Лабораторная работа №14

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

Сунгурова Мариян Мухсиновна

Содержание

Список иллюстраций

1 Цель работы

Приобрести простейшие навыки разработки, анализа, тестирования и отладки при- ложений в ОС типа UNIX/Linux на примере создания на языке программирования C 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

Основная среда взаимодействия с UNIX — командная строка. Суть её в том, что каждая строка, передаваемая пользователем системе, — это команда, которую та должна выполнить. Более подробно об Unix см. в [1–6].

4 Выполнение лабораторной работы

1. 1. В домашнем каталоге создайте подкаталог ~/work/os/lab_prog.(рис. 4.1)

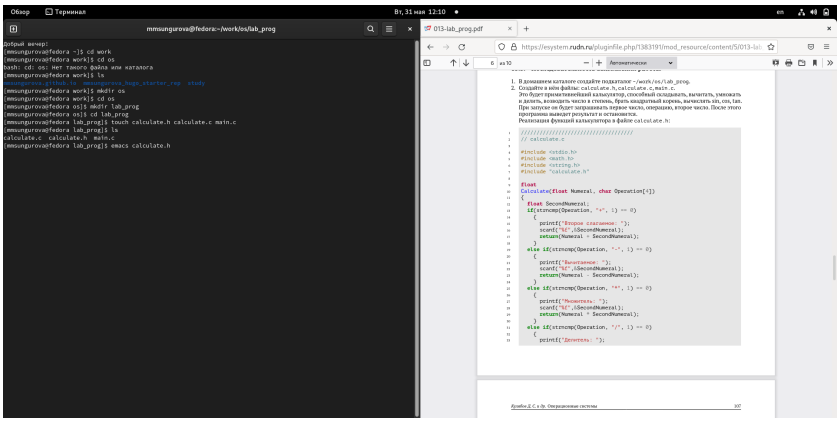


Рис. 4.1: Терминал

2.2. 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. (рис. 4.2, 4.3, 4.4)

