

РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ
Факультет физико-математических и естественных наук
Кафедра прикладной информатики и теории вероятностей

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ № 14
Средства, применяемые при разработке программного обеспечения в ОС
типа UNIX/Linux
дисциплина: Операционные системы

Студент: Тазаева Анастасия Анатольевна

Группа: НПИбд-02-20

МОСКВА 2021г.

Содержание:

1. Цель работы
2. Задания
3. Ход работы
4. Контрольные вопросы(ответы)
5. Выводы

Цель работы:

Приобрести простейшие навыки разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linux на примере создания на языке программирования C калькулятора с простейшими функциями

Задания:

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.

Реализация функций калькулятора в файле **`calculate.c`**

Интерфейсный файл **calculate.h**, описывающий формат вызова функции калькулятора

Основной файл **main.c**, реализующий интерфейс пользователя к калькулятору

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 со содержанием
 6. Поясните в отчёте его содержание.
 7. 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 точка останова:
info breakpoints
– Запустите программу внутри отладчика и убедитесь, что программа остановится в момент прохождения точки останова:
run
5
-- backtrace
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
 8. С помощью утилиты splint попробуйте проанализировать коды файлов calculate.c и main.c.
-

Ход работы:

1. В домашнем каталоге создала подкаталог `~/work/os/lab_prog` с помощью команды `MKDIR`(рис.1)

```
aatazaeva@aatazaeva-VirtualBox:~$ cd ~/work/os
aatazaeva@aatazaeva-VirtualBox:~/work/os$ mkdir lab_prog
```

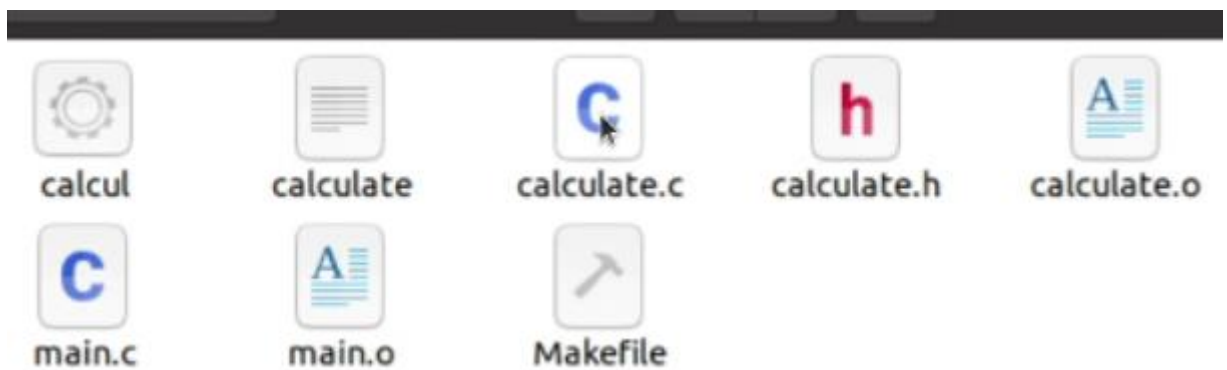
(рис.1)

2. Создала в нём файлы: `calculate.h`, `calculate.c`, `main.c`. С помощью команды `touch`(рис.2)

```
aatazaeva@aatazaeva-VirtualBox:~/work/os/lab_prog$ touch calculate.c calculate.h main.c
```

