ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №13

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

Анастасия Павловна Баранова, НБИбд-01-21

Содержание

# 1 Цель работы

Целью данной лабораторной работы является 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. Поясните в отчёте его содержание.
6. С помощью gdb выполните отладку программы calcul (перед использованием gdb исправьте Makefile): – Запустите отладчик GDB, загрузив в него программу для отладки: gdb ./calcul – Для запуска программы внутри отладчика введите команду run: run – Для постраничного (по 9 строк) просмотра исходного код используйте команду list: list – Для просмотра строк с 12 по 15 основного файла используйте list с параметрами: list 12,15 – Для просмотра определённых строк не основного файла используйте list с параметрами: list calculate.c:20,29 – Установите точку останова в файле calculate.c на строке номер 21: list calculate.c:20,27 break 21 – Выведите информацию об имеющихся в проекте точка останова: info breakpoints – Запустите программу внутри отладчика и убедитесь, что программа остановится в момент прохождения точки останова: run 5

* backtrace – Отладчик выдаст следующую информацию: #0 Calculate (Numeral=5, Operation=0x7fffffffd280 “-”) at calculate.c:21 #1 0x0000000000400b2b in main () at main.c:17 а команда backtrace покажет весь стек вызываемых функций от начала программы до текущего места. – Посмотрите, чему равно на этом этапе значение переменной Numeral, введя: 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

1. С помощью утилиты splint попробуйте проанализировать коды файлов calculate.c и main.c.

# 3 Выполнение лабораторной работы

В домашнем каталоге создам подкаталог ~/work/os/lab\_prog. (рис. 1)

Рис. 1: В домашнем каталоге создам подкаталог ~/work/os/lab_prog.

Рис. 1: В домашнем каталоге создам подкаталог ~/work/os/lab\_prog.

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

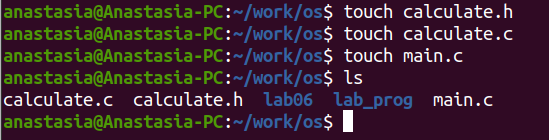


Рис. 2: Создам в нём файлы: calculate.h, calculate.c, main.c.

