Отчет по лабораторной работе №13

Операционные системы

Ничипорова Елена Дмитриевна

Содержание

# 1 Цель работы

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

# 2 Выполнение лабораторной работы

1. Создаю новый подкаталог ~/work/os/lab\_prog(рис. 1)

Рис. 1: Создание нового подкаталога

Рис. 1: Создание нового подкаталога

* Создала в каталоге файлы: calculate.h, calculate.c, main.c.(рис. 2)

Рис. 2: Создание файлов

Рис. 2: Создание файлов

* 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.Реализация функций калькулятора в файле calculate.с:(рис. 3)(рис. 4)



Рис. 3: текст программы

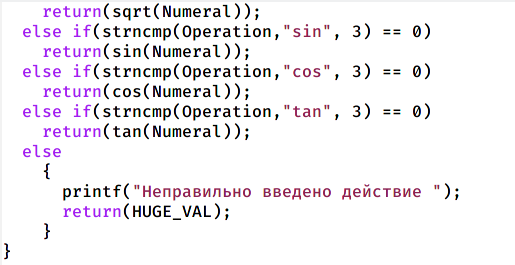


Рис. 4: текст программы

* Интерфейсный файлcalculate.h, описывающий формат вызова функции-калькулятора:(рис. 5)

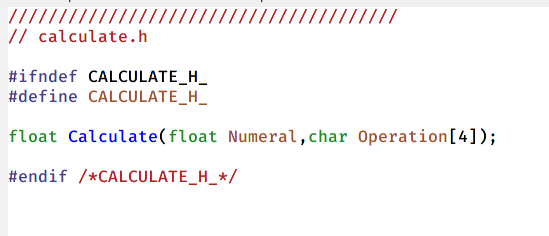


Рис. 5: текст программы

* Основной файл main.c,реализующий интерфейс пользователя к калькулятору:(рис. 6)

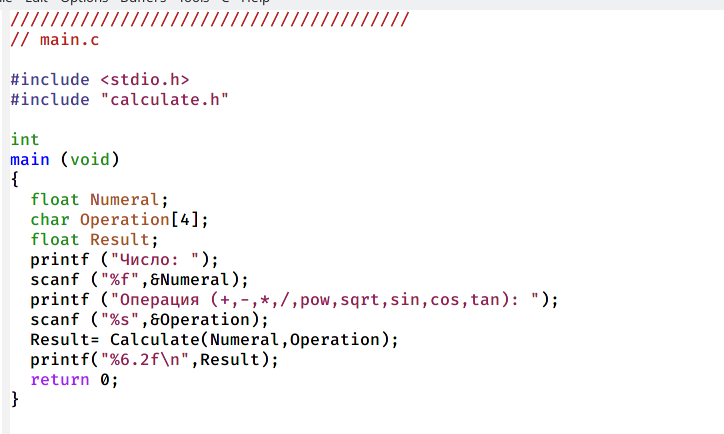


Рис. 6: текст программы

* выполнила компиляцию программы посредством gcc(рис. 7). Обнаружила предупреждение в файле main.c и исправила программу(рис. 8). Заново скомпилировала, после этого ошибок не возникло.(рис. 9)

