Лабораторная работа №14

Операционные системы

Саттарова Вита Викторовна

Содержание

List of Figures

1 Цели и задачи

1.1 Цель

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.

1.2 Задачи

1. Изучить теорию относительно разработки, анализа, тестирования и отладки приложений в ОС UNIX/Linux.
2. Написать свою программу калькулятор.
3. Рассмотреть на практике возможности анализа, тестирования и отладки приложений в ОС UNIX/Linux.

2 Объект и предмет исследования

2.1 Объект исследования

Разработка, анализ, тестирование и отладка приложений в ОС UNIX/Linux.

2.2 Предмет исследования

Изучение основной информации, связанной с разработкой, анализом, тестированием и отладкой приложений, реализация калькулятора, простейшие анализ, тестирование и отладка созданной программы.

3 Условные обозначения и термины

Условные обозначения и термины отсутствуют

4 Теоретические вводные данные

4.1 Этапы разработки приложений

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.

4.2 Компиляция исходного текста и построение исполняемого файла

Стандартным средством для компиляции программ в ОС типа UNIX является GCC (GNU Compiler Collection). Это набор компиляторов для разного рода языков программирования (C, 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 C, файлы с расширением .cc или .C — как файлы на языке C++, а файлы с расширением .o считаются объектными. Для компиляции файла main.c, содержащего написанную на языке C простейшую программу достаточно в командной строке ввести: gcc -c main.c. Таким образом, gcc по расширению (суффиксу) .c распознает тип файла для компиляции и формирует объектный модуль — файл с расширением .o. Если требуется получить исполняемый файл с определённым именем (например, hello), то требуется воспользоваться опцией -o и в качестве параметра задать имя создаваемого файла: gcc -o hello main.c.

Для сборки разрабатываемого приложения и собственно компиляции полезно воспользоваться утилитой make. Она позволяет автоматизировать процесс преобразования файлов программы из одной формы в другую, отслеживает взаимосвязи между файлами. Для работы с утилитой make необходимо в корне рабочего каталога с Вашим проектом создать файл с названием makefile или Makefile, в котором будут описаны правила обработки файлов Вашего программного комплекса. Сначала задаётся список целей, разделённых пробелами, за которым идёт двоеточие и список зависимостей. Затем в следующих строках указываются команды. Строки с командами обязательно должны начинаться с табуляции. В качестве цели в Makefile может выступать имя файла или название какого-то действия. Зависимость задаёт исходные параметры (условия) для достижения указанной цели. Зависимость также может быть названием какого-то действия.

4.3 Тестирование и отладка

Во время работы над кодом программы программист неизбежно сталкивается с появлением 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). Для использования GDB необходимо скомпилировать анализируемый код программы таким образом, чтобы отладочная информация содержалась в результирующем бинарном файле. Для этого следует воспользоваться опцией -g компилятора gcc. После этого для начала работы с gdb необходимо в командной строке ввести одноимённую команду, указав в качестве аргумента анализируемый бинарный файл. Затем можно использовать по мере необходимости различные команды gdb.

4.4 Анализ исходного текста программы

Ещё одним средством проверки исходных кодов программ, написанных на языке C, является утилита splint. Эта утилита анализирует программный код, проверяет корректность 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.

5 Техническое оснащение и выбранные методы проведения работы

5.1 Техническое оснащение

Персональный компьютер, интернет, виртуальная машина.

5.2 Методы

Анализ предложенной информации, выполнение указанных заданий, получение дополнительной информации из интернета.

6 Выполнение лабораторной работы

1. В домашнем каталоге создала подкаталог ~/work/os/lab_prog. (рис. -fig. 6.1)