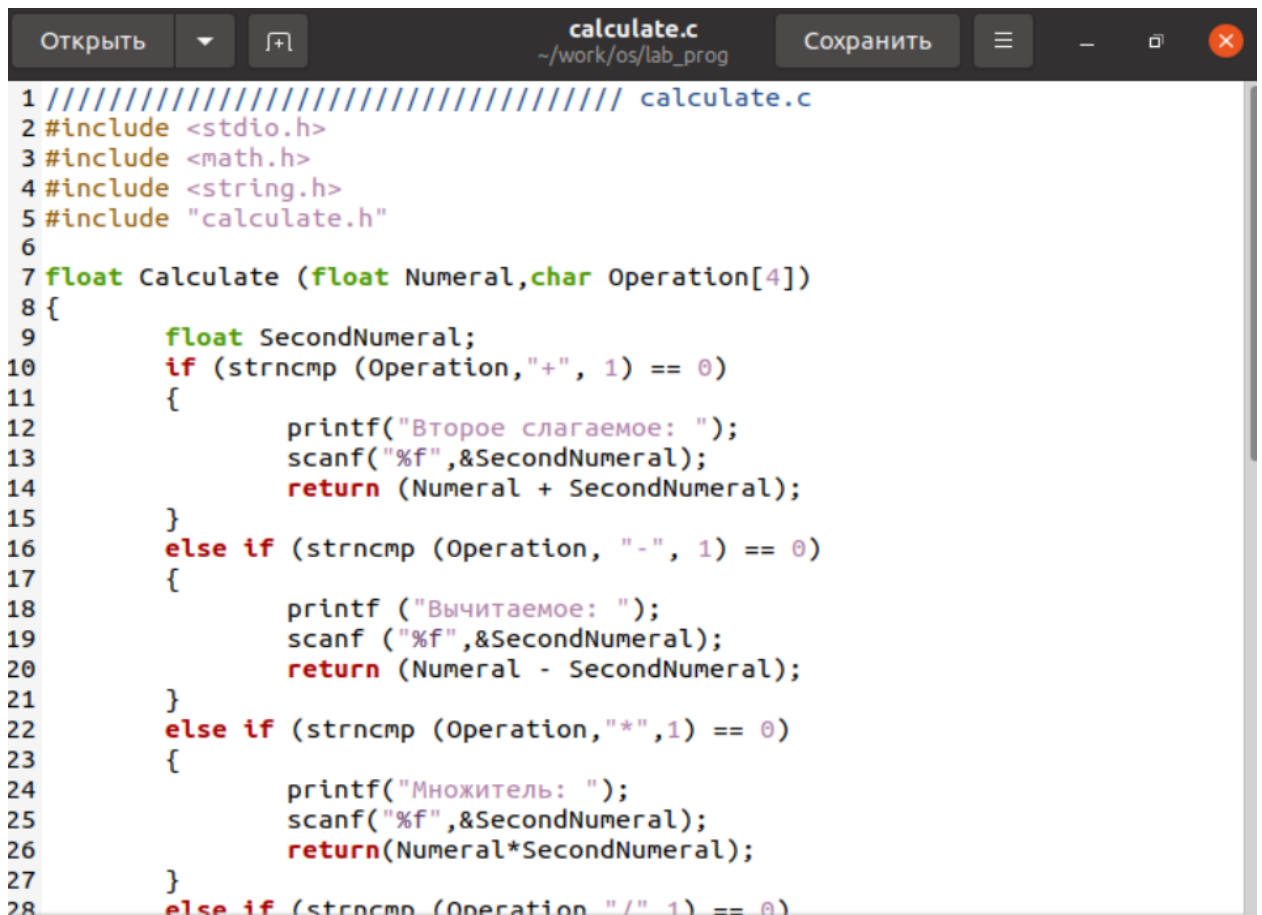
(рис.2)

Эти файлы будут содержать в себе код(рис.3). Создаем простой калькулятор.



