

# **Лабораторная работа-13**

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

Кузнецов Алексей НБИбд-02-21

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Задание</b>	<b>6</b>
<b>3</b>	<b>Выполнение лабораторной работы</b>	<b>8</b>
<b>4</b>	<b>Вывод:</b>	<b>17</b>
<b>5</b>	<b>Ответы на контрольные вопросы:</b>	<b>18</b>

## Список иллюстраций

3.1	Создаю подкаталог . . . . .	8
3.2	Создаю файлы . . . . .	9
3.3	Скрипт . . . . .	9
3.4	Пишу скрипт . . . . .	9
3.5	Скрипт . . . . .	10
3.6	Основной файл мэйн . . . . .	10
3.7	Компиляция . . . . .	11
3.8	Файлы . . . . .	11
3.9	Создал Мэйкфайл . . . . .	11
3.10	Исправил мэйкфайл . . . . .	12
3.11	Запустил программу . . . . .	12
3.12	Использую команды . . . . .	13
3.13	Использую команды . . . . .	13
3.14	Параметры Лист . . . . .	14
3.15	Информация о точках останова . . . . .	14
3.16	Запуск программы . . . . .	15
3.17	Сравнение . . . . .	15
3.18	Удаляю точки останова . . . . .	15
3.19	splint . . . . .	16
3.20	splint . . . . .	16

## 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 результаты и остановится. Реализация функций калькулятора в файле `calculate.h`:

Интерфейсный файл `calculate.h`, описывающий формат вызова функции-калькулятора:

3. Выполните компиляцию программы посредством `gcc`:
4. При необходимости исправьте синтаксические ошибки.
5. Создайте `Makefile` со следующим содержанием: Поясните в отчёте его содержание.
6. С помощью `gdb` выполните отладку программы `calcul` (перед использованием `gdb` исправьте `Makefile`): – Запустите отладчик `GDB`, загрузив в него программу для отладки: Кулябов Д.С. и др. Операционные системы 109 1  
`gdb ./calcul` – Для запуска программы внутри отладчика введите команду `run`: 1 `run` – Для постраничного (по 9 строк) просмотра исходного код

используйте команду list: 1 list – Для просмотра строк 12 по 15 основного файла используйте list с параметрами: 1 list 12,15 – Для просмотра определённых строк не основного файла используйте list с параметрами: 1 list calculate.c:20,29 – Установите точку останова в файле calculate.c на строке номер 21: 1 list calculate.c:20,27 2 break 21 – Выведите информацию об имеющихся в проекте точках останова: 1 info breakpoints – Запустите программу внутри отладчика и убедитесь, что программа остановится в момент прохождения точки останова: 1 run 2 5 3 - 4 backtrace – Отладчик выдаст следующую информацию: 1 #0 Calculate (Numeral=5, Operation=0x7fffffff280 “-”) 2 at calculate.c:21 3 #1 0x00000000400b2b 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: 1 10 Лабораторная работа No 13. Средства, применяемые при разработке программного... 1 print Numeral На экран должно быть выведено число 5. – Сравните с результатом вывода на экран после использования команды: 1 display Numeral – Уберите точки останова: 1 info breakpoints 2 delete 1

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

### 3 Выполнение лабораторной работы

1. В домашнем каталоге создал подкаталог ~/work/os/lab\_prog.(рис. 3.1)

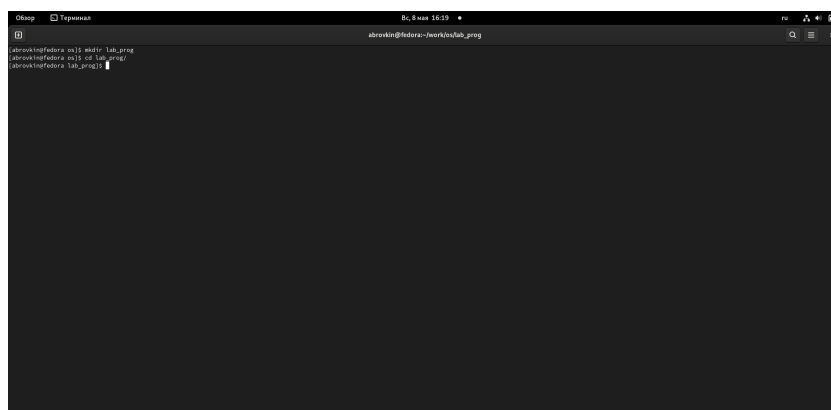


Рис. 3.1: Создаю подкаталог

2. Создал в нём файлы: calculate.h, calculate.c, main.c. Это примитивнейший калькулятор, способный складывать, вычитать, умножать, делить, возводить число в степень, вычислять квадратный корень, вычислять sin, cos, tan. При запуске он запрашивает первое число, операцию, второе число. После этого программа выводит результат и останавливается.(рис. 3.2)



