

Отчет по лабораторной работе №13

Дисциплина: Операционные системы

Шишук Владислав Олегович

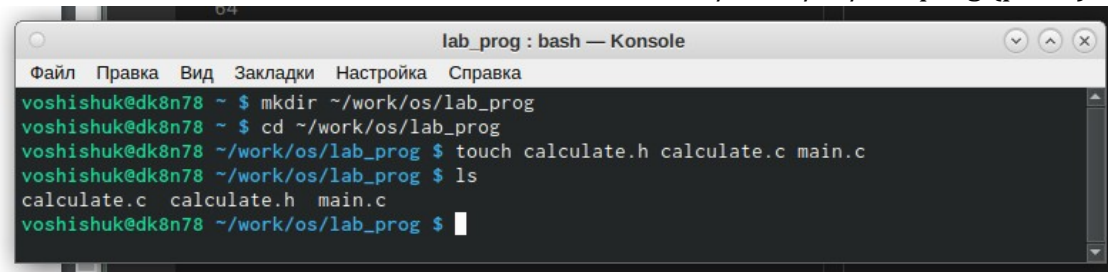
Содержание

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.

Выполнение лабораторной работы

1. В домашнем каталоге создаем подкаталог `~/work/os/lab_prog` (рис.1)



```
lab_prog : bash — Konsole
Файл  Правка  Вид  Закладки  Настройка  Справка
voshishuk@dk8n78 ~ $ mkdir ~/work/os/lab_prog
voshishuk@dk8n78 ~ $ cd ~/work/os/lab_prog
voshishuk@dk8n78 ~/work/os/lab_prog $ touch calculate.h calculate.c main.c
voshishuk@dk8n78 ~/work/os/lab_prog $ ls
calculate.c calculate.h main.c
voshishuk@dk8n78 ~/work/os/lab_prog $
```

рис.1

2. Создаем в нем файлы `calculate.h`, `calculate.c`, `main.c`. (рис.1) Далее реализуем функции калькулятора в файлах: `calculate.c` (рис.2, рис.3), `calculate.h` (рис.4), `main.c` (рис.5)

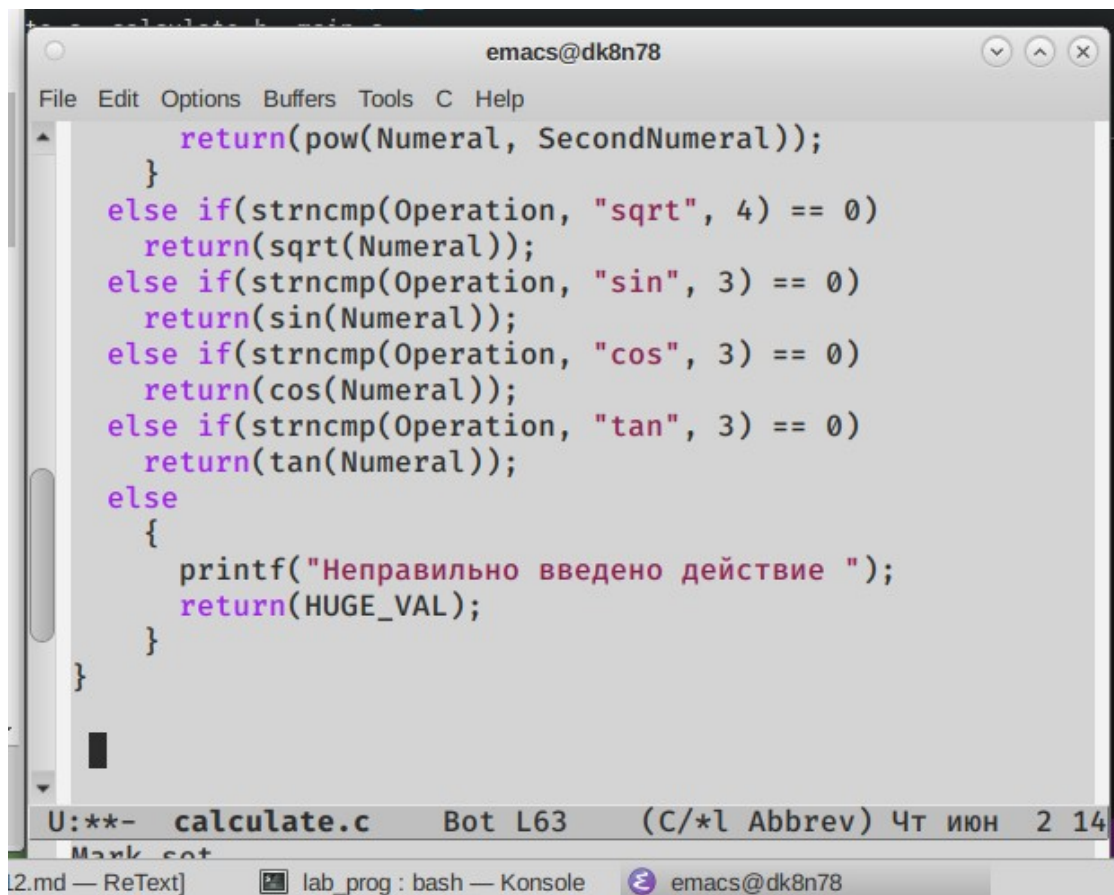
```
emacs@dk8n78
File Edit Options Buffers Tools C Help
////////////////////////////////////
// calculate.c

#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) == 1)
    {
        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));
    }
}

U:***- calculate.c Top L46 (C/*l Abbrev) Чт июн 2 14:52 0.81
```

