Операционные системы

Лабораторная работа №14

Безрук Мария Андреевна

Содержание

# Цель работы

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

# Выполнение лабораторной работы

1. В домашнем каталоге создаю подкаталог ~/work/os/lab\_progс помощью команды «mkdir-p~/work/os/lab\_prog»

Figure 1: Создание подкаталога

Figure 1: Создание подкаталога

2)Создала в каталоге файлы: calculate.h, calculate.c, main.c, используя команды «cd~/work/os/lab\_prog» и «touch calculate.h calculate.c main.c»

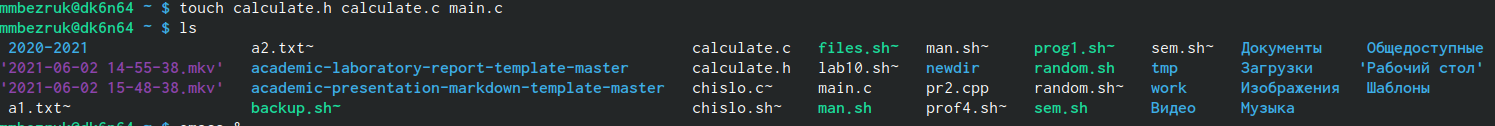


Figure 2: Создание файлов для работы

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. Открыв редактор Emacs, приступила к редактированию созданных файлов. Реализация функций калькулятора в файле calculate.с



Figure 3: Код для калькулятора

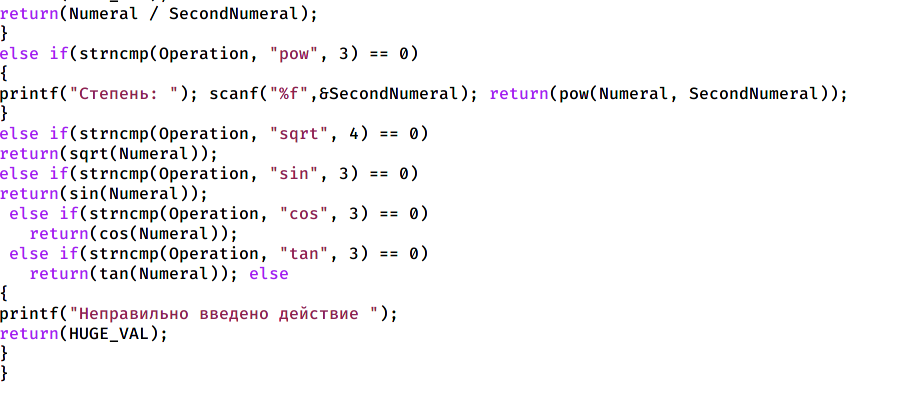


Figure 4: Продолжение кода

Интерфейсный файл calculate.h, описывающий формат вызова функции калькулятора.

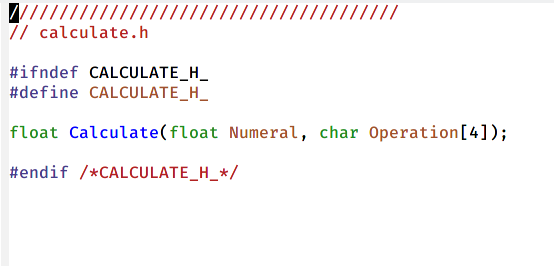


Figure 5: calculate.h

Основной файл main.c, реализующий интерфейс пользователя к калькулятору

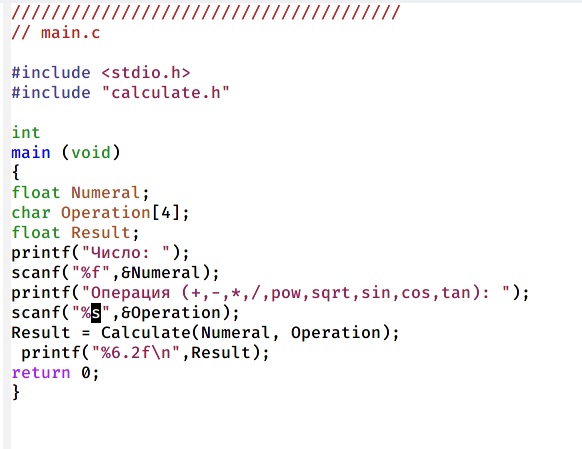


Figure 6: main.c

1. Выполнила компиляцию программы посредством gcc (версия компилятора:8.3.0-19), используя команды «gcc-ccalculate.c», «gcc-cmain.c» и «gcccalculate.omain.o-ocalcul-lm»

