Лабораторная работа №14 по предмету

Операционные системы

Группа НПМбв-01-19

Бондаренко Артем Федорович

Содержание

|  |  |
| --- | --- |
| [Цель работы](#page5) | 5 |
| [Задание](#page6) | 6 |
| [Выполнение лабораторной работы](#page13) | 13 |
| [Выводы](#page21) | 21 |
| [Ответы на контрольные вопросы](#page22) | 22 |

2

Список иллюстраций

* [Процесс создания директорий в терминале](#page13) . . . . . . . . . . . . . . . . 13

[2 Процесс создания необходимых файлов в терминале](#page13) . . . . . . . . . . 13

|  |  |  |
| --- | --- | --- |
| [3](#page14) | [Фрагмент одного из заполненных файлов](#page14) . . . . . . . . . . . . . . . . | 14 |
| [4](#page14) | [Результат компиляции](#page14) . . . . . . . . . . . . . . . . . . . . . . . . . . . | 14 |
| [5](#page15) | [Makefile с содержанием из лабораторной работы](#page15) . . . . . . . . . . . . | 15 |

* [Исправленный Makefile](#page16) . . . . . . . . . . . . . . . . . . . . . . . . . . . 16

|  |  |  |
| --- | --- | --- |
| [7](#page16) | [Запуск отладчика и введённая команда run](#page16) . . . . . . . . . . . . . . . | 16 |
| [8](#page17) | [Результат выполнения команд в терминале](#page17) . . . . . . . . . . . . . . . | 17 |

* [Команда list с параметрами calculate.c:20,29](#page17) . . . . . . . . . . . . . . . 17

10 [Установка точки останова](#page18) . . . . . . . . . . . . . . . . . . . . . . . . . 18

1. [Информацию о точках останова](#page18) . . . . . . . . . . . . . . . . . . . . . . 18
2. [Информация о точках останова](#page18) . . . . . . . . . . . . . . . . . . . . . . 18
3. [Информация по backtrace](#page18) . . . . . . . . . . . . . . . . . . . . . . . . . . 18
4. [Информация после ввода print Numeral и display Numeral](#page19) . . . . . . . 19
5. [Удаление точек останова](#page19) . . . . . . . . . . . . . . . . . . . . . . . . . . 19
6. [Анализ кода файла calculate.c утилитой splint](#page19) . . . . . . . . . . . . . . 19
7. [Анализ кода файла main.c утилитой splint](#page20) . . . . . . . . . . . . . . . . 20

3

Список таблиц

4

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.

5

Задание

1. В домашнем каталоге создайте подкаталог ~/work/os/lab\_prog.
2. Создайте в нём файлы: calculate.h, calculate.c, main.c. Это будет примитив-нейший 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.

Реализация функций калькулятора в файле calculate.h:

////////////////////////////////////

* 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);

6

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: ");

7

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);

}

}

Интерфейсный файл calculate.h, описывающий формат вызова функциикалькуля-тора:

///////////////////////////////////////

// calculate.h

#ifndef CALCULATE\_H\_

#define CALCULATE\_H\_

float Calculate(float Numeral, char Operation[4]); #endif /\*CALCULATE\_H\_\*/

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

////////////////////////////////////////

8

// 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;

}

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:

#

* Makefile

CC = gcc

9

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 \*~

# End Makefile

Поясните в отчёте его содержание.

* 1. С помощью gdb выполните отладку программы calcul (перед использованием gdb исправьте Makefile):

– Запустите отладчик GDB, загрузив в него программу для отладки:

gdb ./calcul

– Для запуска программы внутри отладчика введите команду run: run

– Для постраничного (по 9 строк) просмотра исходного код используйте команду list:

list

– Для просмотра строк с 12 по 15 основного файла используйте list с параметрами: list 12,15

10

– Для просмотра определённых строк не основного файла используйте list с параметрами:

list calculate.c:20,29

– 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

* 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

* 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

11

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. С помощью утилиты splint попробуйте проанализировать коды файлов calculate.c и main.c.