рис.2



```
emacs@dk8n78
File Edit Options Buffers Tools C Help
return(pow(Numeral, SecondNumeral));
}
else if(strncmp(Operation, "sqrt", 4) == 0)
return(sqrt(Numeral));
else if(strncmp(Operation, "sin", 3) == 0)
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);
}
}
U:**~ calculate.c Bot L63 (C/*l Abbrev) ЧТ июн 2 14
Mark set
lab_prog : bash — Konsole emacs@dk8n78
```

рис.3

```
emacs@dk8n78
File Edit Options Buffers Tools C Help
////////////////////////////////////
// calculate.h

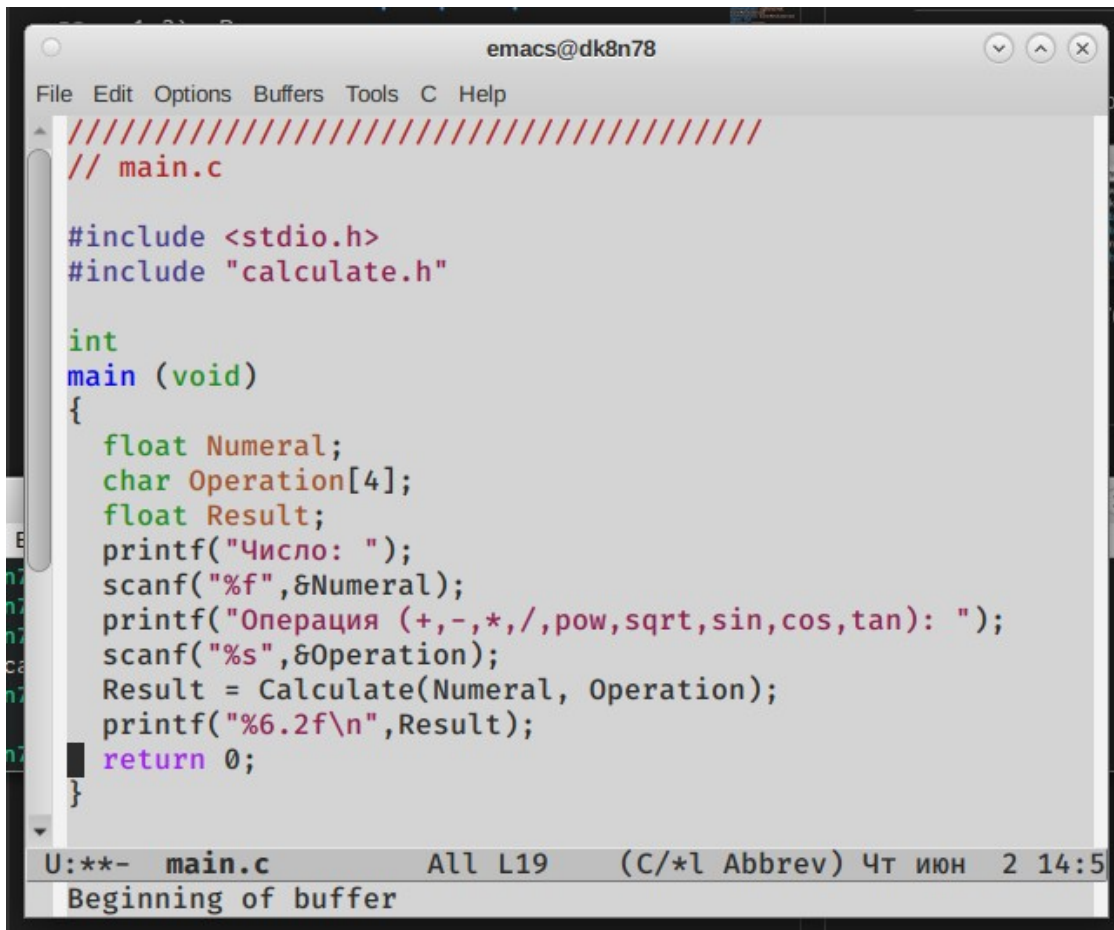
#ifndef CALCULATE_H_
#define CALCULATE_H_

float Calculate(float Numeral, char Operation[4]);

#endif /*CALCULATE_H_*/

U:--- calculate.h All L9 (C/*l Abbrev) ЧТ июн 2 14
Wrote /afs/.dk.sci.pfu.edu.ru/home/v/o/voshishuk/work/os/
slab_prog/calculate.h
```

