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

Отчёт по лабораторной работе №14

Мурашко В.В.

Содержание

# Теоретическое введение

Интегрированные средства (среды) разработки (IDE) не являются критически необходимым компонентом программной разработки. В традициях UNIX вполне достаточным для ведения программной разработки считается использование текстового редактора, обладающего дополнительными развитыми свойствами, такими как цветовая разметка текста, функции контекстного поиска и замены… Удовлетворяющих таким требованиям редакторов в Linux великое множество, начиная с традиционных vim и Emacs, и до простого редактирования в mc по F4. Опыт использования показывает, что этих средств вполне достаточно вплоть до средних размеров проектов.

Но использование IDE часто позволяет более производительно вести отработку программного кода, оперативнее выполнять в связке цикл: редактирование кода — сборка проекта — отладка. Значительно возрастает роль IDE в разработке GUI приложений, потому как большинство IDE предполагают в своём составе визуальные построители графических экранов.

Под Linux доступно весьма много разных IDE, различной степени интегрированности. Их уже настолько много, что становится бессмысленным описывать все, или значительную их часть в деталях: использование тех или иных IDE становится, в значительной мере, вопросом субъективных предпочтений и привычек.

# Цель работы

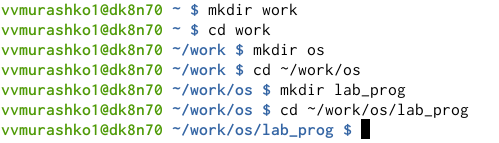
Приобрести простейшие навыки разработки, анализа, тестирования и отладкиприложений в ОС типа UNIX/Linux на примере создания на языке программирования С калькулятора с простейшими функциями.

# 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 посредствомgcc: gcc -c calculate.cgcc -c main.cgcc calculate.o main.o -o calcul -lm
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.

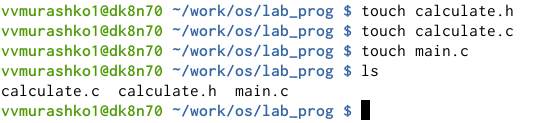
# Выполнение лабораторной работы

1. В домашнем каталоге я создала подкаталог ~/work/os/lab\_prog.

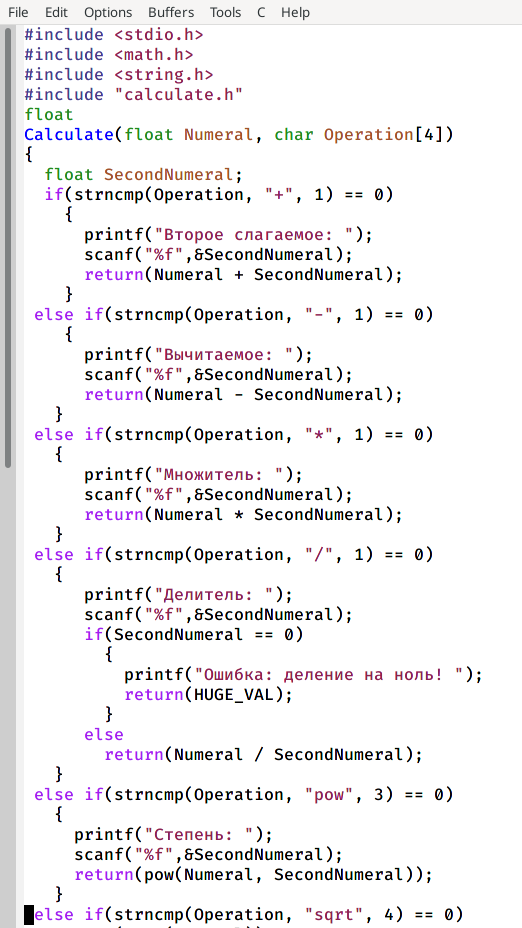


Создание подкаталога

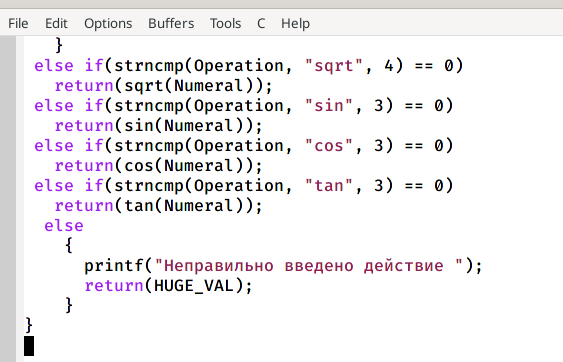
1. Я создала в нём файлы: calculate.h, calculate.c, main.c.



