Лабораторная работа №13

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

Медникова Екатерина Михайловна

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 Выполнение лабораторной работы

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

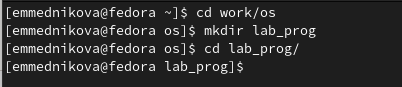


Figure 1: Создание подкаталога

1. Создала в нём файлы: calculate.h, calculate.c, main.c.

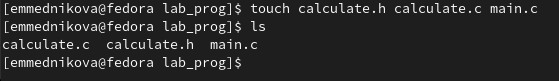


Figure 2: Создание файлов

Написала программы в созданные файлы.

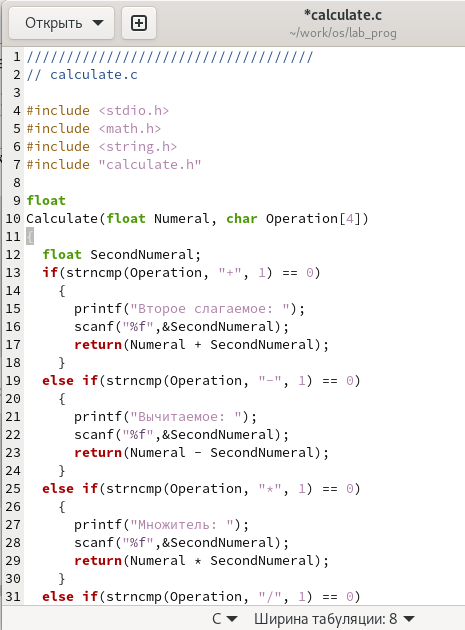


Figure 3: Вставка программы в файл calculate.c

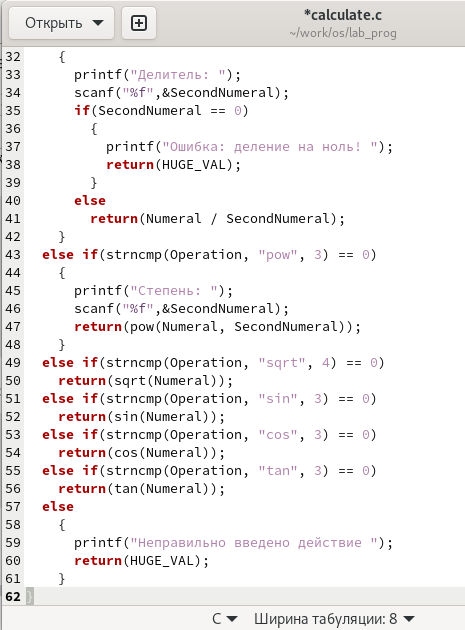


Figure 4: Вставка программы в файл calculate.c

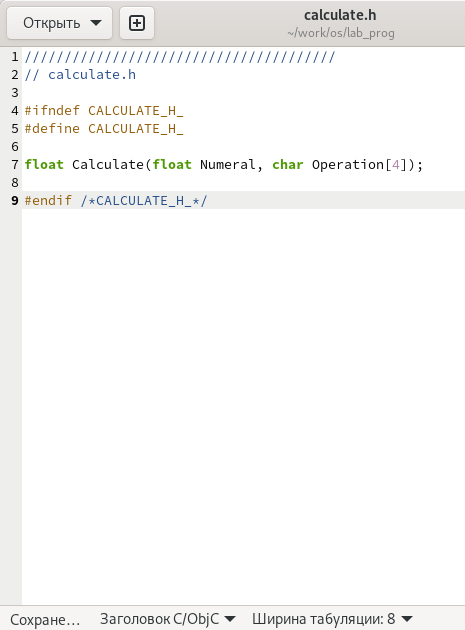


Figure 5: Вставка программы в файл calculate.h



Figure 6: Вставка программы в файл main.c

1. Выполнила компиляцию программы посредством gcc:

Figure 7: Компиляция программы

Figure 7: Компиляция программы

1. При компиляции программы синтаксических ошибок выявлено не было.
2. Создала Makefile со следующим содержанием:

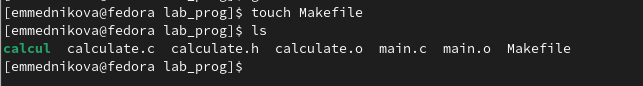


Figure 8: Создание файла

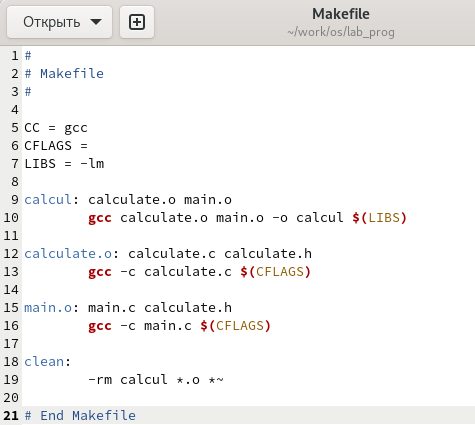


Figure 9: Содержание файла

Данный файл нужен для автоматической компиляции файлов calculate.c, main.c, а также их объединения в один исполняемый файл calcul. Функция clean - автоматическое удаление файлов. Переменная СС отвечает за утилиту для компиляции. Переменная CFLAGS отвечает за опции в данной утилите. Переменная LIBS отвечает за опции для объединения объектных файлов в один исполняемый файл.

1. Перед использованием gdb исправила Makefile:

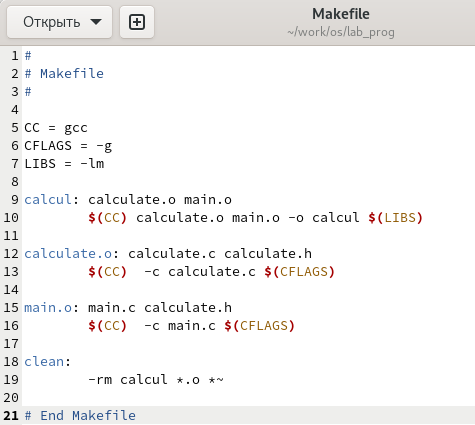


Figure 10: Исправление файла Makefile

Выполнила компиляцию файлов.

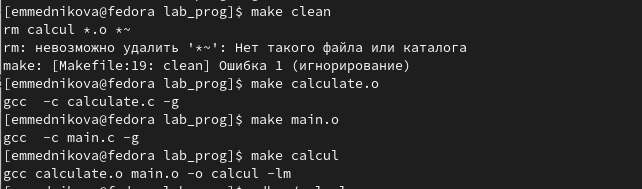


Figure 11: Компиляция файлов

С помощью gdb выполнила отладку программы calcul.

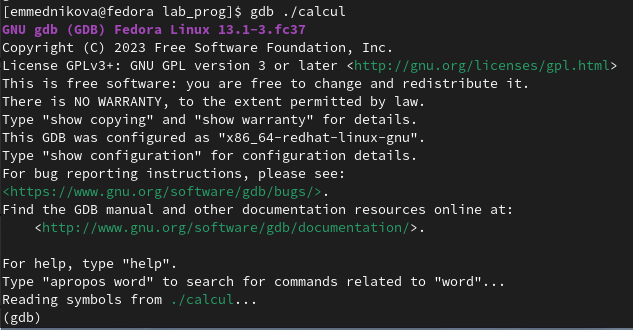


Figure 12: Отладка программы

Для запуска программы внутри отладчика ввела команду run.

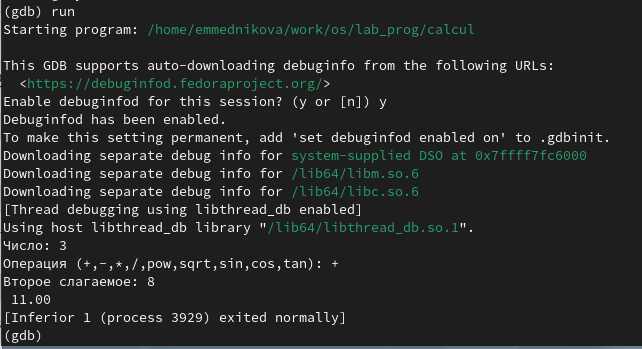


Figure 13: Запуск программы

Для постраничного (по 9 строк) просмотра исходного код использовала команду list.

