Отчёт по лабораторной работе №13

Средства, применяемые при разработке программного обеспечения в ОС типа UNIX/Linux

Грузинова Елизавета Константиновна, НКНбд-02-21

Содержание

# 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 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. Тексты реализаций функций калькулятора в файле calculate.h, интерфейсного файла calculate.h, описывающий формат вызова функции-калькулятора и основного файла main.c, реализующий интерфейс пользователя к калькулятору приведены при выполнении лабораторной работы.

1. Выполните компиляцию программы посредством gcc:

1 gcc -c calculate.c 2 gcc -c main.c 3 gcc calculate.o main.o -o calcul -lm

1. При необходимости исправьте синтаксические ошибки.
2. Создайте Makefile со следующим содержанием (текст также приведен при выполнении лабораторной работы). Поясните в отчёте его содержание.
3. С помощью gdb выполните отладку программы calcul (перед использованием gdb исправьте Makefile):

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

1 gdb ./calcul

– Для запуска программы внутри отладчика введите команду run:

1 run

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

1 list

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

1 list 12,15

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

1 list calculate.c:20,29

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

1 list calculate.c:20,27

2 break 21

– Выведите информацию об имеющихся в проекте точка останова:

1 info breakpoints

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

1 run 2 5 3 - 4 backtrace

– Отладчик выдаст следующую информацию:

1 #0 Calculate (Numeral=5, Operation=0x7fffffffd280 “-”) 2 at calculate.c:21 3 #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

1 print Numeral

На экран должно быть выведено число 5.

– Сравните с результатом вывода на экран после использования команды:

1 display Numeral

– Уберите точки останова:

1 info breakpoints 2 delete 1

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

# 3 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. [1]

# 4 Выполнение лабораторной работы

1. В домашнем каталоге создание подкаталога ~/work/os/lab\_prog. (рис. 1)

Рис. 1: Создала каталог lab_prog и перешла в него

Рис. 1: Создала каталог lab\_prog и перешла в него

1. Создание в нём файлов: calculate.h, calculate.c, main.c. (рис. 2)

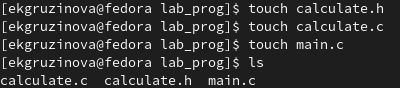


Рис. 2: Создала необходимые файлы и проверила с помощью ls

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. Тексты реализаций функций калькулятора в файле calculate.h, интерфейсного файла calculate.h, описывающий формат вызова функции-калькулятора и основного файла main.c, реализующий интерфейс пользователя к калькулятору приведены при выполнении лабораторной работы. (рис. 3-7)

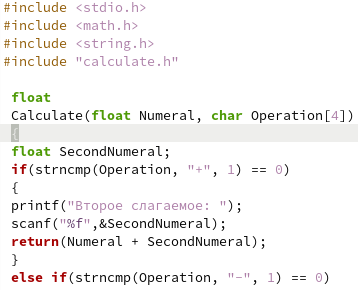


Рис. 3: Текст файла calculate.c (1)

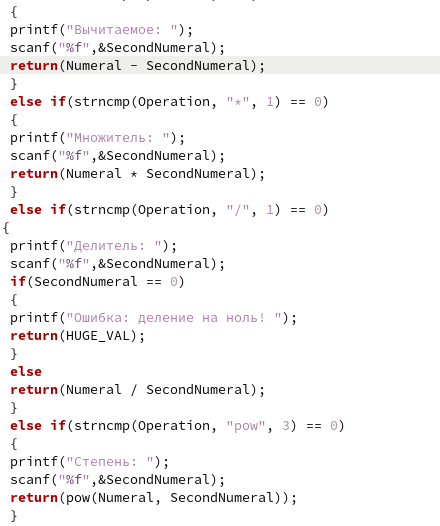


Рис. 4: Текст файла calculate.c (2)

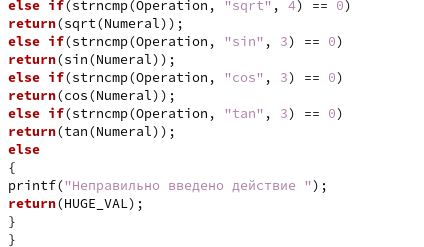


Рис. 5: Текст файла calculate.c (3)