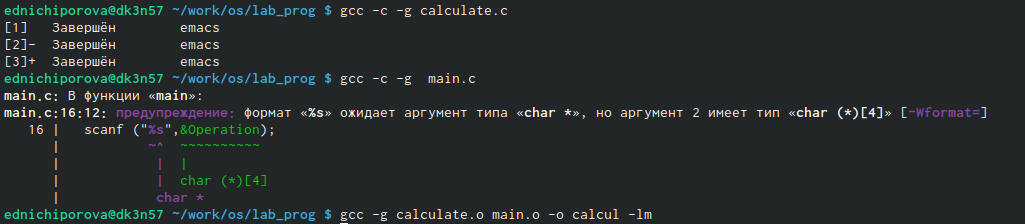


Рис. 7: компиляция программы

Рис. 8: исправленный текст программы

Рис. 8: исправленный текст программы

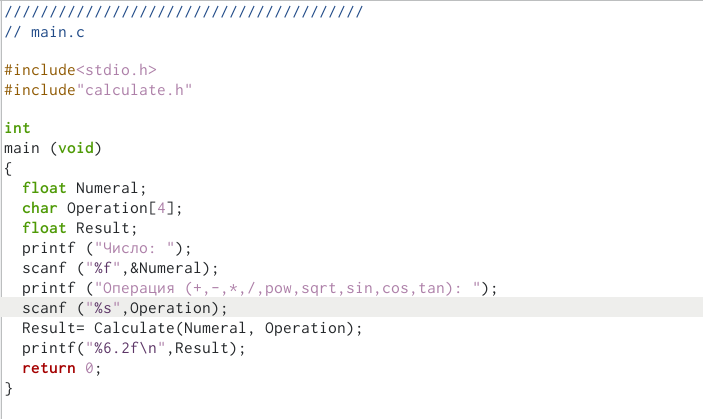


Рис. 9: компиляция программы

* Создала Makefile с необходимым содержанием и исправила его(рис. 10) Данный файл необходим для автоматической компиляции файлов calculate.c (цель calculate.o), main.c (цель main.o), а также их объединения в один исполняемый файл calcul (цель calcul). Цель clean нужна для автоматического удаления файлов. Переменная CC 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.

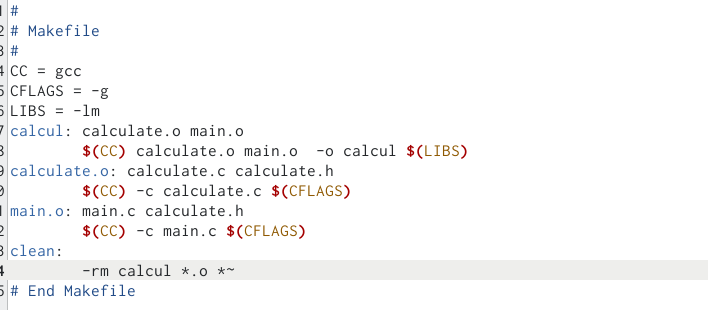


Рис. 10: текст программы Makefile

* Удалила исполняемые и объектные файлы из каталога(рис. 11)

Рис. 11: удаление файлов

Рис. 11: удаление файлов

* Выполнила компиляцию файлов (рис. 12)

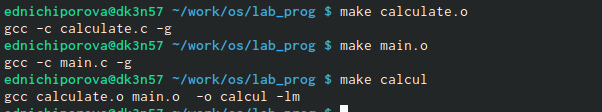


Рис. 12: компиляция файлов

* С помощью gdb выполнила отладку программы calcul. Запустила отладчик GDB(рис. 13)

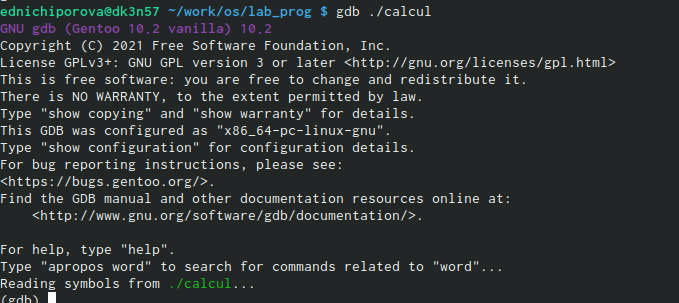


Рис. 13: Работа с gdb

* Запустила программу внутри отладчика(рис. 14)

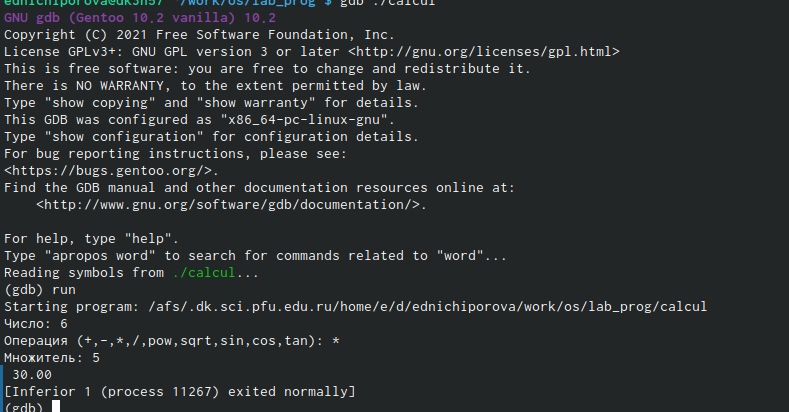


Рис. 14: run - запуск программы

* Постранично просмотрела исходный код с помощью команды “list”(рис. 15)

