

# **Отчет по лабораторной работе №13**

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

Абдулгалимов Мурад

# **Содержание**

<b>1 Цель работы</b>	<b>5</b>
<b>2 Задание</b>	<b>6</b>
<b>3 Выполнение лабораторной работы</b>	<b>8</b>
<b>4 Выводы</b>	<b>11</b>
<b>5 Контрольные вопросы</b>	<b>12</b>
<b>Список литературы</b>	<b>16</b>

# Список иллюстраций

3.1 Создание каталогов . . . . .	8
3.2 Создание файла . . . . .	8
3.3 Компиляция . . . . .	9
3.4 Компиляция . . . . .	9
3.5 Отладка . . . . .	10
3.6 Установка утилиты . . . . .	10

# **Список таблиц**

# **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. 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:

```
gcc -c calculate.c gcc -c main.c gcc calculate.o main.o -o calcul -lm
```

4. При необходимости исправьте синтаксические ошибки.
5. Создайте Makefile.
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 точках 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=0x7fffffff280 “-”) at calculate.c:21 #1 0x000000000400b2b 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

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

# 3 Выполнение лабораторной работы

В домашнем каталоге создал подкаталог `~/work/os/lab_prog` (рис. 3.1)

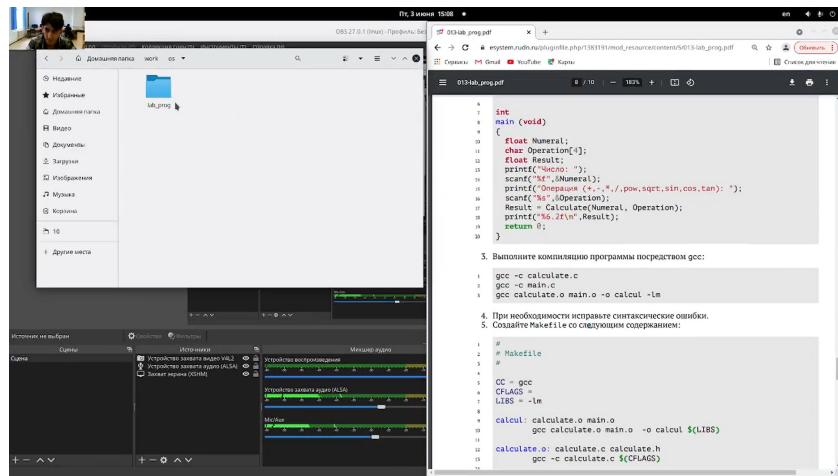


Рис. 3.1: Создание каталогов

Создал в нём файлы: `calculate.h`, `calculate.c`, `main.c` (рис. 3.2)

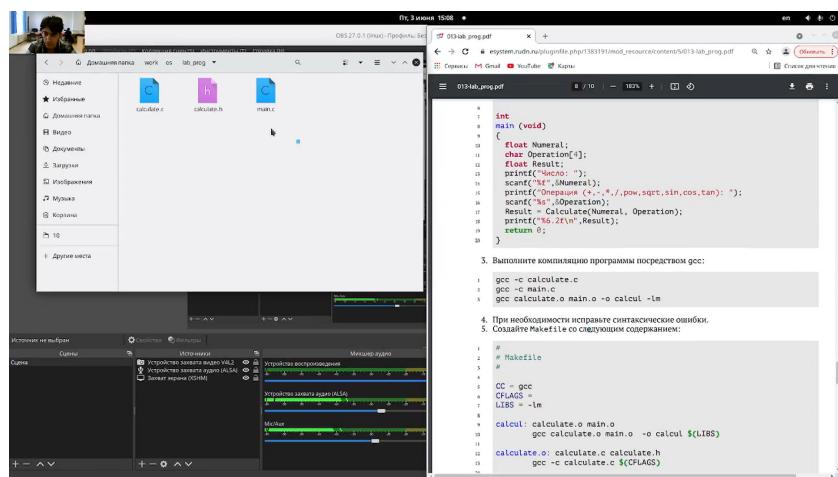


Рис. 3.2: Создание файла

Выполнил компиляцию программы посредством gcc: gcc -c calculate.c gcc -c main.c gcc calculate.o main.o -o calcul -lm (рис. 3.3)

```

03Lab_prog.pdf
1. Введите в терминал команду:
2. Исправьте ошибку в Makefile:
3. Выполните компиляцию программы посредством gcc:
4. Для необходимости исправьте синтаксические ошибки.
5. Создайте Makefile со следующим содержанием:
# Makefile
# Compiler
CC = gcc
CFLAGS =
LIBS = -lm

# Targets
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

