Отчёт по лабораторной работе №14

дисциплина: Операционные системы

Быстров Глеб Андреевич

Содержание

# Цель работы

В данной лабораторной работе мне будет необходимо 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.

# Теория

Bash — популярный командный интерпретатор, используемый в юниксоподобных системах, например, в GNU/Linux. Это программа, которую называют оболочка либо шелл (shell), а само название «bash» является сокращением от «Bourne Again Shell». Интерпретатор Bash принимает ваши команды, передавая их операционной системе. Чтобы осуществлялось взаимодействие с ОС, применяются терминалы (gnome-terminal, nxterm и прочие).

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 специфи- каций, 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.

Стандартным средством для компиляции программ в ОС типа UNIX является GCC (GNU Compiler Collection). Это набор компиляторов для разного рода языков программирования (С, C++, Java, Фортран и др.). Работа с GCC производится при помощи одноимённой управляющей программы gcc , которая интерпретирует аргументы командной строки, определяет и осуществляет запуск нужного компилятора для входного файла. 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.

# 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 посредством gcc: gcc -c calculate.c gcc -c main.c gcc calculate.o main.o -o calcul -lm
4. При необходимости исправьте синтаксические ошибки.
5. Создайте Makefile со следующим содержанием: CC = gcc CFLAGS = LIBS = -lm calcul: calculate.o main.o gcc calculate.o main.o -o calcul $(LIBS) calculate.o: calculate.c calculate.h gcc -c calculate.c $(CFLAGS) main.o: main.c calculate.h gcc -c main.c $(CFLAGS) clean: -rm calcul *.o* ~ Поясните в отчёте его содержание.
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 создал подкаталог ~/work/os/lab\_prog. Использовал команды mkdir и cd. (рис. 1)

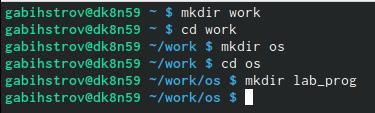


Figure 1: Создание подкаталога

1. Создал в нём файлы: calculate.h, calculate.c, main.c. (рис. 2)

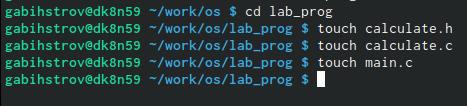


Figure 2: Создание файлов

1. Реализация функций калькулятора в файле calculate.c. (рис. 3)