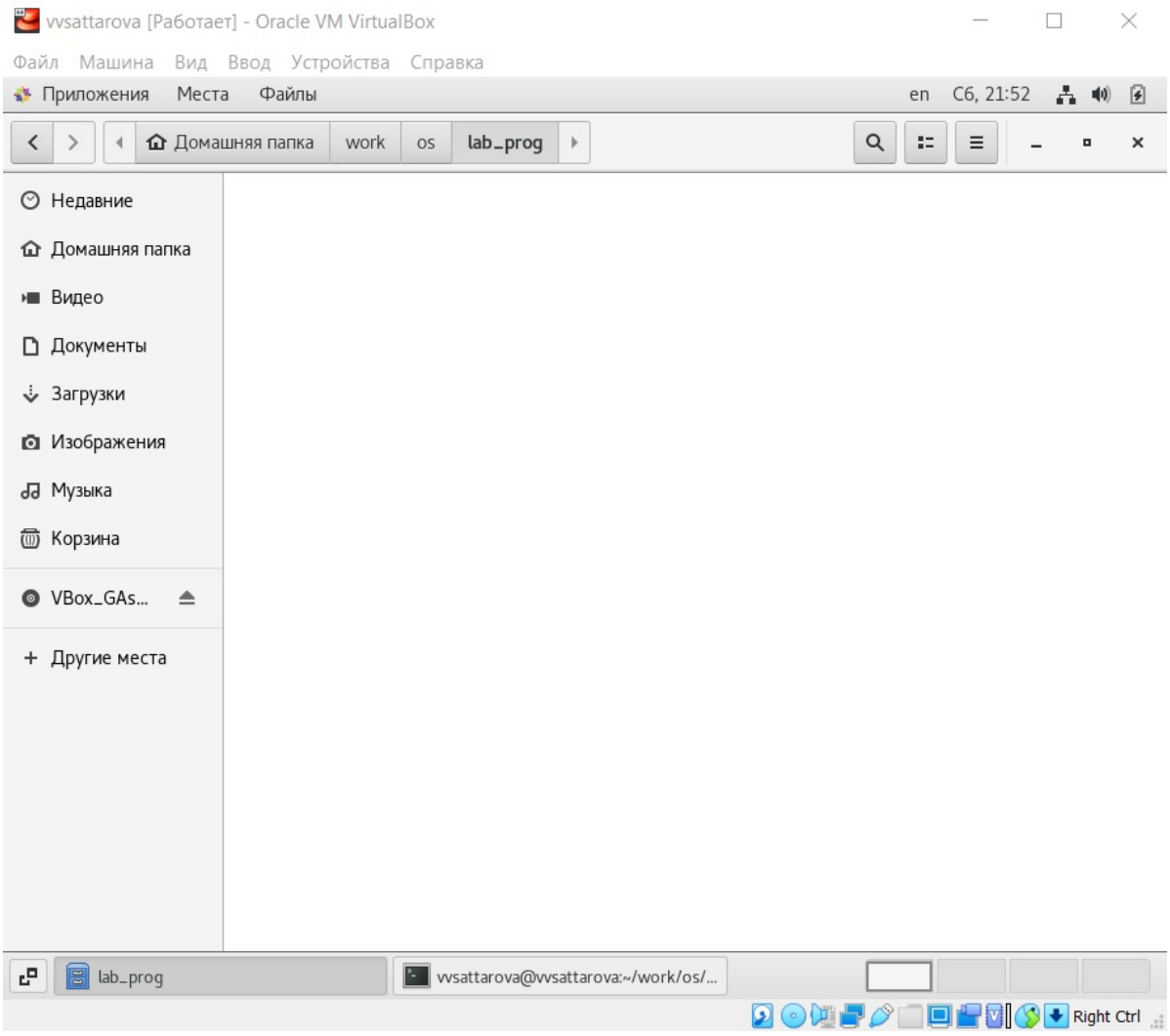

Figure 6.1: Рис. 1 Создание подкаталога

2. Создала в нём файлы: calculate.h, calculate.c, main.c для калькулятора. 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.

```
////////////////////////////////////
// calculate.c
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "calculate.h"

float Calculate(float Numeral, char Operation[4])
{
  float SecondNumeral;
  if(strncmp(Operation, "+", 1) == 0)
    {
      printf("Второе слагаемое: ");
      scanf("%f",&SecondNumeral);
      return(Numeral + SecondNumeral);
    }
  else if(strncmp(Operation, "-", 1) == 0)
    {
      printf("Вычитаемое: ");
      scanf("%f",&SecondNumeral);
      return(Numeral - SecondNumeral);
    }
  else if(strncmp(Operation, "*", 1) == 0)
```

```
  {
    printf("Множитель: ");
    scanf("%f",&SecondNumeral);
    return(Numeral * SecondNumeral);
  }
else if(strncmp(Operation, "/", 1) == 0)
  {
    printf("Делитель: ");
    scanf("%f",&SecondNumeral);
    if(SecondNumeral == 0)
      {
        printf("Ошибка: деление на ноль! ");
        return(HUGE_VAL);
      }
    else
      return(Numeral / SecondNumeral);
  }
else if(strncmp(Operation, "pow", 3) == 0)
  {
    printf("Степень: ");
    scanf("%f",&SecondNumeral);
    return(pow(Numeral, SecondNumeral));
  }
else if(strncmp(Operation, "sqrt", 4) == 0)
  return(sqrt(Numeral));
else if(strncmp(Operation, "sin", 3) == 0)76 Лабораторная работа № 11.
  return(sin(Numeral));
else if(strncmp(Operation, "cos", 3) == 0)
  return(cos(Numeral));
```

```
  else if(strncmp(Operation, "tan", 3) == 0)
    return(tan(Numeral));
  else
    {
      printf("Неправильно введено действие ");
      return(HUGE_VAL);
    }
}
////////////////////////////////////
// calculate.h
#ifndef CALCULATE_H_
#define CALCULATE_H_
float Calculate(float Numeral, char Operation[4]);
#endif /*CALCULATE_H_*/
////////////////////////////////////
// main.c
#include <stdio.h>
#include "calculate.h"
int
main (void)
{
float Numeral;
char Operation[4];
float Result;
printf("Число: ");
scanf("%f",&Numeral);
printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): ");
scanf("%s",&Operation);
Result = Calculate(Numeral, Operation);
```

```
printf("%6.2f\n",Result);
return 0;
}
```

(рис. -fig. 6.2)

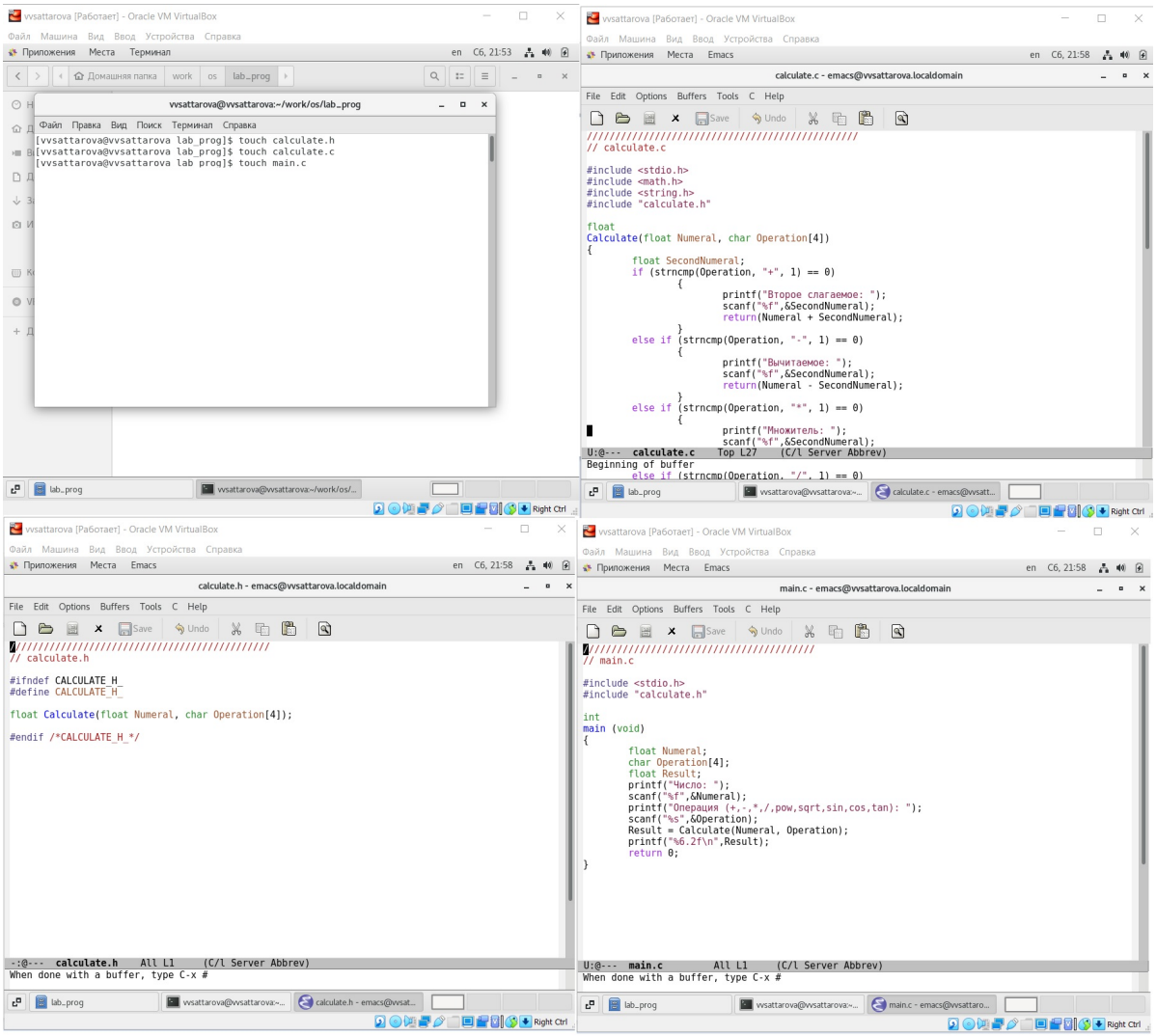

Figure 6.2: Рис. 2 Написание файлов калькулятора

3. Выполнила компиляцию программы посредством gcc:

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

(рис. -fig. 6.3)