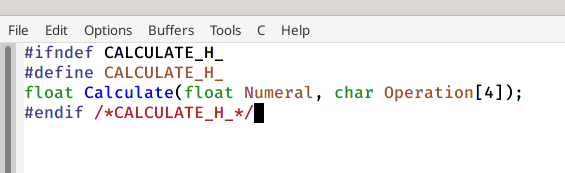
Создание файлов



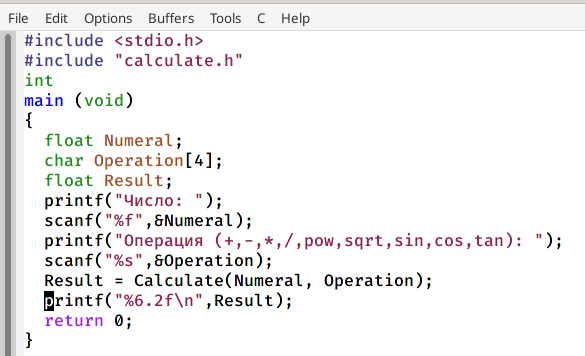
Реализация функций калькулятора в файлеcalculate.h



Реализация функций калькулятора в файлеcalculate.h



Интерфейсный файл calculate.h



Основной файл main.c

1. Я выполнила компиляцию программы посредством gcc: gcc -c calculate.cgcc -c main.c gcc calculate.o main.o -o calcul -lm

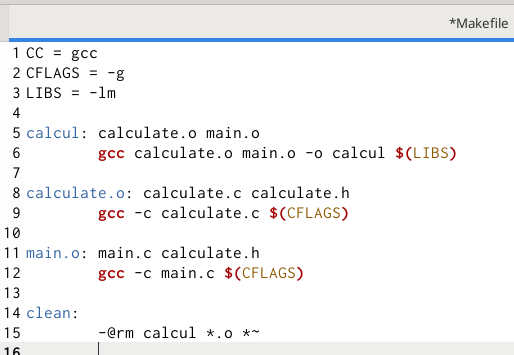
Команда gcc

Команда gcc

Компиляция

Компиляция

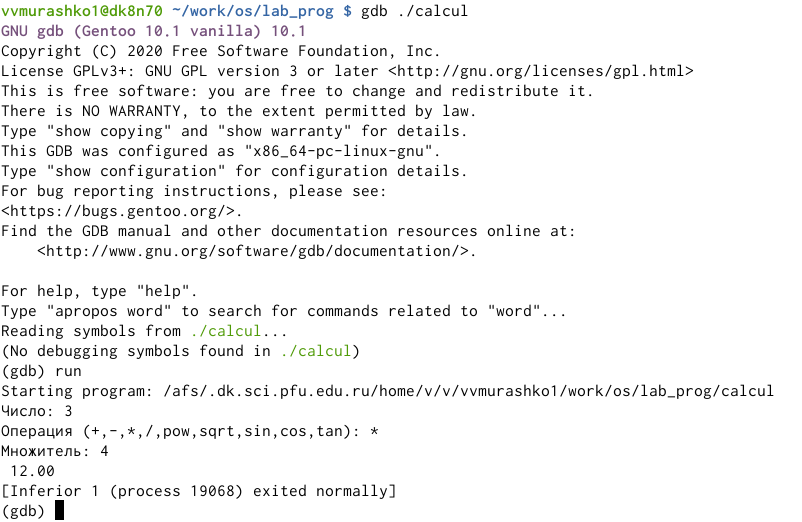
1. Я исправила синтаксические ошибки.
2. Я coздала Makefile со следующим содержанием.



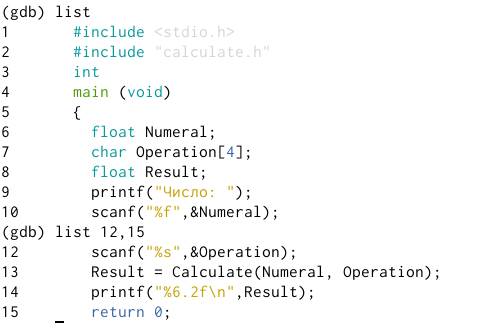
Makefile

В содержании файла указаны флаги компиляции, тип компилятора и файлы, которые должен собрать сборщик.