Figure 7: Команды gcc

Figure 7: Команды gcc

Figure 8: Команды gcc

Figure 8: Команды gcc

1. В ходе компиляции программы никаких ошибок выявлено не было.
2. Создала Makefile с необходимым содержанием

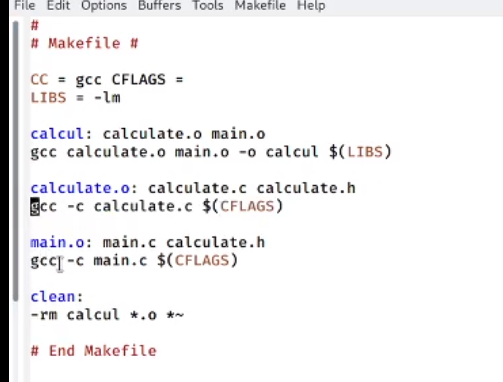


Figure 9: Makefile

Данный файл необходим для автоматической компиляции файлов calculate.c (цель calculate.o), main.c (цельmain.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.

1. Далее исправила Makefile

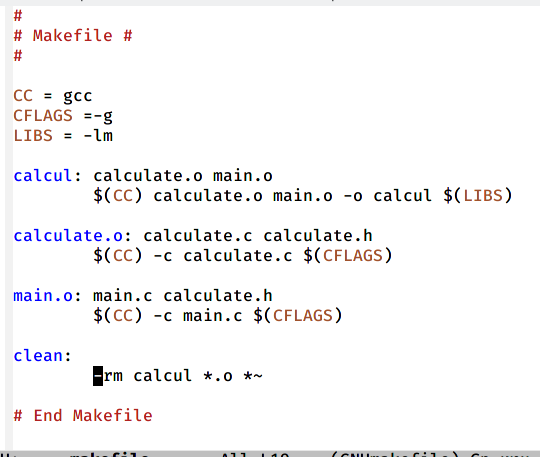


Figure 10: Измененный Makefile

В переменную CFLAGS добавила опцию -g, необходимую для компиляции объектных файлов и их использования в программе отладчика GDB. Сделала так, что утилита компиляции выбирается с помощью переменной CC.

После этого я удалила исполняемые и объектные файлы из каталога с помощью команды «make clean».

Figure 11: Измененный Makefile

Figure 11: Измененный Makefile

Выполнила компиляцию файлов, используя команды «make calculate.o», «make main.o», «make calcul».

Figure 12: Измененный Makefile

Figure 12: Измененный Makefile

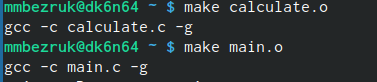


Figure 13: Измененный Makefile

Далее с помощью gdb выполнила отладку программы calcul. Запустила отладчик GDB, загрузив в него программу для отладки, используя команду: «gdb ./calcul»

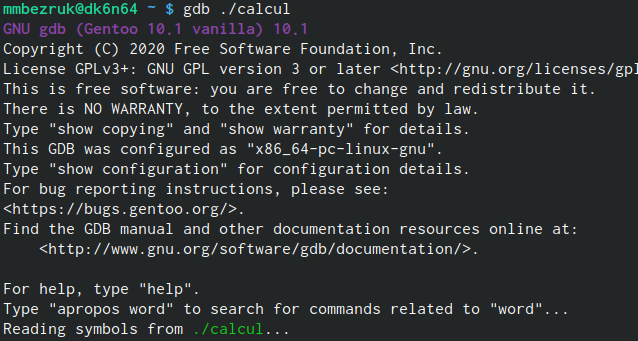


Figure 14: Запуск GDB

Для запуска программы внутри отладчика ввела команду «run»

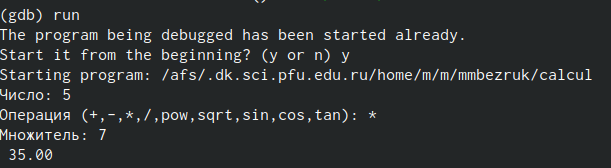


Figure 15: Команда «run»

Для постраничного (по 10 строк) просмотра исходного кода использовала команду «list»



Figure 16: Команда «list»

Для просмотра строк с 12 по 15 основного файла использовала команду «list 12,15»

Для просмотра определённых строк не основного файла использовала команду «list calculate.c:20,29»