12

Выполнение лабораторной работы

В домашнем каталоге создал подкаталог ~/work/os/lab\_prog. (Ссылка: Рис. 1)



Рис. 1: Процесс создания директорий в терминале

Создал в нём файлы: calculate.h, calculate.c, main.c. (Ссылка: Рис. 2)

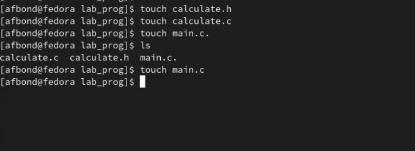


Рис. 2: Процесс создания необходимых файлов в терминале

Каждый из созданных файлов заполнил кодом из лабораторной работы. (Ссылка:

Рис. 3)

13

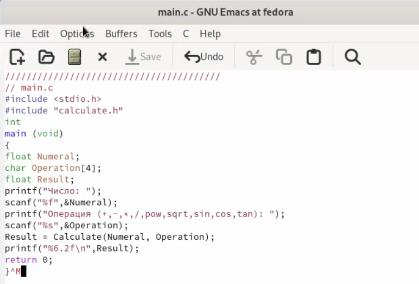


Рис. 3: Фрагмент одного из заполненных файлов

Получился примитивнейший калькулятор, способный складывать, вычитать, умно-жать 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.

Выполнил компиляцию программы посредством gcc. (Ссылка: Рис. 4)

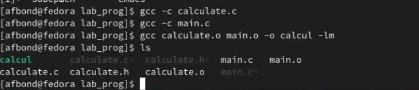


Рис. 4: Результат компиляции

Создал Makefile со следующим содержанием: (Ссылка: Рис. 5)

14

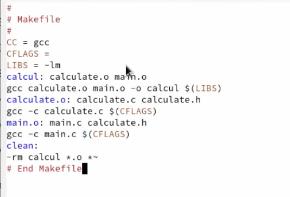


Рис. 5: Makefile с содержанием из лабораторной работы

Пояснение по Makefile, он используется для сборки программы calcul.

* 1. = gcc: это задание переменной CC, которая используется для определения компилятора.

CFLAGS =: это задание переменной CFLAGS, которая используется для добавле-ния дополнительных флагов компилятора.

LIBS = -lm: это задание переменной LIBS, которая используется для добавления библиотек в программу.

calcul: calculate.o main.o: это задание цели calcul, которая зависит от файлов calculate.o и main.o.

gcc calculate.o main.o -o calcul $(LIBS): это команда сборки для создания исполня-емого файла calcul.

calculate.o: calculate.c calculate.h: это задание для сборки файла calculate.o из файлов calculate.c и calculate.h.

gcc -c calculate.c $(CFLAGS): это команда компиляции для создания файла calculate.o.

main.o: main.c calculate.h: это задание для сборки файла main.o из файлов main.c

* calculate.h.

gcc -c main.c $(CFLAGS): это команда компиляции для создания файла main.o.

clean: -rm calcul .o ~: это задание для очистки (удаления) всех созданных файлов и объектов.

15

* помощью gdb выполняю отладку программы calcul (перед использованием gdb исправил Makefile(Ссылка: Рис. 6)):

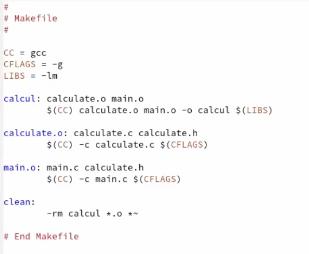


Рис. 6: Исправленный Makefile

Запустил отладчик GDB, загрузив в него программу для отладки. (Ссылка: Рис.

7)

Для запуска программы внутри отладчика ввел команду run.(Ссылка: Рис. 7)

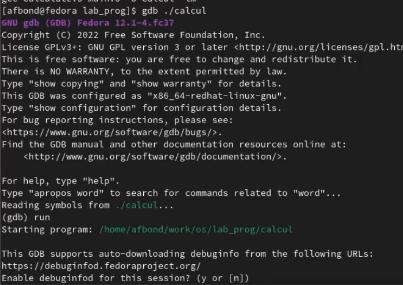


Рис. 7: Запуск отладчика и введённая команда run

16

Для постраничного (по 9 строк) просмотра исходного код использовал команду list.(Ссылка: Рис. 8)

Для просмотра строк с 12 по 15 основного файла использовал list с параметрами 12,15. (Ссылка: Рис. 8)

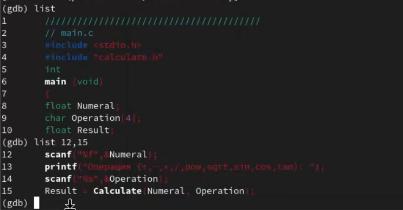


Рис. 8: Результат выполнения команд в терминале

Для просмотра определённых строк не основного файла использовал list с пара-метрами calculate.c:20,29. (Ссылка: Рис. 9)

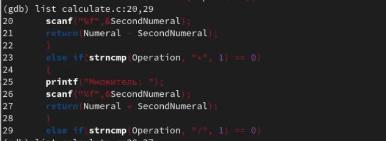


Рис. 9: Команда list с параметрами calculate.c:20,29

Установил точку останова в файле calculate.c на строке номер 21. (Ссылка: Рис. 10)

