Отчёт по лабораторной работе №13

Дисциплина: Операционные системы

Батова Ирина Сергеевна, НММбд-01-22

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. В домашнем каталоге создайте подкаталог ~/work/os/lab\_prog.
2. Создайте в нём файлы: calculate.h, calculate.c, main.c. В соответствие с лабораторной работы внести скрипт в файлы.

* Реализация функций калькулятора в файле calculate.h
* Интерфейсный файл calculate.h, описывающий формат вызова функции-калькулятора
* Основной файл main.c, реализующий интерфейс пользователя к калькулятору

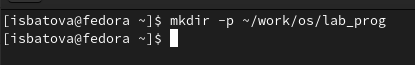
1. Выполните компиляцию программы посредством gcc.
2. При необходимости исправьте синтаксические ошибки.
3. Создайте Makefile
4. С помощью gdb выполните отладку программы calcul (перед использованием gdb исправьте Makefile):

* Запустите отладчик GDB, загрузив в него программу для отладки:
* Для запуска программы внутри отладчика введите команду run
* Для постраничного (по 9 строк) просмотра исходного код используйте команду list
* Для просмотра строк с 12 по 15 основного файла используйте list с параметрами
* Для просмотра определённых строк не основного файла используйте list с параметрами – Установите точку останова в файле calculate.c на строке номер 21 – Выведите информацию об имеющихся в проекте точках останова – Запустите программу внутри отладчика и убедитесь, что программа остановится в момент прохождения точки останова – Посмотрите, чему равно на этом этапе значение переменной Numeral – Сравните с результатом вывода на экран после использования другой команды – Уберите точки останова

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

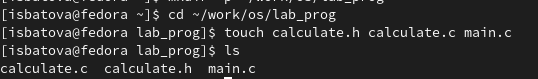
# 3 Выполнение лабораторной работы

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



Создание подкаталога

1. Создаем в нём файлы: calculate.h, calculate.c, main.c (рис. ??).



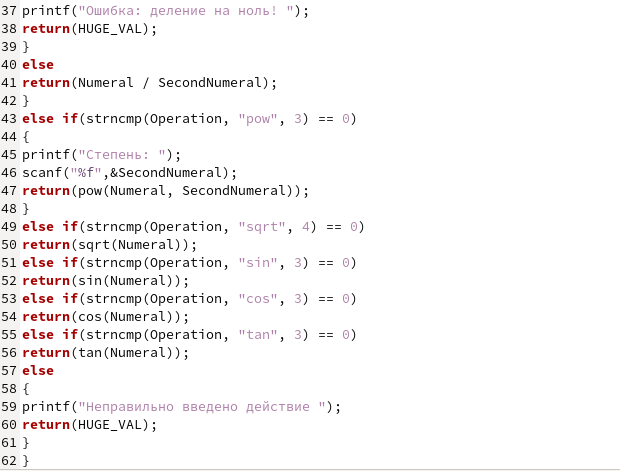
Создание файлов

Далее вносим в файлы скрипты соответственно лабораторной работе для создания примитивного калькуляторы.

Вводим скрипт для реализации функций калькулятора в файле calculate.с (рис. ??, ??).

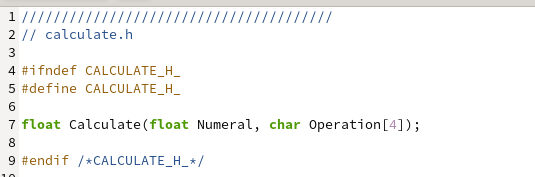


Скрипт в calculate.c



Скрипт в calculate.c

Вводим скрипт в интерфейсный файл calculate.h, описывающий формат вызова функции-калькулятора (рис. ??).



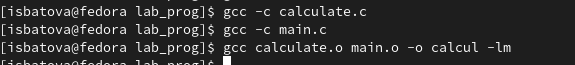
Скрипт в calculate.h

Вводим скрипт в основной файл main.c, реализующий интерфейс пользователя к калькулятору (рис. ??).



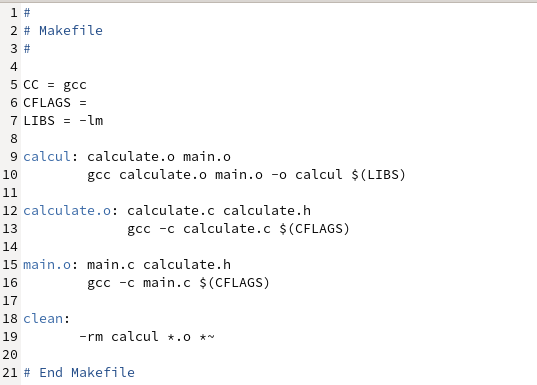
Скрипт в main.c

1. Выполните компиляцию программы посредством gcc (рис. ??).