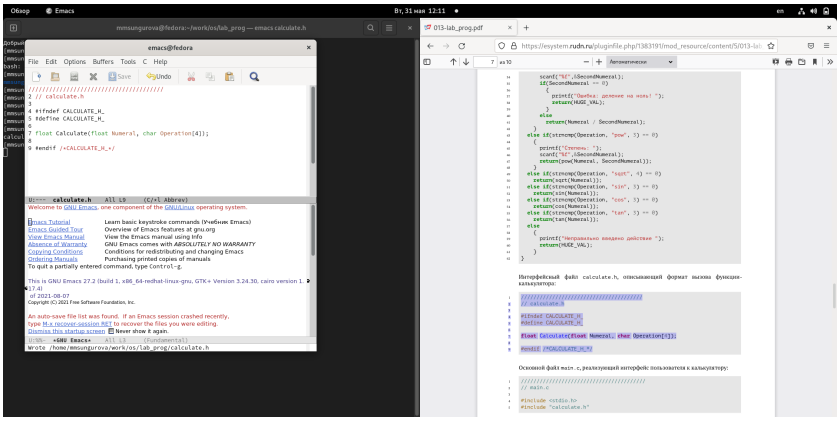


Рис. 4.2: Терминал

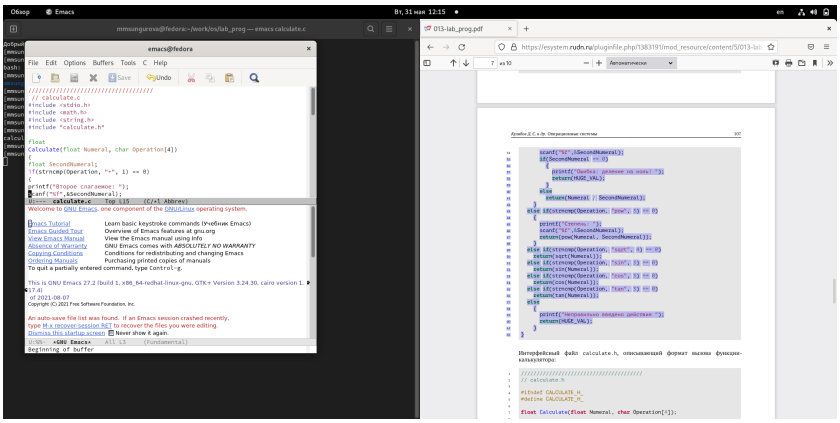


Рис. 4.3: Терминал

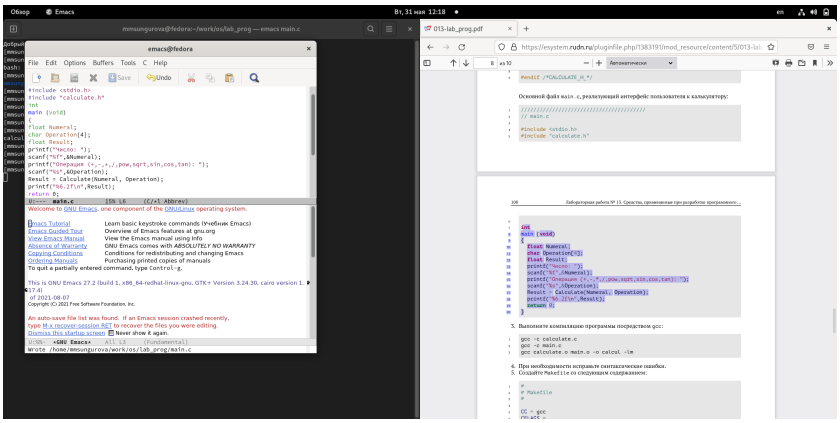


Рис. 4.4: Терминал

3.Выполните компиляцию программы посредством gcc:(рис. 4.5)

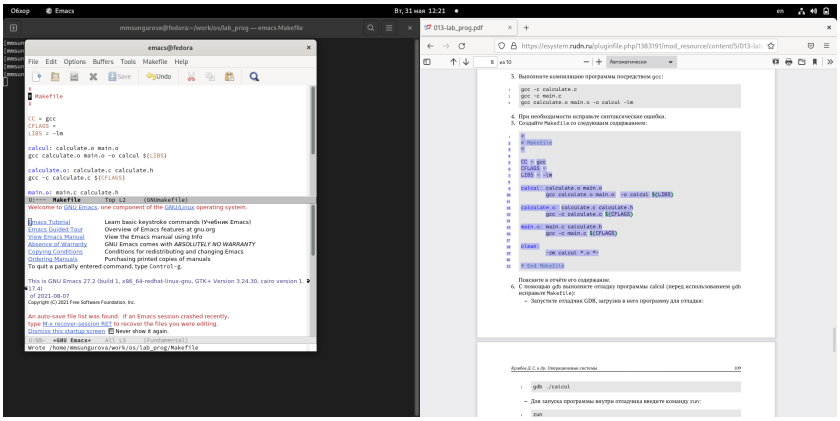


Рис. 4.5: Терминал

4. При необходимости исправьте синтаксические ошибки.
5. Создайте Makefile со следующим содержанием(рис. 4.6)

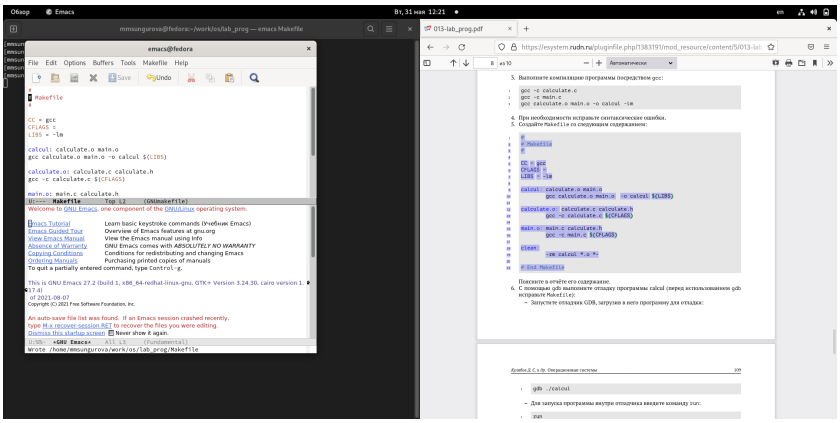


Рис. 4.6: Терминал

6. С помощью gdb выполните отладку программы calcul (перед использованием gdb исправьте Makefile): – Запустите отладчик GDB, загрузив в него программу для отладки:(рис. 4.7)