17

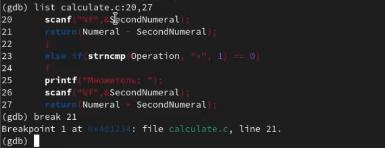


Рис. 10: Установка точки останова

Вывел информацию об имеющихся в проекте точка останова. (Ссылка: Рис. 11)



Рис. 11: Информацию о точках останова

Запустил программу внутри отладчика и убедился, что программа остановилась в момент прохождения точки останова. (Ссылка: Рис. 12)

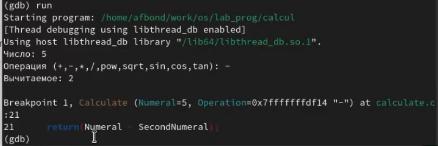


Рис. 12: Информация о точках останова

Ввёл backtrace и отладчик выдал следующую информацию: (Ссылка: Рис. 13)



Рис. 13: Информация по backtrace

18

Посмотрел, чему равно на этом этапе значение переменной Numeral, введя: print Numeral. Затем ввёл display Numeral и сравнил результаты. И там и там значение 5, но с небольшими отличиями. (Ссылка: Рис. 14)

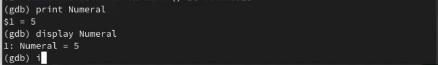


Рис. 14: Информация после ввода print Numeral и display Numeral

Убрал точки останова (Ссылка: Рис. 15)



Рис. 15: Удаление точек останова

* + помощью утилиты splint попробовал проанализировать коды файлов calculate.c

main.c. (Ссылка: Рис. 16) (Ссылка: Рис. 17)

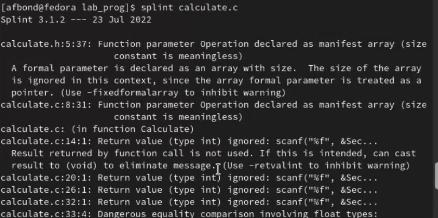


Рис. 16: Анализ кода файла calculate.c утилитой splint

19

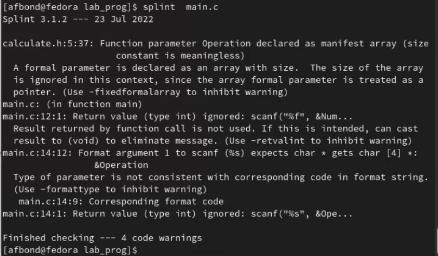


Рис. 17: Анализ кода файла main.c утилитой splint

20

Выводы

Таким образом, были приобретены простейшие навыки разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linux на примере создания на языке программирования С калькулятора с простейшими функциями.

21

Ответы на контрольные вопросы

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

Для получения информации о возможностях программ gcc, make, gdb и дру-гих утилит, установленных в системе, можно использовать команды man и info в терминале Linux.

Команда man позволяет просмотреть мануал (руководство пользователя) по конкретной утилите.

1. Назовите и дайте краткую характеристику основным этапам разработки при-ложений 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 специ-фикаций, 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.

22

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

Суффикс в контексте языка программирования - это часть имени файла, которая указывает на его тип или назначение. Например, в операционной системе Unix/Linux суффикс “.c” обычно используется для исходных файлов программ на языке С,

* суффикс “.o” - для объектных файлов, получаемых в результате компиляции исходных файлов.

Еще одним примером может быть суффикс “.py” для файлов программ на языке Python, суффикс “.java” для файлов программ на языке Java, и т.д.

Суффиксы используются для облегчения процесса компиляции, линковки и сбор-ки программного кода, так как позволяют автоматически определять тип файлов и выполнять соответствующие действия. Кроме того, суффиксы также использу-ются для определения типа файлов в операционных системах и приложениях, что позволяет им обрабатывать файлы соответствующим образом.

4. Каково основное назначение компилятора языка С в UNIX?

Основное назначение компилятора языка C в UNIX - это преобразование исход-ного кода программы, написанной на языке C, в машинный код, который может быть исполнен процессором компьютера. Компилятор языка C является одним из основных инструментов разработки программного обеспечения в UNIX-подобных операционных системах. Он позволяет разработчикам писать эффективный и пор-тируемый код на языке C, который может быть выполнен на различных аппаратных платформах и операционных системах. Кроме того, компилятор языка C может использоваться для оптимизации кода, анализа его качества и обнаружения ошибок во время компиляции.

5. Для чего предназначена утилита make?