Компиляция программы

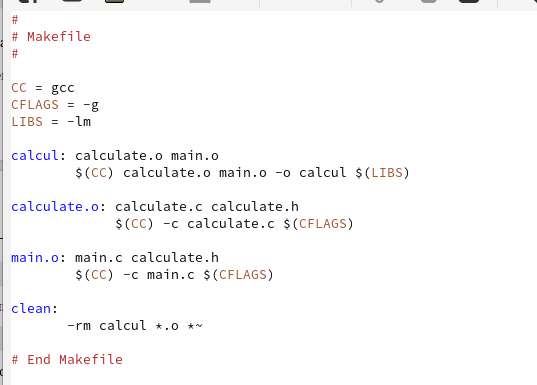
1. Синтаксические ошибки не обнаружены.
2. Создаем файл с именем “makefile” и вводим в него скрипт соответственно лабораторной работе (рис. ??).



Создание makefile

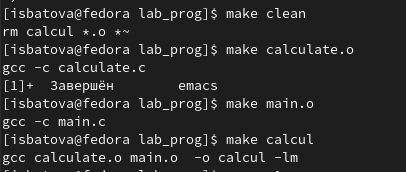
Файл используется для автоматической компиляции main.c, calculate.c и создание из них исполняемого файла calcul. Помимо этого, в файле также есть функция ‘clean’, используемая для удаления всех файлов. В начале скрипта также вводятся переменные: СС - команда для компиляции gcc, СFLAGS - опции к команде компиляции, LIBS - опции при создании исполняемого файла.

1. Немного редактируем makefile - заменяем в тексте скрипта команду gcc на соответствующую ей переменную CC, а также придаем переменной СFLAGS значение опции ‘-g’, чтобы объектные файлы можно было использовать в отладчике.



Редактирование makefile

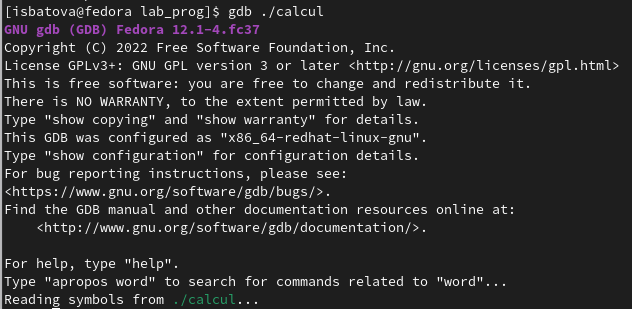
Проверяем работу makefile - удаляем файлы, затем вновь их компилируем (рис. ??).



Проверка makefile

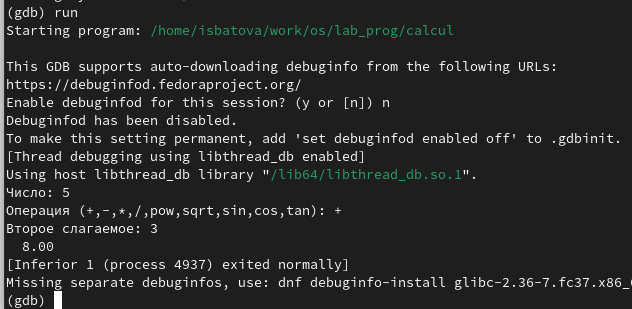
Далее с помощью gdb выполняем отладку программы calcul.

Запускаем отладчик GDB, загрузив в него программу для отладки (рис. ??).



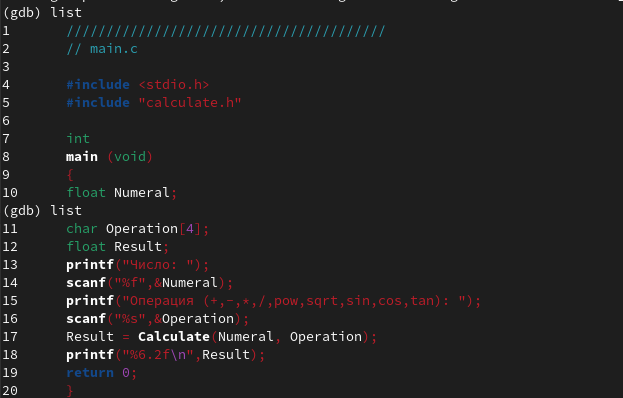
Запуск отладчика

Для запуска программы внутри отладчика вводим команду run (рис. ??).



