Отчёта по лабораторной работе №12:

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

Кононов Алексей Сергеевич

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

1. Провести сборку программы простейшего калькулятора
2. Произвести отладку программы используя gdb
3. Изучить предупреждения линтера splint

# 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

1. Создаем подкаталог ~/work/os/lab\_prog, и в нем 3 файла: calculate.h, calculate.c, main.c (рис. 1):

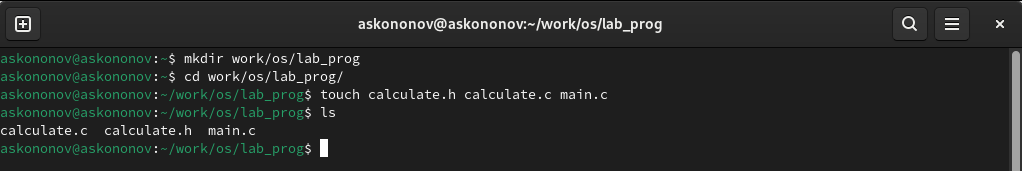


Рис. 1: Создание

1. Запишем в файлы тексты программ, которые даны в руководстве к лабораторной работе (рис. 2), (рис. 3), (рис. 4).



Рис. 2: calculate.c

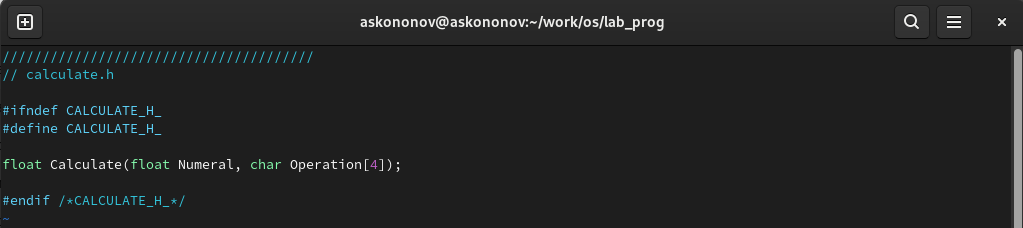


Рис. 3: calculate.h

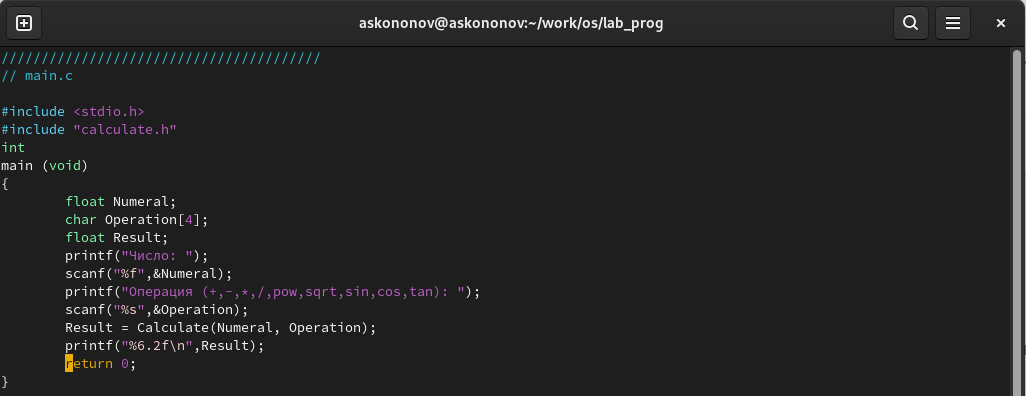


Рис. 4: main.c

1. Выполним компиляцию с помощью gcc (рис. 5):

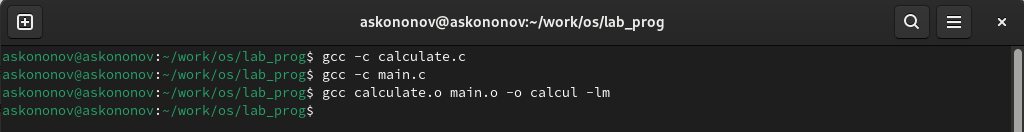


Рис. 5: компиляция

1. Создим Makefile (рис. 6):