Утилита make предназначена для автоматизации процесса компиляции программ. Она позволяет автоматически определять, какие файлы программы были изменены,

23

* перекомпилировать только их, а также устанавливать программу в систему и за-пускать тесты. В результате использования утилиты make значительно сокращается время, необходимое для сборки программы, и уменьшается вероятность ошибок, возникающих при ручной компиляции. Основным принципом работы make является использование файла Makefile, в котором задаются правила сборки программы и зависимости между файлами.
  1. Приведите пример структуры Makefile. Дайте характеристику основным эле-ментам этого файла.

Общий синтаксис Makefile имеет вид:

target1 [target2. . . ]:[:] [dependment1. . . ]

[(tab)commands] [#commentary]

[(tab)commands] [#commentary]

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

Основным свойством, присущим всем программам отладки, является возмож-ность остановки выполнения программы на определенном месте, наблюдения за ее состоянием и изменения этого состояния в процессе выполнения.

Чтобы использовать это свойство, необходимо подготовить исходный код про-граммы и запустить ее в среде отладки, установив точки останова в нужных местах. При достижении точки останова выполнение программы приостанавливается, и пользователь может проанализировать текущее состояние программы и выполнить необходимые действия для ее дальнейшего исполнения.

1. Назовите и дайте основную характеристику основным командам отладчика gdb.

backtrace - вывод на экран пути к текущей точке останова (по сути вывод названий всех функций)

24

break - установить точку останова (в качестве параметра может быть указан номер строки или название функции)

clear - удалить все точки останова в функции continue продолжить выполнение программы

delete - удалить точку останова

display - добавить выражение в список выражений, значения которых отобража-ются при достижении точки останова программы

finish - выполнить программу до момента выхода из функции

info breakpoints - вывести на экран список используемых точек останова

info watchpoints - вывести на экран список используемых контрольных выражений list - вывести на экран исходный код (в качестве параметра может быть указано

название файла и через двоеточие номера начальной и конечной строк)

next - выполнить программу пошагово, но без выполнения вызываемых в про-грамме функций

print - вывести значение указываемого в качестве параметра выражения run - запуск программы на выполнение set - установить новое значение переменной

step - пошаговое выполнение программы watch установить контрольное выражение, при изменении значения которого программа будет остановлена

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

Исправил Makwfile, после запустил отладку gdb ./calcul проверил на работо-способность программу, установил точку останова и удалил её, далее попробовал проанализировать код с помощью утилиты splint.

1. Прокомментируйте реакцию компилятора на синтаксические ошибки в про-грамме при его первом запуске.

Компилятор не выдал никаких ошибок при первом компилированнии созданных файлов через gcc. Сам исполняемый файл работал корректно.

25

1. Назовите основные средства, повышающие понимание исходного кода програм-мы.

Основные средства, которые могут помочь повысить понимание исходного кода программы, включают:

Комментарии: они могут предоставить дополнительную информацию о том, что делает код и как он это делает.

Документация: она может содержать инструкции по использованию программы, описание функций, классов и методов, которые содержатся в программе, и другую полезную информацию.

Отладчики: они позволяют выполнять код по шагам, отслеживать значения пере-менных и тестировать различные сценарии выполнения программы.

Редакторы кода: они обычно имеют функции подсветки синтаксиса, автодопол-нения и форматирования кода, которые упрощают чтение и понимание исходного кода.

Программы анализа кода: такие программы могут автоматически проверять исход-ный код на наличие ошибок, обнаруживать недостатки и предлагать рекомендации по улучшению кода.

Книги и руководства: они могут содержать общие принципы программирования, описывать конкретные алгоритмы и структуры данных, а также предоставлять примеры кода для изучения и анализа.

12. Каковы основные задачи, решаемые программой splint?

Splint (Secure Programming Lint) - это инструмент статического анализа кода на языке C, который помогает обнаруживать ошибки в коде, связанные с безопасностью и потенциальными уязвимостями.

Основные задачи, решаемые программой Splint, включают:

Поиск ошибок в коде: Splint может обнаружить ошибки в коде, такие как неиници-ализированные переменные, неправильное использование указателей, переполнение буфера и другие.

26

Анализ потока данных: Splint может проанализировать поток данных в программе

* найти возможные ошибки в обработке данных, такие как некорректное приведение типов, ошибки при работе с файлами и сетью и другие.

Проверка безопасности: Splint может проверять код на наличие уязвимостей безопасности, связанных с ошибками в обработке пользовательского ввода, уязви-мостями буфера и другими проблемами безопасности.

Проверка соответствия стандартам: Splint может проверять код на соответствие стандартам языка C, таким как ANSI C и POSIX.

Улучшение качества кода: Splint помогает улучшить качество кода, предостав-ляя рекомендации по улучшению стиля программирования и снижению уровня сложности кода.

27