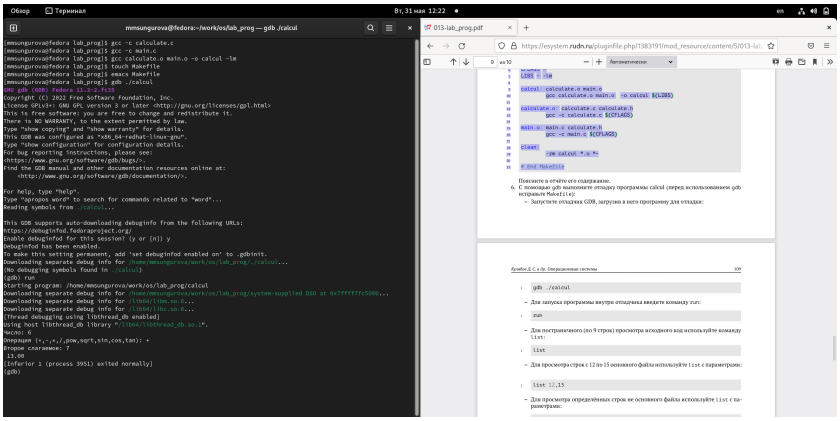


Рис. 4.7: Терминал

– Для запуска программы внутри отладчика введите команду run:

– Установите точку останова в файле calculate.c на строке номер 21:

Выведите информацию об имеющихся в проекте точка останова:(рис. 4.8)

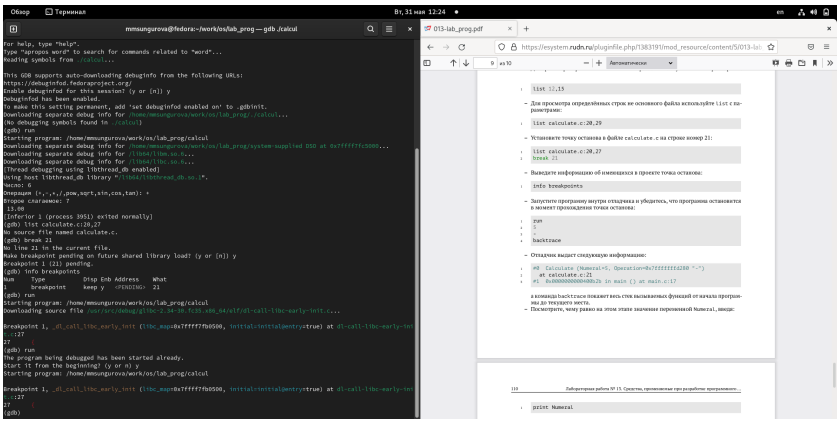


Рис. 4.8: Терминал

– Посмотрите, чему равно на этом этапе значение переменной Numeral, введя:

1 print Numeral На экран должно быть выведено число 5.(рис. 4.9)

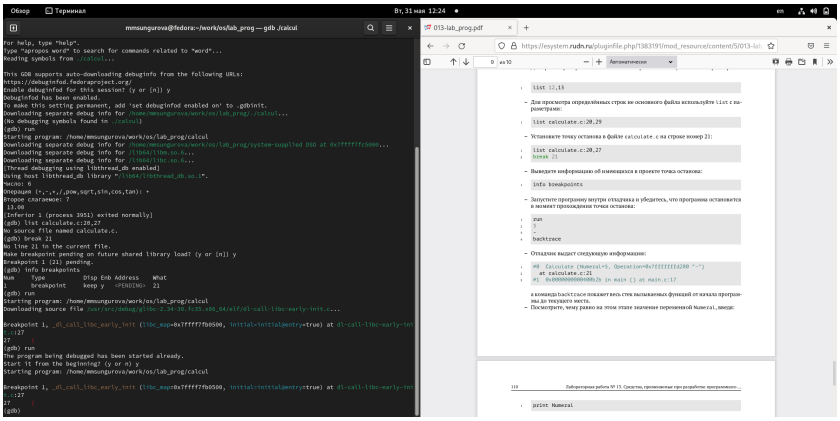


Рис. 4.9: Терминал

– Сравните с результатом вывода на экран после использования команды: 1 display Numeral – Уберите точки останова: 1 info breakpoints 2 delete 1 (рис. 4.10)

