Лабораторная работа №13

Операционные системы

Серёгина Ирина Андреевна

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.
3. Выполните компиляцию программы посредством gcc.
4. При необходимости исправьте синтаксические ошибки.
5. Создайте Makefile.
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

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. Во время работы над кодом программы программист неизбежно сталкивается с по- явлением ошибок в ней. Использование отладчика для поиска и устранения ошибок в программе существенно облегчает жизнь программиста. В комплект программ GNU для ОС типа UNIX входит отладчик GDB (GNU Debugger). Ещё одним средством проверки исходных кодов программ, написанных на языке C, является утилита splint. Эта утилита анализирует программный код, проверяет коррект- ность задания аргументов использованных в программе функций и типов возвращаемых значений, обнаруживает синтаксические и семантические ошибки. В отличие от компилятора C анализатор splint генерирует комментарии с описанием разбора кода программы и осуществляет общий контроль, обнаруживая такие ошибки, как одинаковые объекты, определённые в разных файлах, или объекты, чьи значения не используются в работе программы, переменные с некорректно заданными значениями и типами и многое другое.

# 4 Выполнение лабораторной работы

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

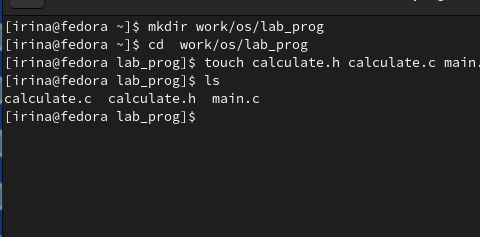


Рис. 1: создаю подкаталог

1. Создаю в нём файлы: calculate.h, calculate.c, main.c. (рис. fig. 2).

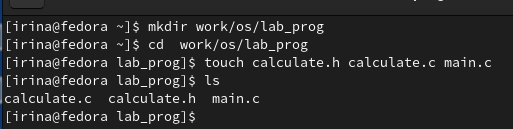


Рис. 2: создаю нужные файлы

1. Реализация функций калькулятора в файле calculate.h (рис. fig. 3).



Рис. 3: реализация функций

1. Интерфейсный файл calculate.h, описывающий формат вызова функции калькулятора (рис. fig. 4).

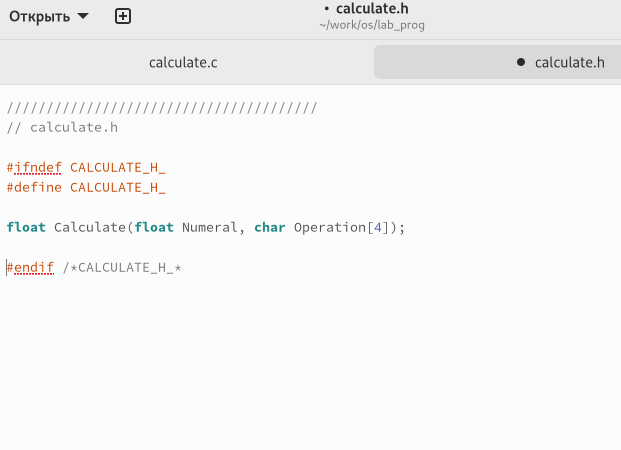


Рис. 4: интерфейсный файл

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

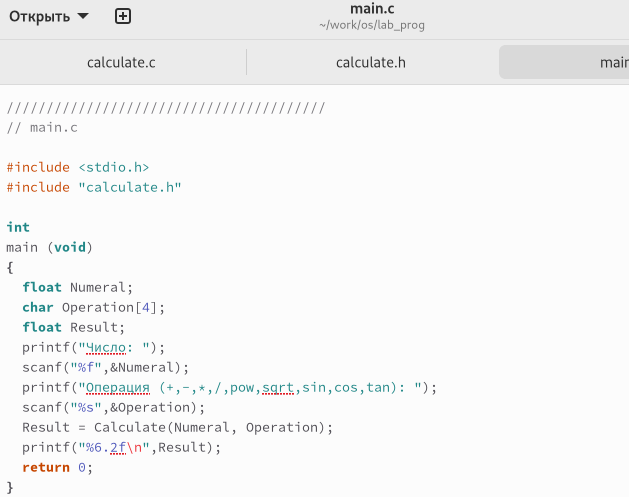


Рис. 5: основной файл

1. Выполняю компиляцию программы посредством gcc (рис. fig. 6).