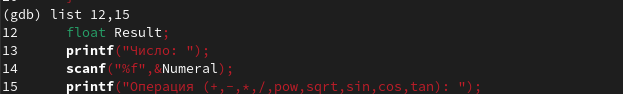
Запуск программы внутри отладчика

Для постраничного просмотра исходного код вводим команду list (рис. ??).



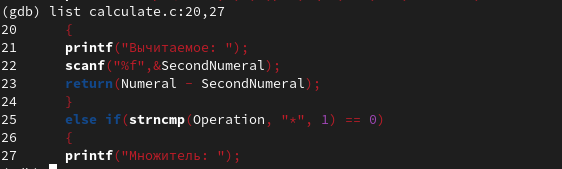
Постраничный просмотр исходного кода

Для просмотра строк с 12 по 15 основного файла вводим команду list с параметрами - номерами строк (рис. ??).



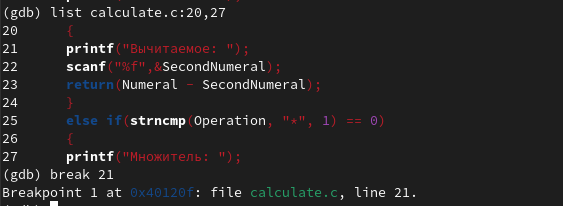
Просмотр определенных строк основного файла

Для просмотра определённых строк не основного файла вводим команду list с параметрами - номерами строк и указанием файла (рис. ??).



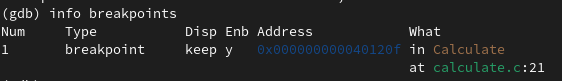
Просмотр определенных строк не основного файла

Устанавливаем точку останова в файле calculate.c на строке номер 21 командой break (рис. ??).



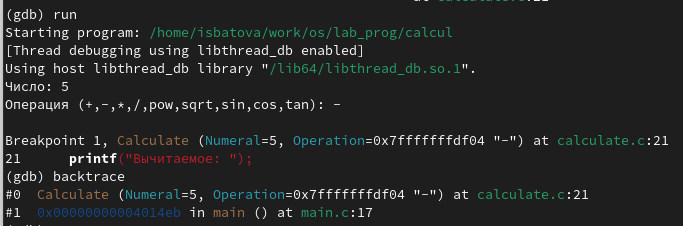
Установка точки останова

Выводим информацию об имеющихся в проекте точках останова командой info breakpoints (рис. ??).



Информация о точках останова

Чтобы убедиться, что программа остановится в момент прохождения точки останова, запускаем программу внутри отладчика (рис. ??).



Запуск программы с точкой останова

Проверяем, чему равно на этом этапе значение переменной Numeral командой print (рис. ??).

Проверка значения Numeral командой print

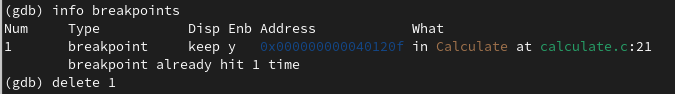
Проверка значения Numeral командой print

Проверяем, чему равно на этом этапе значение переменной Numeral командой display (рис. ??).

Проверка значения Numeral командой display

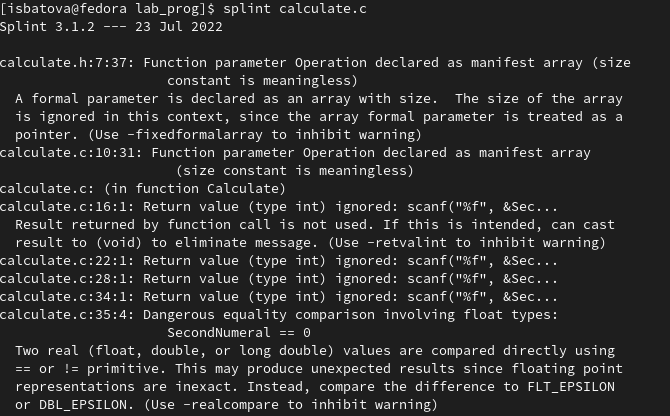
Проверка значения Numeral командой display

Убираем точки останова командой delete (рис. ??).