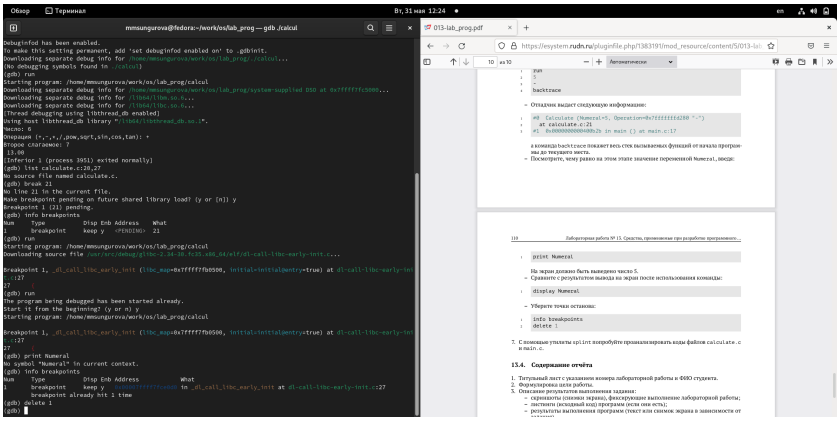
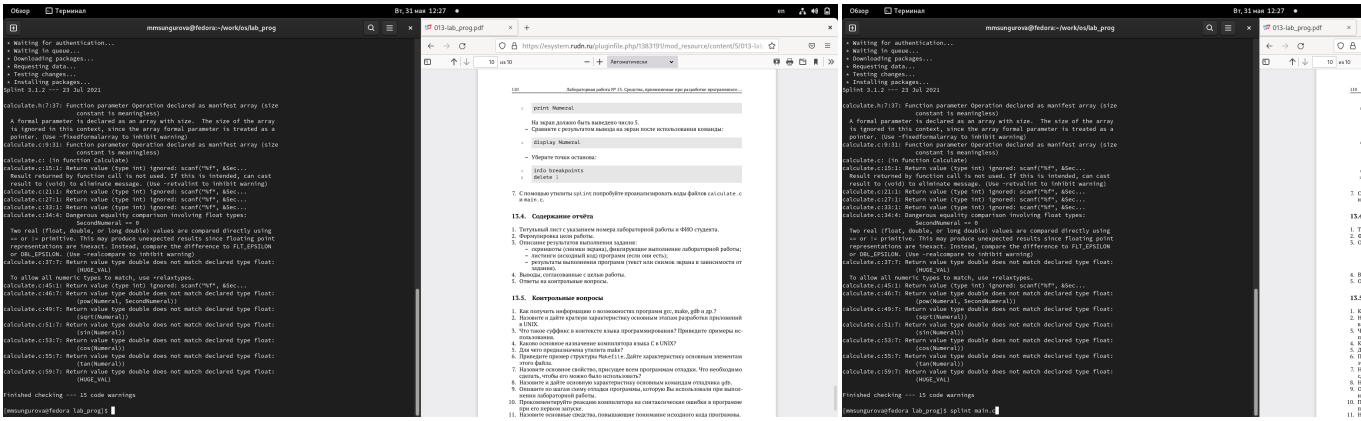


Рис. 4.10: Терминал

7. С помощью утилиты splint попробуйте проанализировать коды файлов calculate.c и main.c. (рис. **??**, **??**)


