Отчет по лабораторной работе №13

Операционные системы

Скворцова Анастасия Дмитриевна

Содержание

[1 Цель работы 1](#_Toc197780548)

[2 Задание 1](#_Toc197780549)

[3 Теоретическое введение 1](#_Toc197780550)

[4 Выполнение лабораторной работы 1](#_Toc197780551)

[5 Выводы 1](#_Toc197780552)

[6 Ответы на контрольные вопросы 1](#_Toc197780553)

# 1 Цель работы

Цель данной лабораторной 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

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.
4. При необходимости исправьте синтаксические ошибки.
5. Создайте Makefile со следующим содержанием.
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.

# 3 Теоретическое введение

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 (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.

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

# 4 Выполнение лабораторной работы

Создаю директорию ~/work/os/lab\_prog, перехожу в нее и в этой директории создаю файлы calculate.h, calculate.c, main.c (рис. [1](#fig:001)).

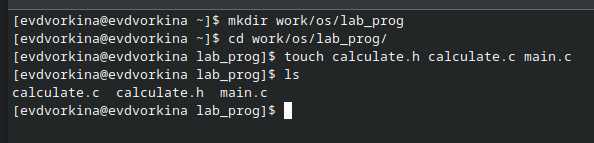


Figure 1: Создание файлов и директорий

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.h (рис. [2](#fig:002)).

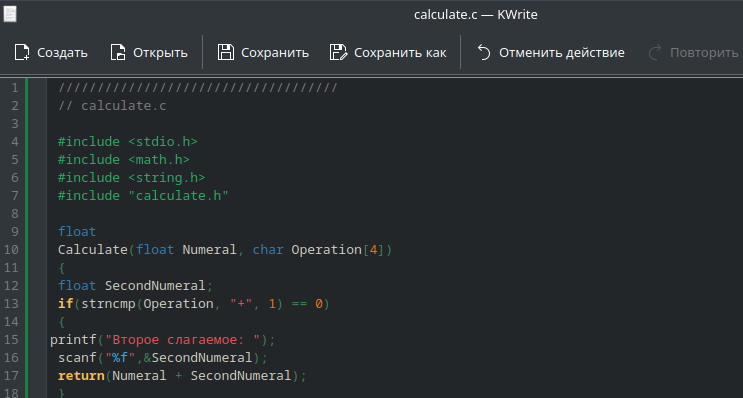


Figure 2: Функционал калькулятора в calculate.c

Интерфейсный файл calculate.h, описывающий формат вызова функции- калькулятора (рис. [3](#fig:003)).

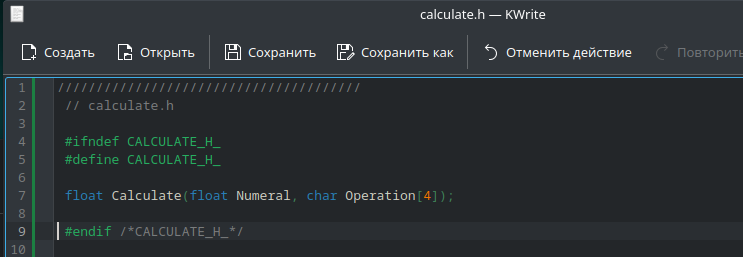


Figure 3: Код файла calculate.h

Основной файл main.c, реализующий интерфейс пользователя к калькулятору (рис. [4](#fig:004)).

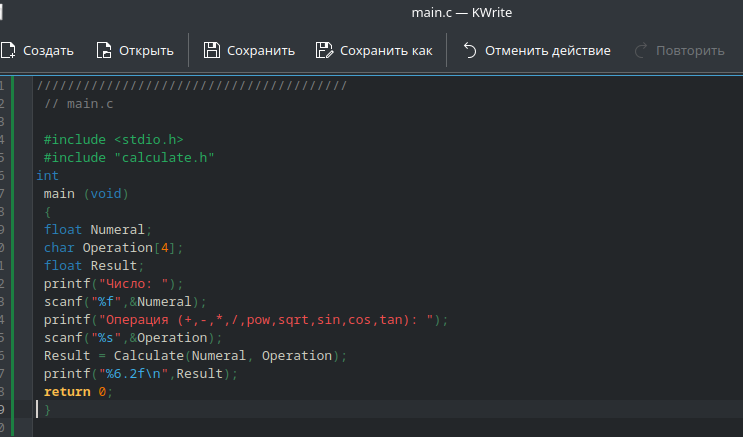


Figure 4: Код файла main.c

Далее выполняю компиляцию программ посредством gcc (как выяснится позже, тут не хватает флажочка) (рис. [5](#fig:005)).