puc.4



```
emacs@dk8n78
File Edit Options Buffers Tools C Help

////////////////////////////////////
// main.c

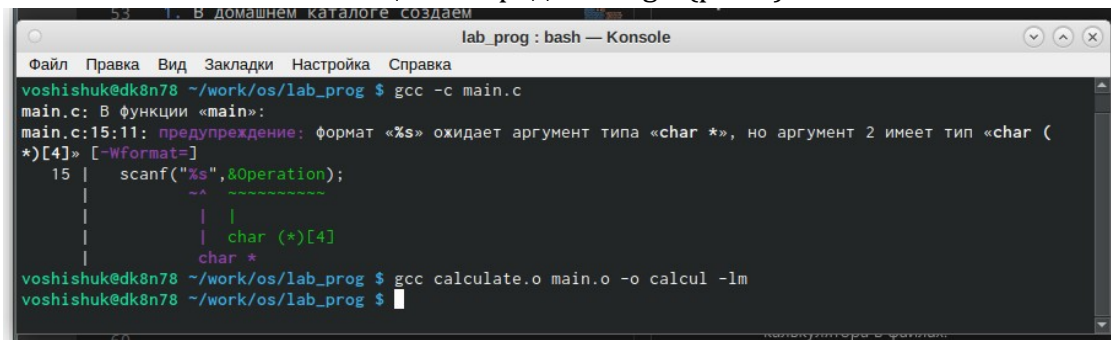
#include <stdio.h>
#include "calculate.h"

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;
}

U:***- main.c All L19 (C/*l Abbrev) Чт июн 2 14:5
Beginning of buffer
```

рис.5

3. Выполнили компиляцию посредством gcc(рис.6):