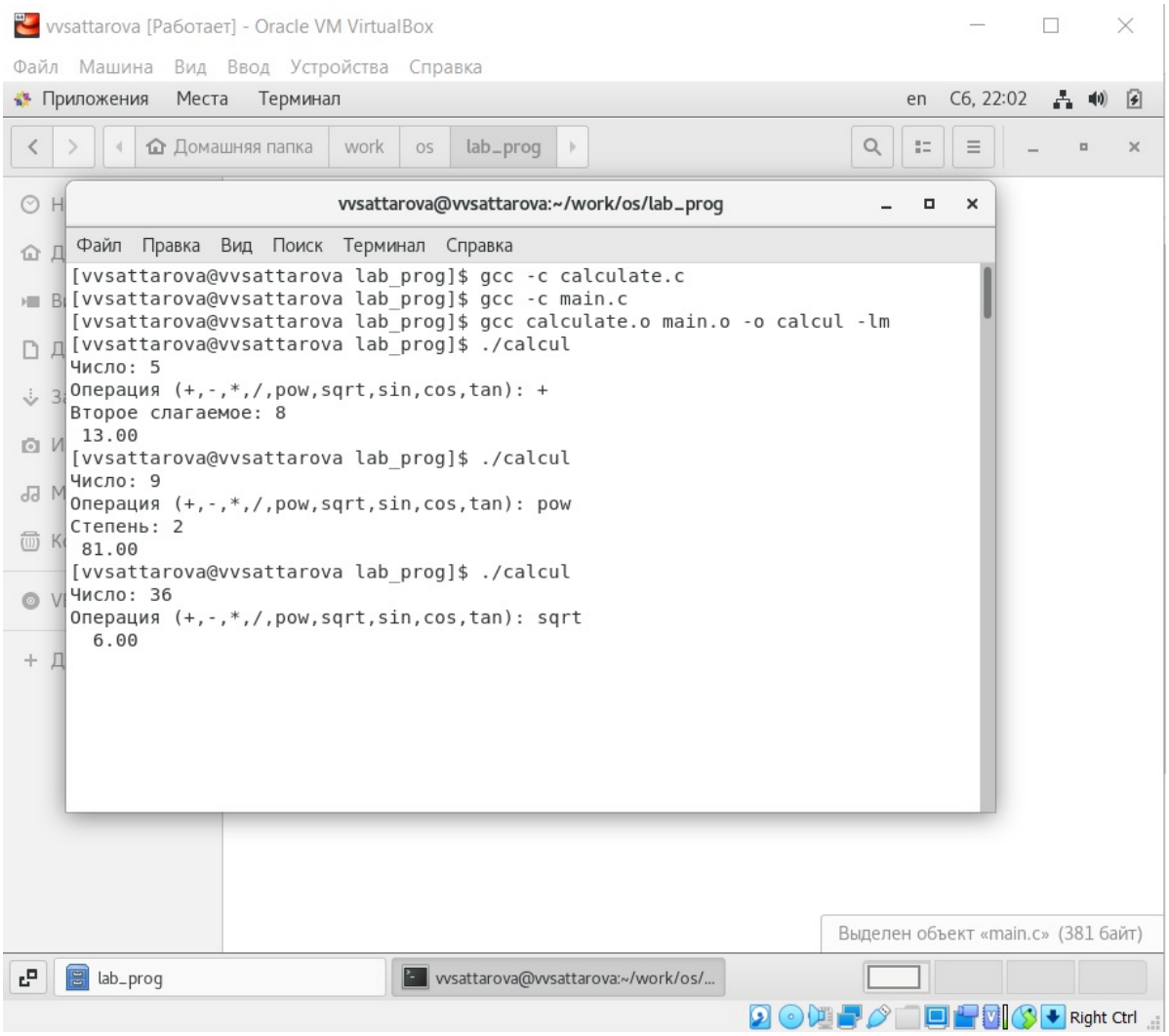

Figure 6.3: Рис. 3 Компиляция программы

4. У меня не возникло необходимости исправлять синтаксические ошибки.

5. Создала Makefile со следующим содержанием.

```
#
# Makefile
#
CC = gcc # Компилятор
CFLAGS = # ключи для C файлов
LIBS = -lm
```

```
calcul: calculate.o main.o              #  создание исполняемого файла из об
    gcc calculate.o main.o -o calcul $(LIBS)
calculate.o: calculate.c calculate.h # создание объектного файла калькуля
    gcc -c calculate.c $(CFLAGS)
main.o: main.c calculate.h              # создание основного объектного файл
    gcc -c main.c $(CFLAGS)
clean:
    -rm calcul *.o *~                   # удаление объектных файлов
# End Makefile
```

(рис. -fig. 6.4)

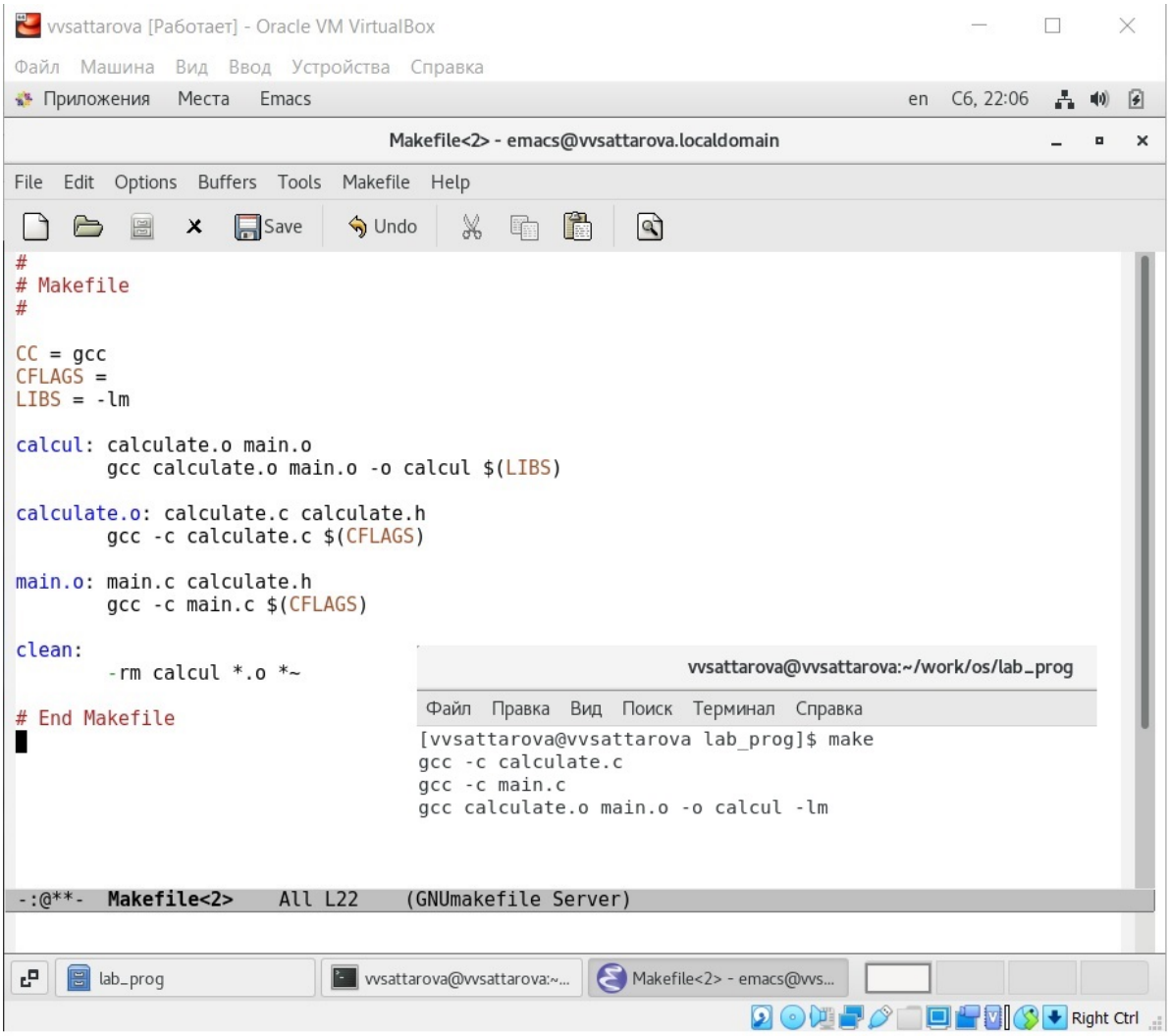

Figure 6.4: Рис. 4 Makefile

6. С помощью gdb выполнила отладку программы calcul (перед использованием gdb исправила Makefile, добавив ключ для отладки файлов).

- Запустила отладчик GDB, загрузив в него программу для отладки: gdb ./calcul
- Для запуска программы внутри отладчика ввела команду run: run. (рис. -fig. 6.5)