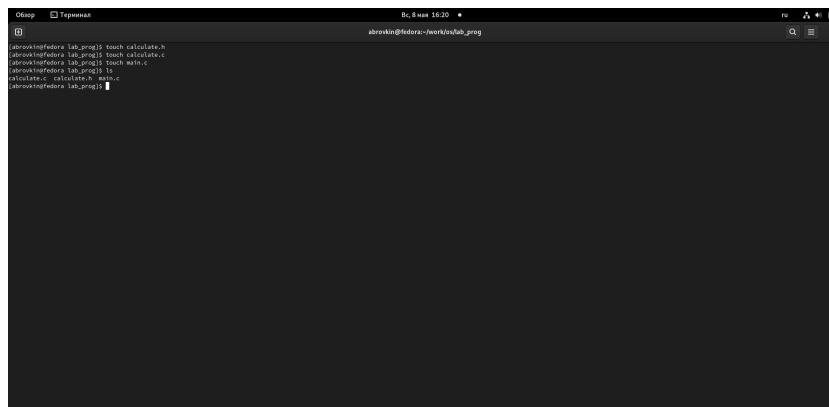


Рис. 3.2: Создаю файлы

Реализация функций калькулятора в файле calculate.c:(рис. 3.3)(рис. 3.4)

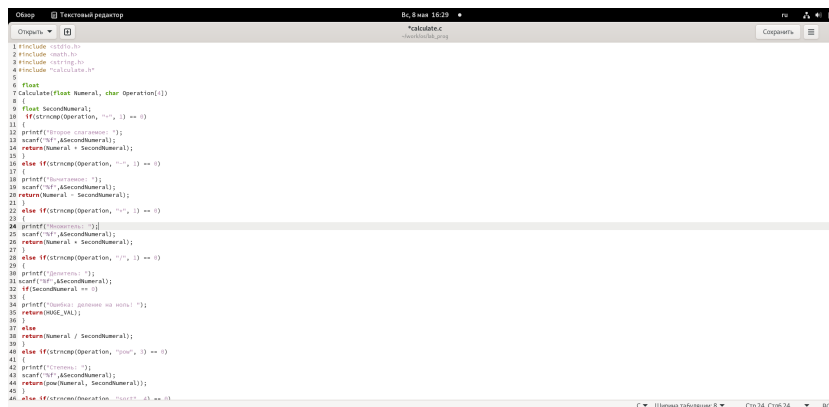


Рис. 3.3: Скрипт

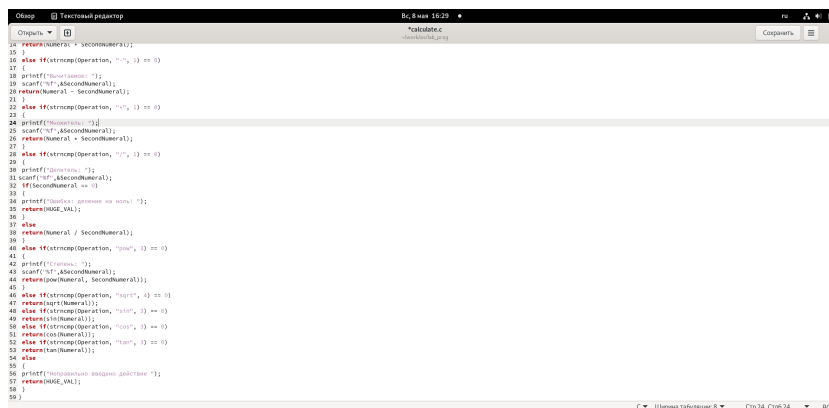
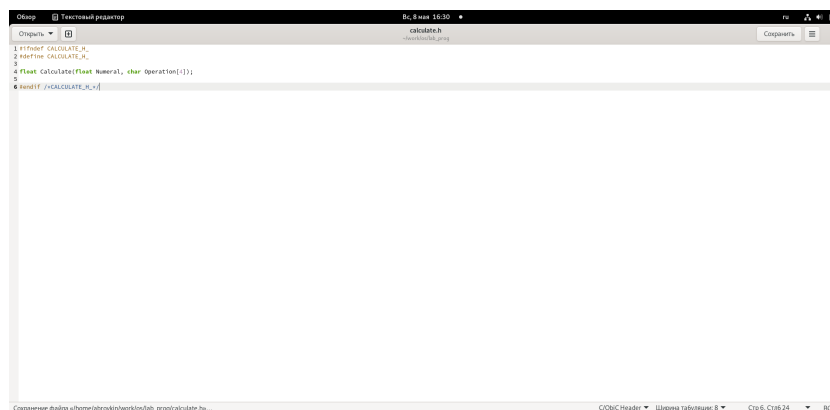