Figure 5: Компиляция программ посредством gcc

Figure 5: Компиляция программ посредством gcc

Далее создаю Makefile с указанным в лабораторной работе содержанием (рис. [6](#fig:006)). Но в нем нужно кое-что исправить.

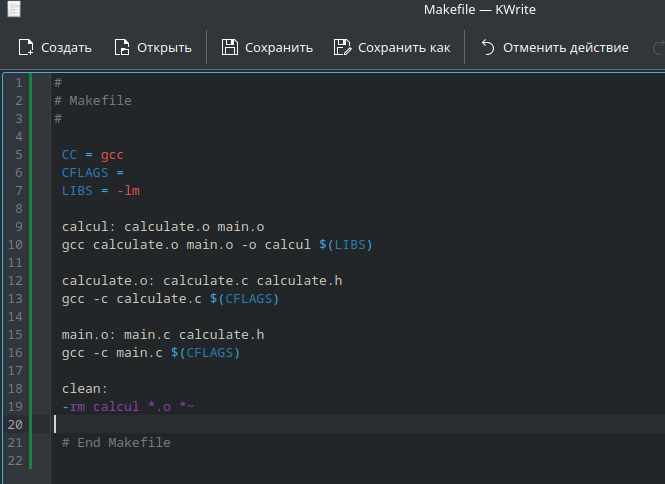


Figure 6: Makefile без исправлений

Исправляю Makefile:

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

В переменную CFLAGS добавляю значение -g (рис. [7](#fig:007)).

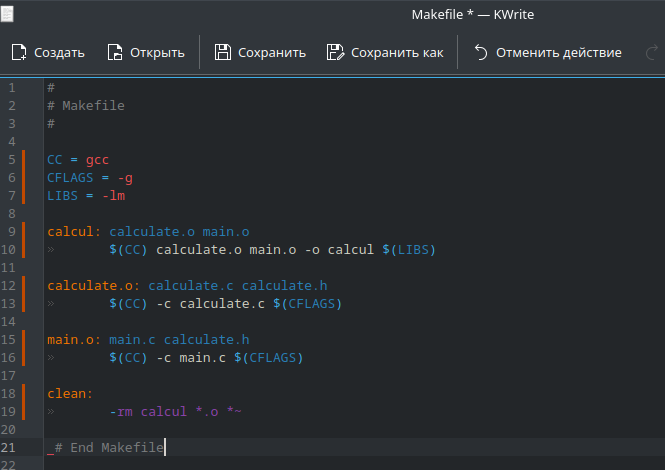


Figure 7: Makefile с исправлениями

#Объявление переменных  
CC = gcc #компилятор  
CFLAGS = -g #опция, которая отладочную информацию положит в результирующий бинарный файл  
LIBS = -lm #  
#Создаем файл calcul из файлов calculate.o main.o  
calcul: calculate.o main.o #ниже обращаемся к содержимому переменной  
 $(CC) calculate.o main.o -o calcul $(LIBS) #добавляем опцию  
#Здесь отражена строчка: gcc calculate.o main.o -o calcul -lm  
#Создаем файл calculate.o  
calculate.o: calculate.c calculate.h #  
 $(CC) -c calculate.c $(CFLAGS) #gcc -c calculate.c -g  
#Создаем файл main.o  
main.o: main.c calculate.h #gcc -c main.c -g  
 $(CC) -c main.c $(CFLAGS) #  
#  
clean: #при вызове make clean будем удалять все файлы с разрешением .о  
 -rm calcul \*.o \*~ #

Далее использую make clean, чтобы удалить не совсем верно скомпилированные файлы, и использую make (рис. [8](#fig:008)).

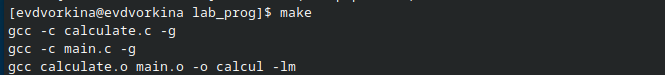


Figure 8: Компиляция с помощью Makefile

Запусткаю отладчик GDB, загрузив в него программу для отладки (рис. [9](#fig:009)).