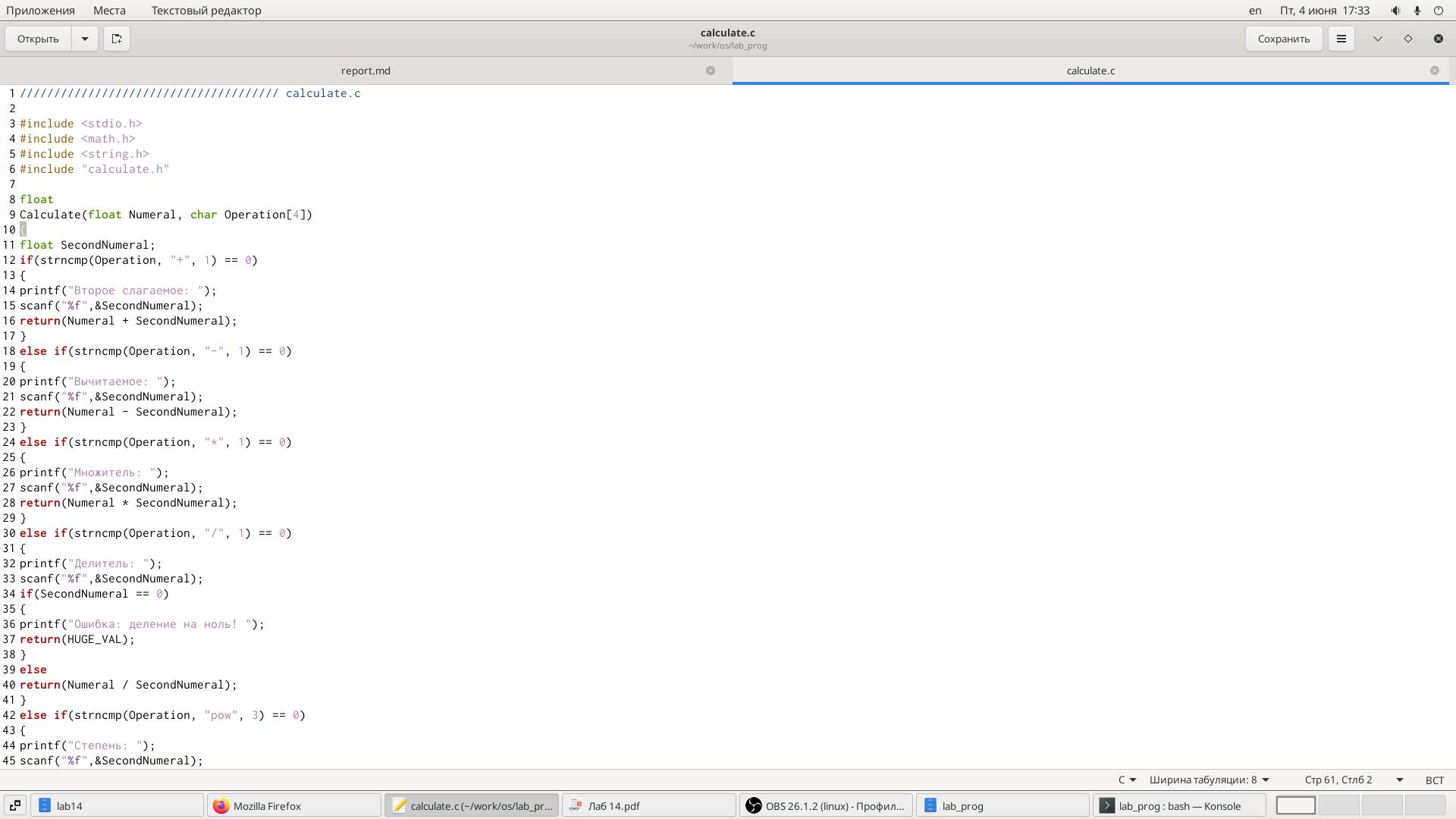


Figure 3: Файл calculate.c

1. Интерфейсный файл calculate.h, описывающий формат вызова функции калькулятора. (рис. 4)

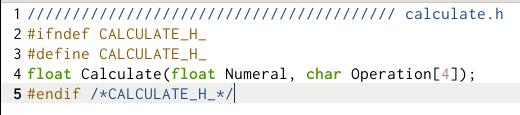


Figure 4: Файл calculate.h

1. Основной файл main.c, реализующий интерфейс пользователя к калькулятору. (рис. 5)

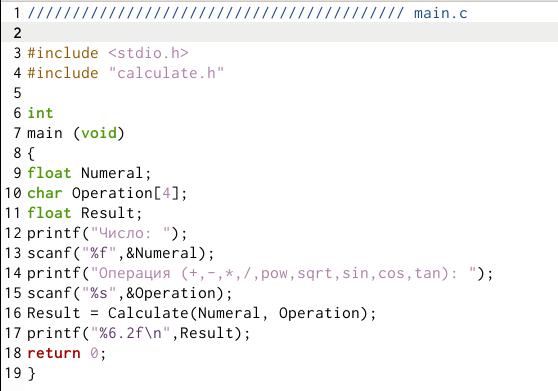


Figure 5: Файл main.c

1. Выполнил компиляцию программы посредством gcc: gcc -c calculate.c gcc -c main.c gcc calculate.o main.o -o calcul -lm (рис. 6)