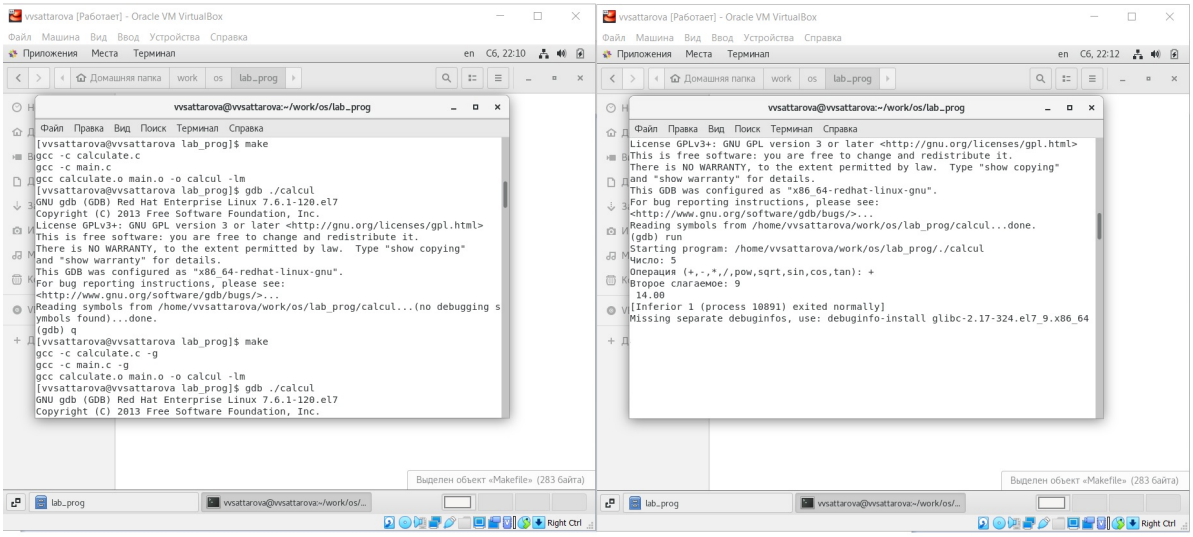

Figure 6.5: Рис. 5 Запуск отладчика с калькулятором

7. Для постраничного (по 9 строк) просмотра исходного код использовала команду list. (рис. -fig. 6.6)

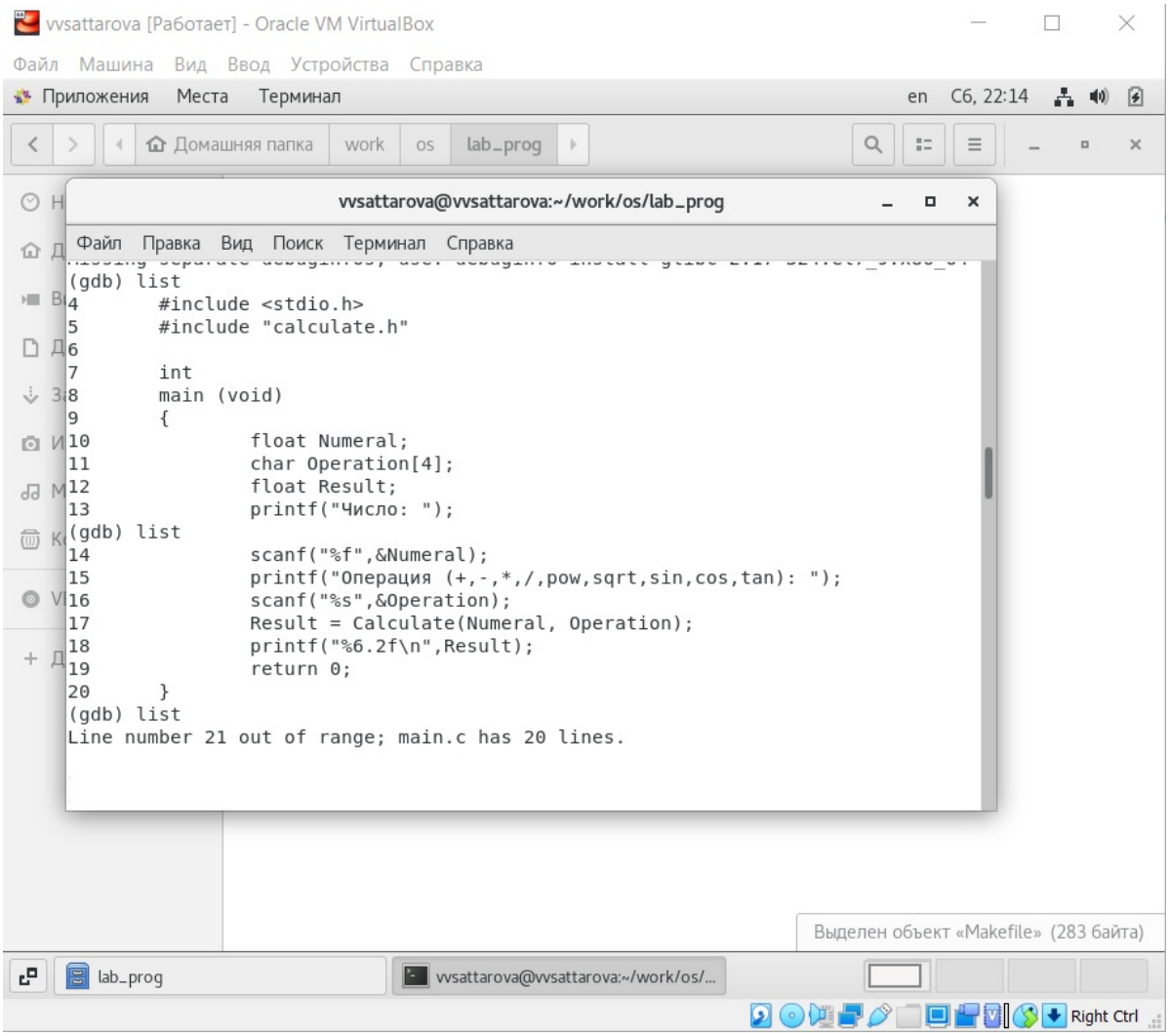

Figure 6.6: Рис. 6 Постраничный просмотр

8. Для просмотра строк с 12 по 15 основного файла использовала list с параметрами: list 12,15. (рис. -fig. 6.7)