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

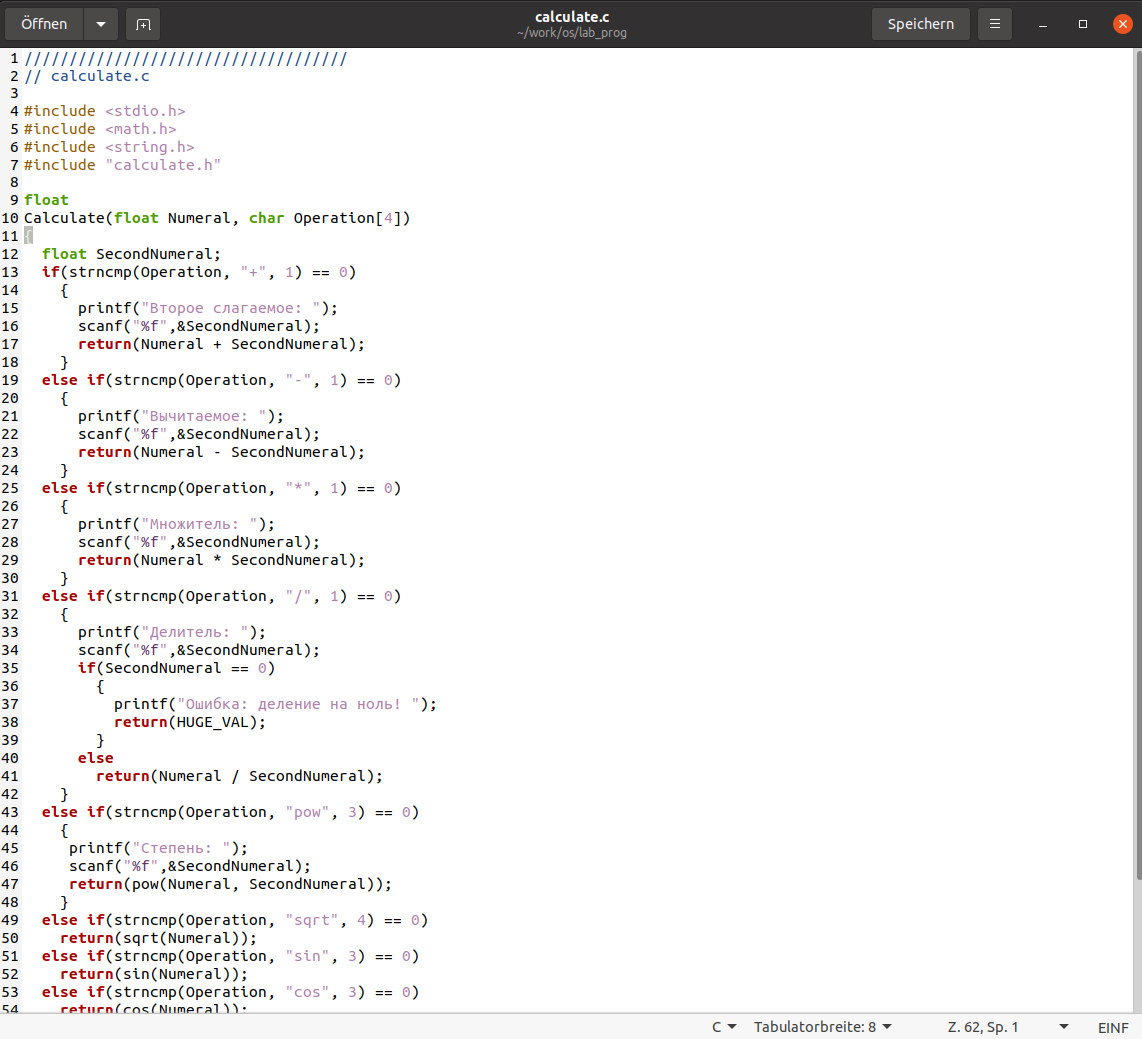


Рис. 3: Реализация функций калькулятора в файле calculate.c.

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



Рис. 4: Интерфейсный файл calculate.h.

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



Рис. 5: Основной файл main.c.

Выполню компиляцию программы посредством gcc (рис. 6):

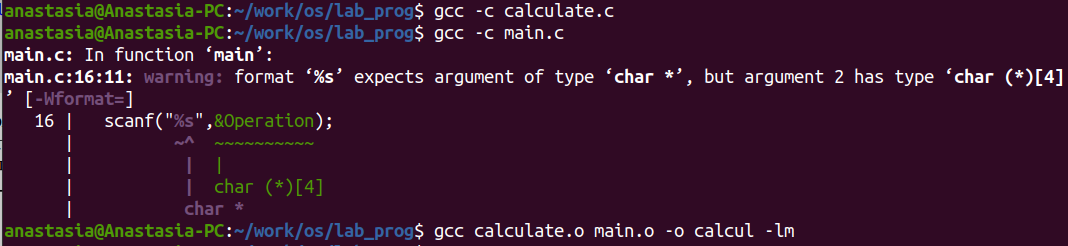


Рис. 6: Выполню компиляцию программы посредством gcc.

Создам Makefile со следующим содержанием (рис. 7):

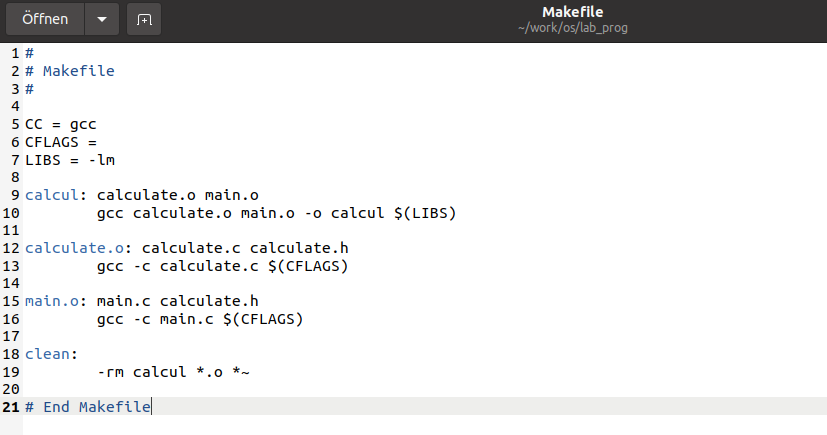


Рис. 7: Создам Makefile со следующим содержанием.