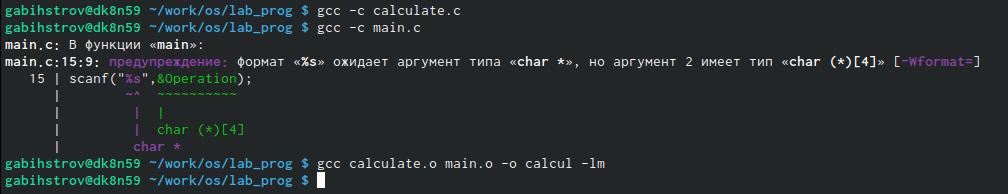


Figure 6: Компиляция

1. Создал Makefile. (рис. 7)

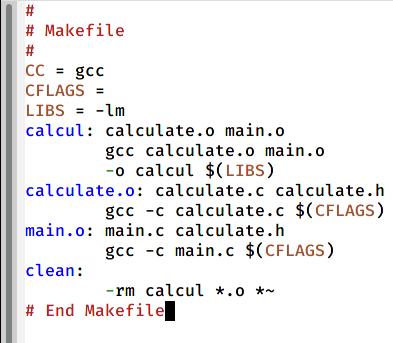


Figure 7: Makefile

1. С помощью gdb выполнил отладку программы calcul (перед использованием gdb исправил Makefile). Запустил отладчик GDB, загрузив в него программу для отладки gdb ./calcul. Для запуска программы внутри отладчика ввёл команду run. (рис. 8)

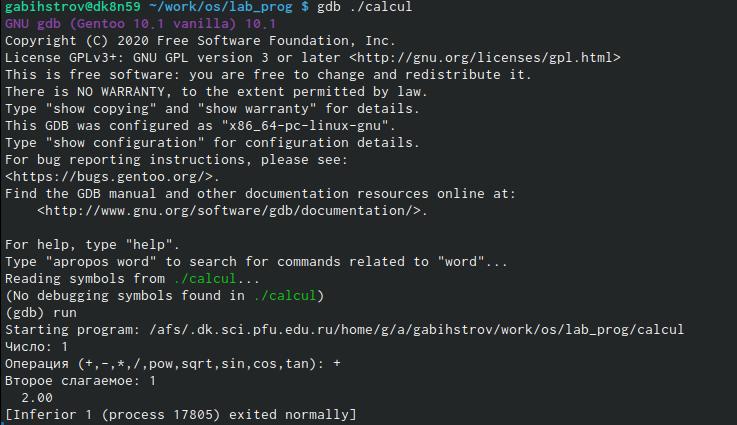


Figure 8: Отладка программы calcul

1. Для постраничного (по 9 строк) просмотра исходного код использовал команду list. (рис. 9)

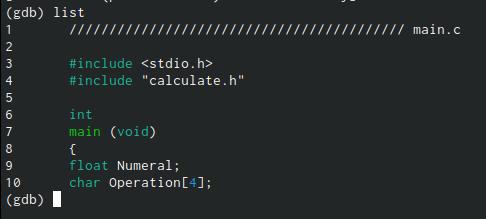


Figure 9: Команда list

1. Для просмотра строк с 12 по 15 основного файла использовал list с параметрами list 12,15. (рис. 10)

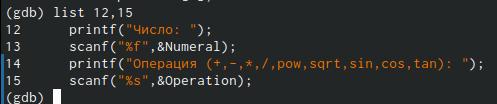


Figure 10: Просмотр строк

1. Для просмотра определённых строк не основного файла используйте list с параметрами: list calculate.c:20,29. (рис. 11)

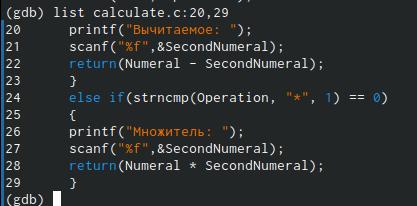


Figure 11: Просмотр определённых строк

1. Установите точку останова в файле calculate.c на строке номер 21: list calculate.c:20,27 break 21 (рис. 12)