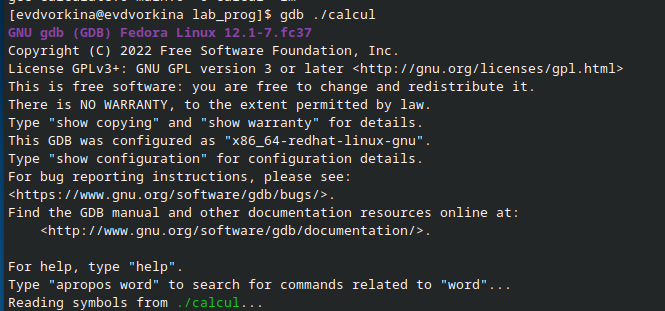


Figure 9: Запуск GDB

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

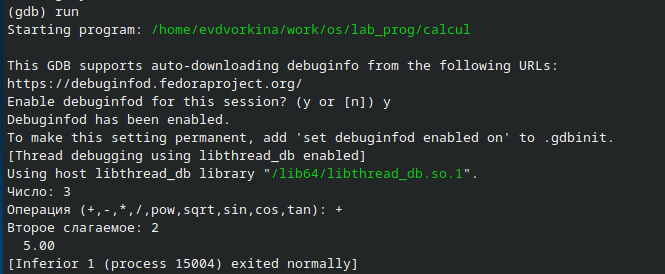


Figure 10: Запуск программы

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

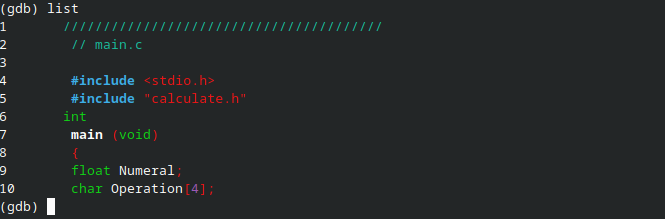


Figure 11: Постраничный просмотр кода

Для просмотра строк с 12 по 15 основного файла использую list с параметрами (рис. [12](#fig:012)).

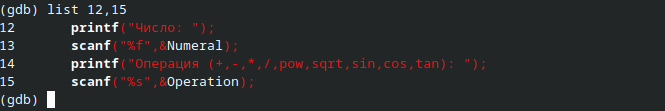


Figure 12: Просмотр определенных строк кода

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

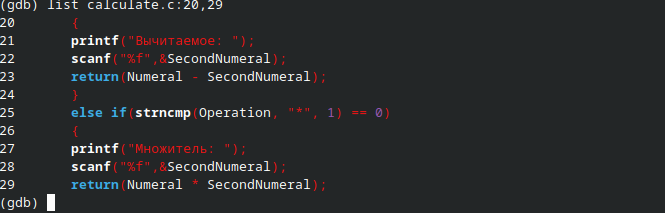


Figure 13: Просмотр определенных строк не основного файла

Устанавливаю точку остановки в файле calculate.c на строке номер 21 (break 21) (рис. [14](#fig:014)).

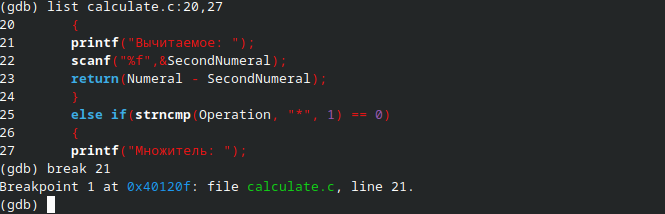


Figure 14: Установка точки остановки

Вывожу информацию об имеющихся в проекте точках остановки (рис. [15](#fig:015)).

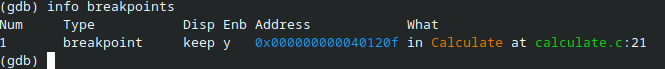


Figure 15: Информация об имеющихся точках остановки

Запускаю программу внутри отладчика и убеждаюсь, что программа останавливается в момент прохождения точки остановки: (рис. [16](#fig:016)).