В содержании файла указаны флаги компиляции, тип компилятора и файлы, которые должен собрать сборщик.

С помощью gdb выполню отладку программы calcul (перед использованием gdb исправлю Makefile):

* Запущу отладчик GDB, загрузив в него программу для отладки: gdb ./calcul (рис. 8, рис. 9)

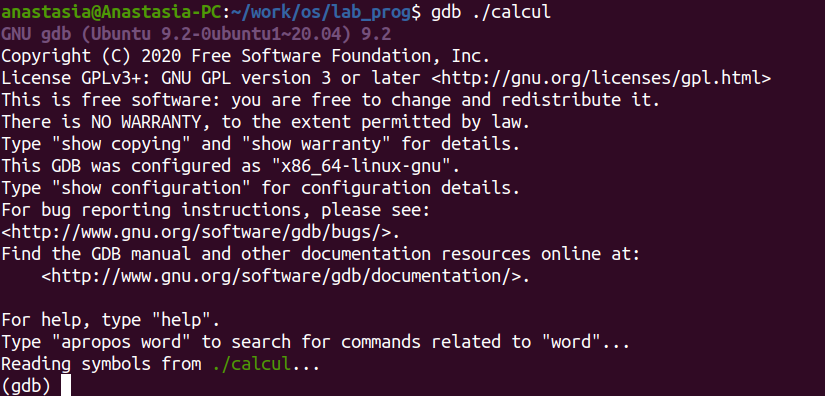


Рис. 8: Запущу отладчик GDB.

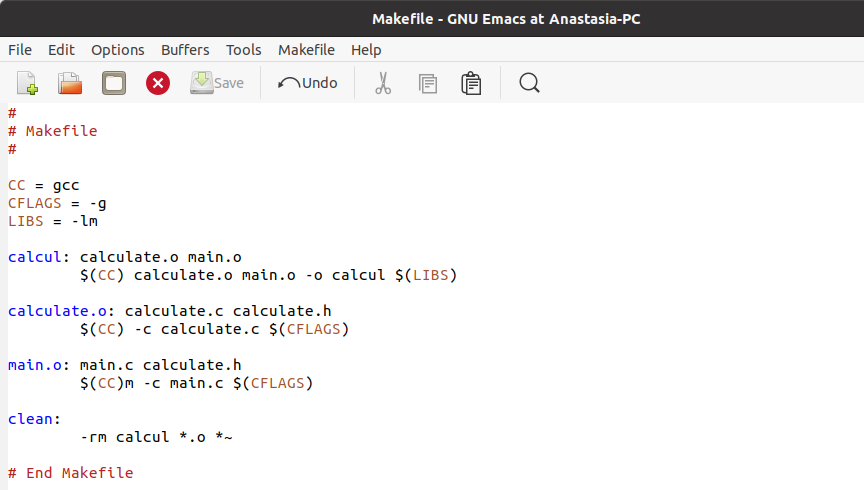


Рис. 9: Перед использованием gdb исправлю Makefile.

* Для запуска программы внутри отладчика введу команду run: run (рис. 10)

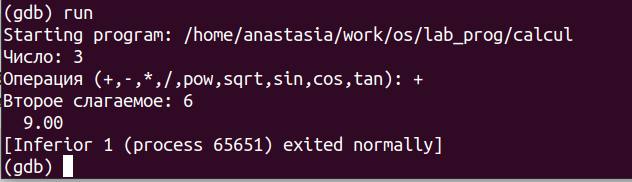


Рис. 10: Для запуска программы внутри отладчика введу команду run.