Рис. 3.4: Пишу скрипт

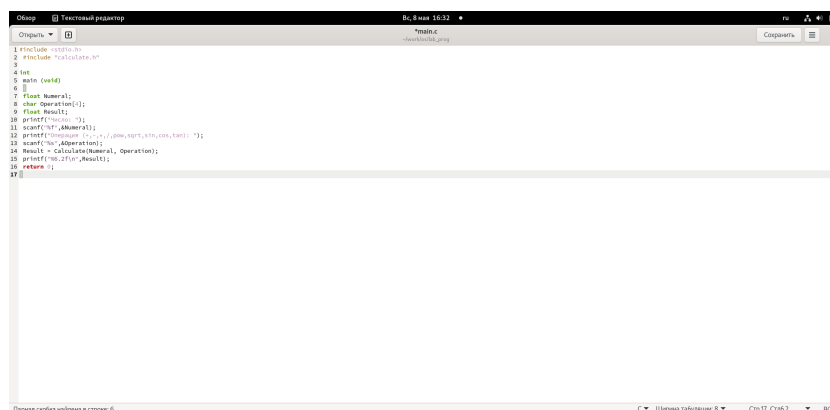
Интерфейсный файл calculate.h, описывающий формат вызова функции калькулятора:(рис. 3.5)



```
1 #ifndef CALCULATE_H
2 #define CALCULATE_H
3
4 float Calculate(float Numeral, char Operation[]);
5
6 #endif // !CALCULATE_H
```

Рис. 3.5: Скрипт

Основной файл main.c, реализующий интерфейс пользователя к калькулятору:(рис. 3.6)



```
1 #include <stdio.h>
2 #include "calculate.h"
3
4 int
5 main(void)
6 {
7     float Numeral;
8     char Operation[];
9     float Result;
10    printf("Numeral: ");
11    scanf("%f", &Numeral);
12    printf("Operation (+, -, *, /, pow, sqrt, xln, cos, tan): ");
13    scanf("%s", &Operation);
14    Result = Calculate(Numeral, Operation);
15    printf("Result: %f\n", Result);
16    return 0;
17 }
```

Рис. 3.6: Основной файл мейн

3.Выполнил компиляцию программы посредством gcc:(рис. 3.7)(рис. 3.8)