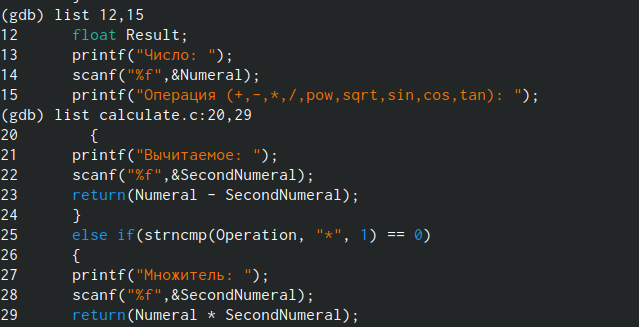


Figure 17: Команда «list calculate.c:20,29»

Установила точку останова в файле calculate.c на строке номер 21, используя команды «list calculate.c:20,27» и «break 21»

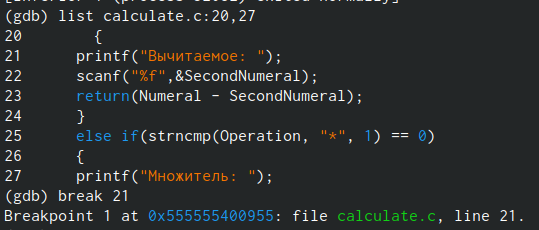


Figure 18: Точка останова

Вывела информацию об имеющихся в проекте точках останова с помощью команды «info breakpoints»

Figure 19: Вывод информации

Figure 19: Вывод информации

Запустила программу внутри отладчика и убедилась, что программа остановилась в момент прохождения точки останова. Использовала команды «run», «5», «−» и «backtrace»

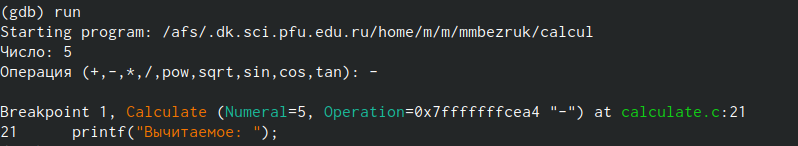


Figure 20: Запуск программы

Посмотрела, чему равно на этом этапе значение переменной Numeral, введя команду «print Numeral»

Figure 21: Переменная Numeral

Figure 21: Переменная Numeral

Сравнила с результатом вывода на экран после использования команды «display Numeral». Значения совпадают.

Figure 22: Сравнение

Figure 22: Сравнение

Убрала точки останова с помощью команд «info breakpoints» и «delete 1»

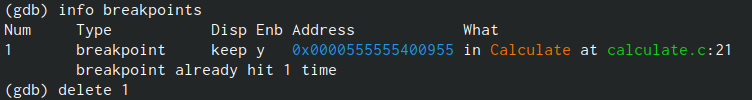


Figure 23: Удаление точки останова

1. С помощью утилиты splint проанализировала коды файлов calculate.c и main.c. Далее воспользовалась командами «splint calculate.c» и «splint main.c»