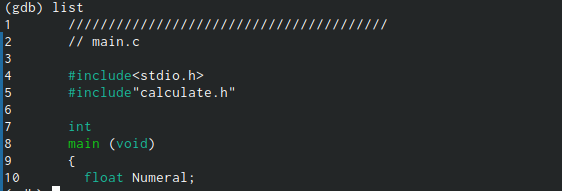


Рис. 15: list

* просмотрела строки 12-15 с помощью команды “list 12,15”(рис. 16)

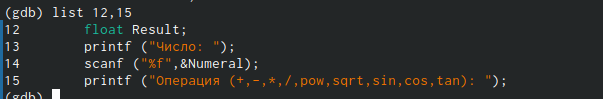


Рис. 16: list 12,15

* Просмотрела определенные строки(рис. 17)

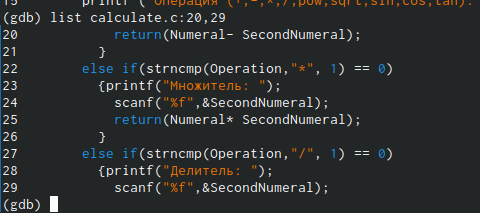


Рис. 17: list calculate:20,29

* Поставила точку останова в файле calculate.c на 21 строке(рис. 18)

Рис. 18: точка останова

Рис. 18: точка останова

* Вывела информацию о имеющихся точках останова (рис. 19)

Рис. 19: info breakpoint

Рис. 19: info breakpoint

* Запустила программу внутри отладчика и убедилась, что программа остановилась в момент прохождения точки останова. Посмотрела, чему равно на этом этапе значениие переменной Numeral, сравнила с результатом вывода на экран после инспользования команды “display Numeral”. Значения совпадают(рис. 20)

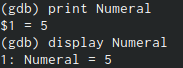


Рис. 20: проверка работы программы

* Убрала точки останова(рис. 21)

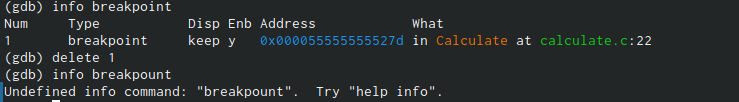


Рис. 21: delet 1

* С помощью утилиты splint проанализируем коды файлов calculate.c и main.c. Воспользуемся командами splint calculate.c и splint main.c (рис. **¿fig:022?**) (рис. **¿fig:023?**). C помощью утилиты splint выяснилось, что в файлах calculate.c и main.c присутствует функция чтения scanf, возвращающая целое число (тип int), но эти числа не используются и нигде не сохранятся. Утилита вывела предупреждение о том, что в файле calculate.c происходит сравнение вещественного числа с нулем. Также возвращаемые значения (тип double) в функциях pow, sqrt, sin, cos и tan записываются в переменную типа float, что свидетельствует о потери данных.