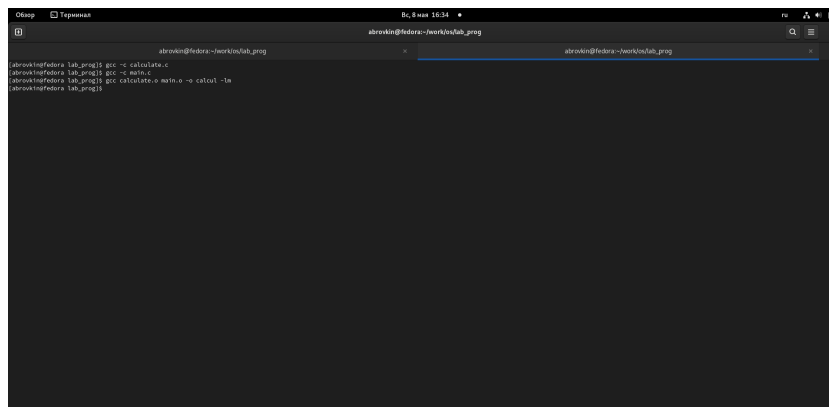


Рис. 3.7: Компиляция

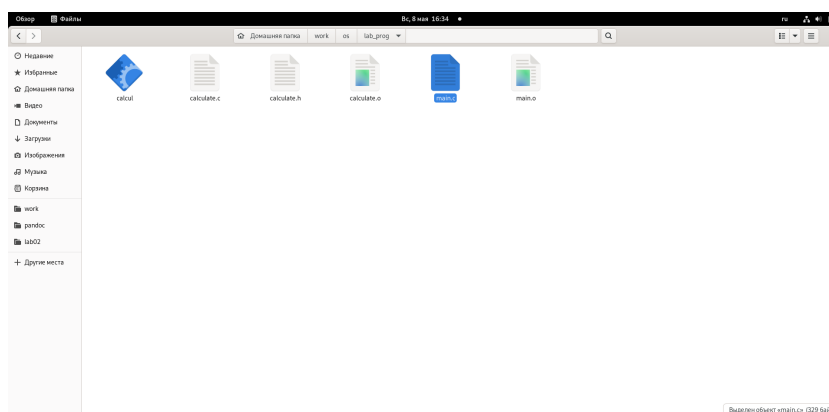


Рис. 3.8: Файлы

4. Исправил синтаксические ошибки.

5. Создал Makefile.(рис. 3.9)

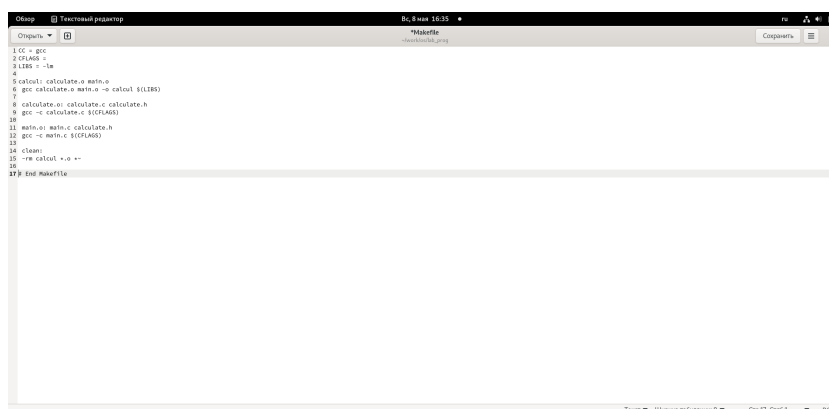
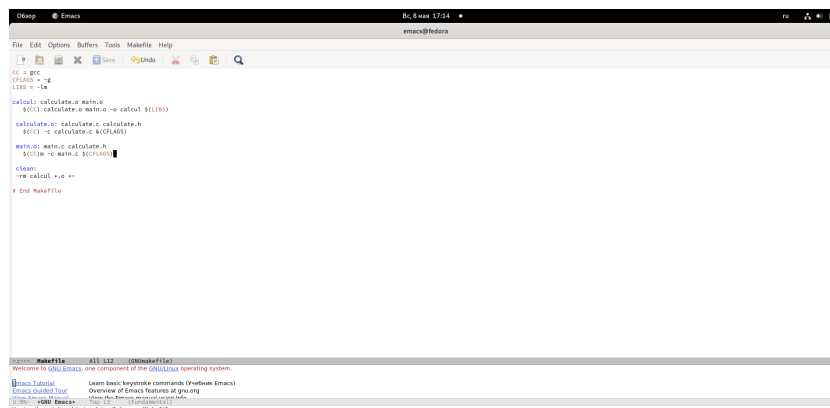


Рис. 3.9: Создал Мэйкфайл

В содержании файла указаны флаги компиляции, тип компилятора и файлы, которые должен собрать сборщик.