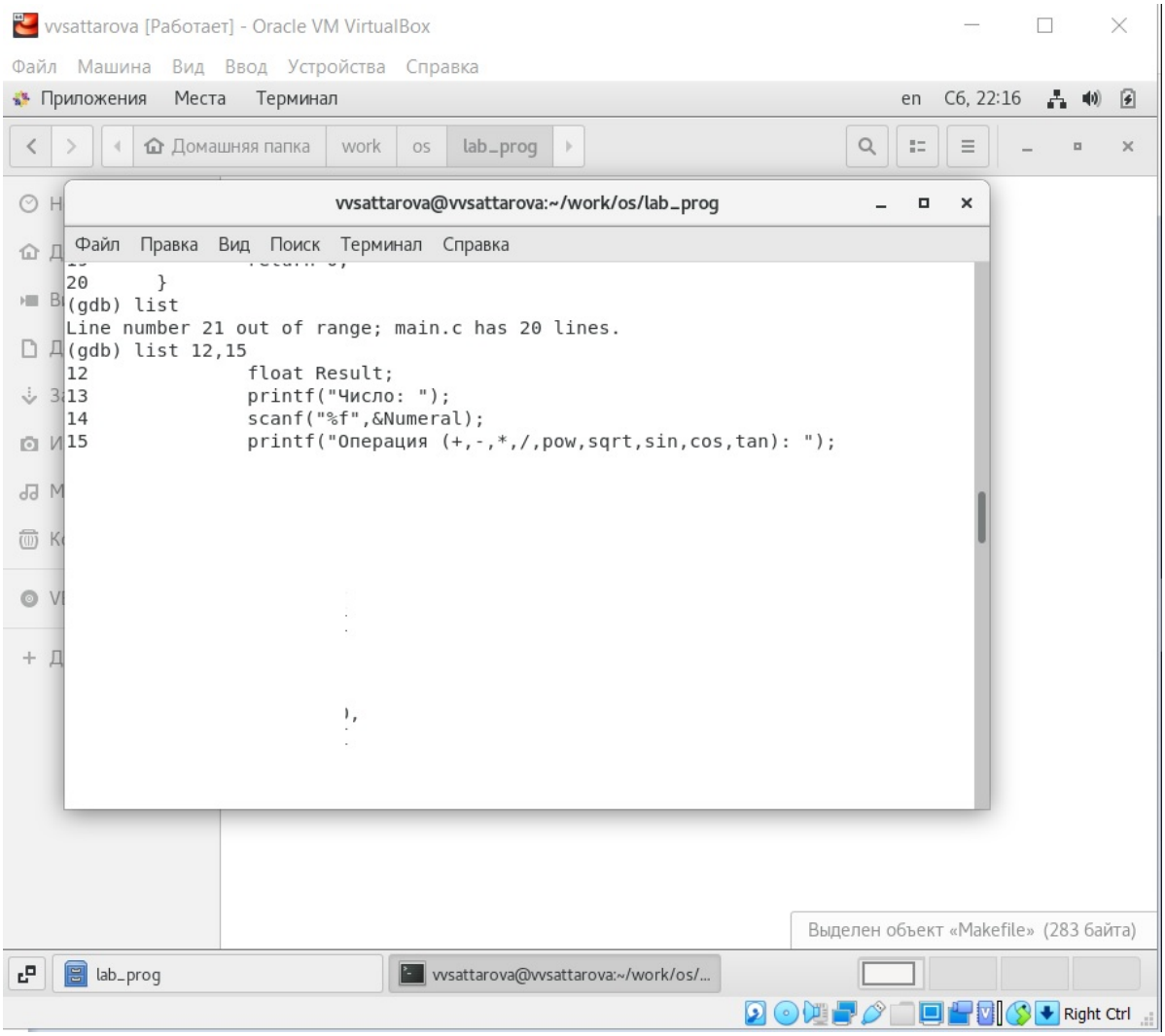

Figure 6.7: Рис. 7 Просмотр отпределённых строк

9. Для просмотра определённых строк не основного файла использовала list с параметрами: list calculate.c:20,29. (рис. -fig. 6.8)

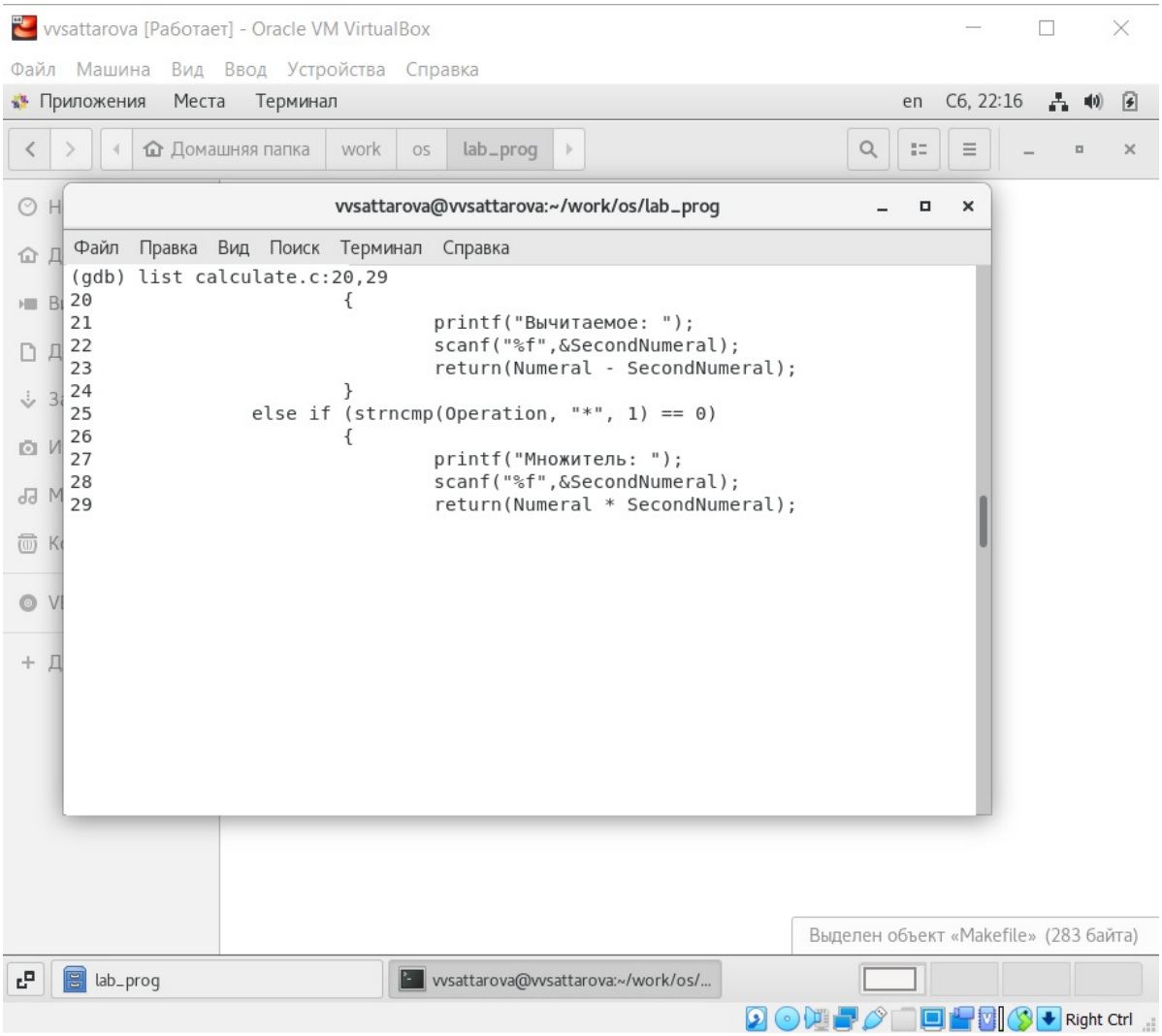

Figure 6.8: Рис. 8 Просмотр строк не основного файла

10. Установила точку останова в файле calculate.c на строке номер 21: list calculate.c:20,27; break 21. Вывела информацию об имеющихся в проекте точка останова: info breakpoints. (рис. -fig. 6.9)