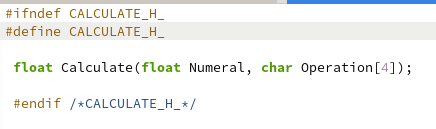


Рис. 6: Текст файла calculate.h (1)

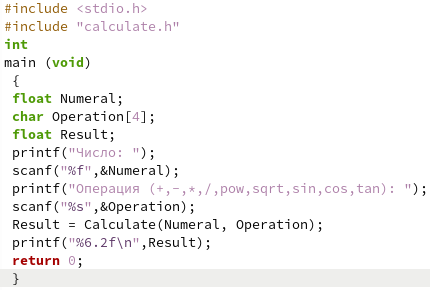


Рис. 7: Текст файла main.c

1. Выполнение компиляции программы посредством gcc (рис. 8):

1 gcc -c calculate.c 2 gcc -c main.c 3 gcc calculate.o main.o -o calcul -lm

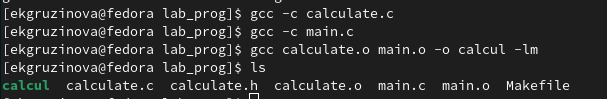


Рис. 8: Компиляция через терминал

1. При необходимости исправьте синтаксические ошибки (их не было обнаружено).
2. Создание Makefile со следующим содержанием (текст также приведен при выполнении лабораторной работы). Поясните в отчёте его содержание. (рис. 9)

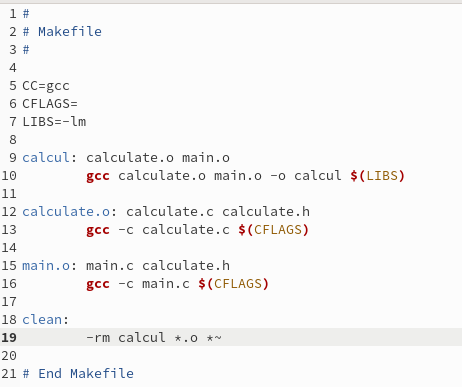


Рис. 9: Текст в файле Makefile

1. С помощью gdb выполнение отладки программы calcul (перед использованием gdb исправьте Makefile) (рис. 10, 11):

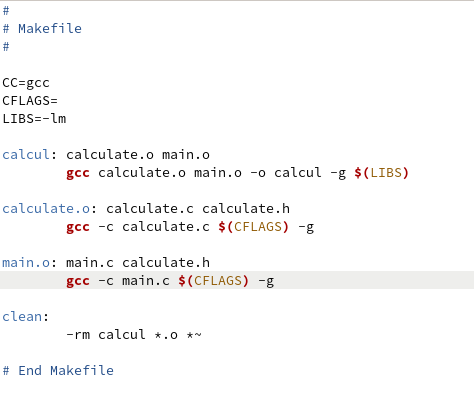


Рис. 10: Изменения в файле Makefile

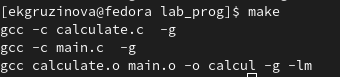


Рис. 11: Запуск изменненого Makefile

– Запуcтите отладчик GDB, загрузив в него программу для отладки (рис. 12):

1 gdb ./calcul

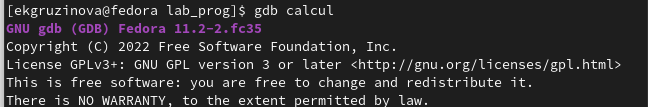


Рис. 12: Вызов для программы calcul откладчика GDB

– Для запуска программы внутри отладчика введите команду run (рис. 13):

1 run

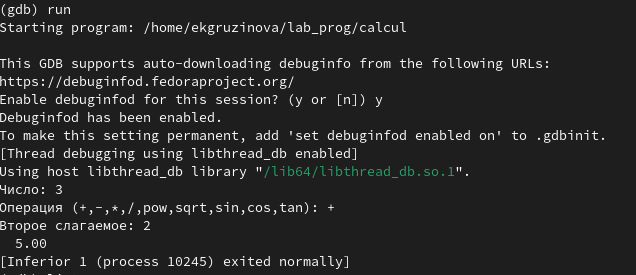


Рис. 13: Запуск программы внутри откладчика

– Для постраничного (по 9 строк) просмотра исходного код используйте команду list (рис. 14):