6. С помощью gdb выполнил отладку программы calcul (перед использованием gdb исправил Makefile): – запустите отладчик GDB, загрузив в него программу для отладки: `gdb ./calcul` – для запуска программы внутри отладчика ввел команду `run`(рис. 3.10)(рис. 3.11)



```
cc = gcc
CFLAGS = -g
LDFLAGS = -lm

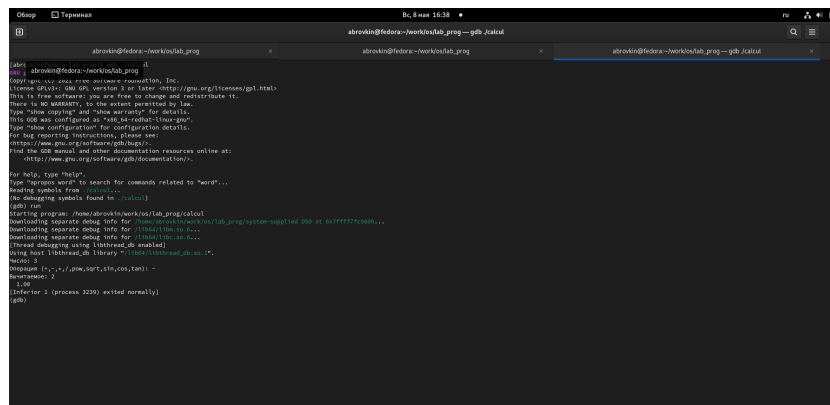
calcul: calcul.o main.o
    $(CC) calcul.o main.o -o calcul $(LDFLAGS)

calcul.o: calcul.c calcul.h
    $(CC) -c calcul.c $(CFLAGS)

main.o: main.c calcul.h
    $(CC) -c main.c $(CFLAGS)

clean:
    rm calcul *.o *~
+ End Makefile
```

Рис. 3.10: Исправил мейкфайл



```
abrokin@fedora:~/work/lab_prog$ gdb ./calcul
GNU gdb (Fedora 7.6.20120327) 7.6.20120327
Copyright (C) 2012 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./calcul...
(no debugging symbols found in ./calcul)
(gdb) run
Starting program: /home/abrokin/work/lab_prog/calcul
Downloading separate debug info for /lib64/libc.so.6...
Downloading separate debug info for /lib64/libpthread.so.0...
Thread debugging using libthread_db enabled.
Using host libthread_db library "/lib64/libthread_db.so.1".
120
Program terminated normally.
(gdb)
```

Рис. 3.11: Запустил программу

– для страничного (по 9 строк) просмотра исходного код использовал команду `list` – для просмотра строк с 12 по 15 основного файла использовал `list` с параметрами: `list 12,15`(рис. 3.12)(рис. 3.13)

```

abrovin@fedora-lab-prog ~$ gcc -o calculate.o calculate.c
abrovin@fedora-lab-prog ~$ ./calculate.o
Result: 12.3456789

```

Рис. 3.12: Используя команды

```

(gdb) list calculate.c:20,29
20  int main()
21  {
22      int a = 1;
23      int b = 2;
24      int c = 3;
25      int d = 4;
26      int e = 5;
27      int f = 6;
28      int g = 7;
29      int h = 8;

```

Рис. 3.13: Используя команды

– для просмотра определённых строк не основного файла использовал list с параметрами: list calculate.c:20,29 – установил точку останова в файле calculate.c на строке номер 21: list calculate.c:20,27 break 20 – вывел информацию об имеющихся в проекте точка останова: info breakpoints(рис. 3.14)(рис. 3.15)

```

abrokin@fedora-workbook:~$ gdb ./calcul
(gdb) list
19      }
20      }
21      }
22      }
23      }
24      }
25      }
26      }
27      }
(gdb) list calculate.c:20,27
20      }
21      }
22      }
23      }
24      }
25      }
26      }
27      }
(gdb) list calculate.c:20,27
20      }
21      }
22      }
23      }
24      }
25      }
26      }
27      }
(gdb) break 21
Breakpoint 1 at calculate.c:21:22.
(gdb)

```

Рис. 3.14: Параметры Лист

```

abrokin@fedora-workbook:~$ gdb ./calcul
(gdb) info breakpoints
Breakpoint 1 at 0x0000000000401022: File calculate.c, line 22.
Breakpoint 2 at 0x0000000000401023: File calculate.c, line 23.
Breakpoint 3 at 0x0000000000401024: File calculate.c, line 24.
Breakpoint 4 at 0x0000000000401025: File calculate.c, line 25.
(gdb)