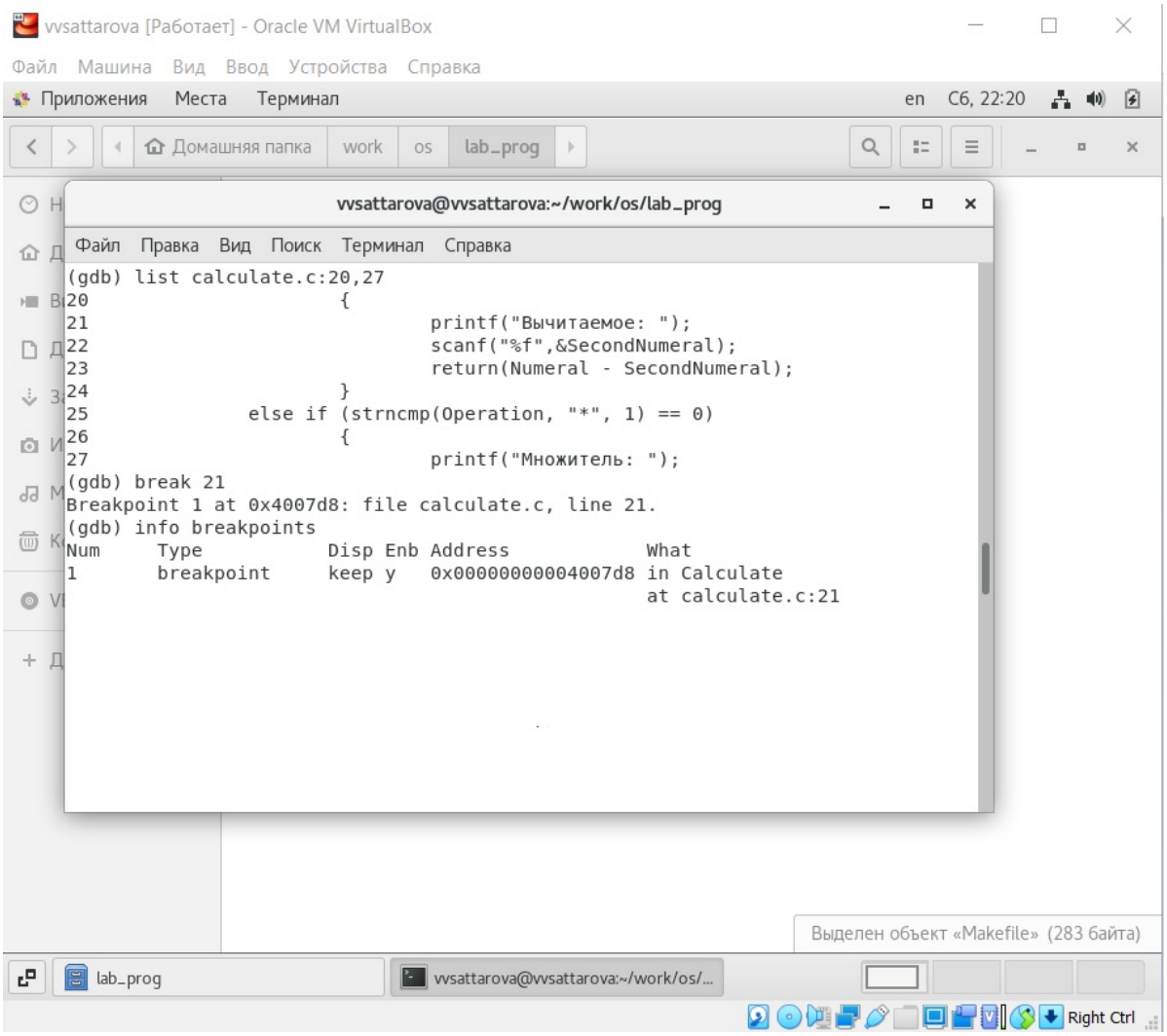

Figure 6.9: Рис. 9 Создание точек останова и получение информации

11. Запустила программу внутри отладчика и убедилась, что программа остановится в момент прохождения точки останова: run, 5- backtrace. (рис. - fig. 6.10)

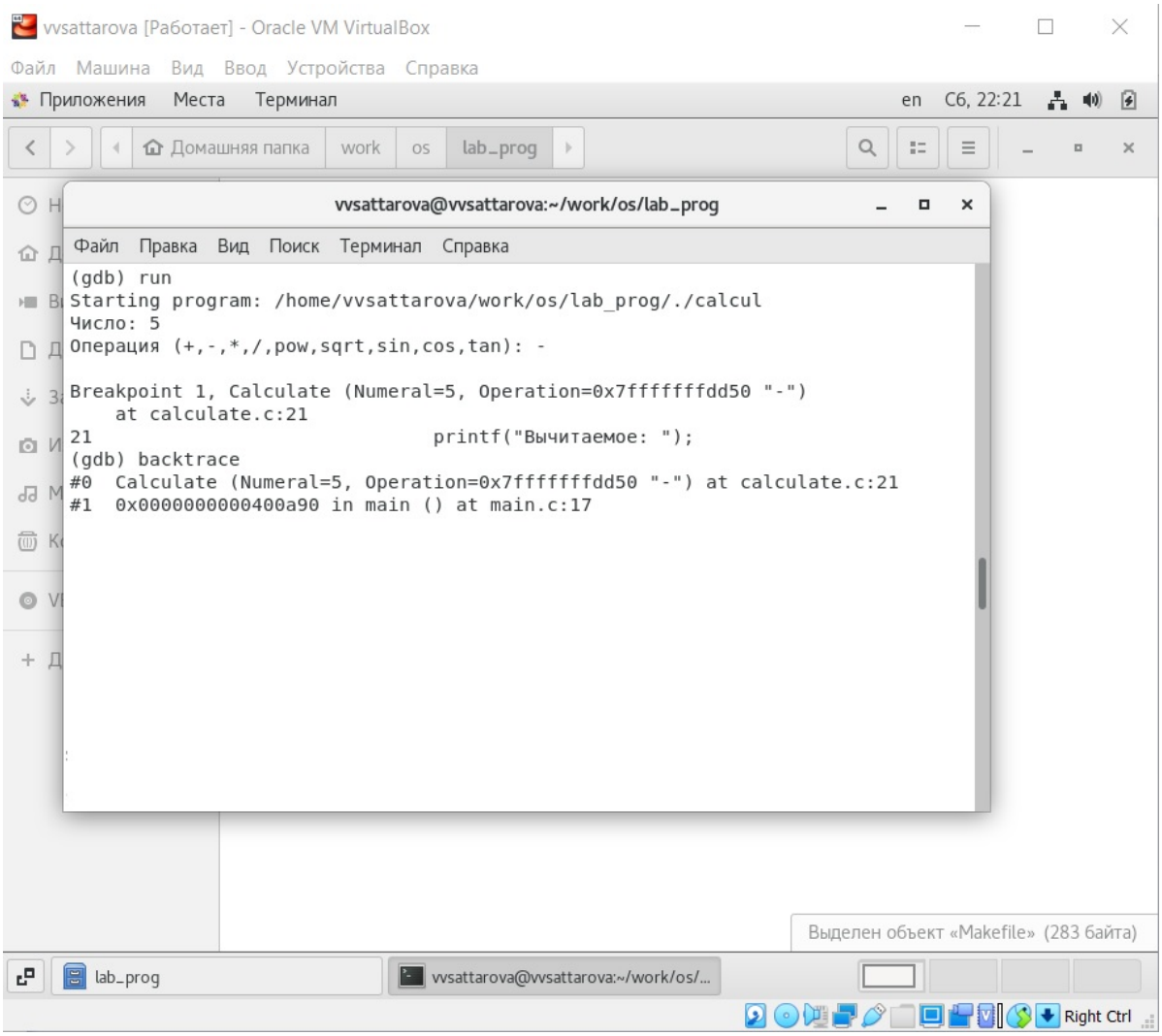

Figure 6.10: Рис. 10 Работа точек останова

12. Посмотрела, чему равно на этом этапе значение переменной Numeral, введя: print Numeral (рис. -fig. 6.11).