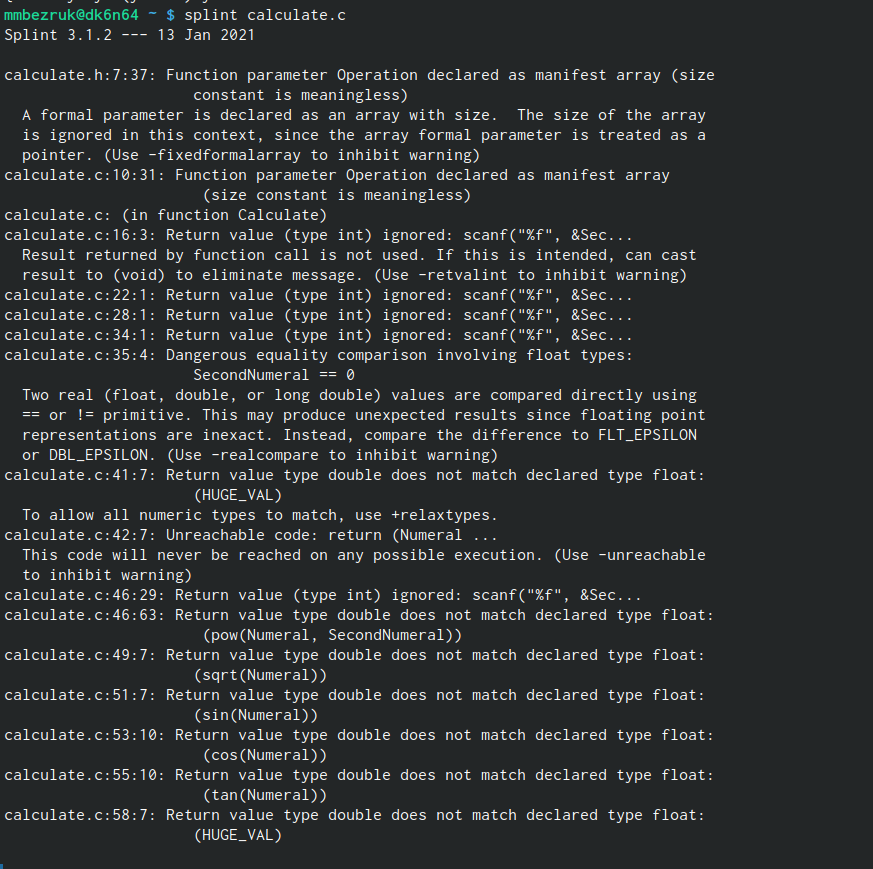


Figure 24: Анализ файлов

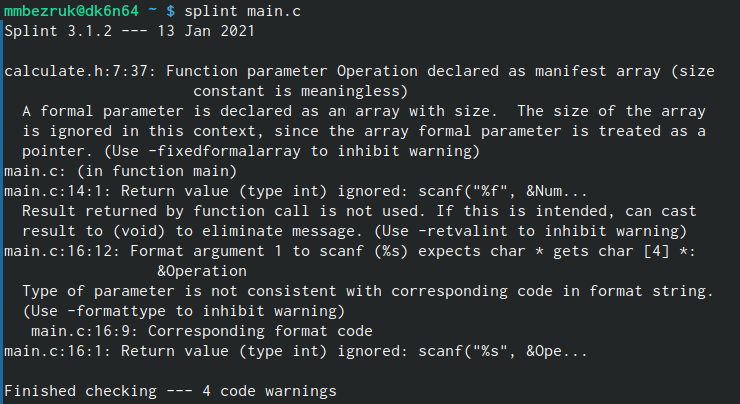


Figure 25: Анализ файлов

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.

# Контрольные вопросы:

1. Чтобы получить информацию о возможностях программ gcc, make, gdbи др. нужно воспользоваться командой man или опцией -help(-h)для каждой команды.
2. 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: oкодирование − по сути создание исходного текста программы (возможно в нескольких вариантах); – анализ разработанного кода; oсборка, компиляция и разработка исполняемого модуля;
* oтестирование и отладка, сохранение произведённых изменений;
* документирование.

Для создания исходного текста программы разработчик может воспользоваться любым удобным для него редактором текста: vi, vim, mceditor, emacs, geanyи др. После завершения написания исходного кода программы (возможно состоящей из нескольких файлов), необходимо её скомпилировать и получить исполняемый модуль.

1. Для имени входного файла суффикс определяет какая компиляция требуется. Суффиксы указывают на тип объекта. Файлы с расширением (суффиксом) .cвоспринимаются gccкак 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»: gccпо 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-ohellomain.c».
2. Основное назначение компилятора языка Си в UNIX заключается в компиляции всей программы и получении исполняемого файла/модуля.

5)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.

1. 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 Makefileможет 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 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. Пример более сложного синтаксиса Makefile:## Makefile for abcd.c#CC = gccCFLAGS =# Compile abcd.c normalyabcd: abcd.c$(CC) -o abcd $(CFLAGS) abcd.cclean:-rm abcd *.o* ~# EndMakefileforabcd.c В этом примере в начале файла заданы три переменные: CCи CFLAGS. Затем указаны цели, их зависимости и соответствующие команды. В командах происходит обращение к значениям переменных. Цель с именем clean производит очистку каталога от файлов, полученных в результате компиляции. Для её описания использованы регулярные выражения.

1. 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 UNIXвходит отладчик GDB (GNUDebugger).

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-cfile.c -g После этого для начала работы с gdb необходимо в командной строке ввести одноимённую команду, указав в качестве аргумента анализируемый бинарный файл: gdbfile.o

1. Основные команды отладчика gdb:

* backtrace−вывод на экран пути к текущей точке останова (по сутивывод − названий всех функций)
* break −установить точку останова (в качестве параметра можетбыть 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 начальнойи 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 mangdb.

1. Cхема отладки программы показана в 6 пункте лабораторной работы.
2. 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 массивасимволов уже является указателемна первый элементэтого массива.
3. 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.12)Утилита splintанализирует программный код, проверяет корректность задания аргументов использованных в программе функций и типов возвращаемых значений, обнаруживает синтаксические и семантические ошибки. В отличие от компилятора Cанализатор splintгенерирует 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 значениямии типами и многое другое.

# Выводы

В ходе выполнения данной лабораторной работы я 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.