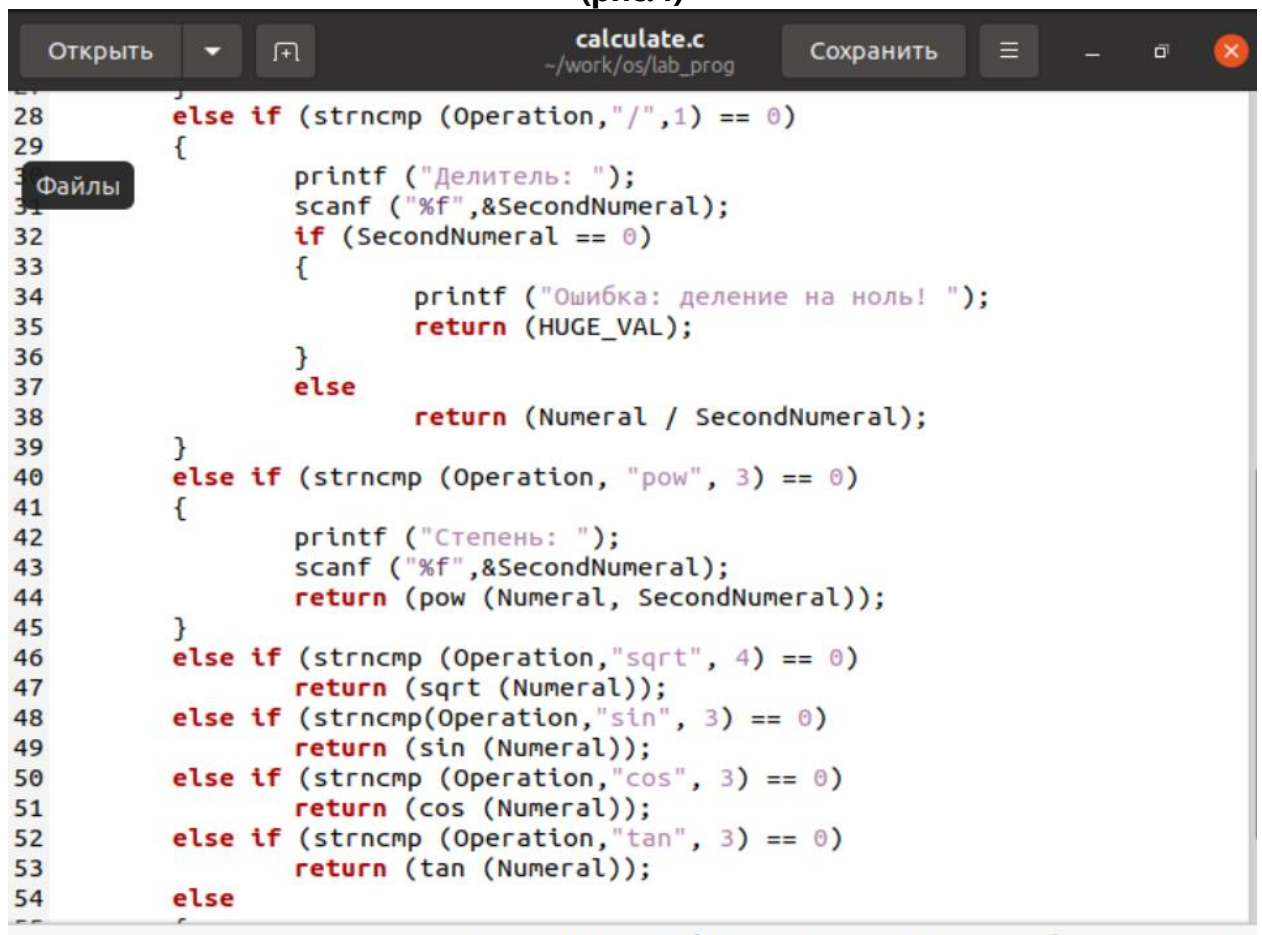
(рис.3)

Файл `calculate.c` с кодом(рис.4-6)



```
1 ////////////////////////////////////////////////// calculate.c
2 #include <stdio.h>
3 #include <math.h>
4 #include <string.h>
5 #include "calculate.h"
6
7 float Calculate (float Numeral, char Operation[4])
8 {
9     float SecondNumeral;
10    if (strncmp (Operation, "+", 1) == 0)
11    {
12        printf("Второе слагаемое: ");
13        scanf("%f",&SecondNumeral);
14        return (Numeral + SecondNumeral);
15    }
16    else if (strncmp (Operation, "-", 1) == 0)
17    {
18        printf ("Вычитаемое: ");
19        scanf ("%f",&SecondNumeral);
20        return (Numeral - SecondNumeral);
21    }
22    else if (strncmp (Operation, "*", 1) == 0)
23    {
24        printf("Множитель: ");
25        scanf("%f",&SecondNumeral);
26        return(Numeral*SecondNumeral);
27    }
28    else if (strncmp (Operation, "/", 1) == 0)
```