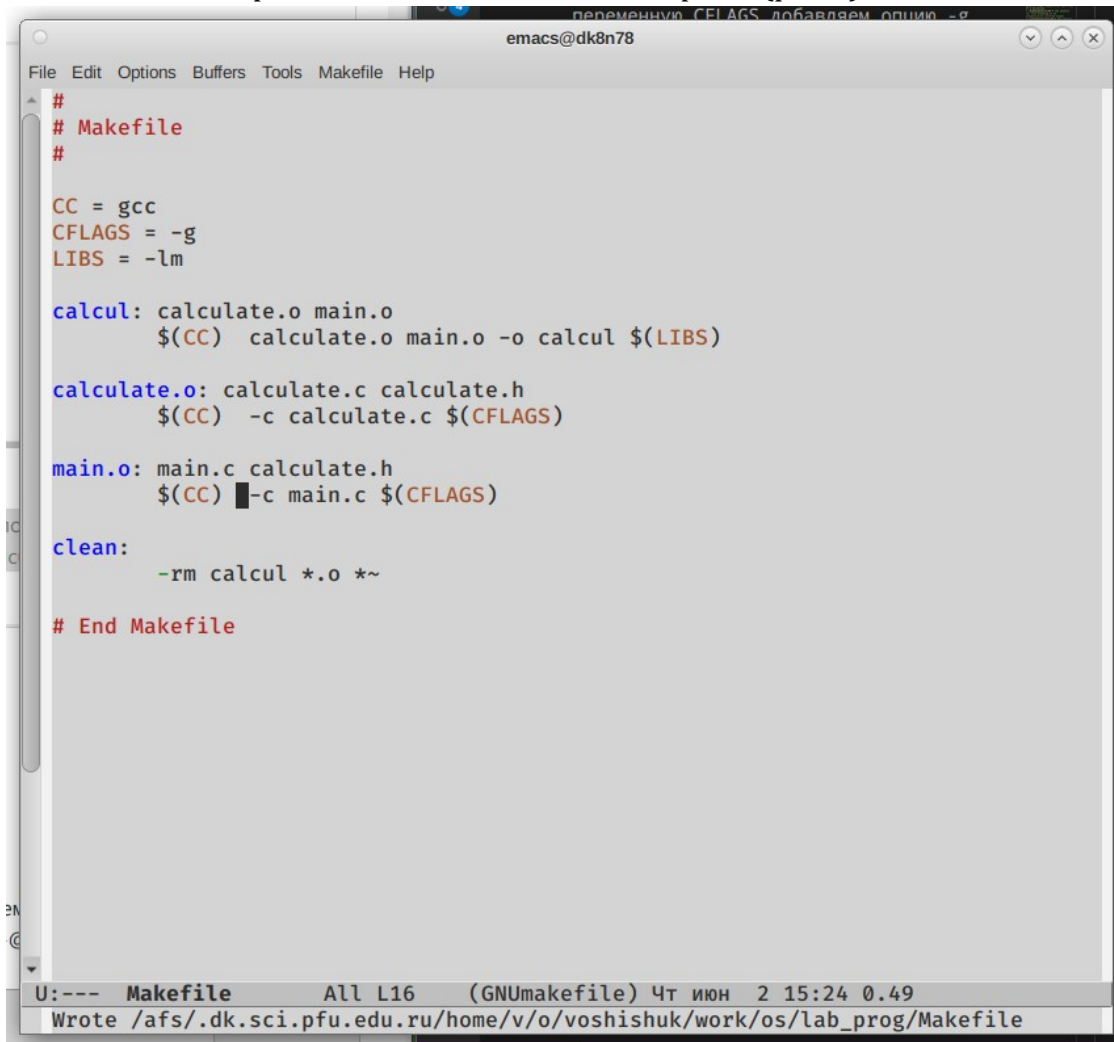


```
lab_prog : bash — Konsole
Файл Правка Вид Закладки Настройка Справка
voshishuk@dk8n78 ~/work/os/lab_prog $ gcc -c main.c
main.c: В функции «main»:
main.c:15:11: предупреждение: формат «%s» ожидает аргумент типа «char *», но аргумент 2 имеет тип «char (
*)[4]» [-Wformat=]
   15 |     scanf("%s",&Operation);
      |           ^~
      |           |
      |           | char (*)[4]
      |           |
      |           | char *
voshishuk@dk8n78 ~/work/os/lab_prog $ gcc calculate.o main.o -o calcul -lm
voshishuk@dk8n78 ~/work/os/lab_prog $
```

рис.6

4. Ошибок не возникло.
5. Создаем Makefile с нужным содержанием. Данный файл необходим для автоматической компиляции файлов calculate.c (цель calculate.o), main.c (цель main.o), а также их объединения в один исполняемый файл calcul (цель calcul). Цель clean нужна для автоматического удаления файлов. Переменная CC отвечает за утилиту для компиляции. Переменная CFLAGS отвечает за опции в