1 list

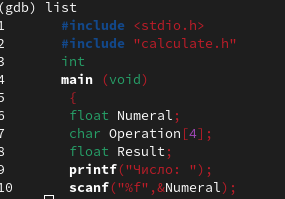


Рис. 14: Вывод текста программы в откладчике

– Для просмотра строк с 12 по 15 основного файла используйте list с параметрами (рис. 15):

1 list 12,15

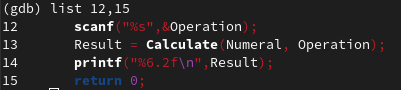


Рис. 15: Вывод определенного текста программы в откладчике

– Для просмотра определённых строк не основного файла используйте list с параметрами (рис. 16):

1 list calculate.c:20,29

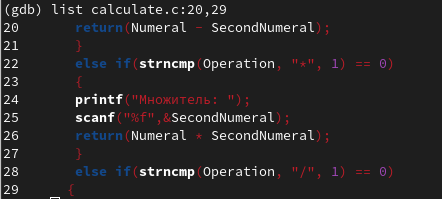


Рис. 16: Вывод определенного текста не основного файла в откладчике

– Установите точку останова в файле calculate.c на строке номер 21 (рис. 17, 18):

1 list calculate.c:20,27

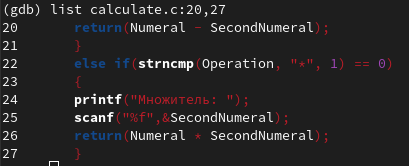


Рис. 17: Вывод необходимого образца текста

2 break 21

Рис. 18: Создание точки останова

Рис. 18: Создание точки останова

– Выведите информацию об имеющихся в проекте точка останова (рис. 19):

1 info breakpoints

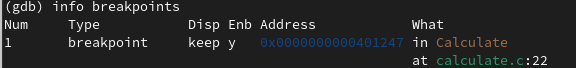


Рис. 19: Информация о точках останова

– Запустите программу внутри отладчика и убедитесь, что программа остановится в момент прохождения точки останова (рис. 20):

1 run 2 5 3 - 4 backtrace

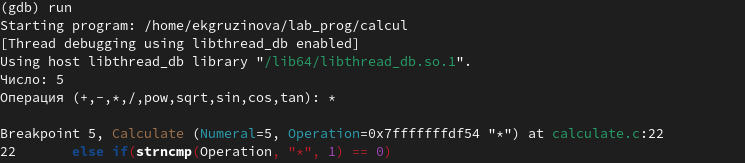


Рис. 20: Запуск программы после установления точки остановы

а команда backtrace покажет весь стек вызываемых функций от начала программы до текущего места (рис. 21).

Рис. 21: Работа команды backtrace

Рис. 21: Работа команды backtrace

– Посмотрите, чему равно на этом этапе значение переменной Numeral, введя

1 print Numeral

На экран должно быть выведено число 5 (рис. 22).

Рис. 22: Работа команды print в откладчике

Рис. 22: Работа команды print в откладчике

– Сравните с результатом вывода на экран после использования команды (рис. 23):

1 display Numeral

Рис. 23: Работа команды display в откладчике

Рис. 23: Работа команды display в откладчике

– Уберите точки останова (рис. 24):

1 info breakpoints 2 delete 1

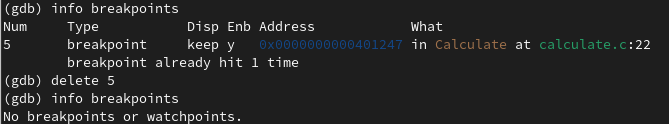


Рис. 24: Удаление точки остановы

1. С помощью утилиты splint попробуйте проанализировать коды файлов calculate.c и main.c. (рис. 25, 26)