(рис.4)



```
28    else if (strncmp (Operation, "/", 1) == 0)
29    {
30        printf ("Делитель: ");
31        scanf ("%f",&SecondNumeral);
32        if (SecondNumeral == 0)
33        {
34            printf ("Ошибка: деление на ноль! ");
35            return (HUGE_VAL);
36        }
37        else
38            return (Numeral / SecondNumeral);
39    }
40    else if (strncmp (Operation, "pow", 3) == 0)
41    {
42        printf ("Степень: ");
43        scanf ("%f",&SecondNumeral);
44        return (pow (Numeral, SecondNumeral));
45    }
46    else if (strncmp (Operation, "sqrt", 4) == 0)
47        return (sqrt (Numeral));
48    else if (strncmp (Operation, "sin", 3) == 0)
49        return (sin (Numeral));
50    else if (strncmp (Operation, "cos", 3) == 0)
51        return (cos (Numeral));
52    else if (strncmp (Operation, "tan", 3) == 0)
53        return (tan (Numeral));
54    else
```

(рис.5)

```
53         return (tan (Numeral));
54     else
55     {
56         printf ("Неправильно введено действие ");
57         return [HUGE_VAL];
58     }
59 }
60
```

(рис.6)

Файл *calculate.h* с кодом(рис.7)

```
calculate.h
~/work/os/lab_prog

1 #ifndef CALCULATE_H_
2 #define CALCULATE_H_
3 float Calculate(float Numeral, char Operation[4]);
4 #endif /*CALCULATE_H_*/
```

(рис.7)

Файл *main.c* с кодом(рис.8)

```
main.c
~/work/os/lab_prog

1 #include <stdio.h>
2 #include "calculate.h"
3 #include <math.h>
4
5 int main(void)
6 {
7     float Numeral;
8     char Operation[4];
9     float Result;
10    printf("Число: ");
11    scanf("%f",&Numeral);
12    printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): ");
13    scanf("%s",&Operation);
14    Result = Calculate(Numeral, Operation);
15    printf("%6.2f\n",Result);
16    return 0;
17 }
```

(рис.8)

3. Далее выполнила компиляцию с помощью команды(рис.9)

- gcc -g -c calculate.c

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

```
aatazaeva@aatazaeva-VirtualBox:~/work/os/lab_prog$ gcc -g -c calculate.c
aatazaeva@aatazaeva-VirtualBox:~/work/os/lab_prog$ gcc -g -c main.c
main.c: In function 'main':
main.c:13:10: warning: format '%s' expects argument of type 'char *', but argument 2 has type 'char (*)[4]' [-Wformat=]
   13 |     scanf("%s",&Operation);
      |           ^~
      |           |
      |           | char (*)[4]
      |           char *
```

```
aatazaeva@aatazaeva-VirtualBox:~/work/os/lab_prog$ gcc calculate.o main.o -o calcul -lm -g
```

(рис.8)

4. Синтаксических ошибок не было, было простое предупреждение
5. Создала Makefile с помощью команды `touch Makefile`, далее пользовалась `emacs`'ом(рис.10-11). С кодом внутри(рис.12).