данной утилите. Переменная LIBS отвечает за опции для объединения объектных файлов в один исполняемый файл. (рис.7)



```
#
# Makefile
#

CC = gcc
CFLAGS = -g
LIBS = -lm

calcul: calculate.o main.o
    $(CC) calculate.o main.o -o calcul $(LIBS)

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

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

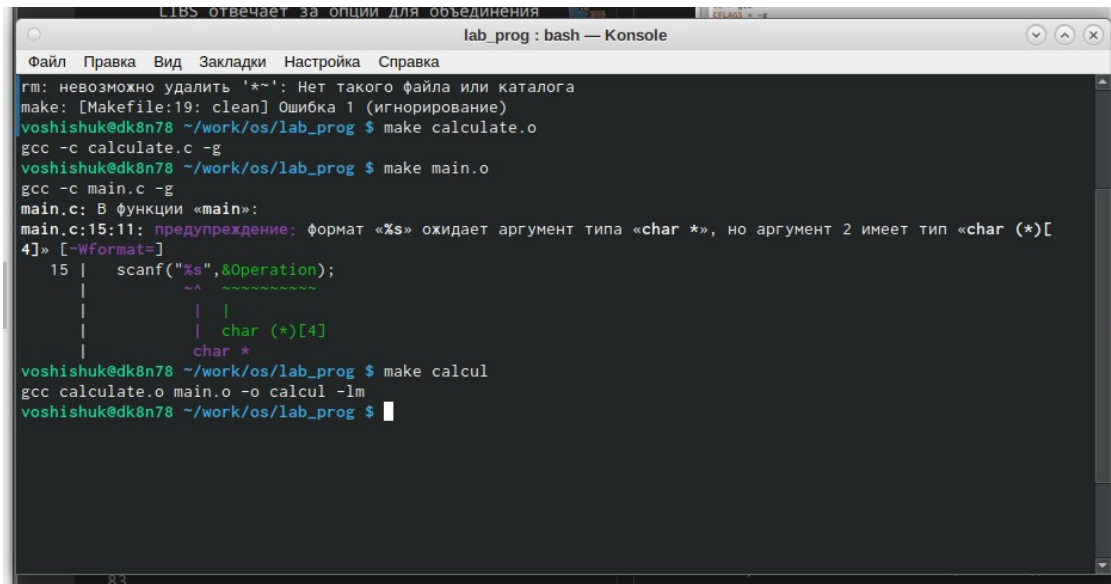
clean:
    -rm calcul *.o *~

# End Makefile
```

U:--- Makefile All L16 (GNUmakefile) Чт июн 2 15:24 0.49
Wrote /afs/.dk.sci.pfu.edu.ru/home/v/o/voshishuk/work/os/lab_prog/Makefile

рис.7

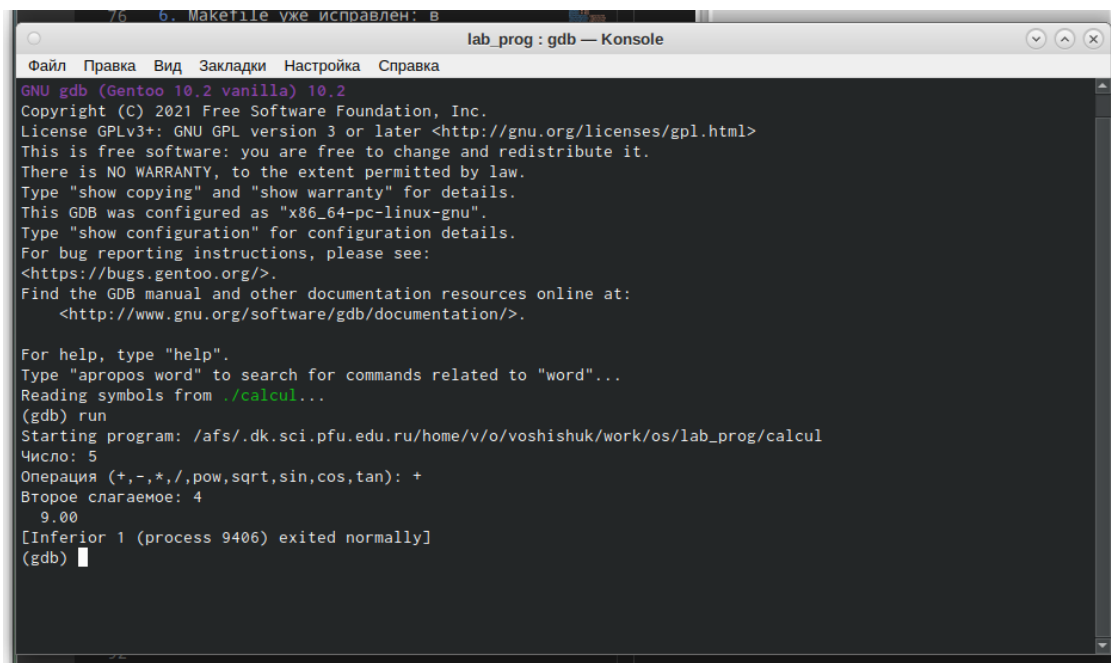
6. Makefile уже исправлен: в переменную CFLAGS добавляем опцию -g, необходимую для компиляции объектных файлов и их использования в программе отладчика GDB; делаем так, что утилита компиляции выбирается с помощью переменной CC. Выполняем компиляцию файлов, используя команды «make calculate.o», «make main.o», «make calcul».(рис.8)



```
LIBS отвечает за опции для объединения
lab_prog : bash — Konsole
Файл Правка Вид Закладки Настройка Справка
rm: невозможно удалить '*~': Нет такого файла или каталога
make: [Makefile:19: clean] Ошибка 1 (игнорирование)
voshishuk@dk8n78 ~/work/os/lab_prog $ make calculate.o
gcc -c calculate.c -g
voshishuk@dk8n78 ~/work/os/lab_prog $ make main.o
gcc -c main.c -g
main.c: В функции «main»:
main.c:15:11: предупреждение: формат «%s» ожидает аргумент типа «char *», но аргумент 2 имеет тип «char (*)[4]» [-Wformat=]
   15 |     scanf("%s", &Operation);
       |           ^~
       |           |
       |           | char (*)[4]
       |           |
       |           | char *
voshishuk@dk8n78 ~/work/os/lab_prog $ make calcul
gcc calculate.o main.o -o calcul -lm
voshishuk@dk8n78 ~/work/os/lab_prog $
```

