Отчёт по лабораторной работе № 13

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

Татур Стефан Андреевич

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

* Ознакомиться с теоретическим материалом.
* Выполнить упражнения.
* Ответить на контрольные вопросы.

# 3 Выполнение лабораторной работы

## 3.1 Работа с программой калькулятор

1. В домашнем каталоге создали подкаталог ~/work/os/lab\_prog.(рис. [[1](#fig:001)])

Figure 1: lab_prog

Figure 1: lab\_prog

1. Создали в нём файлы: calculate.h, calculate.c, main.c. (рис. [[2](#fig:002)])

Figure 2: calculate.h, calculate.c, main.c

Figure 2: calculate.h, calculate.c, main.c

Это примитивнейший калькулятор, способный складывать, вычитать, умножать и делить, возводить число в степень, брать квадратный корень, вычислять sin, cos, tan. При запуске он запрашивает первое число, операцию, второе число. После этого программа выводит результат и останавливается. Реализация функций калькулятора в файле calculate.с:(рис. [[3](#fig:003)])



Figure 3: calculate.c

Интерфейсный файл calculate.h, описывающий формат вызова функциикалькулятора: (рис. [[4](#fig:004)])

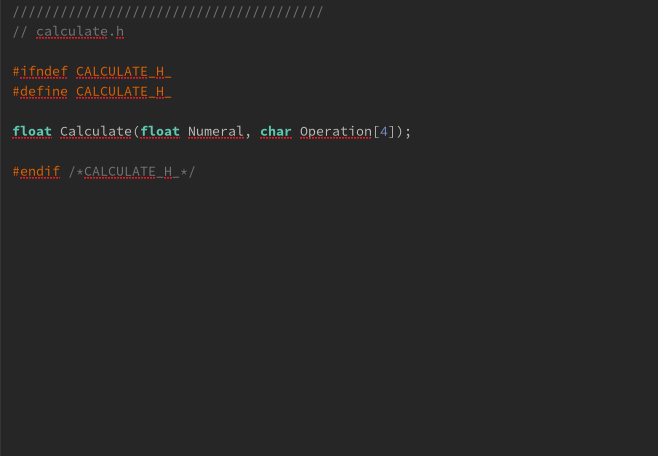


Figure 4: calculate.h

Основной файл main.c, реализующий интерфейс пользователя к калькулятору:(рис. [[5](#fig:005)])



Figure 5: main.c

1. Выполнили компиляцию программы посредством gcc: (рис. [[6](#fig:006)])

Figure 6: gcc

Figure 6: gcc

1. Исправили синтаксические ошибки.
2. Создали Makefile. (рис. [[7](#fig:007)]), (рис. [[8](#fig:008)])

Figure 7: Makefile

Figure 7: Makefile

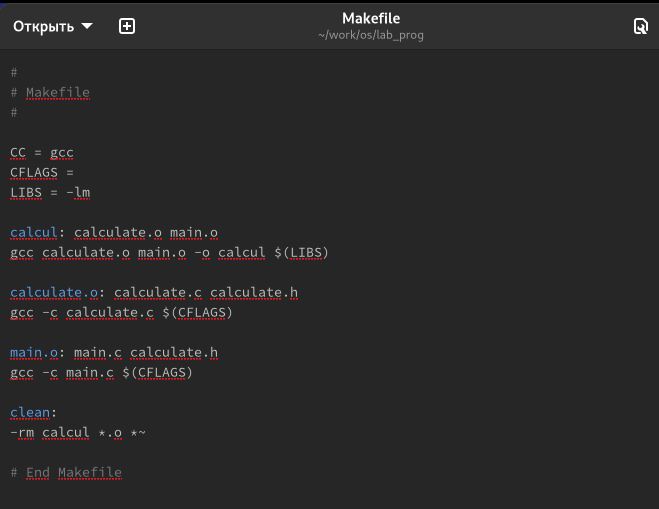


Figure 8: Makefile

1. С помощью gdb выполнили отладку программы calcul: – Запустили отладчик GDB, загрузив в него программу для отладки:(рис. [[9](#fig:009)])

Figure 9: gdb

Figure 9: gdb

– Для запуска программы внутри отладчика ввели команду run:(рис. [[10](#fig:010)])