```

Рис. 3.15: Информация о точках останова

– запустил программу внутри отладчика и убедился, что программа остановится в момент прохождения точки останова – отладчик выдал следующую информацию, а команда `backtrace` показала весь стек вызываемых функций от начала программы до текущего места: – посмотрел, чему равно на этом этапе значение переменной `Numeral`, введя: `print Numeral` – сравнил с результатом вывода на экран после использования команды: `display Numeral` – убрал точки останова: `info breakpoints delete 1`(рис. 3.16)(рис. 3.17)(рис. 3.18)

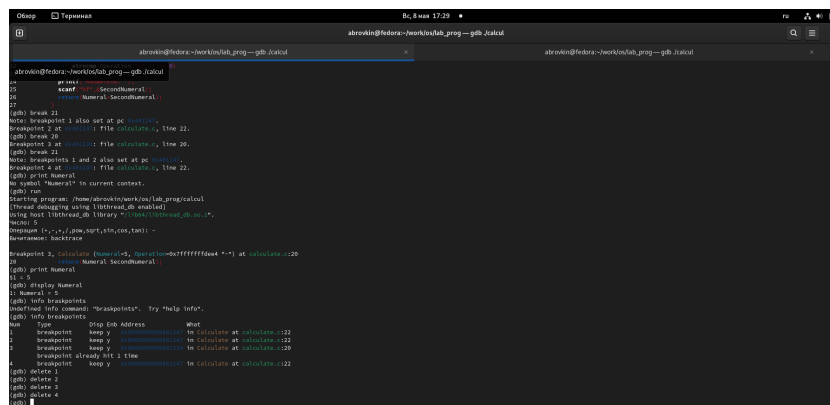
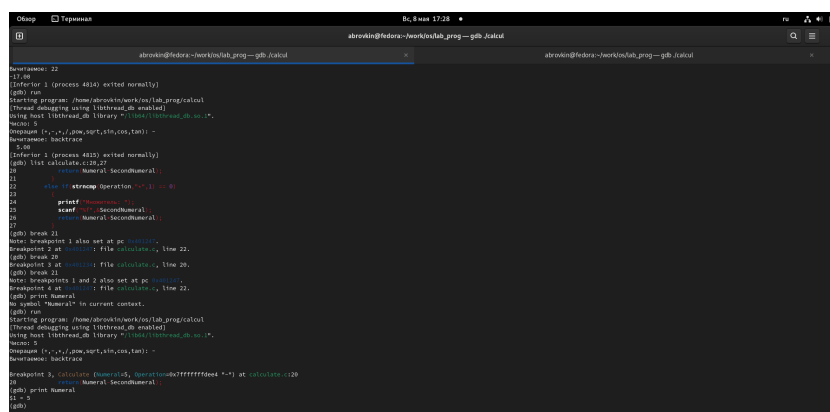
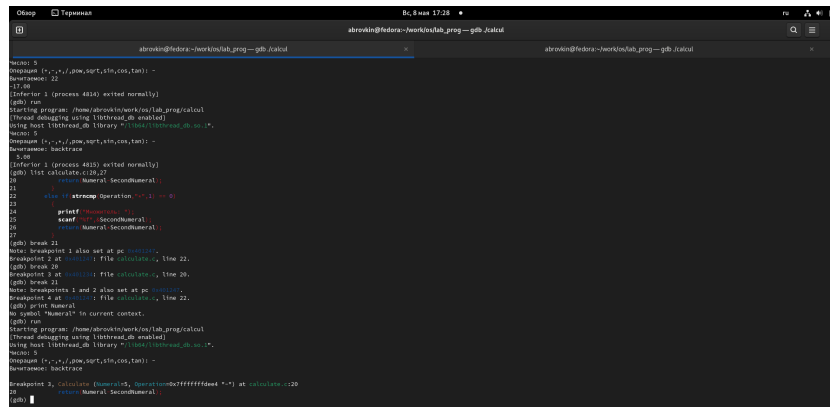
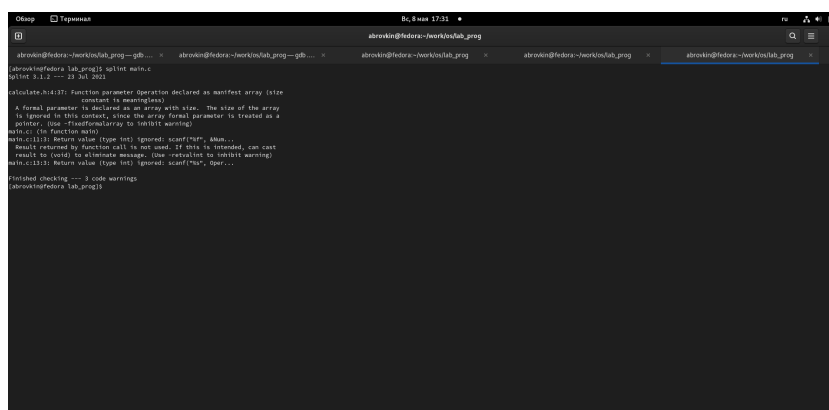
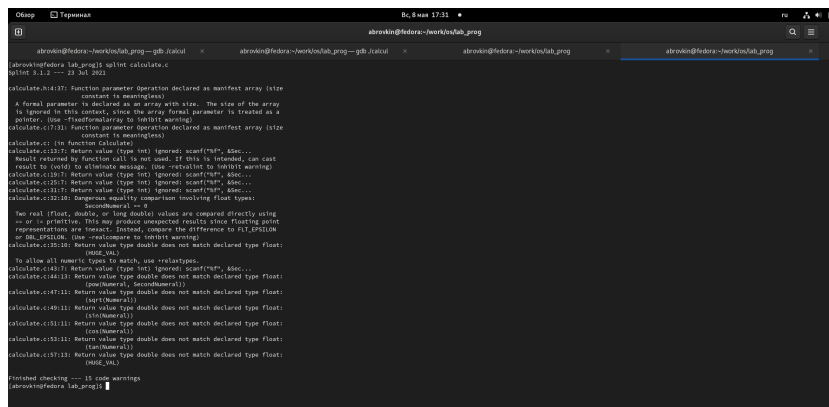