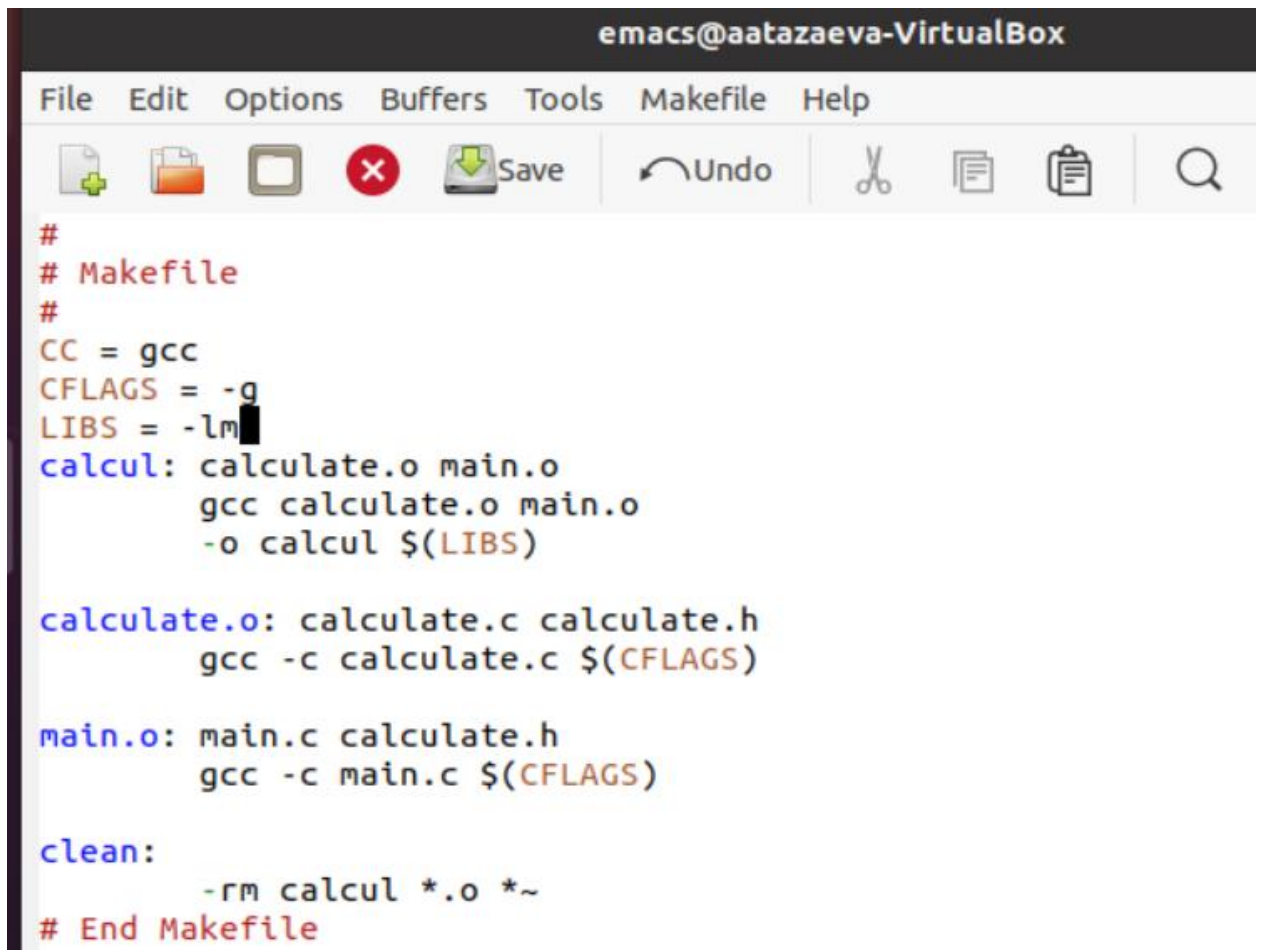
Чтобы **не компилировать все по отдельности** мы и создали Makefile. Makefile является списком правил. Каждое правило начинается с указателя, называемого «Цель». После него стоит двоеточие, а далее через пробел указываются зависимости. В нашем случае ясно, что конечный файл `kalkul` зависит от объектных файлов `calculate.o` и `main.o`. Поэтому они должны быть собраны прежде сборки `kalkul`. После зависимостей пишутся команды. Каждая команда должна находиться на отдельной строке, и отделяться от начала строки клавишей `Tab`. Структура правила Makefile может быть очень сложной. Там могут присутствовать переменные, конструкции ветвления, цикла.

```
aatazaeva@aatazaeva-VirtualBox:~/work/os/lab_prog$ touch Makefile
```

(рис.10)

```
aatazaeva@aatazaeva-VirtualBox:~/work/os/lab_prog$ emacs Makefile
```

(рис.11)



```
#
# Makefile
#
CC = gcc
CFLAGS = -g
LIBS = -lm
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.12)

6. С помощью *gdb* выполнила отладку программы *calcul*:

- Запустила отладчик GDB, загрузив в него программу для отладки:
`gdb ./calcul` (рис.13)


```

aatazaeva@aatazaeva-VirtualBox:~/work/os/lab_prog$ gdb ./calcul
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04) 9.2
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/g
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-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...
(gdb) run

```

(рис.13)

- Для запуска программы внутри отладчика ввела команду run и проверила работу(2+2; 2^10;)(рис.14):
run

```

(gdb) run
Starting program: /home/aatazaeva/work/os/lab_prog
Число: 2
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): +
Второе слагаемое: 2
    4.00
[Inferior 1 (process 58090) exited normally]
(gdb) run
Starting program: /home/aatazaeva/work/os/lab_prog
Число: 2
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): pow
Степень: 10
    1024.00

```

(рис.14)

- Для постраничного (по 9 строк) просмотра исходного код использовала команду list(рис.15):
list