* Для постраничного (по 9 строк) просмотра исходного кода использую команду list: list (рис. 11)

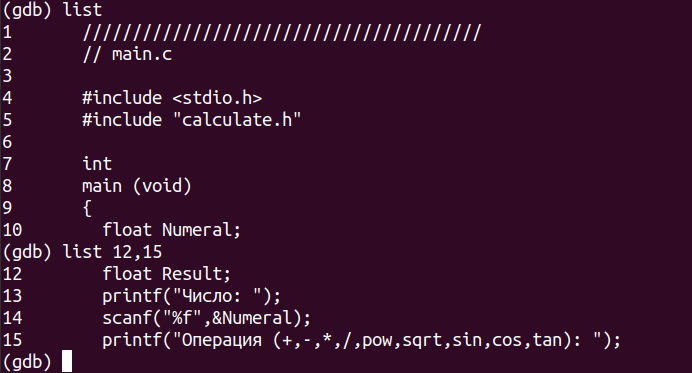


Рис. 11: Для постраничного просмотра кода использую команду list.

* Для просмотра строк с 12 по 15 основного файла использую list с параметрами: list 12,15 (рис. 11)
* Для просмотра определённых строк не основного файла использую list с параметрами: list calculate.c:20,29 (рис. 12)

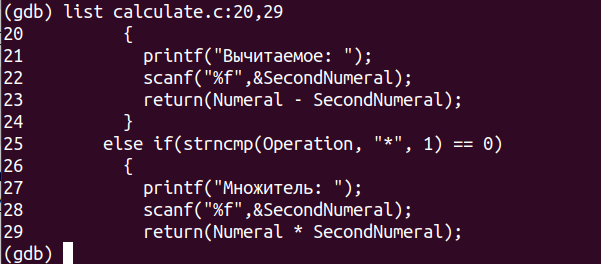


Рис. 12: Использую list с параметрами: list calculate.c:20,29.

* Установлю точку останова в файле calculate.c на строке номер 21 (рис. 13): list calculate.c:20,27 break 21

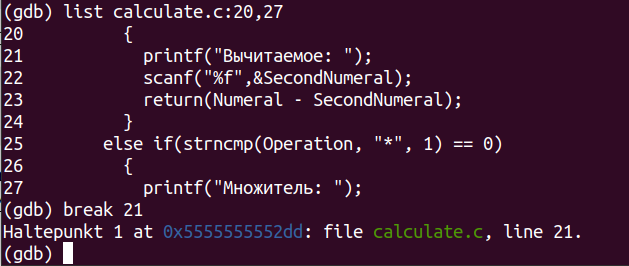


Рис. 13: Установлю точку останова.

* Выведу информацию об имеющихся в проекте точках останова (рис. 14): info breakpoints

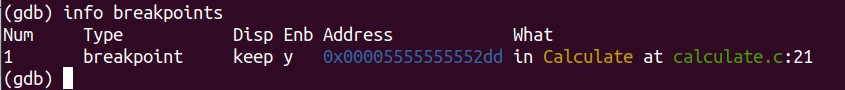


Рис. 14: Выведу информацию об имеющихся точках останова.

* Запущу программу внутри отладчика и проверю, что программа остановится в момент прохождения точки останова (рис. 15): run 5
  + backtrace

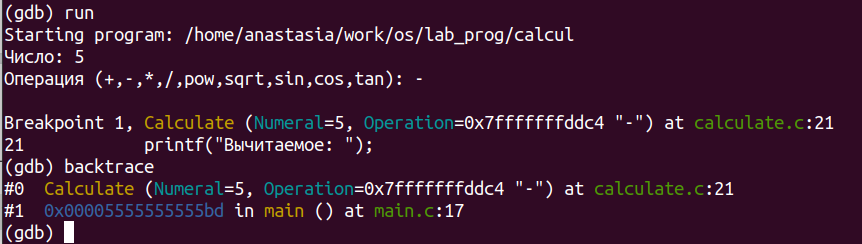


Рис. 15: Запущу программу внутри отладчика.

* Отладчик выдал следующую информацию (рис. 15): #0 Calculate (Numeral=5, Operation=0x7fffffffd280 “-”) at calculate.c:21 #1 0x0000000000400b2b in main () at main.c:17 а команда backtrace показала весь стек вызываемых функций от начала программы до текущего места.
* Посмотрю, чему равно на этом этапе значение переменной Numeral, введя: print Numeral На экран было выведено число 5 (рис. 16).

Рис. 16: Посмотрю, чему равно на этом этапе значение переменной Numeral.

Рис. 16: Посмотрю, чему равно на этом этапе значение переменной Numeral.