рис.8

- Запустили отладчик GDB, загрузив в него программу для отладки
- Для запуска программы внутри отладчика ввели команду run(рис.9)



```
lab_prog : gdb — Konsole
Файл Правка Вид Закладки Настройка Справка
GNU gdb (Gentoo 10.2 vanilla) 10.2
Copyright (C) 2021 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
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-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://bugs.gentoo.org/>.
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
Starting program: /afs/.dk.sci.pfu.edu.ru/home/v/o/voshishuk/work/os/lab_prog/calcul
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): +
Второе слагаемое: 4
9.00
[Inferior 1 (process 9406) exited normally]
(gdb)
```

Для постраничного (по 9 строк) просмотра исходного код использовали команду list
- Для просмотра строк с 12 по 15 основного файла использовали list с параметрами(рис.10)

```
переменную CFLAGS добавляем опцию -g,
lab_prog : gdb — Konsole
Файл  Правка  Вид  Закладки  Настройка  Справка

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./calcul...
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/v/o/voshishuk/work/os/lab_prog/calcul
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): +
Второе слагаемое: 4
9.00
[Inferior 1 (process 9406) exited normally]
(gdb) list
1  // main.c
2
3
4  #include <stdio.h>
5  #include "calculate.h"
6
7  int main (void)
8  {
9      float Numeral;
10     char Operation[4];
(gdb) list 12,15
12     printf("Число: ");
13     scanf("%f",&Numeral);
14     printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): ");
15     scanf("%s",&Operation);
(gdb) █
```

рис.10

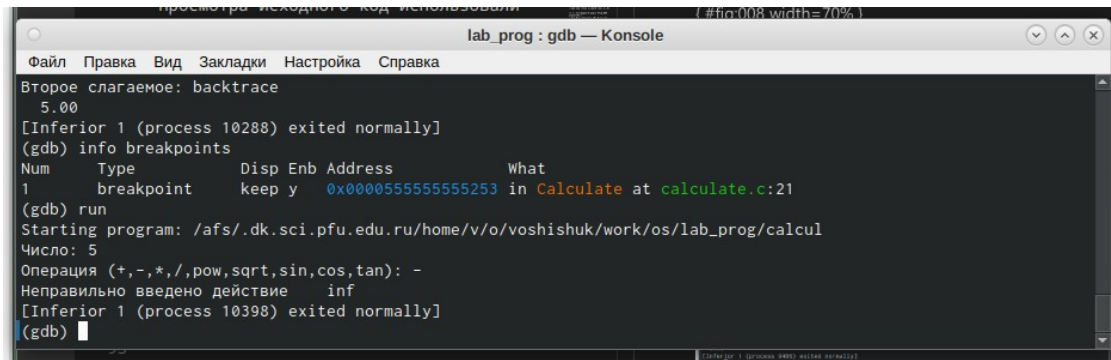
- Для просмотра определённых строк не основного файла использовали list с параметрами
- Установили точку останова в файле calculate.c на строке номер 21(рис.11)

```
переменную CFLAGS добавляем опцию -g,
lab_prog : gdb — Konsole
Файл  Правка  Вид  Закладки  Настройка  Справка

2  // main.c
3
4  #include <stdio.h>
5  #include "calculate.h"
6
7  int main (void)
8  {
9      float Numeral;
10     char Operation[4];
(gdb) list 12,15
12     printf("Число: ");
13     scanf("%f",&Numeral);
14     printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): ");
15     scanf("%s",&Operation);
(gdb) list calculate.c:20,29
20     printf("Вычитаемое: ");
21     scanf("%f",&SecondNumeral);
22     return(Numeral - SecondNumeral);
23 }
24 else if(strncmp(Operation, "*", 1) == 0)
25 {
26     printf("Множитель: ");
27     scanf("%f",&SecondNumeral);
28     return(Numeral * SecondNumeral);
29 }
(gdb) break 21
Breakpoint 1 at 0x55555555253: file calculate.c, line 21.
(gdb) █
```