```
(gdb) list
1      #include <stdio.h>
2      #include "calculate.h"
3      #include <math.h>
4
5      int main(void)
6      {
7          float Numeral;
8          char Operation[4];
9          float Result;
10         printf("Введите число: ");
```

(рис.15)

- Для просмотра строк с 12 по 15 основного файла использовала list с параметрами:
list 12,15

```
(gdb) list 12,15
12         printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): ");
13         scanf("%s",&Operation);
14         Result = Calculate(Numeral, Operation);
15         printf("%.2f\n",Result);
```

(рис.16)

- Для просмотра определённых строк не основного файла использовала list с параметрами:
list calculate.c:20,29

```
(gdb) list calculate.c:20,29
20         return (Numeral - SecondNumeral);
21     }
22     else if (strncmp (Operation,"*",1) == 0)
23     {
24         printf("Множитель: ");
25         scanf("%f",&SecondNumeral);
26         return(Numeral*SecondNumeral);
27     }
28     else if (strncmp (Operation,"/",1) == 0)
29     {
```

(рис.17)

- Установила точку останова в файле calculate.c на строке номер 21:
list calculate.c:20,27
break 21

```

(gdb) list calculate.c:20,27
20             return (Numeral - SecondNumeral);
21         }
22         else if (strcmp (Operation,"*",1) == 0)
23         {
24             printf("Множитель: ");
25             scanf("%f",&SecondNumeral);
26             return(Numeral*SecondNumeral);
27         }
(gdb) break 21
Breakpoint 1 at 0x555555555319: file calculate.c, line 22.

```

(рис.18)

- Вывела информацию об имеющихся в проекте точка останова:
info breakpoints

```

(gdb) info breakpoints
Num      Type          Disp Enb Address                      What
1        breakpoint    keep y  0x0000555555555319 in Calculate
                                     at calculate.c:22

```

(рис.19)

- Убрала точки останова:
info breakpoints
delete 1

```

(gdb) info breakpoints
Num      Type          Disp Enb Address                      What
1        breakpoint    keep y  0x0000555555555319 in Calculate
                                     at calculate.c:22
(gdb) delete 1

```

(рис.20)

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

Контрольные вопросы:

1. Как получить информацию о возможностях программ gcc, make, gdb и др?
воспользоваться командой man, info
2. Назовите и дайте краткую характеристику основным этапам разработки приложений в UNIX.

Unix поддерживает следующие основные этапы разработки приложений:

- создание исходного кода программы;
- 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;
- сохранение всех изменений, выполняемых при тестировании и отладке.

3. Что такое суффикс в контексте языка программирования? Приведите примеры использования.

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.

4. Каково основное назначение компилятора языка C в UNIX?

В компиляции всей программы в целом и получении исполняемого модуля.

5. Для чего предназначена утилита make?

Для упрощения и автоматизации работы пользователя с командной строкой

6. Приведите пример структуры Makefile. Дайте характеристику основным элементам этого файла

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; : — 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. Назовите основное свойство, присущее всем программам отладки. Что необходимо сделать, чтобы его можно было использовать?
Все программы отладки позволяют отслеживать состояние программы на любом из этапов ее исполнения. Для того чтобы эту возможность использовать необходимо изучить документацию по использованию определенного отладчика. Понять общие принципы отладки.
8. Назовите и дайте основную характеристику основным командам 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 передается 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. Опишите по шагам схему отладки программы, которую Вы использовали при выполнении лабораторной работы.

Сначала я запустила отладчик для моей программы. Установила интересующую меня точку остановки. Запустил программу ожидая, что программа остановится на

точке остановки. Узнала необходимые данные моей программы на текущем этапе ее исполнения путем ввода команд. Отобразил данные. Завершила программу, снял точки остановки.

10. Прокомментируйте реакцию компилятора на синтаксические ошибки в программе при его первом запуске.

В моем случае не было синтаксических ошибок.

11. Назовите основные средства, повышающие понимание исходного кода программы.

- cscope - исследование функций, содержащихся в программе;
- lint -- критическая проверка программ, написанных на языке Си.

12. Каковы основные задачи, решаемые программой splint?

Splint- инструмент для статической проверки C-программ на наличие уязвимостей и ошибок

ВЫВОД:

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