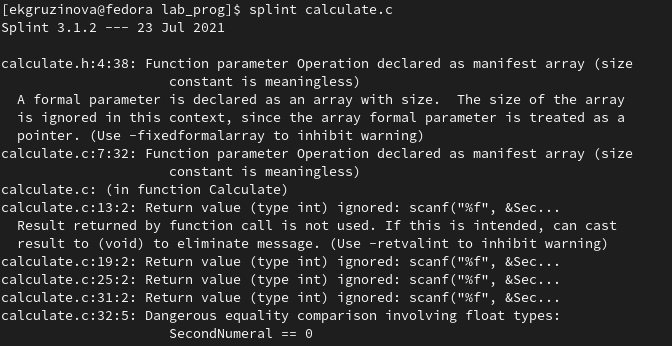


Рис. 25: Аналих файла calculate.c

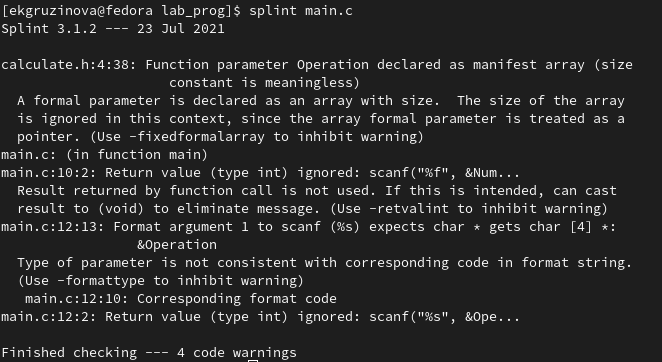


Рис. 26: Аналих файла main.c

# 5 Выводы

При выполнении лабораторной работы приобрела навыки разработки, анализа, тестирования и откладки приложений в ОС типа UNIX/Linux на примере создания на языке программирования С калькулятора с простейшими функциями.

# 6 Контрольные вопросы

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

С помощью команды man (имя\_программы).

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

– 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.

Использование суффикса “.с” для имени файла с программой на языке Си отражает удобное и полезное соглашение, принятое в ОС UNIX. Для любого имени входного файла суффикс определяет какая компиляция требуется. Суффиксы и префиксы указывают тип объекта. Одно из полезных свойств компилятора Си — его способность по суффиксам определять типы файлов. По суффиксу .c компилятор распознает, что файл abcd.c должен компилироваться, а по суффиксу .o, что файл abcd.о является объектным модулем и для получения исполняемой программы необходимо выполнить редактирование связей. Простейший пример командной строки для компиляции программы abcd.c и построения исполняемого модуля abcd имеет вид: gcc -o abcd abcd.c.

1. Каково основное назначение компилятора языка С в UNIX?

Основное назначение компилятора с языка Си заключается в компиляции всей программы в целом и получении исполняемого модуля.

1. Для чего предназначена утилита make?

Утилита make автоматически определяет какие части большой программы должны быть перекомпилированы, и выполняет необходимые для этого действия.

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

Текст, следующий за точкой с запятой, и все последующие строки, начинающиеся с литеры табуляции, являются командами OC UNIX, которые необходимо выполнить для обновления целевого файла. Таким образом, спецификация взаимосвязей имеет формат:

target1 [ target2…]: [:] [dependment1…]

[(tab)commands]

[#commentary]

[(tab)commands]

[#commentary],

где # — специфицирует начало комментария, так как содержимое строки, начиная с # и до конца строки, не будет обрабатываться командой make; : — последовательность команд ОС UNIX должна содержаться в одной строке make-файла (файла описаний), есть возможность переноса команд (), но она считается как одна строка; :: — последовательность команд ОС UNIX может содержаться в нескольких последовательных строках файла описаний.

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

Запуск программы внутри отладки. В gdb используют команду run.

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.

Добавляла после цели программы -g, чтобы выолнялась отладка программы.

1. Прокомментируйте реакцию компилятора на синтаксические ошибки в программе при его первом запуске.
2. Назовите основные средства, повышающие понимание исходного кода программы.

Если вы работаете с исходным кодом, который не вами разрабатывался, то назначение различных конструкций может быть не совсем понятным. Система разработки приложений UNIX предоставляет различные средства, повышающие понимание исходного кода. К ним относятся:

– cscope - исследование функций, содержащихся в программе;

– lint — критическая проверка программ, написанных на языке Си.

1. Каковы основные задачи, решаемые программой splint?

Эта утилита анализирует программный код, проверяет корректность задания аргументов использованных в программе функций и типов возвращаемых значений, обнаруживает синтаксические и семантические ошибки.

# Список литературы

1. Лабораторная работа No 13. Средства, применяемые при разработке программного обеспечения в ОС типа UNIX/Linux [Электронный ресурс]. URL: <https://esystem.rudn.ru/pluginfile.php/1383191/mod_resource/content/5/013-lab_prog.pdf>.