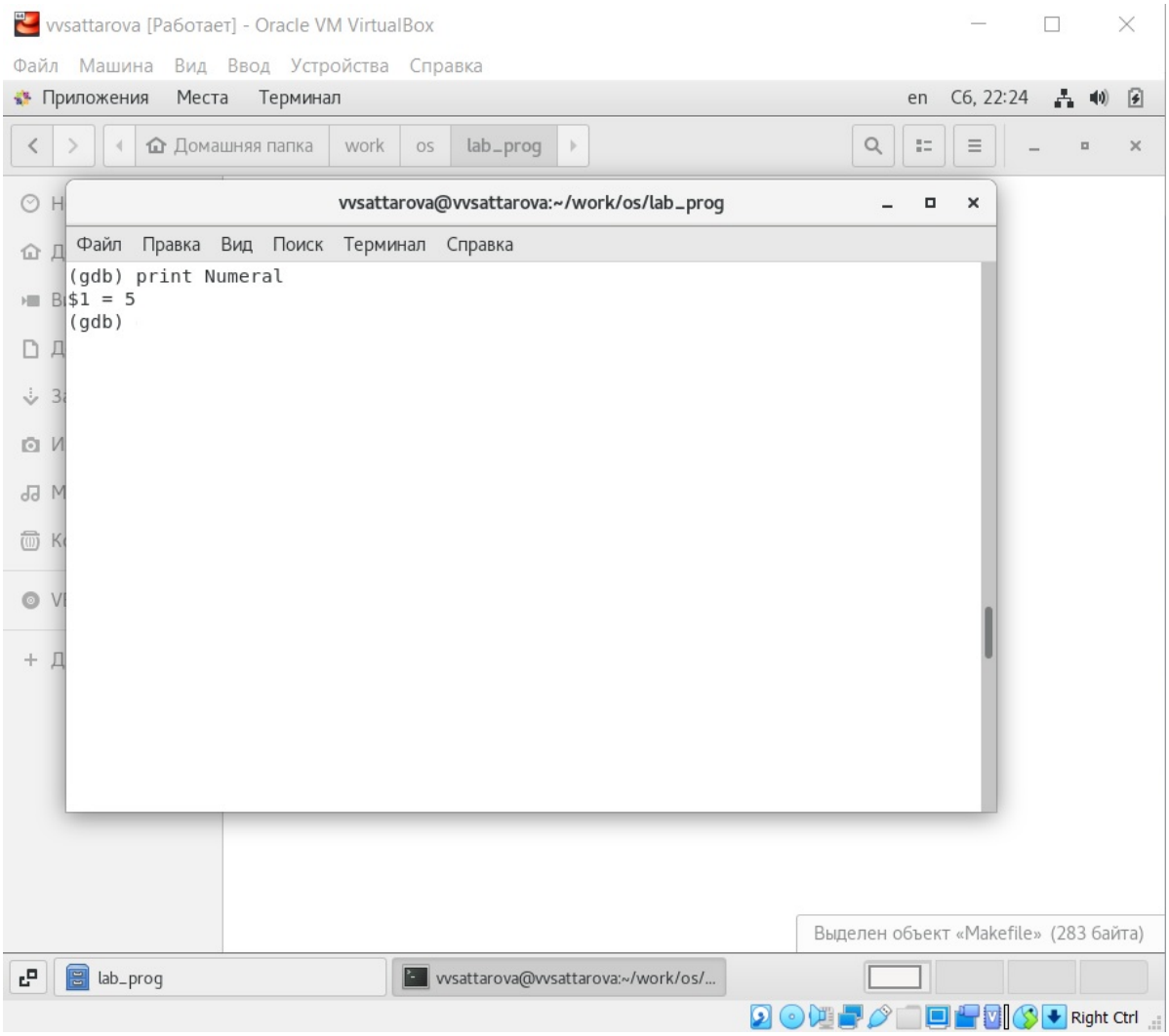

Figure 6.11: Рис. 11 Просмотр через печать

13. Сравнила с результатом вывода на экран после использования команды: display Numeral. (рис. -fig. 6.12)

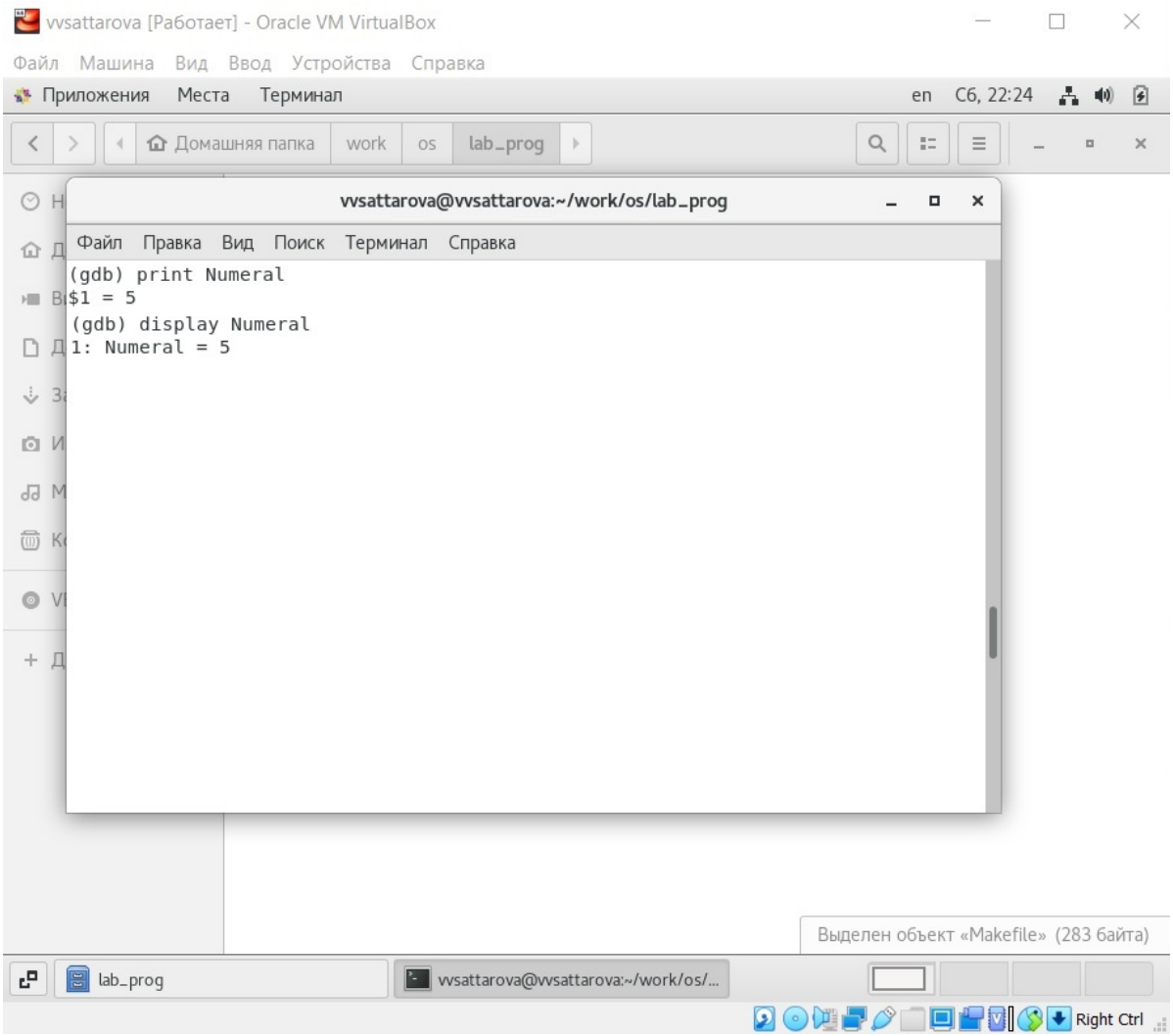

Figure 6.12: Рис. 12 Просмотр через вывод на экран

14. Убрала точки останова: info breakpoints, delete 1. (рис. -fig. 6.13)