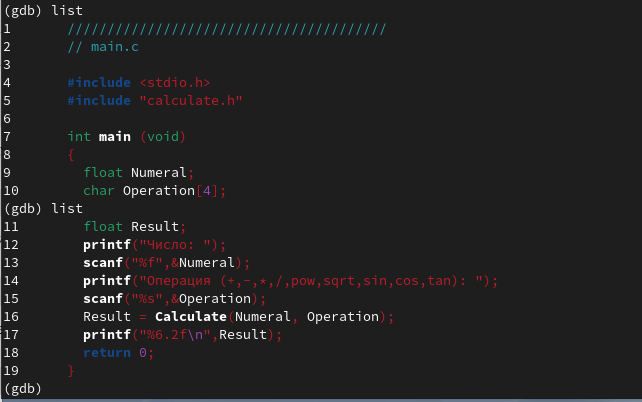


Figure 14: Использование команды list

Для просмотра строк с 12 по 15 основного файла использовала list с параметрами.

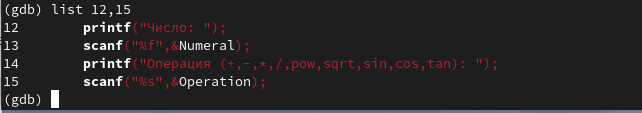


Figure 15: Использование команды list с параметрами

Для просмотра определённых строк не основного файла использовала list с параметрами.

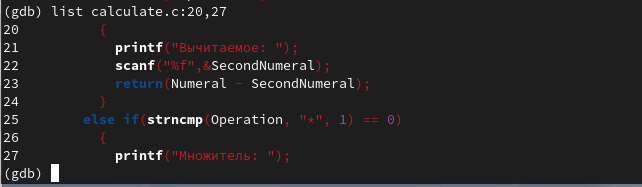


Figure 16: Просмотр определённых строк

Установила точку останова в файле calculate.c на строке номер 21.

Figure 17: Точка останова

Figure 17: Точка останова

Вывела информацию об имеющихся в проекте точках останова.

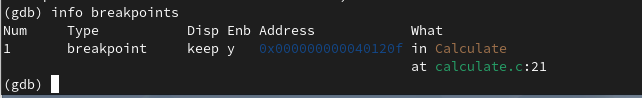


Figure 18: Информация об имеющихся точка останова

Запустила программу внутри отладчика и убедилась, что программа остановится в момент прохождения точки останова.

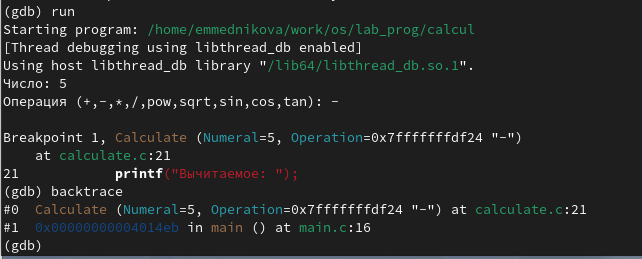


Figure 19: Запуск программы внутри отладчика

Отладчик выдал следующую информацию:

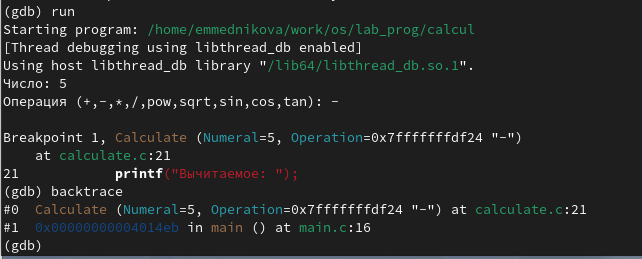


Figure 20: Вывод информации

Посмотрела, чему равно на этом этапе значение переменной Numeral, введя команду print Numeral. На экран было выведено число 5.

Figure 21: Значение переменной Numeral

Figure 21: Значение переменной Numeral

После использования команды display Numeral на экран также было выведено число 5.

Figure 22: Использование команды display Numeral

Figure 22: Использование команды display Numeral

Убрала точки останова.