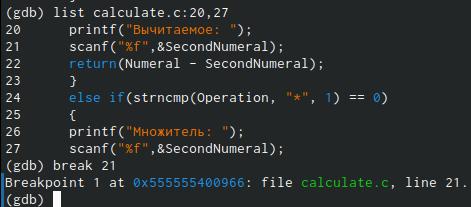


Figure 12: Установил точку останова

1. Выведите информацию об имеющихся в проекте точка останова: info breakpoints. (рис. 13)

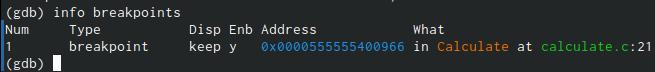


Figure 13: Информация о точке останова

1. Запустил программу внутри отладчика и убедился, что программа остановилась в момент прохождения точки останова (рис. 14)

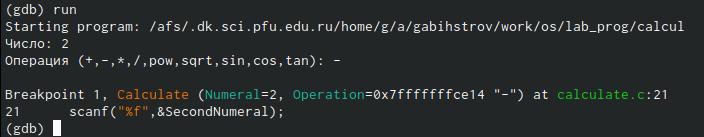


Figure 14: Остановка программы

1. Посмотрите, чему равно на этом этапе значение переменной Numeral, введя: print Numeral. На экран выведено число 2. (рис. 15)

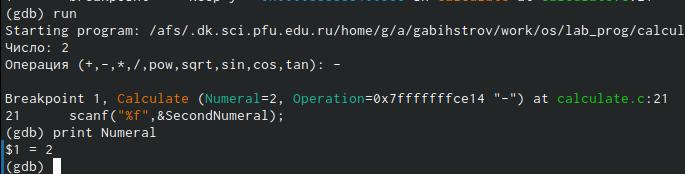


Figure 15: Переменная Numeral

1. Сравнил с результатом вывода на экран после использования команды: display Numeral. Цифры сходятся. (рис. 16)

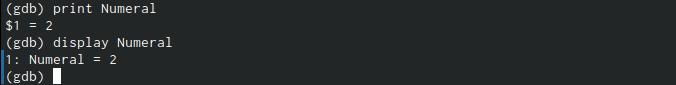


Figure 16: Сравнение цифр

1. Убрал точки останова: info breakpoints delete 1. (рис. 17)

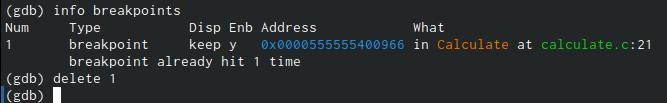


Figure 17: Убрал точки

1. С помощью утилиты splint проанализировал коды файлов calculate.c и main.c. (рис. 18) (рис. 19)