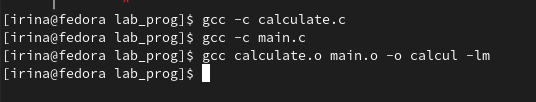


Рис. 6: компиляция программы

1. Создаю Makefile, исправляю ошибки (рис. fig. 7).



Рис. 7: Создаю Makefile

1. Выполняю команду make (рис. fig. 8).

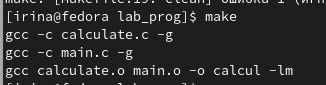


Рис. 8: команда make

1. Запускаю отладчик GDB, загрузив в него программу для отладки (рис. fig. 9).

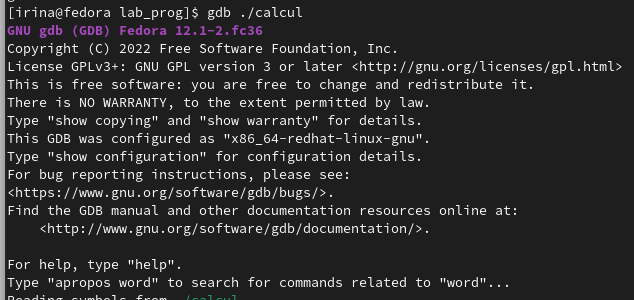


Рис. 9: Запускаю отладчик GDB

1. Для запуска программы внутри отладчика ввожу команду run (рис. fig. 10).

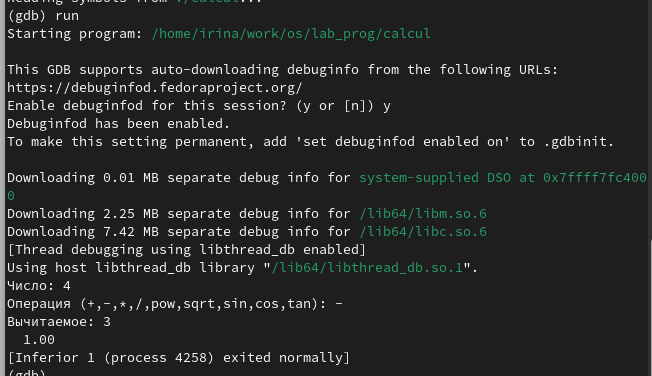


Рис. 10: команда run

1. Для постраничного (по 9 строк) просмотра исходного код использую команду list (рис. fig. 11).

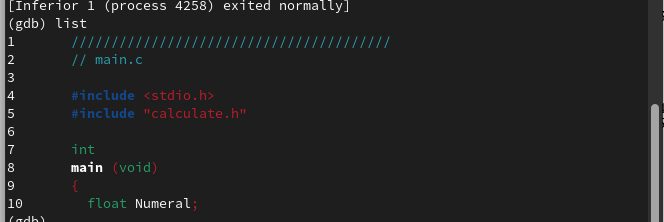


Рис. 11: использую команду list

1. Для просмотра строк с 12 по 15 основного файла использую команду list(рис. fig. 12).

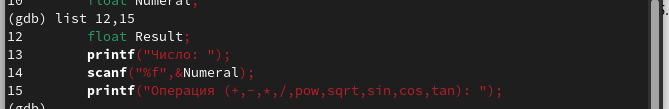


Рис. 12: использую команду list с параметрами

1. Для просмотра определённых строк не основного файла использую list с параметрами (рис. fig. 13).

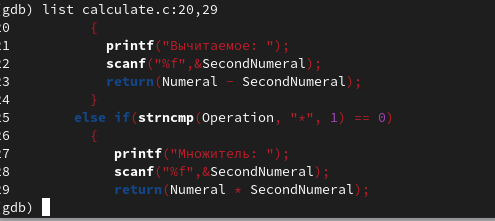


Рис. 13: использую list с параметрами

1. Устанавливаю точку останова в файле calculate.c на строке номер 21 (рис. fig. 14).