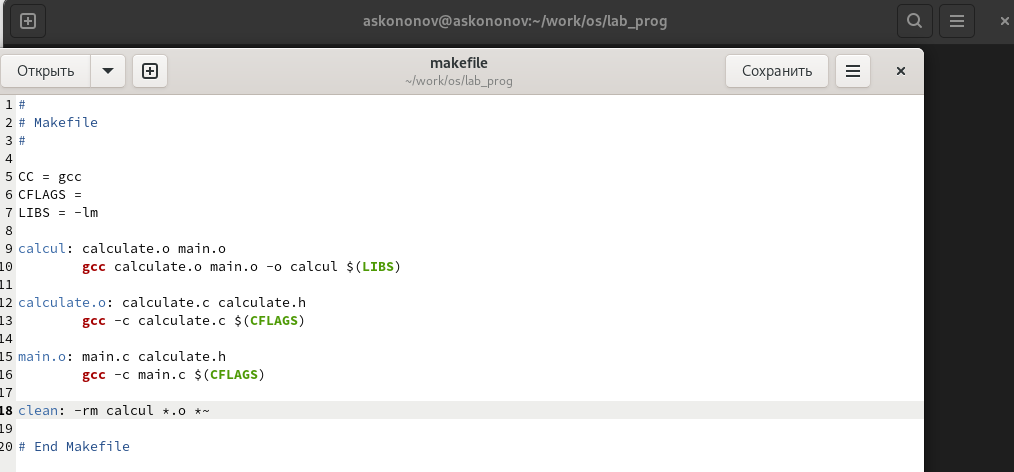


Рис. 6: Makefile

Этот Makefile используется для автоматизации процесса компиляции и сборки программы. Вот объяснение каждой его части:

* CC = gcc: Эта строка устанавливает переменную CC равной gcc, что означает, что для компиляции будет использоваться компилятор GNU C.
* CFLAGS = -g: Здесь устанавливаются флаги компиляции (CFLAGS) для gcc. Флаг -g добавляет отладочную информацию в исполняемые файлы, что полезно при отладке.
* LIBS = -lm: Это определяет переменную LIBS, которая содержит флаги для линкера. -lm указывает на необходимость подключения математической библиотеки.

Цели и правила в Makefile:

* calcul: Это цель для создания исполняемого файла calcul. Она зависит от объектных файлов calculate.o и main.o. Команда $(CC) calculate.o main.o -o calcul $(LIBS) компилирует эти объектные файлы вместе с библиотеками, указанными в LIBS, для создания исполняемого файла.
* calculate.o: Это правило говорит make, как создать файл calculate.o из calculate.c и calculate.h. Команда $(CC) -c calculate.c $(CFLAGS) компилирует исходный файл calculate.c в объектный файл, используя флаги из CFLAGS.
* main.o: Похоже на предыдущее правило, но для создания main.o из main.c и calculate.h.
* clean: Это специальная цель для очистки каталога от файлов, созданных во время сборки. Команда -rm calcul \*.o \*~ удаляет исполняемый файл calcul, все объектные файлы (.o) и временные файлы, созданные редакторами (файлы, заканчивающиеся на ~).

Знак минуса (-) перед командой rm говорит make игнорировать ошибки при удалении файлов (например, если файл уже был удалён).

1. С помощью gdb выполним отладку программы calcul (я привела код уже исправленного Makefile:

* Запустим отладчик GDB gdb ./calcul. Для запуска программы введем run и посчитаем некое выражение (рис. 7)