* Сравню с результатом вывода на экран после использования команды: display Numeral (рис. 17).

Рис. 17: Сравню с результатом вывода на экран после использования команды: display Numeral.

Рис. 17: Сравню с результатом вывода на экран после использования команды: display Numeral.

* Уберу точки останова (рис. 18): info breakpoints delete 1

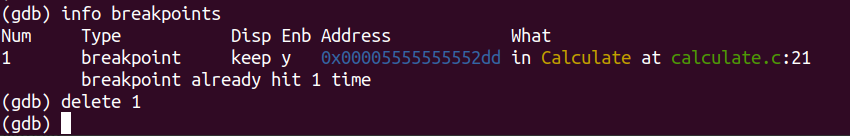


Рис. 18: Уберу точки останова.

С помощью утилиты splint проанализирую коды файлов calculate.c и main.c (рис. 19, рис. 20).

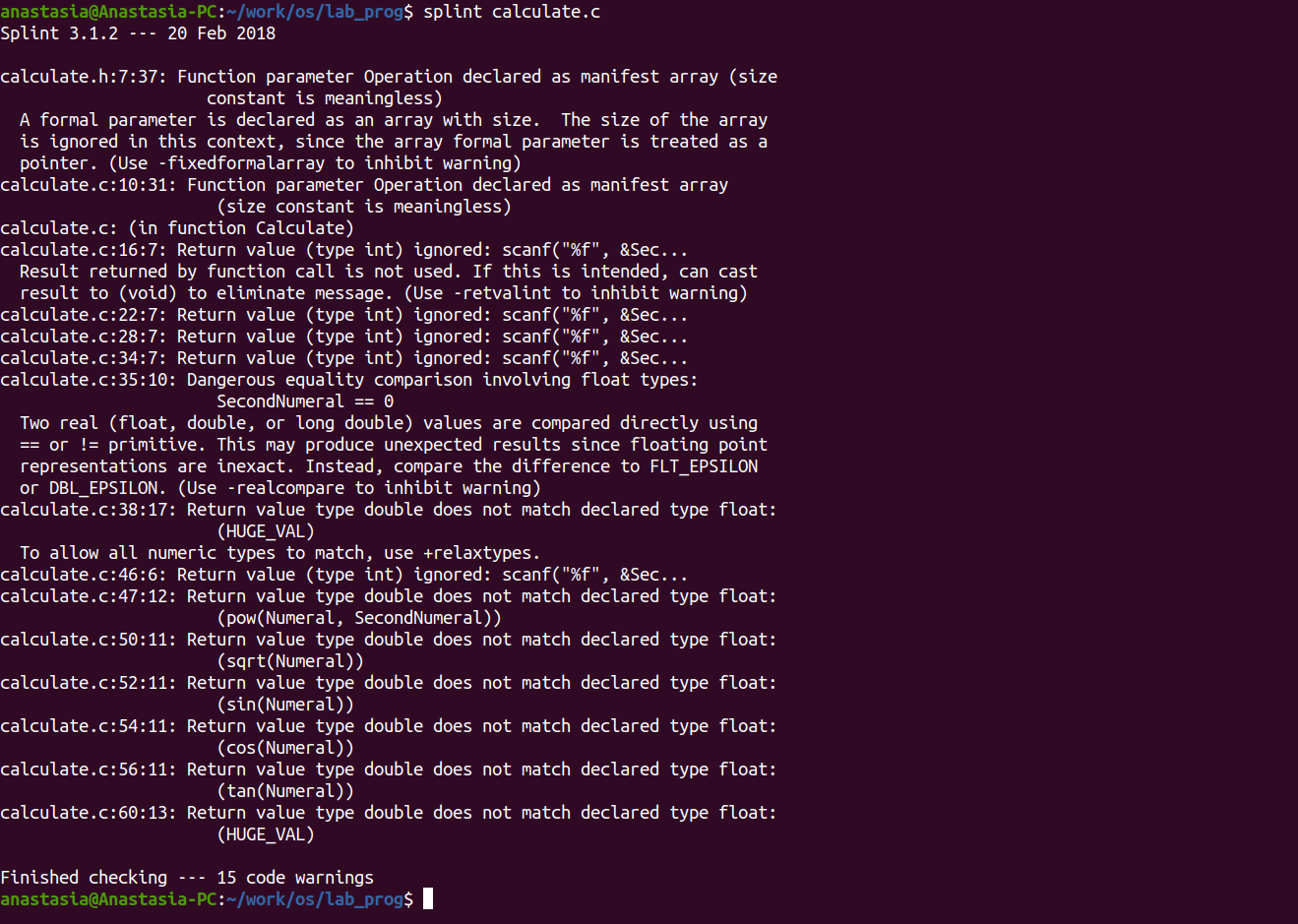


Рис. 19: С помощью утилиты splint проанализирую код файла calculate.c.

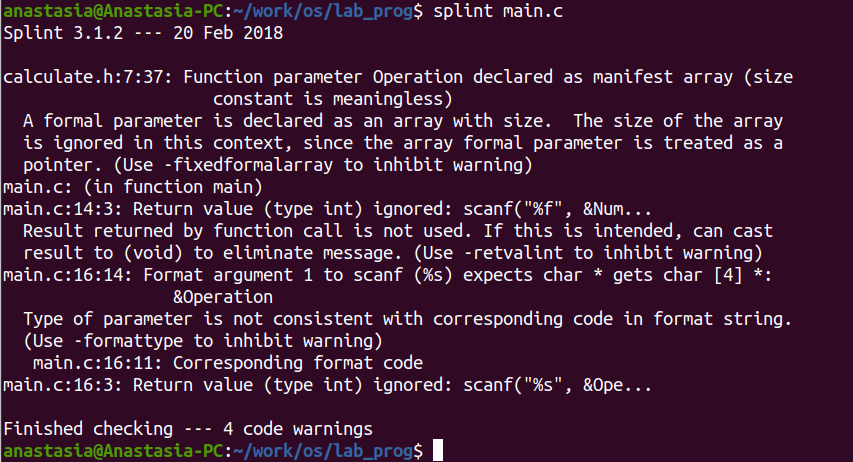


Рис. 20: С помощью утилиты splint проанализирую код файла main.c.

# 4 Вывод

В ходе 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.

# 5 Ответы на контрольные вопросы

1. Как получить информацию о возможностях программ gcc, make, gdb и др.? Ответ: Информацию об этих программах можно получить с помощью функций info и man.
2. Назовите и дайте краткую характеристику основным этапам разработки приложений в UNIX. Ответ: 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. Что такое суффикс в контексте языка программирования? Приведите примеры использования. Ответ: 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.
4. Каково основное назначение компилятора языка С в UNIX? Ответ: Основное назначение компилятора с языка Си заключается в компиляции всей программы в целом и получении исполняемого модуля.
5. Для чего предназначена утилита make? Ответ: 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. Приведите пример структуры Makefile. Дайте характеристику основным элементам этого файла. Ответ: 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. Пример можно найти в задании 5.