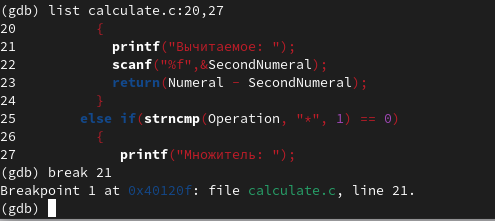


Рис. 14: Устанавливаю точку останова

1. Вывожу информацию об имеющихся в проекте точках останова (рис. fig. 15).

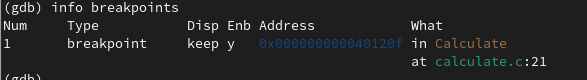


Рис. 15: вывожу информацию точках останова

1. Запускаю программу внутри отладчика и убеждаюсь, что программа остановится в момент прохождения точки останова (рис. fig. 16).

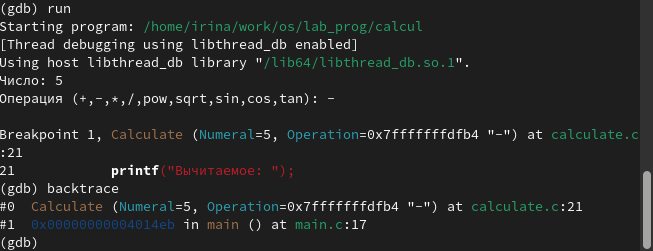


Рис. 16: запускаю программу

1. Смотрю, чему равно на этом этапе значение переменной Numeral, сравниваю с результатом вывода на экран после использования другой команды (рис. fig. 17).

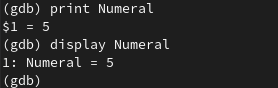


Рис. 17: сравниваю значения переменной

1. Убираю точки останова (рис. fig. 18).

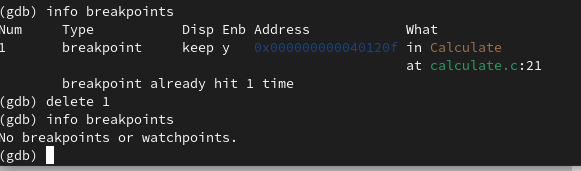


Рис. 18: Убираю точки останова

1. С помощью утилиты splint анализирую коды файла calculate.c (рис. fig. 19).

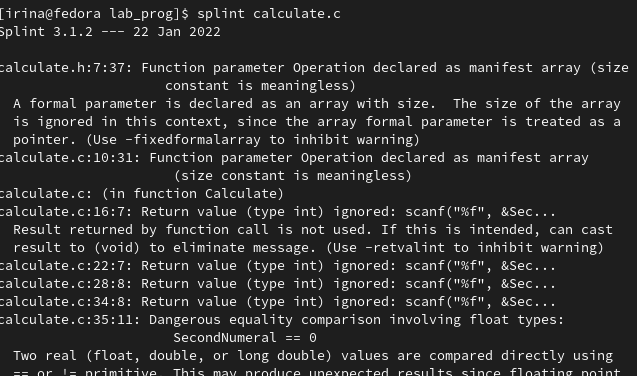


Рис. 19: анализирую коды файла calculate.c

1. С помощью утилиты splint попробуйте анализирую коды файла main.c. (рис. fig. 20).

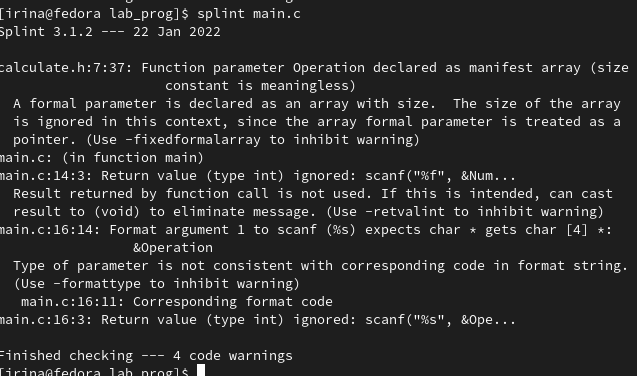


Рис. 20: анализирую коды файла main.c.

# 5 Выводы

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.

# Список литературы