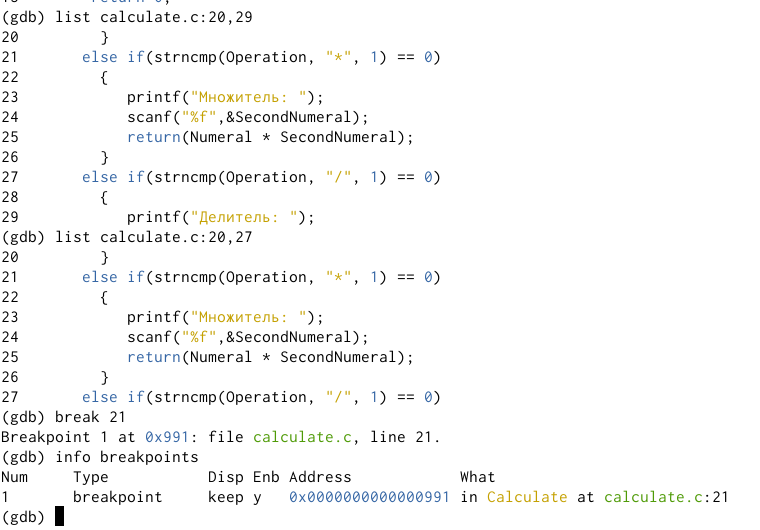
1. С помощью gdb я выполнила отладку программы calcul (перед использованием gdb исправила Makefile).



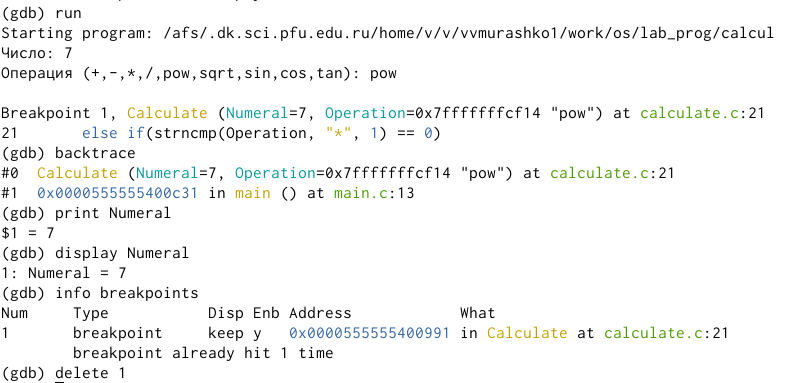
Отладчик GDB и run



Команда list

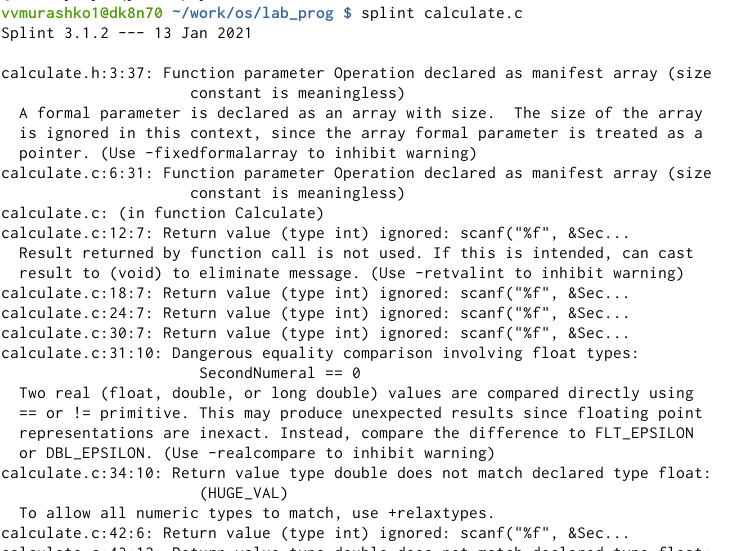


Команда list, точка останова и информация

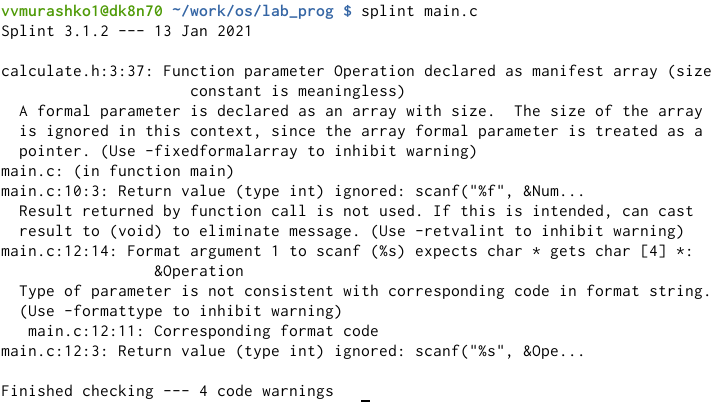


Run, команда backtrace, Numeral и удаление точки останова

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



Анализ calculate.c



Анализ main.c

# Выводы

Я приобрела простейшие навыки разработки, анализа, тестирования и отладкиприложений в ОС типа UNIX/Linux на примере создания на языке программирования С калькулятора с простейшими функциями.