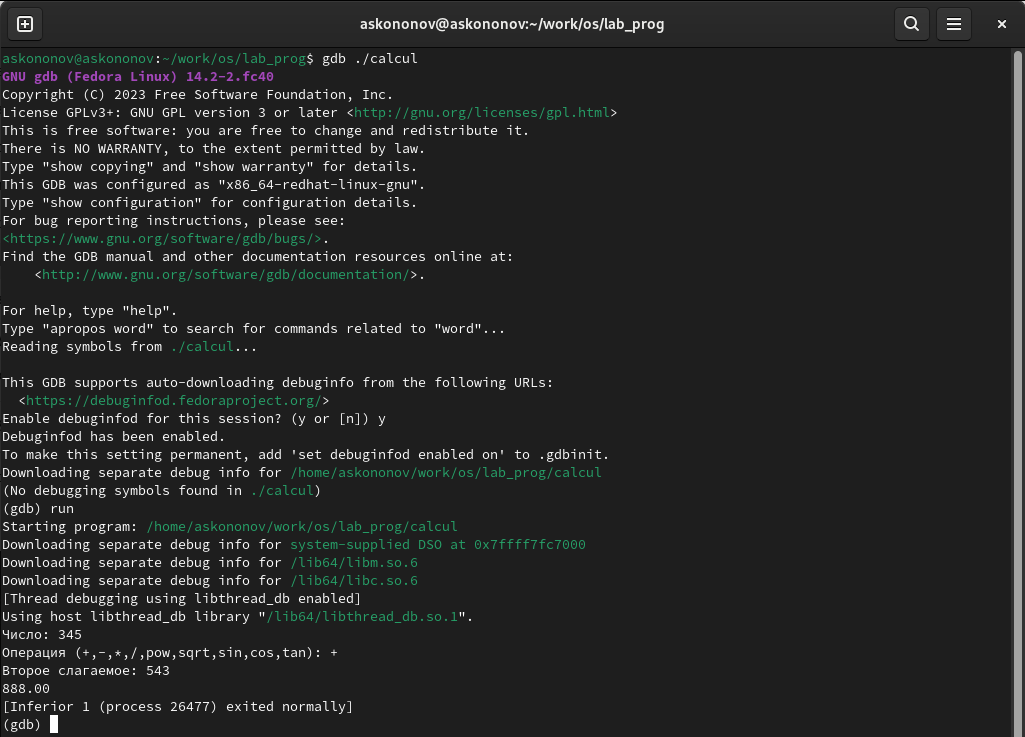


Рис. 7: Запуск отладчика

* Для постраничного (по 9 строк) просмотра исходного код используем команду list, затем для просмотра строк с 12 по 15 основного файла используем list 12,15, просмотрим определённых строк не основного файла, используя list calculate.c:20,29, а также установим точку останова в файле calculate.c на строке номер 21, использовав list calculate.c:20,27 и break 21 (рис. 8):



Рис. 8: Просмотр кода

Запустим программу внутри отладчика с помощью run и убедимся, что программа остановится в момент прохождения точки останова. С помощью команды backtrace покажим весь стек вызываемых функций от начала программы до текущего места. Посмотрим, чему равно на этом этапе значение переменной Numeral, введя print Numeral и сравним с результатом вывода на экран после использования команды, использовав display Numeral. Посмотрим, информацию про точку останова с помощью info breakpoints и удалим эту точку командой delete 1 (рис. 9):



Рис. 9: Проверка и удаление точки остановки

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

Разберем сначала файл main.c. Сначала выводится информация о том, что длина массива, указанная в сигнатуре функции calculate, не имеет никакого смысла и игнорируется. Далее несколько раз выводится информация о том что мы игнорируем возвращаемое значение функции scanf. Всего 3 предупреждения (рис. 10).

