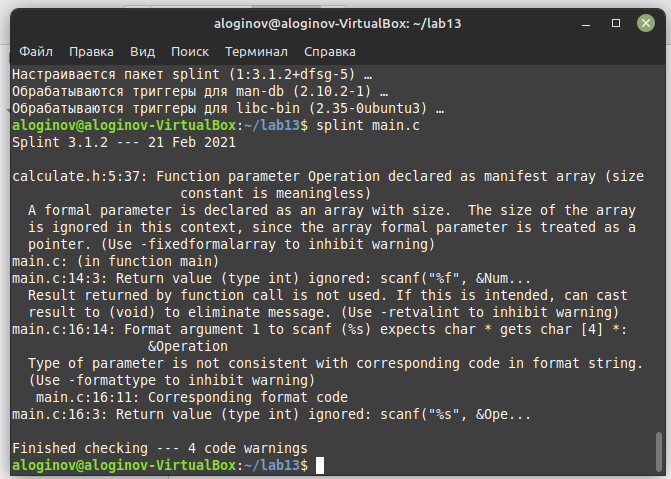


Figure 6: Использование splint

# 3 Вывод

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.

# 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 прило- жений 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 С 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 неуказана 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 «сс -с» с указанием в параметрах этого «.c»-файла и всех файлов-зависимостей.

1. 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.
2. 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;

1. Опишите по шагам схему отладки программы, которую Вы использовали при выполнении лабораторной работы. Ответ:
2. gdb –silent ./calcul
3. run
4. list
5. backtrace
6. breakpoints
7. print Numeral
8. Splint (Не использовался по причине отсутствия команды в консоли).
9. 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.
10. Назовите основные средства, повышающие понимание исходного кода программы. Ответ:
11. Правильный синтаксис
12. Наличие комментариев
13. Разбиение большой сложной программы на несколько сегментов попроще.
14. 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 xaa, xab и т.д. По умолчанию разбиение идёт по 1000 строк в файле.