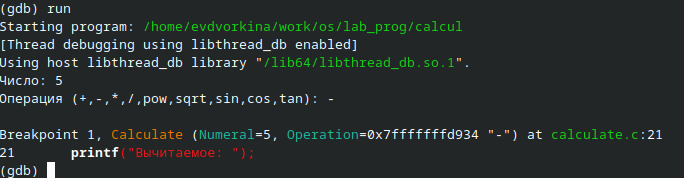


Figure 16: Запуск программы

Команда backtrace покажет весь стек вызываемых функций от начала програм- мы до текущего места (рис. [17](#fig:017)).

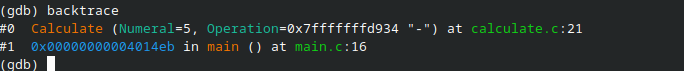


Figure 17: Стек вызываемых функций

Смотрю, чему равно на этом этапе значение переменной Numeral, это можно сделать двумя способами. При первом я получу вывод значения переменной из bash-скрипта, второй способ более интуитивный, там значение соответствует переменной из кода на Си (рис. [18](#fig:018)).

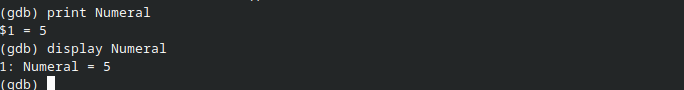


Figure 18: Просмотр промежуточного значения переменной Numeral

Удаляю точки остановки (рис. [19](#fig:019)).

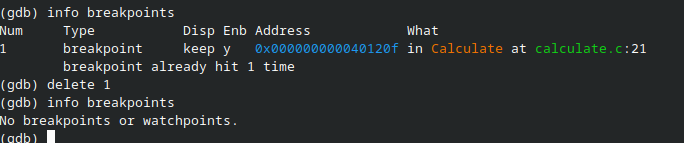


Figure 19: Удаление точек остановки

С помощью утилиты splint пробую проанализировать код файла calculate.c. (рис. [20](#fig:020)).

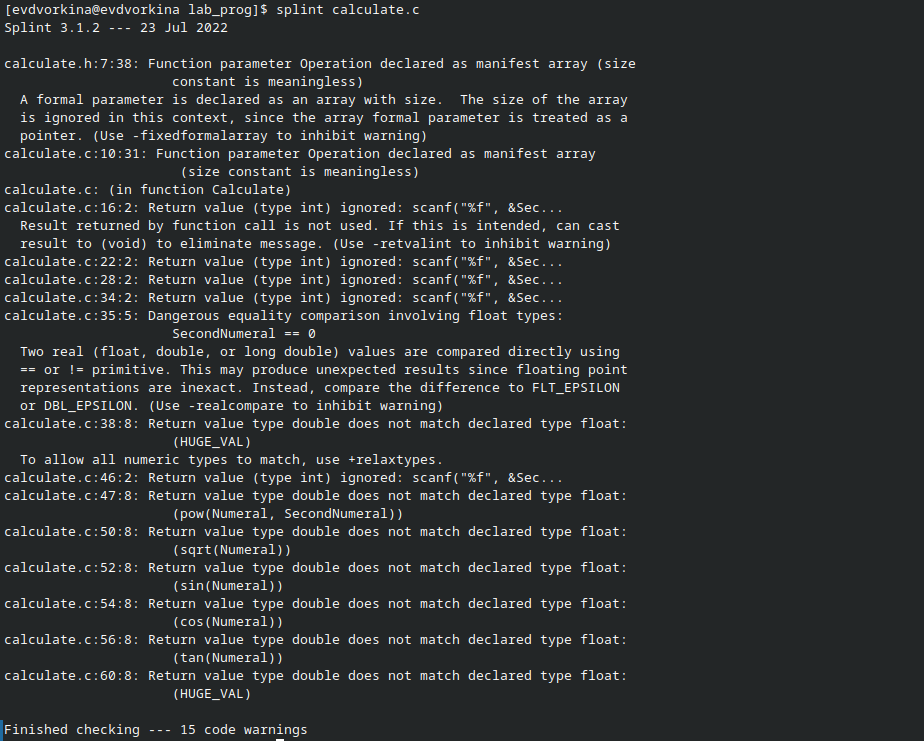


Figure 20: Анализ кода файла calculate.c

С помощью утилиты splint пробую проанализировать код файла main.c. (рис. [21](#fig:021)).