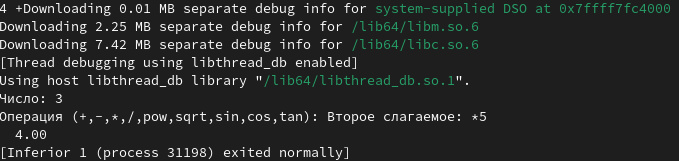


Figure 10: run

– Для постраничного (по 9 строк) просмотра исходного код использовали команду list (рис. [[11](#fig:011)])

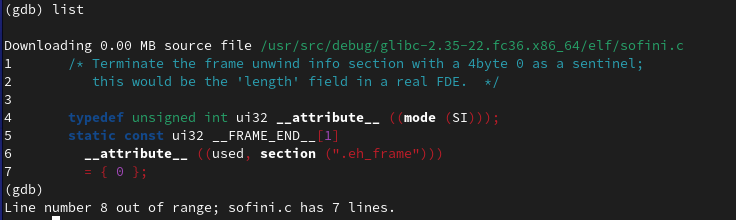


Figure 11: list

– Для просмотра строк с 1 по 4 основного файла использовали list с параметрами:(рис. [[12](#fig:012)])

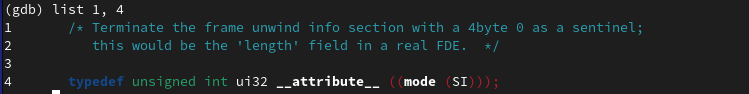


Figure 12: list

– Для просмотра определённых строк не основного файла использовали list с параметрами: (рис. [[13](#fig:013)])

Figure 13: list

Figure 13: list

– Установили точку останова в файле calculate.c на строке номер 21: (рис. [[14](#fig:014)])

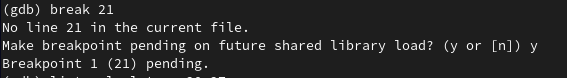


Figure 14: breakpoint

– Вывели информацию об имеющихся в проекте точка останова: (рис. [[15](#fig:015)])

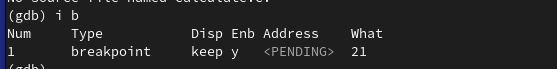


Figure 15: info breakpoints

– Запустили программу внутри отладчика и убедились, что программа остановится в момент прохождения точки останова: (рис. [??])

[breakpoint]](image/16.png){ #fig:016 width=70%}

– Посмотрели, чему равно на этом этапе значение переменной Numeral, введя:(рис. [[16](#fig:017)])

Figure 16: Numeral

Figure 16: Numeral

– Сравнили с результатом вывода на экран после использования команды: (рис. [[17](#fig:018)])

Figure 17: Numeral

Figure 17: Numeral

– Убрали точки останова: (рис. [[18](#fig:019)])

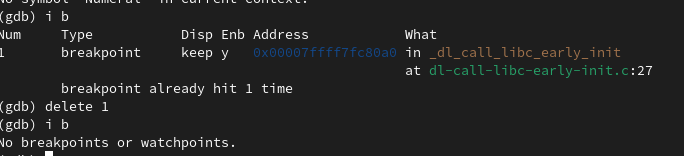


Figure 18: breakpoint

1. С помощью утилиты splint попробовали проанализировать коды файлов calculate.c и main.c. (рис. [[19](#fig:020)]), (рис. [[20](#fig:021)])