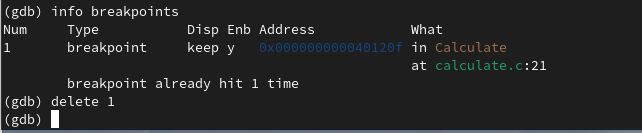


Figure 23: Использование команд info breakpoints и delete 1

1. С помощью утилиты splint было замечено, что в файлах calculate.c и main.c есть функция scanf, которая возвращает целое число, но данные числа не используются и нигде не сохраняются. Далее утилита вывела предупреждение о том, что в файле calculate.c происходит сравнение вещественного числа с нулём.

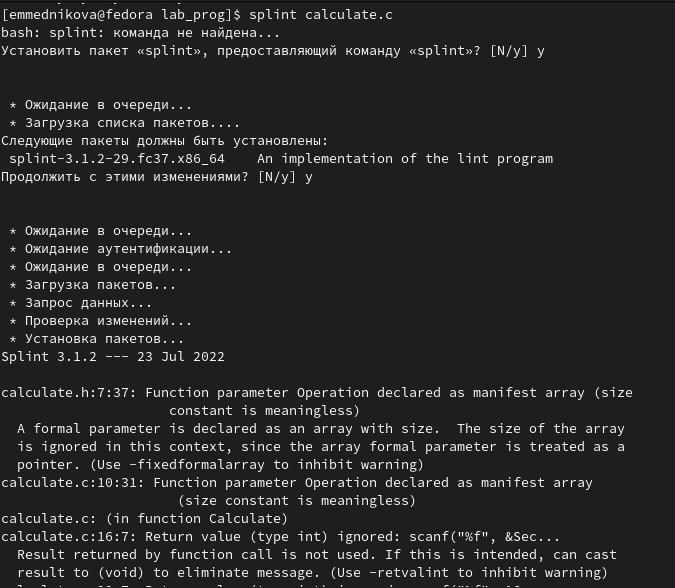


Figure 24: splint calculate.c

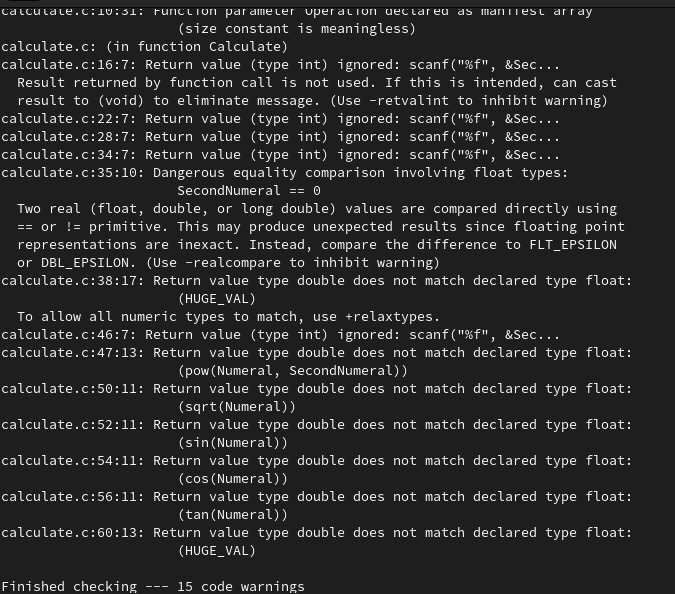


Figure 25: splint calculate.c

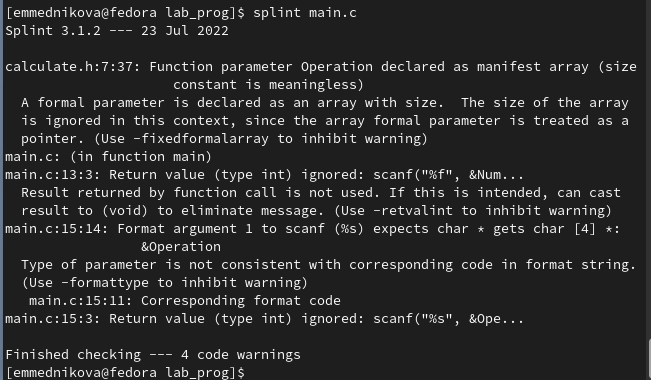


Figure 26: splint main.c

# 3 Выводы

Приобрела 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.?*

Чтобы получить информацию о возможностях программ gcc, make, gdb и др., нужно использовать команду man или -help (-h).