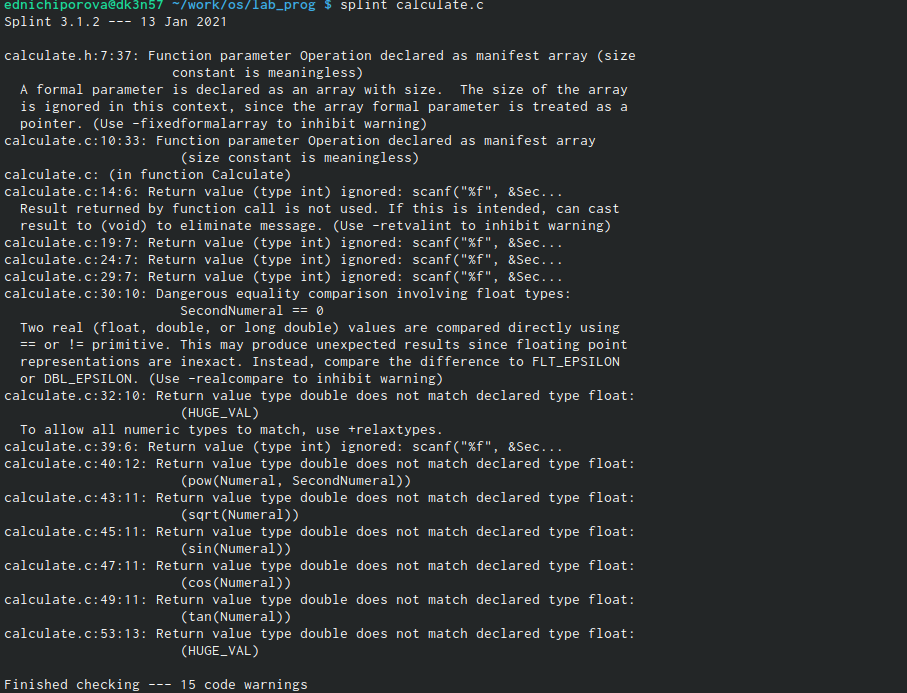


Рис. 22: результат работы splint calculate.c

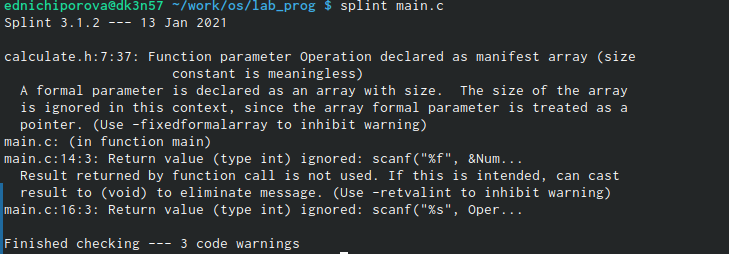


Рис. 23: splint main.c

1. Контрольные вопросы

* Чтобы получить информацию о возможностях программ gcc, make, gdb и др. нужно воспользоваться командой man или опцией -help (-h) для каждой команды.
* Процесс разработки программного обеспечения обычно разделяется на следующие этапы: планирование, включающее сбор и анализ требований кфункционалу 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.
  + 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 -c main.c»: gcc по расширению (суффиксу) .c распознает тип файла для компиляции и формирует объектный модуль − файл с расширением .o. Если требуется получить исполняемый файл с определённым именем (например, hello), то требуется воспользоваться опцией -o и в качестве параметра задать имя создаваемого файла: «gcc -o hello main.c».
* Основное назначение компилятора языка Си в UNIX заключается вкомпиляции всей программы и получении исполняемого файла/модуля.
* Для сборки разрабатываемого приложения и собственно компиляции полезно воспользоваться утилитой make. Она позволяет автоматизировать процесс преобразования файлов программы из одной формы в другую, отслеживает взаимосвязи между файлами.
* Для работы с утилитой make необходимо в корне рабочего каталога с Вашим 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. 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 водной 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.
* 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
* Основные команды 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 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.
* Cхема отладки программы показана в 6 пункте лабораторной работы.
* При первом запуске компилятор не выдал никаких ошибок, но в коде программы main.c допущена ошибка, которую компилятор мог пропустить (возможно, из-за версии 8.3.0-19): в строке scanf(“%s”, &Operation); нужно убрать знак &, потому что имя массива символов уже является указателем на первый элемент этого массива.
* 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.
* Утилита splint 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 работепрограммы, переменные с некорректно заданными значениями и типами и многое другое.

# 3 Выводы

Приобрела простейшие навыки разработки,анализа,тестирования и отладки при-ложений в ОС типа UNIX/Linux на примере создания на языке программированияС калькулятора с простейшими функциями.

# Список литературы