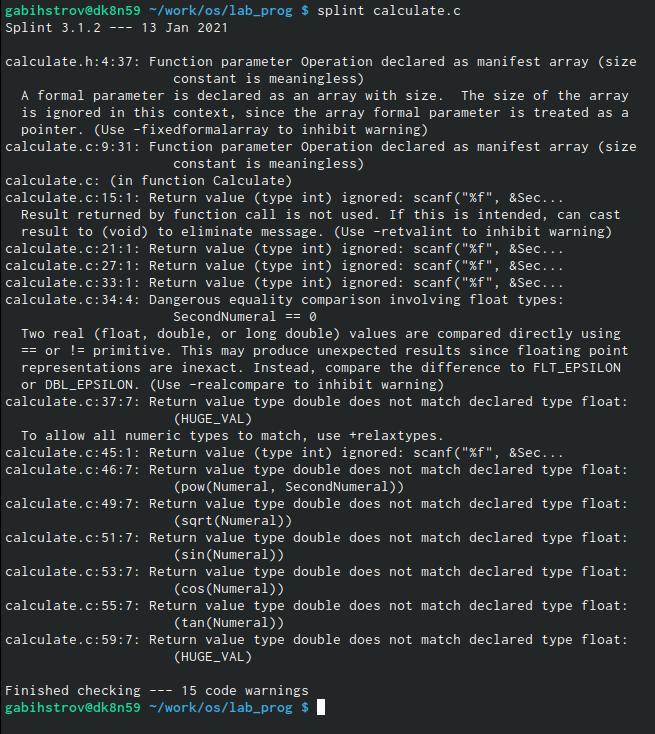


Figure 18: calculate.c

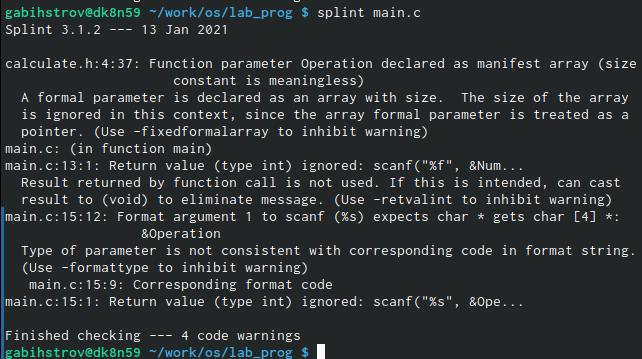


Figure 19: main.c

# Контрольные вопросы

1. Как получить информацию о возможностях программ gcc, make, gdb и др.?

Чтобы получить информацию о возможностях программ gcc, make, gdb необходимо использовать команды man, info или help. Также можно воспользоваться интернетом и узнать необходимую информацию.

1. Назовите и дайте краткую характеристику основным этапам разработки приложений в UNIX.

*Библиографический список ссылка №1*

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 специфи- каций, 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.

1. Что такое суффикс в контексте языка программирования? Приведите примеры использования.

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.

Таким образом, gcc по расширению (суффиксу) .c распознает тип файла для компиляции и формирует объектный модуль — файл с расширением .o.

1. Каково основное назначение компилятора языка С в UNIX?

Стандартным средством для компиляции программ в ОС типа UNIX является GCC (GNU Compiler Collection). Это набор компиляторов для разного рода языков программирования (С, C++, Java, Фортран и др.). Работа с GCC производится при помощи одноимённой управляющей программы gcc, которая интерпретирует аргументы командной строки, определяет и осуществляет запуск нужного компилятора для входного файла. 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.

1. Для чего предназначена утилита make?

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.

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

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.

Рассмотрим пример Makefile для написанной выше простейшей программы, выводящей на экран приветствие ’Hello World!’: hello: main.c gcc -o hello main.c Здесь в первой строке hello — цель, main.c — название файла, который мы хотим скомпилировать; во второй строке, начиная с табуляции, задана команда компиляции gcc с опциями. 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 -g73

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.

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.

* Запустил отладчик для программы;
* установил точку остановки;
* запустил программу;
* узнал необходимые данные путём ввода команд;
* завершил программу.

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

В строке scanf(“%s”,&Operation) необходимо убрать &, так как им массива уже является указателем на первый элемент. Также компилятор писал, что программа принимает указатель на char массив.

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

*Библиографический список ссылка №2*

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;
* lint — критическая проверка программ, написанных на языке Си.

Для постраничного (по 9 строк) просмотра исходного код используйте ко- манду list : list Для просмотра строк с 12 по 15 основного файла используйте list с пара- метрами: list 12,15

1. Каковы основные задачи, решаемые программой splint?

*Библиографический список ссылка №3*

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.

# Выводы

В данной лабораторной работе мне успешно удалось 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.

# Библиографический список

1. Разработка Unix-приложений (https://qna.habr.com/q/17069)
2. Статический анализ кода (https://www.jetinfo.ru/staticheskij-analiz-koda-chto-mogut-instrumentalnye-sredstva/)
3. Sprint layout аналог для linux (https://a174.ru/sprint-layout-analog-dlya-linux/)