```

Рис. 3.3: Компиляция

Создал Makefile (рис. 3.4)

```

03Lab_Prog.mk
1. gcc -c main.c
2. gcc calculate.o main.o -o calcul -lm
# Makefile
# Compiler
CC = gcc
CFLAGS =
LIBS = -lm

# Targets
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

```

Рис. 3.4: Компиляция

С помощью gdb выполнил отладку программы calcul (перед использованием gdb исправьте Makefile): Запустил отладчик GDB, загрузив в него программу для отладки: gdb ./calcul Для постраничного (по 9 строк) просмотра исходного кода использовал команду list Сравнил с результатом вывода на экран после использования команды: display Numeral – Убрал точки останова: info breakpoints delete 1 (рис. 3.5)

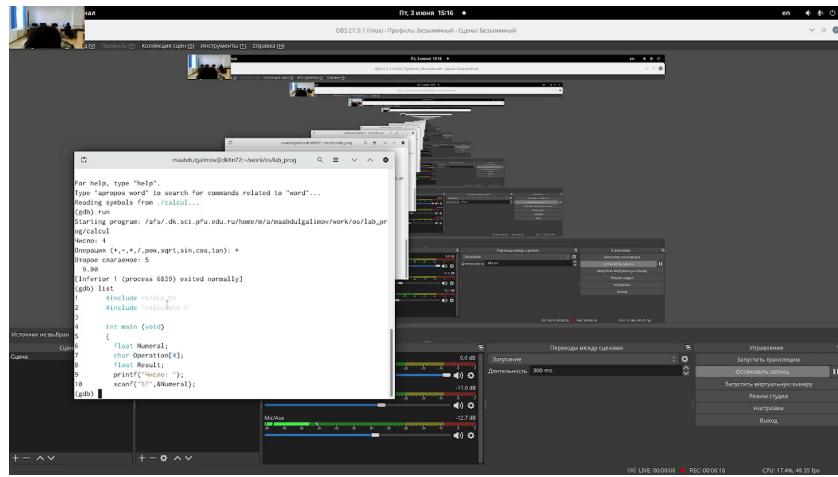


Рис. 3.5: Отладка

Установил утилиту splint. (рис. 3.6)

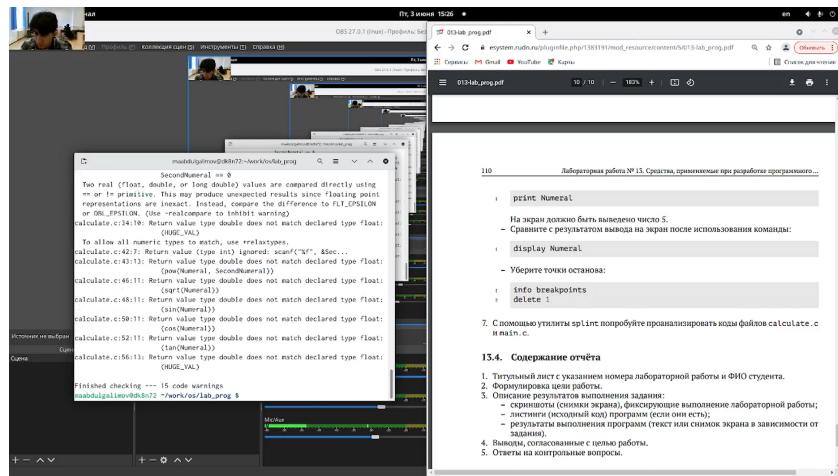


Рис. 3.6: Установка утилиты

## **4 Выводы**

Приобрел 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. С помощью функций `info` и `man`.
2. Этапы разработки приложений в UNIX: создание исходного кода программы, которая 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 “.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 .с компилятор распознает, что файл `abcd.c` должен компилироваться, а по суффиксу .о, 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 предпочтуют 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. Компиляция всей программы в целом и получении исполняемого модуля.
5. 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.
6. 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.

7. 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. Основные команды отладчика gdb: 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; list – 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. Схема отладки программы, которую я использовала при выполнении лабораторной работы: Выполнили компиляцию программы Увидели ошибки в программе Открыли редактор и исправили программу Загрузили программу в отладчик gdb run — отладчик выполнил программу, мы ввели требуемые значения. Программа завершена, gdb не видит ошибок.
10. У меня не было синтаксических ошибок в программе.
11. Средства, повышающие понимание исходного кода программы: cscope - исследование функций, содержащихся в программе; splint — критическая проверка программ, написанных на языке Си.
12. Основные задачи, решаемые программой splint: Проверка корректности задания аргументов всех использованных в программе функций, а также типов возвращаемых ими значений; Поиск фрагментов исходного текста, корректных с точки зрения синтаксиса языка Си, но малоэффективных с точки зрения их реализации или содержащих в себе семантические ошибки; Общая оценка мобильности пользовательской программы.

# **Список литературы**