Рис. 3.18: Удаляю точки останова

7. С помощью утилиты `splint` попробуйте проанализировать коды файлов `calculate.c` и `main.c` (рис. 3.19)(рис. 3.20)





## 4 Вывод:

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 Ответы на контрольные вопросы:

1. Информацию об этих программах можно получить с помощью функций `info` и `man`.
2. Unix поддерживает следующие основные этапы разработки приложений:
  - создание исходного кода программы; - 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 “.c” 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 .c компилятор распознает, что файл `abcd.c` должен компилироваться, а по суффиксу .o, 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 предпочитают 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. Основное назначение компилятора с языка Си заключается в компиляции всей программы в целом и получении исполняемого модуля.
5. При разработке большой программы, состоящей из нескольких исходных файлов заголовков, приходится постоянно следить за файлами, которые требуют перекомпиляции после внесения изменений. Программа `make` освобождает пользователя от такой рутинной работы и служит для документирования взаимосвязей между файлами. Описание взаимосвязей и соответствующих действий хранится в так называемом `make-файле`, который по умолчанию имеет имя `makefile` или `Makefile`.
6. В общем случае `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. 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. Приведённый выше make-файл для программы abcd.c включает два способа компиляции и построения исполняемого модуля. Первый способ предусматривает обычную компиляцию с построением исполняемого модуля с именем abcd. Второй способ позволяет включать в исполняемый модуль testabcd возможность выполнить процесс отладки на уровне исходного текста. Пример можно найти в задании 5.

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. 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 двоеточие 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.   1) Выполнила компиляцию программы 2) Увидела ошибки в программе  
      3) Открыла редактор и исправила программу 4) Загрузила программу в отладчик `gdb` 5) `run` — отладчик выполнил программу, ввела требуемые значения. 6) Использовала другие команды отладчика и проверила работу программы
10. Отладчику не понравился формат `%s` для `&Operation`, т.к `%s` — символьный формат, а значит необходим только `Operation`.
11. Если вы работаете с исходным кодом, который не вами разрабатывался, то назначение различных конструкций может быть не совсем понятным. Система разработки приложений UNIX предоставляет различные средства, повышающие понимание исходного кода. К ним относятся: – `cscope` - исследование функций, содержащихся в программе; – `splint` — критическая проверка программ, написанных на языке Си.
12.   1. Проверка корректности задания аргументов всех использованных в программе функций, а также типов возвращаемых ими значений; 2. Поиск фрагментов исходного текста, корректных с точки зрения синтак-

сиса языка Си, но малоэффективных с точки зрения их реализации или содержащих в себе семантические ошибки; 3.Общая оценка мобильности пользовательской программы.