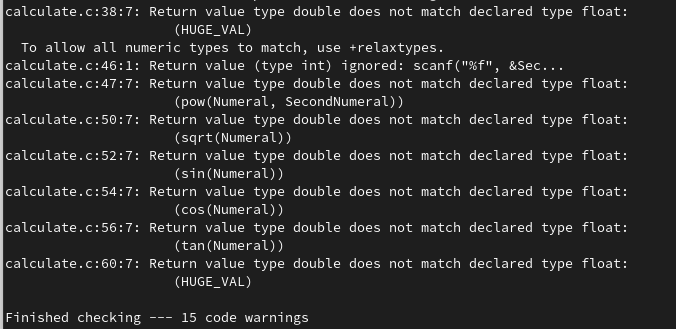


Удаление точек останова

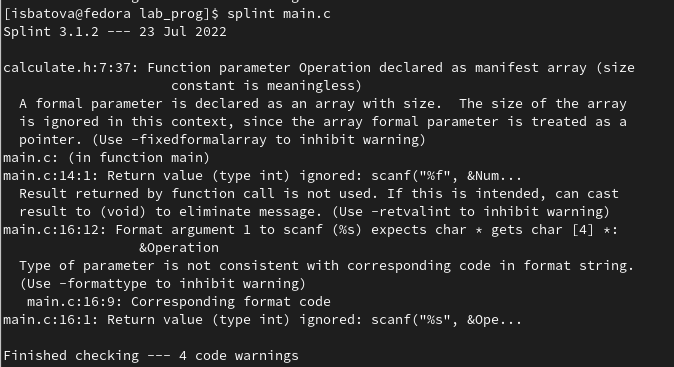
1. Вводим команды ‘splint calculate.c’ (рис. ??, ??) и ‘splint main.c’ (рис. ??) для анализа кодов файлов.



Команда ‘splint calculate.c’



Команда ‘splint calculate.c’



Команда ‘splint main.c’

С помощью данной команды мы узнали, что значения типа double в функциях pow, sin, cos, tan, sqrt записываются в переменную float, а значит, есть потеря данных. Также в обоих файлах есть функция scanf, которая возвращает целое значение, нигде не сохраняющиеся и не использующееся дальше в скрипте.

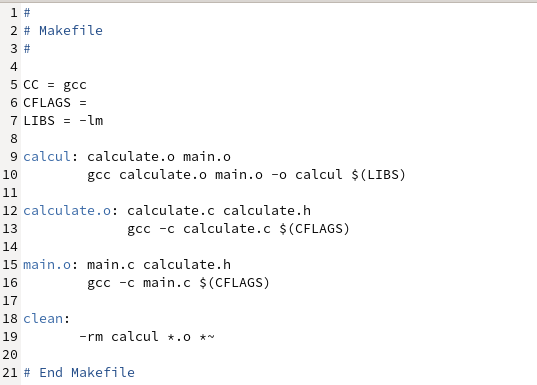
# 4 Выводы

В ходе данной лабораторной работы мной были приобретены 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 Контрольные вопросы

1. Для получения информации о возможностях программ gcc, make, gdb и других можно воспользоваться командой ‘man’.
2. Основные этапы разработки приложений в UNIX:

* Планирование: сбор и анализ требований к функционалу и другим характеристикам разрабатываемого приложения
* Проектирование: разработка базовых алгоритмов и спецификаций, определение языка программирования
* Кодирование: создание исходного текста программы
* Анализ разработанного кода
* Сборка, компиляция, разработка исполняемого модуля
* Тестирование и отладка, сохранение изменений
* Документирование

1. Суффикс определяет какая компиляция требуется для имени входного файла и указывают на тип объекта. Например, в команде ‘gcc -c calculate.c’ по суффиксу .с распознается тип файла как файл на языке Си и формируется объектный файл с суффиксом .o.
2. Компилятор языка Си в UNIX используется для компиляции всей программы и получения исполняемого файла.
3. Утилита make предназначена для автоматизирования процесса преобразования файлов программы из одного формата в другой.
4. 

* Пример файла makefile

Файл используется для автоматической компиляции main.c, calculate.c и создание из них исполняемого файла calcul. Помимо этого, в файле также есть функция ‘clean’, используемая для удаления всех файлов. В начале скрипта также вводятся переменные: СС - команда для компиляции gcc, СFLAGS - опции к команде компиляции, LIBS - опции при создании исполняемого файла.

1. Для того, чтобы можно было использовать программы отладки, необходимо скомпилировать анализируемый код программы так, чтобы отладочная информация содержалась в результирующим бинарном файле (реализуется опцией -g компилятора gcc).
2. Основные команды отладчика gdb:

* backtrace - вывод на экран пути к текущей точке останова
* break - установить точку останова
* 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
* 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 остановле

1. Схема отладки программы пошагово описана в шестом пункте лабораторной работы.
2. Синтаксических ошибок в программе при первом запуске обнаружено не было.
3. Основные средства, повышающие понимание исходного кода программы - cscope (исследование функций программы) и lint (проверка программ языка Си)
4. Программа splint анализирует программный код и выполняет проверку корректности всех аргументов, функций, значений, синтаксиса программы. Помимо этого, программа выдает комментарии с разбором кода программы.