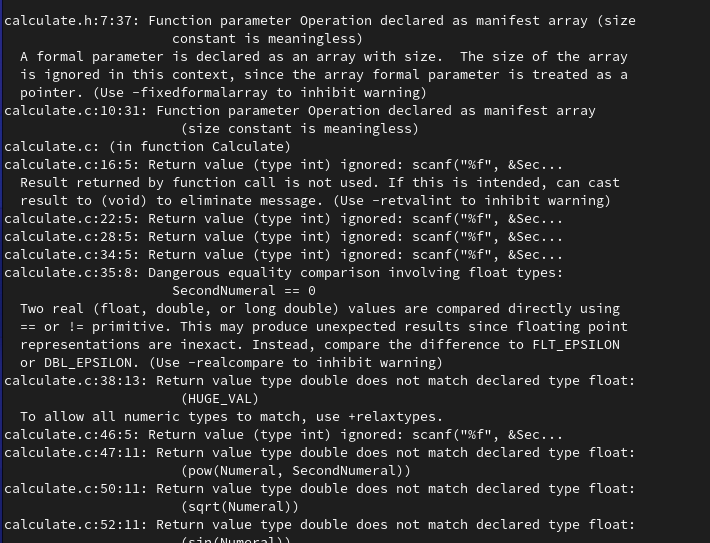


Figure 19: splint

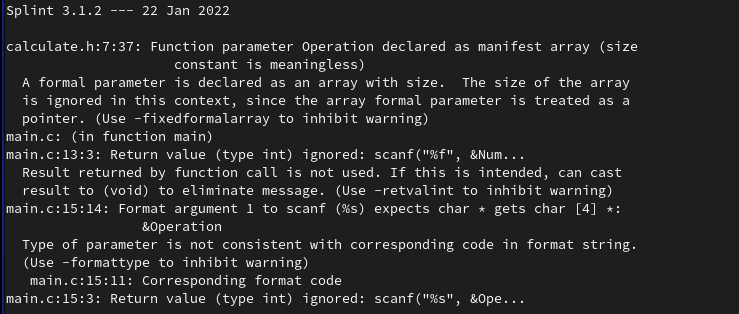


Figure 20: splint

# 4 Выводы

В ходе выполнения были приобретены простейшие навыки разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linux на примере создания на языке программирования С калькулятора с простейшими функциями.

# 5 Ответы на контрольные вопросы

1. Чтобы получить информацию о возможностях программ gcc, make, gdb и 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: – кодирование − по сути создание исходного текста программы (возмож- но 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.
3. 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».
4. Основное назначение компилятора языка Си в UNIX заключается в компи- ляции всей программы и получении исполняемого файла/модуля.
5. Для сборки разрабатываемого приложения и собственно компиляции по- лезно воспользоваться утилитой make. Она позволяет автоматизировать процесс преобразования файлов программы из одной формы в другую, отслеживает взаимосвязи между файлами.
6. 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 названи- ем какого-то действия. Команды − собственно действия, которые необхо- димо 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 указыва- ет на то, что последовательность команд может содержаться в нескольких последовательных строках. Пример более сложного синтаксиса Makefile: Makefile for abcd.c # CC = gcc CFLAGS = #Compile abcd.c normaly abcd: abcd.c (CFLAGS) abcd.c clean: -rm abcd .o ~ #End Makefile for abcd.c В этом примере в начале файла заданы три переменные: CC и CFLAGS. Затем указаны цели, их зависимости и соответствующие команды. В командах происходит обращение к значениям переменных. Цель с именем clean про- изводит очистку каталога от файлов, полученных в результате компиляции. Для её описания использованы регулярные выражения.
7. Во время работы над кодом программы программист неизбежно сталкивается с появлением ошибок в ней. Использование отладчика для поиска иустранения ошибок в программе существенно облегчает жизнь программи- ста. В комплект программ GNU для ОС типа UNIX входит отладчик GDB (GNU Debugger). Для использования GDB необходимо скомпилировать анализи- руемый код программы таким образом, чтобы отладочная информация содержалась в результирующем бинарном файле. Для этого следует вос- пользоваться 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
8. 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 указанномер 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 ко- торого программа будет остановлена Для выхода из gdb можно воспользо- ваться командой quit (или её сокращённым вариантом q) или комбинацией клавиш Ctrl-d. Более подробную информацию по работе с gdb можно полу- чить с помощью команд gdb -h и man gdb. Cхема отладки программы показана в 6 пункте лабораторной работы.

1. 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); нужноубрать знак &, потому что имя массива символов уже является указателем на первый элемент этого массива.
2. 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. Утилита splint анализирует программный код, проверяет корректность задания аргументов использованных в программе функций и типов возвра- щаемых значений, обнаруживает синтаксические и семантические ошибки. В отличие от компилятора C анализатор splint генерирует комментарии с описанием разбора кода программы и осуществляет общий контроль, обна- руживая такие ошибки, как одинаковые объекты, определённые в разных файлах, или объекты, чьи значения не используются в работ программы, переменные с некорректно заданными значениями и типами и многое другое