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 ха- рактеристикам 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. *Что такое суффикс в контексте языка программирования? Приведите примеры использования.*

Для имени входного файла суффикс определяет, какая компиляция требуется. Суффиксы указывают на тип объекта. Файлы с расширением (суффиксом) .c воспринимаются gcc как программы на языке С, файлы с расширением .cc или .C — как файлы на языке C++, а файлы c расширением .o считаются объектными. Например, в команде “gcc-cmain.c”: gcc по расширению .с распознаёт тип файла для компиляции и формирует объектный модуль - файл с расширением .о.

1. *Каково основное назначение компилятора языка С в UNIX?*

Основное назначение компилятора языка С в UNIX заключается в компиляции всей программы и получении исполняемого файла/модуля.

1. *Для чего предназначена утилита make?*

Для сборки разрабатываемого приложения и компиляции лучше всего воспользоваться командой make. Она позволяет автоматизировать процесс преобразования файлов программы из одной формы в другую, отслеживает взаимосвязи между файлами.

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

Для работы с утилитой make необходимо в корне рабочего каталога с проектом создать файл с названием makefile или Makefile, в котором будут описаны правила обработки файлов Вашего программного комплекса. В самом простом случае Makefile имеет следующий синтаксис:

… : … <команда 1> …

Сначала задаётся список целей, разделённых пробелами, за которым идёт двоеточие и список зависимостей. Затем в следующих строках указываются команды. Строки с командами обязательно должны начинаться с табуляции. В качестве цели в Makefile может выступать имя файла или название какого-то действия. Зависимость задаёт исходные параметры (условия) для достижения указанной цели. Зависимость также может быть названием какого-то действия. Команды - собственно действия, которые необходимо выполнить для достижения цели. Для запуска программы необходимо в командной строке набрать команду make:

make

Общий синтаксис Makefile имеет вид:

target1 [target2…]:[:] [dependment1…] [(tab)commands] [#commentary] [(tab)commands] [#commentary]

Здесь знак # определяет начало комментария (содержимое от знака # и до конца строки не будет обрабатываться. Одинарное двоеточие указывает на то, что последовательность команд должна содержаться в одной строке. Для переноса можно в длинной строке команд можно использовать обратный слэш (). Двойное двоеточие указывает на то, что последовательность команд может содержаться в нескольких последовательных строках.

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

Во время работы над кодом программы программист неизбежно сталкивается с появлением ошибок в ней. Использование отладчика для поиска и устранения ошибок в программе существенно облегчает жизнь программиста. В комплект программ GNU для ОС типа UNIX входит отладчик GDB (GNU Debugger). Для использования GDB необходимо скомпилировать анализируемый код программы таким образом, чтобы отладочная информация содержалась в результирующем бинарном файле. Для этого следует воспользоваться опцией -g компилятора gcc:

gcc -c file.c -g

После этого для начала работы с gdb необходимо в командной строке ввести одноимённую команду, указав в качестве аргумента анализируемый бинарный файл:

gdb file.o

Затем можно использовать по мере необходимости различные команды gdb. Для выхода из gdb можно воспользоваться командой quit (или её сокращённым вариантом q) или комбинацией клавиш Ctrl-d . Более подробную информацию по работе с gdb можно получить с помощью команд gdb -h и man gdb.

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

Схема отладки программы показана в 6 пункте лабораторной работы.

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

При первом запуске компилятор не выдал никаких синтаксических ошибок.

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

Система разработки приложений UNIX предоставляет различные средства, повышающие понимание исходного кода. К ним относятся:

scope - исследование функций, содержащихся в программе;

lint - критическая проверка программ, написанных на языке С.

1. *Каковы основные задачи, решаемые программой splint?*

Ещё одним средством проверки исходных кодов программ, написанных на языке C, является утилита splint. Эта утилита анализирует программный код, проверяет корректность задания аргументов использованных в программе функций и типов возвращаемых значений, обнаруживает синтаксические и семантические ошибки. В отличие от компилятора C анализатор splint генерирует комментарии с описанием разбора кода программы и осуществляет общий контроль, обнаруживая такие ошибки, как одинаковые объекты, определённые в разных файлах, или объекты, чьи значения не используются в работе программы, переменные с некорректно заданными значениями и типами и многое другое.