# Библиография

http://rus-linux.net/MyLDP/BOOKS/Linux-tools/IDE\_01.html

# Контрольные вопросы

1. Информацию об этих программах можно получить с помощью функций info и man.
2. Unix поддерживает следующие основные этапы разработки приложений: -создание исходного кода программы; - представляется в виде файла -сохранение различных вариантов исходного текста; -анализ исходного текста; необходимо отслеживать изменения исходного кода, а также при работе более двух программистов над проектом программы нужно, чтобы они не делали изменений кода в одно время. -компиляция исходного текста и построение исполняемого модуля; -тестирование и отладка; - проверка кода на наличие ошибок -сохранение всех изменений, выполняемых при тестировании и отладке.
3. 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. Некоторые проекты предпочитают показывать префиксы в начале текста изменений для старых (old) и новых (new) файлов. Опция – prefix может быть использована для установки такого префикса. Плюс к этому команда bzr diff -p1 выводит префиксы в форме которая подходит для команды patch -p1.
4. Основное назначение компилятора с языка Си заключается в компиляции всей программы в целом и получении исполняемого модуля.
5. При разработке большой программы, состоящей из нескольких исходных файлов заголовков, приходится постоянно следить за файлами, которые требуют перекомпиляции после внесения изменений. Программа make освобождает пользователя от такой рутинной работы и служит для документирования взаимосвязей между файлами. Описание взаимосвязей и соответствующих действий хранится в так называемом make-файле, который по умолчанию имеет имя makefile или Makefile.
6. В общем случае make-файл содержит последовательность записей (строк), определяющих зависимости между файлами. Первая строка записи представляет собой список целевых (зависимых) файлов, разделенных пробелами, за которыми следует двоеточие и список файлов, от которых зависят целевые. 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. Приведённый выше make-файл для программы abcd.c включает два способа компиляции и построения исполняемого модуля. Первый способ предусматривает обычную компиляцию с построением исполняемого модуля с именем abcd. Второй способ позволяет включать в исполняемый модуль testabcd возможность выполнить процесс отладки на уровне исходного текста. Пример можно найти в задании 5.
7. Пошаговая отладка программ заключается в том, что выполняется один оператор программы и, затем контролируются те переменные, на которые должен был воздействовать данный оператор. Если в программе имеются уже отлаженные подпрограммы, то подпрограмму можно рассматривать, как один оператор программы и воспользоваться вторым способом отладки программ. Если в программе существует достаточно большой участок программы, уже отлаженный ранее, то его можно выполнить, не контролируя переменные, на которые он воздействует. Использование точек останова позволяет пропускать уже отлаженную часть программы. Точка останова устанавливается в местах, где необходимо проверить содержимое переменных или просто проконтролировать, передаётся ли управление данному оператору. Практически во всех отладчиках поддерживается это свойство (а также выполнение программы до курсора и выход из подпрограммы). Затем отладка программы продолжается в пошаговом режиме с контролем локальных и глобальных переменных, а также внутренних регистров микроконтроллера и напряжений на выводах этой микросхемы.
8. backtrace - вывод на экран пути к текущей точке останова (по сути вывод названий всех функций) 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 отображаются 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 программе 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. Выполнила компиляцию программы 2)Увидела ошибки в программе 3) Открыла редактор и исправила программу 4) Загрузила программу в отладчик gdb 5) run — отладчик выполнил программу, ввела требуемые значения. 6) Использовала другие команды отладчика и проверила работу программы
9. Отладчику не понравился формат %s для &Operation, т.к %s — символьный формат, а значит необходим только Operation.
10. 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; – splint — критическая проверка программ, написанных на языке Си.
    1. Проверка корректности задания аргументов всех использованных в программе функций, а также типов возвращаемых ими значений; 2. Поиск фрагментов исходного текста, корректных с точки зрения синтаксиса языка Си, но малоэффективных с точки зрения их реализации или содержащих в себе семантические ошибки; 3. Общая оценка мобильности пользовательской программы.