Обзор
Терминал
Вт, 31 мая 12:27
mmsungurova@fedora:~/work/os/lab_prog
013-lab_prog.pdf
https://esystem.rudn.ru/pluginfile.php/1383191/mod_resource/content/5/013-lab
Автоматически
Лабораторная работа № 13. Средства, применяемые при разработке программного ...
print Numeral
На экран должно быть выведено число 5.
Сравните с результатом вывода на экран после использования команды:
display Numeral
Уберите точки останова:
info breakpoints
delete 1
7. С помощью утилиты cpl (cc) попробуйте проанализировать коды файлов calculate.c и main.c.
13.4. Содержание отчёта
1. Титульный лист с указанием номера лабораторной работы и ФИО студента.
2. Формулировка цели работы.
3. Описание результатов выполнения задания:
скриншоты (снимки экрана), фиксирующие выполнение лабораторной работы;
листинги (исходный код) программ (если они есть);
результаты выполнения программ (текст или снимок экрана в зависимости от задания).
4. Выводы, согласованные с целью работы.
5. Ответы на контрольные вопросы.
13.5. Контрольные вопросы
1. Как получить информацию о возможностях программ gcc, make, gdb и др.?
2. Назовите и дайте краткую характеристику основным этапам разработки приложений в UNIX.
3. Что такое суффикс в контексте языка программирования? Приведите примеры использования.
4. Каково основное назначение компилятора языка C в UNIX?
5. Для чего предназначена утилита make?
6. Приведите пример структуры Makefile. Дайте характеристику основным элементам этого файла.
7. Назовите основное свойство, присущее всем программам отладки. Что необходимо сделать, чтобы его можно было использовать?
8. Назовите и дайте основную характеристику основным командам отладчика gdb.
9. Опишите по шагам схему отладки программы, которую Вы использовали при выполнении лабораторной работы.
10. Прокомментируйте реакцию компилятора на синтаксические ошибки и программе при его первом запуске.
11. Назовите основные средства, повышающие понимание исходного кода программы.
Waiting for authentication...
Waiting in queue...
Downloading packages...
Requesting data...
Testing changes...
Installing packages...
Splint 3.1.2 --- 23 Jul 2021
calculate.h:7:37: Function parameter Operation declared as manifest array (size constant is meaningless)
A formal parameter is declared as an array with size. The size of the array is ignored in this context, since the array formal parameter is treated as a pointer. (Use -fixedformalarray to inhibit warning)
calculate.c:9:31: Function parameter Operation declared as manifest array (size constant is meaningless)
calculate.c: (in function Calculate)
calculate.c:15:1: Return value (type int) ignored: scanf("%f", &Sec...
Result returned by function call is not used. If this is intended, can cast result to (void) to eliminate message. (Use -retvalint to inhibit warning)
calculate.c:21:1: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:27:1: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:33:1: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:34:4: Dangerous equality comparison involving float types: SecondNumeral == 0
Two real (float, double, or long double) values are compared directly using == or != primitive. This may produce unexpected results since floating point representations are inexact. Instead, compare the difference to FLT_EPSILON or DBL_EPSILON. (Use -realcompare to inhibit warning)
calculate.c:37:7: Return value type double does not match declared type float: (HUGE_VAL)
To allow all numeric types to match, use +relaxtypes.
calculate.c:45:1: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:46:7: Return value type double does not match declared type float: (pow(Numeral, SecondNumeral))
calculate.c:49:7: Return value type double does not match declared type float: (sqrt(Numeral))
calculate.c:51:7: Return value type double does not match declared type float: (sin(Numeral))
calculate.c:53:7: Return value type double does not match declared type float: (cos(Numeral))
calculate.c:55:7: Return value type double does not match declared type float: (tan(Numeral))
calculate.c:59:7: Return value type double does not match declared type float: (HUGE_VAL)
Finished checking --- 15 code warnings
[mmsungurova@fedora lab_prog]$
splint main.c

5 Выводы

В результате 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.

6 Контрольные вопросы

1. Дополнительную информацию о этих программах можно получить с помощью функций info и man.

2. 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.

3. Использование суффикса “.c” 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.o 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.

4. Основное назначение компилятора с языка Си заключается в компиляции всей программы в целом и получении исполняемого модуля.

5. 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.

6. 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

7. 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 ОС 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.

   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.

8. – 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;

9. Выполнили компиляцию программы 2)Увидели ошибки в программе 3) Открыли ре

10. 1 и 2.) Мы действительно забыли закрыть комментарии; 3.) отладчику не понравился формат %s для &Operation, т.к %s — символьный формат, а значит необходим только Operation.

Если вы работаете с исходным кодом, который не вами разрабатывался, то назначение

– cscope - исследование функций, содержащихся в программе;

– splint — критическая проверка программ, написанных на языке Си.

12. Проверка корректности задания аргументов всех исп

функций, а также типов возвращаемых ими значений;

Поиск фрагментов исходного текста, корректных с точки зрения синтаксиса языка Си,

Общая оценка мобильности пользовательской программы.

Список литературы

1. GNU Bash Manual [Электронный ресурс]. Free Software Foundation, 2016. URL: https://www.gnu.org/software/bash/manual/.
2. Newham C. Learning the bash Shell: Unix Shell Programming. O'Reilly Media, 2005. 354 с.
3. Zarrelli G. Mastering Bash. Packt Publishing, 2017. 502 с.
4. Robbins A. Bash Pocket Reference. O'Reilly Media, 2016. 156 с.
5. Таненбаум Э. Архитектура компьютера. 6-е изд. СПб.: Питер, 2013. 874 с.
6. Таненбаум Э., Бос Х. Современные операционные системы. 4-е изд. СПб.: Питер, 2015. 1120 с.