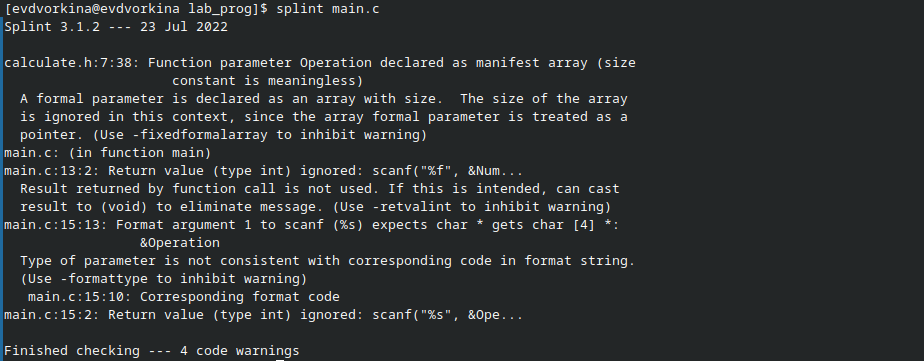


Figure 21: Анализ кода файла main.c

# 5 Выводы

При выполнениии 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

# 6 Ответы на контрольные вопросы

1. Как получить информацию о возможностях программ gcc, make, gdb и др.? Дополнительную информацию об этих программах можно получить с помощью функций info и man.
2. Назовите и дайте краткую характеристику основным этапам разработки приложений в UNIX. Unix поддерживает следующие основные этапы разработки приложений:

* создание исходного кода программы;
* представляется в виде файла;
* сохранение различных вариантов исходного текста;
* анализ исходного текста; Необходимо отслеживать изменения исходного кода, а также при работе более двух программистов над проектом программы нужно, чтобы они не делали изменений кода в одно время.
* компиляция исходного текста и построение исполняемого модуля;
* тестирование и отладка;
* проверка кода на наличие ошибок
* сохранение всех изменений, выполняемых при тестировании и отладке.

1. Что такое суффикс в контексте языка программирования? Приведите примеры использования. Использование суффикса “.с” для имени файла с программой на языке Си отражает удобное и полезное соглашение, принятое в ОС UNIX. Для любого имени входного файла суффикс определяет какая компиляция требуется. Суффиксы и префиксы указывают тип объекта. Одно из полезных свойств компилятора Си — его способность по суффиксам определять типы файлов. По суффиксу .c компилятор распознает, что файл abcd.c должен компилироваться, а по суффиксу .o, что файл abcd.о является объектным модулем и для получения исполняемой программы необходимо выполнить редактирование связей. Простейший пример командной строки для компиляции программы abcd.c и построения исполняемого модуля abcd имеет вид: gcc -o abcd abcd.c. Некоторые проекты предпочитают показывать префиксы в начале текста изменений для старых (old) и новых (new) файлов. Опция – prefix может быть использована для установки такого префикса. Плюс к этому команда bzr diff -p1 выводит префиксы в форме которая подходит для команды patch -p1.
2. Каково основное назначение компилятора языка С в UNIX? Основное назначение компилятора с языка Си заключается в компиляции всей программы в целом и получении исполняемого модуля.
3. Для чего предназначена утилита make? При разработке большой программы, состоящей из нескольких исходных файлов заголовков, приходится постоянно следить за файлами, которые требуют перекомпиляции после внесения изменений. Программа make освобождает пользователя от такой рутинной работы и служит для документирования взаимосвязей между файлами. Описание взаимосвязей и соответствующих действий хранится в так называемом make-файле, который по умолчанию имеет имя makefile или Makefile.
4. Приведите пример структуры Makefile. Дайте характеристику основным элементам этого файла. makefile для программы abcd.c мог бы иметь вид:

Makefile  
  
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 \*~  
  
End Makefile

В общем случае make-файл содержит последовательность записей (строк), определяющих зависимости между файлами. Первая строка записи представляет собой список целевых (зависимых) файлов, разделенных пробелами, за которыми следует двоеточие и список файлов, от которых зависят целевые. Текст, следующий за точкой с запятой, и все последующие строки, начинающиеся с литеры табуляции, являются командами OC UNIX, которые необходимо выполнить для обновления целевого файла. Таким образом, спецификация взаимосвязей имеет формат: target1 [ target2…]: [:][dependment1…] [(tab)commands] [#commentary] [(tab)commands] [#commentary], где # — специфицирует начало комментария, так как содержимое строки, начиная с # и до конца строки, не будет обрабатываться командой make; : — последовательность команд ОС UNIX должна содержаться в одной строке make-файла (файла описаний), есть возможность переноса команд (), но она считается как одна строка; :: — последовательность команд ОС UNIX может содержаться в нескольких последовательных строках файла описаний. Приведённый выше make-файл для программы abcd.c включает два способа компиляции и построения исполняемого модуля. Первый способ предусматривает обычную компиляцию с построением исполняемого модуля с именем abcd. Второй способ позволяет включать в исполняемый модуль testabcd возможность выполнить процесс отладки на уровне исходного текста.

1. Назовите основное свойство, присущее всем программам отладки. Что необходимо сделать, чтобы его можно было использовать? Пошаговая отладка программ заключается в том, что выполняется один оператор программы и, затем контролируются те переменные, на которые должен был воздействовать данный оператор. Если в программе имеются уже отлаженные подпрограммы, то подпрограмму можно рассматривать, как один оператор программы и воспользоваться вторым способом отладки программ. Если в программе существует достаточно большой участок программы, уже отлаженный ранее, то его можно выполнить, не контролируя переменные, на которые он воздействует. Использование точек останова позволяет пропускать уже отлаженную часть программы. Точка останова устанавливается в местах, где необходимо проверить содержимое переменных или просто проконтролировать, передаётся ли управление данному оператору. Практически во всех отладчиках поддерживается это свойство (а также выполнение программы до курсора и выход из подпрограммы). Затем отладка программы продолжается в пошаговом режиме с контролем локальных и глобальных переменных, а также внутренних регистров микроконтроллера и напряжений на выводах этой микросхемы.
2. Назовите и дайте основную характеристику основным командам отладчика gdb.

backtrace – выводит весь путь к текущей точке останова, то есть названия всех функций, начиная от main(); иными словами, выводит весь стек функций;

break – устанавливает точку останова; параметром может быть номер строки или название функции;

clear – удаляет все точки останова на текущем уровне стека (то есть в текущей функции);

continue – продолжает выполнение программы от текущей точки до конца;

delete – удаляет точку останова или контрольное выражение;

display – добавляет выражение в список выражений, значения которых отображаются каждый раз при остановке программы;

finish – выполняет программу до выхода из текущей функции; отображает возвращаемое значение,если такое имеется;

info breakpoints – выводит список всех имеющихся точек останова;

info watchpoints – выводит список всех имеющихся контрольных выражений;

splist – выводит исходный код; в качестве параметра передаются название файла исходного кода, затем, через двоеточие, номер начальной и конечной строки;

next – пошаговое выполнение программы, но, в отличие от команды step, не выполняет пошагово вызываемые функции;

print – выводит значение какого-либо выражения (выражение передаётся в качестве параметра);

run – запускает программу на выполнение;

set – устанавливает новое значение переменной

step – пошаговое выполнение программы;

watch – устанавливает контрольное выражение, программа остановится, как только значение контрольного выражения изменится;

1. Опишите по шагам схему отладки программы, которую Вы использовали при выпол- нении лабораторной работы. Выполнили компиляцию программы 2)Увидели ошибки в программе Открыли редактор и исправили программу Загрузили программу в отладчик gdb run — отладчик выполнил программу, мы ввели требуемые значения. программа завершена, gdb не видит ошибок.
2. Прокомментируйте реакцию компилятора на синтаксические ошибки в программе при его первом запуске. Отладчику не понравился формат %s для &Operation, т.к %s — символьный формат, а значит необходим только Operation.
3. Назовите основные средства, повышающие понимание исходного кода программы. 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;

splint — критическая проверка программ, написанных на языке Си.

1. Каковы основные задачи, решаемые программой splint?

Проверка корректности задания аргументов всех исполняемых функций , а также типов возвращаемых ими значений;

Поиск фрагментов исходного текста, корректных с точки зрения синтаксиса языка Си, но малоэффективных с точки зрения их реализации или содержащих в себе семантические ошибки;

Общая оценка мобильности пользовательской программы.