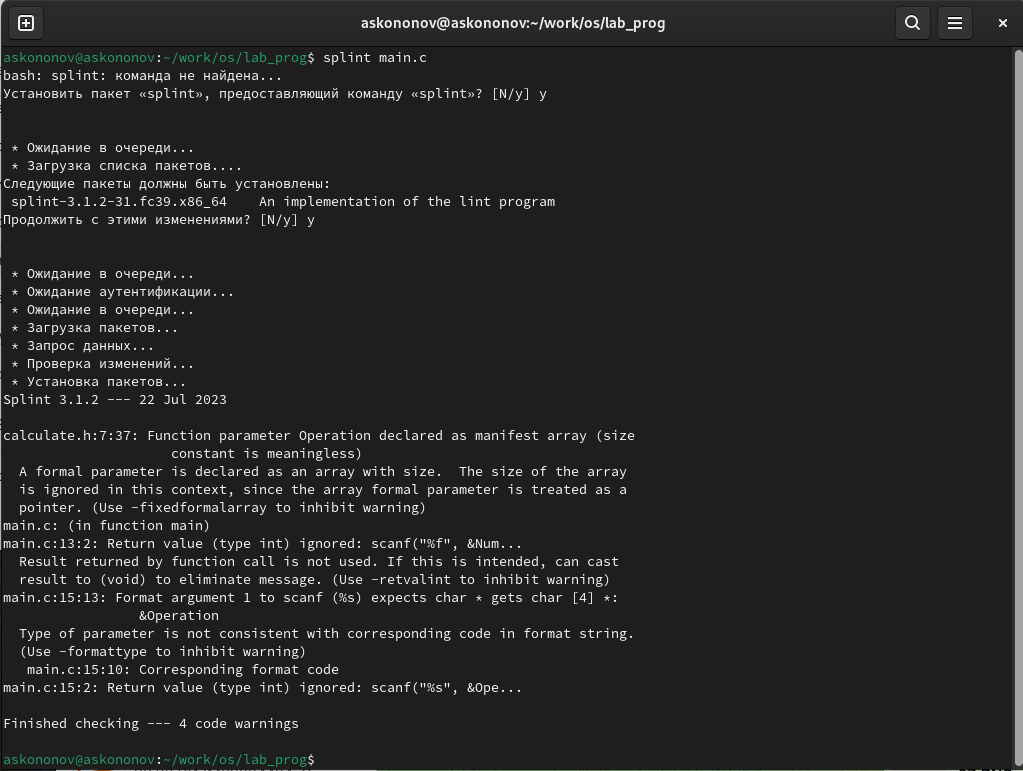


Рис. 10: Анализ main.c

Теперь разберем файл calculate.c. Те же предупреждения о игнорировании длинны массива и возвращаемого значения функции scanf, а также предупреждения о неявном преобразовании типа double в тип float. В сумме 15 предупреждений.