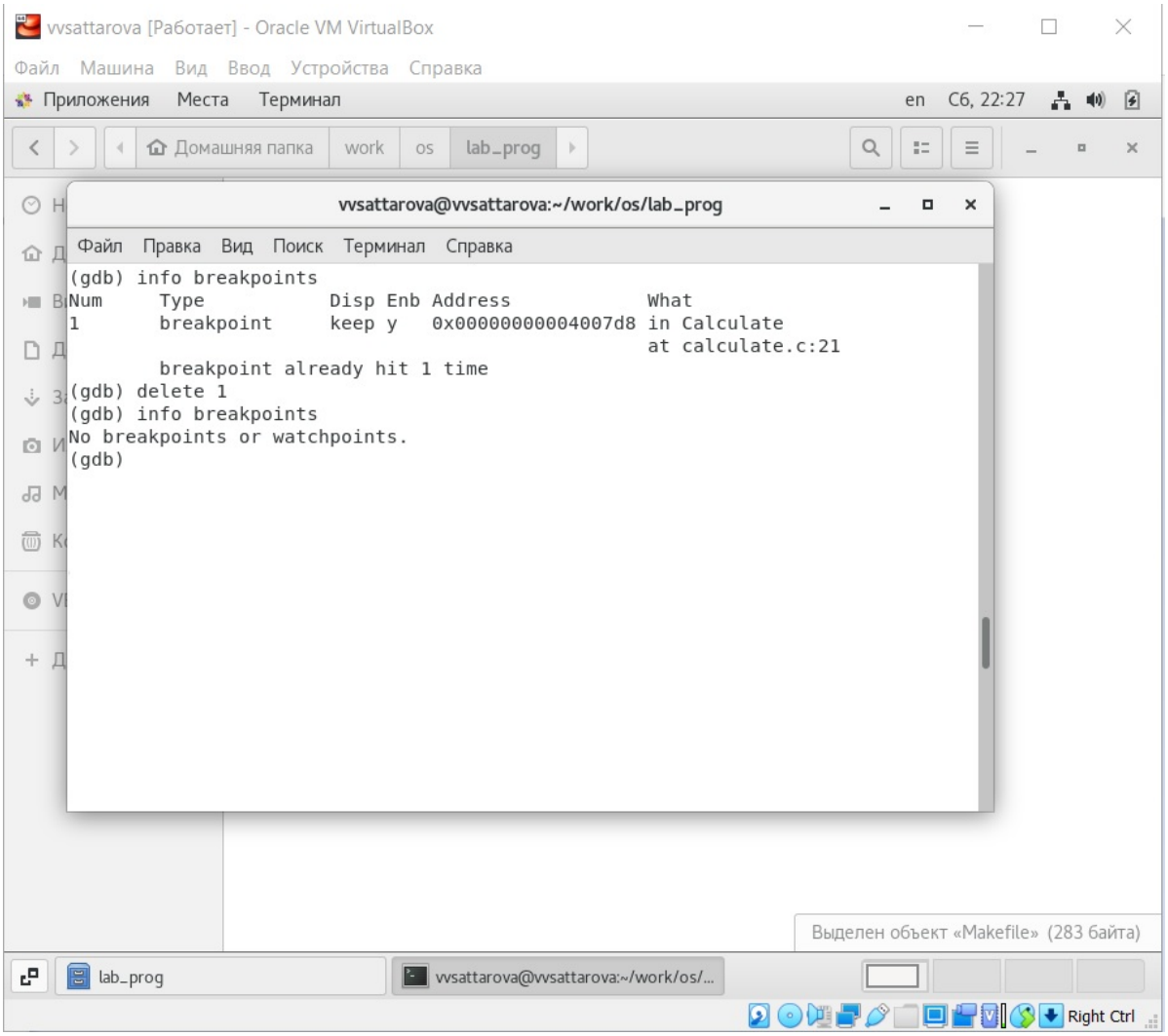

Figure 6.13: Рис. 13 Снятие точек останова

15. С помощью утилиты splint попробовала проанализировать коды файлов calculate.c и main.c. (рис. -fig. 6.14)

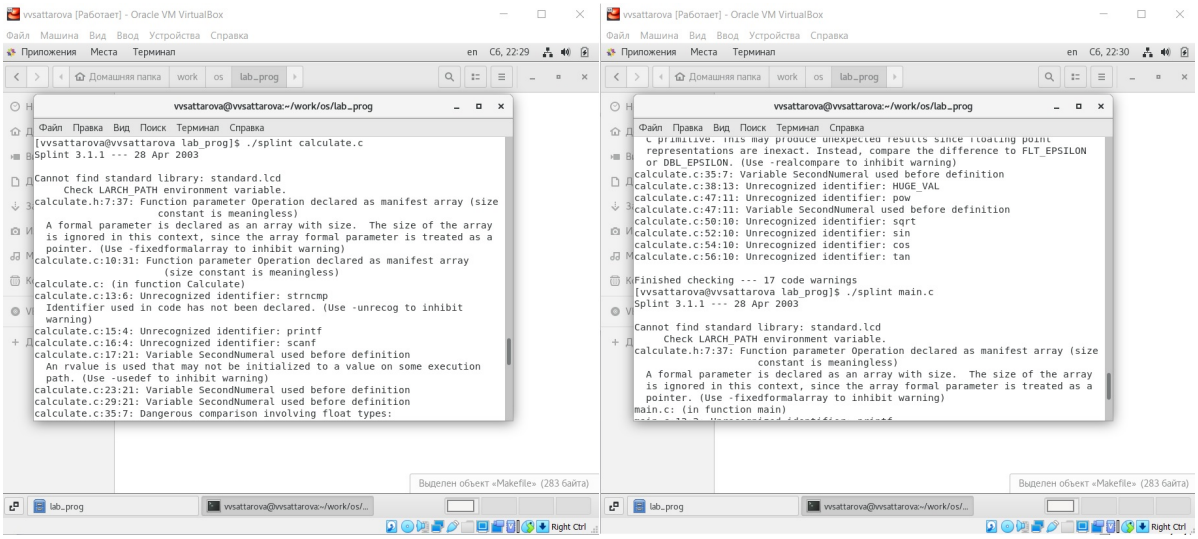

Figure 6.14: Рис. 14 Анализ кода

Подробное пояснение хода работы можно увидеть на видео.

7 Полученные результаты

Изучена информация, касающаяся разработки, анализа, тестирования и отладки приложений в ОС UNIX/Linux. Написана собственная программа калькулятор. Были рассмотрены на практике возможности анализа, тестирования и отладки приложений в ОС UNIX/Linux.

8 Анализ результатов

Работу получилось выполнить по инструкции, проблем с использованием команд по алгоритму, а также работы с файлами C не возникло. Был реализован калькулятор, затем он был запущен в Debugger. Также необходимо было поработать с утилитой splint.

9 Заключение и выводы

В ходе работы я gриобрела простейшие навыки разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linux на примере создания на языке программирования C калькулятора с простейшими функциями.

10 Контрольные вопросы

1. Как получить информацию о возможностях программ gcc, make, gdb и др.?
2. Назовите и дайте краткую характеристику основным этапам разработки приложений в UNIX.
3. Что такое суффикс в контексте языка программирования? Приведите примеры использования.
4. Каково основное назначение компилятора языка C в UNIX?
5. Для чего предназначена утилита make?
6. Приведите пример структуры Makefile. Дайте характеристику основным элементам этого файла.
7. Назовите основное свойство, присущее всем программам отладки. Что необходимо сделать, чтобы его можно было использовать?
8. Назовите и дайте основную характеристику основным командам отладчика gdb.
9. Опишите по шагам схему отладки программы, которую Вы использовали при выполнении лабораторной работы.
10. Прокомментируйте реакцию компилятора на синтаксические ошибки в программе при его первом запуске.
11. Назовите основные средства, повышающие понимание исходного кода программы.
12. Каковы основные задачи, решаемые программой splint?

11 Ответы на контрольные вопросы

1. Прочитать man-файл.
2. 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.

3. Суффикс – это расширение файла. Позволяет определить тип файла, т.е., что с ним можно делать. Например, файлы с расширением (суффиксом) .c воспринимаются gcc как программы на языке C, файлы с расширением .cc или .C — как файлы на языке C++, а файлы с расширением .o считаются объектными.
4. Компиляция файлов с исходным кодом в объектные модули и получение исполняемых файлов.

5. Make позволяет автоматизировать процесс преобразования файлов программы из одной формы в другую, отслеживает взаимосвязи между файлами.
6. Общий синтаксис Makefile имеет вид: target1 [target2…]:[:] [dependment1…] [(tab)commands] [#commentary] [(tab)commands] [#commentary] Сначала задаётся список целей, разделённых пробелами, за которым идёт двоеточие и список зависимостей. Затем в следующих строках указываются команды. Строки с командами обязательно должны начинаться с табуляции. В качестве цели в Makefile может выступать имя файла или название какого-то действия. Зависимость задаёт исходные параметры (условия) для достижения указанной цели. Зависимость также может быть названием какого-то действия. Команды — собственно действия, которые необходимо выполнить для достижения цели. После # пишутся комментарии, они не обрабатываются.
7. Программы отладки позволяют найти и устранить ошибки в программе. Чтобы использовать их, необходимо скомпилировать анализируемый код программы таким образом, чтобы отладочная информация содержалась в результирующем бинарном файле.
8. Основные команды: 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 двое-

точие 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. Запуск отладчика Запуск программы в отладчике Просмотр исходного кода основного и не основного файлов Установка точек останова и просмотр информации о них Запуск программы с установленными точками останова. Просмотр стека вызываемых функций при достижении точки останова Просмотр значения переменной в момент достижения точки останова Удаление точек останова
10. Компилятор выводит найденные им ошибки с комментариями. Это такие ошибки, которые могут повлиять на работу программы.
11. Само по себе грамотное написание кода (с переносами и отступами в нужных местах) уже повышает его понимание. Также этой цели служат комментарии.
12. Эта утилита анализирует программный код, проверяет корректность 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 та- кие ошибки, как одинаковые объекты, определённые в разных файлах, или объекты, чьи значения не используются в работе программы, переменные с некорректно за- данными значениями и типами и многое другое.