7. Назовите основное свойство, присущее всем программам отладки. Что необходимо сделать, чтобы его можно было использовать? Ответ: 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. Назовите и дайте основную характеристику основным командам отладчика gdb. Ответ: 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
9. Опишите по шагам схему отладки программы, которую Вы использовали при выполнении лабораторной работы. Ответ: 1) Выполнила компиляцию программы 2)Увидела ошибки в программе 3) Открыла редактор и исправила программу 4) Загрузила программу в отладчик gdb 5) run — отладчик выполнил программу, ввела требуемые значения. 6) Использовала другие команды отладчика и проверила работу программы
10. Прокомментируйте реакцию компилятора на синтаксические ошибки в программе при его первом запуске. Ответ: Отладчику не понравился формат %s для &Operation, т.к %s — символьный формат, а значит необходим только Operation.
11. Назовите основные средства, повышающие понимание исходного кода программы. Ответ: 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.
12. Каковы основные задачи, решаемые программой splint? Ответ: 1.Проверка корректности задания аргументов всех использованных в программе функций, а также типов возвращаемых ими значений; 2.Поиск фрагментов исходного текста, корректных с точки зрения синтаксиса языка Си, но малоэффективных с точки зрения их реализации или содержащих в себе семантические ошибки; 3.Общая оценка мобильности пользовательской программы.