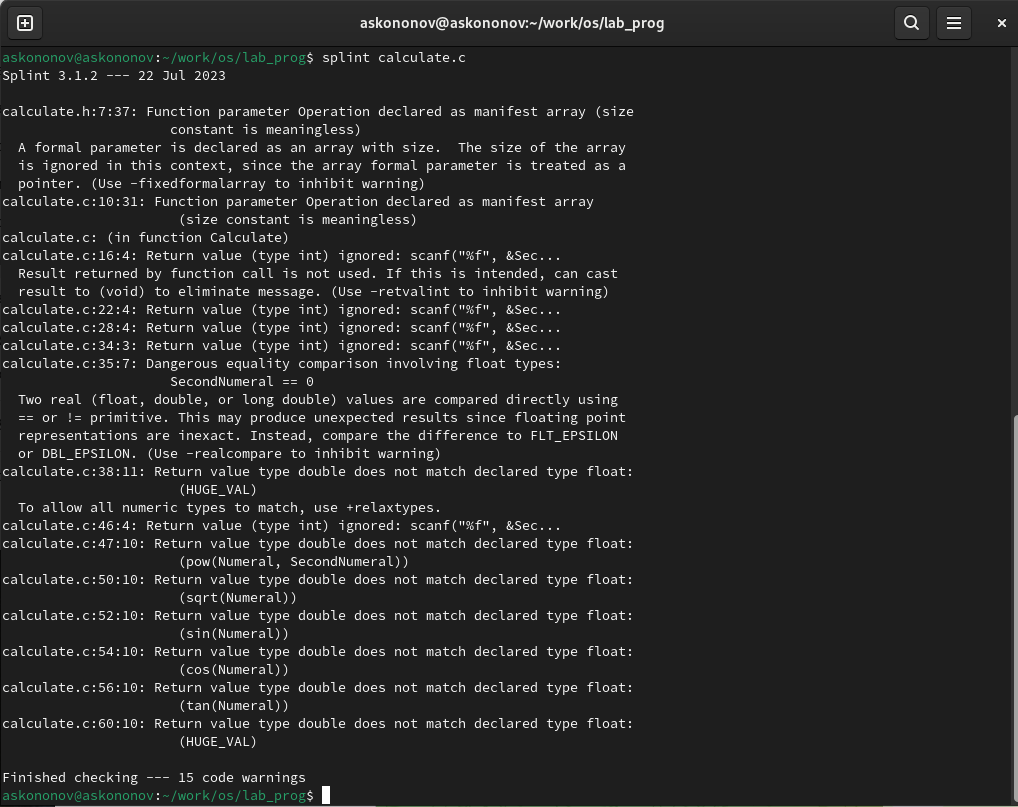


Рис. 11: Анализ calculate.c

# 5 Контрольные вопросы

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

Можно использовать название\_программы --help для общей помощи, man название\_программы для руководства пользователя или info название\_программы для более подробной информации.

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

* **Дизайн**: Определение требований и архитектуры системы.
* **Кодирование**: Написание исходного кода приложения.
* **Компиляция**: Преобразование исходного кода в исполняемый файл.
* **Тестирование**: Проверка функциональности и поиск ошибок.
* **Отладка**: Исправление обнаруженных ошибок.
* **Установка**: Размещение программы в системе для использования.
* **Сопровождение**: Обновление и улучшение программы.

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

Суффикс — это расширение файла, указывающее на тип содержимого. Например, .c для исходных файлов C, .h для заголовочных файлов C.

1. Каково основное назначение компилятора языка С в UNIX?

Компилятор C преобразует исходный код на языке C в машинный код, который может выполняться операционной системой UNIX.

1. Для чего предназначена утилита make?

make автоматизирует процесс компиляции и сборки программы, используя файл Makefile для определения зависимостей между файлами и правил сборки.

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

**Пример структуры Makefile**:

all: program  
  
 program: main.o lib.o  
 gcc -o program main.o lib.o  
  
 main.o: main.c  
 gcc -c main.c  
  
 lib.o: lib.c  
 gcc -c lib.c  
  
 clean:  
 rm -f \*.o program

**Элементы Makefile**:

* **Цели**: all, program, main.o, lib.o, clean.
* **Зависимости**: Файлы, от которых зависит цель.
* **Правила**: Команды для создания цели из зависимостей.
* **Псевдоцели**: Цели, не связанные с файлами, например clean.

1. Назовите основное свойство, присущее всем программам отладки. Что необходимо сделать, чтобы его можно было использовать?

Возможность остановить выполнение программы, просмотреть и изменить значения переменных. Для использования требуется скомпилировать программу с опцией отладки (например, gcc -g).

1. Назовите и дайте основную характеристику основным командам отладчика gdb.

**Основные команды gdb**:

* run: Запуск программы.
* break: Установка точки останова.
* next: Выполнение следующей строки кода.
* continue: Продолжение выполнения до следующей точки останова.
* print: Вывод значения переменной.
* quit: Выход из gdb.

1. Опишите по шагам схему отладки программы, которую Вы использовали при выполнении лабораторной работы.

* Компиляция с опцией -g.
* Запуск gdb.
* Установка точек останова.
* Запуск программы с помощью run.
* Просмотр и изменение переменных.
* Продолжение выполнения и наблюдение за поведением программы.

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

Компилятор выдаст сообщения об ошибках, указывая местоположение и возможную причину ошибки.

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

* Комментарии.
* Читаемые имена переменных и функций.
* Структурирование кода.
* Документация.

1. Каковы основные задачи, решаемые программой splint?

splint выполняет статический анализ кода на C для обнаружения ошибок программирования, уязвимостей безопасности и некачественного кода.

# 6 Выводы

В данной лабораторной работе мы приобрели простейшие навыки разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linux на примере создания на языке программирования С калькулятора с простейшими функциями.

# 7 Список литературы