рис.11

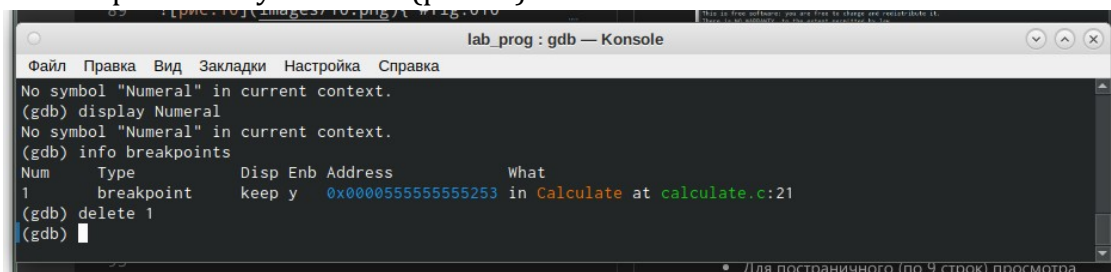
- Вывели информацию об имеющихся в проекте точка останова
- Запустили программу внутри отладчика и убедились, что программа остановится в момент прохождения точки останова:



```
просмотр исходного кода использован (#fn-008 width=70%)
lab_prog : gdb — Konsole
Файл Правка Вид Закладки Настройка Справка
Второе слагаемое: backtrace
5.00
[Inferior 1 (process 10288) exited normally]
(gdb) info breakpoints
Num Type Disp Enb Address What
1 breakpoint keep y 0x0000555555555253 in Calculate at calculate.c:21
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/v/o/voshishuk/work/os/lab_prog/calcul
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): -
Неправильно введено действие inf
[Inferior 1 (process 10398) exited normally]
(gdb)
```

рис.12

- Убрали точку останова(рис.13)



```
lab_prog : gdb — Konsole
Файл Правка Вид Закладки Настройка Справка
No symbol "Numeral" in current context.
(gdb) display Numeral
No symbol "Numeral" in current context.
(gdb) info breakpoints
Num Type Disp Enb Address What
1 breakpoint keep y 0x0000555555555253 in Calculate at calculate.c:21
(gdb) delete 1
(gdb)
```

рис.13

7. С помощью утилиты splint анализируем коды файлов calculate.c и main.c. Предварительно устанавливаем данную утилиту с помощью команды «yum install splint». Далее используем команду «splint calculate.c» и «splint main.c». С помощью утилиты splint выяснилось, что в файлах calculate.c и main.c присутствует функция чтения scanf, возвращающая целое число (тип int), но эти числа не используются и нигде не сохраняются. Утилита вывела предупреждение о том, что в файле calculate.c происходит сравнение вещественного числа с нулем. Также возвращаемые значения (тип double) в функциях pow, sqrt, sin, cos и tan записываются в переменную типа float, что свидетельствует о потере данных.(рис.14-15)

```
lab_prog : bash — Konsole
Файл Правка Вид Закладки Настройка Справка
(gdb) delete 1
(gdb) q
voshishuk@dk8n78 ~/work/os/lab_prog $ splint calculate.c
Splint 3.1.2 --- 13 Jan 2021

calculate.h:7:37: Function parameter Operation declared as manifest array (size
        constant is meaningless)
    A formal parameter is declared as an array with size. The size of the array
    is ignored in this context, since the array formal parameter is treated as a
    pointer. (Use -fixedformalarray to inhibit warning)
calculate.c:9:37: Function parameter Operation declared as manifest array (size
        constant is meaningless)
calculate.c: (in function Calculate)
calculate.c:15:7: Return value (type int) ignored: scanf("%f", &Sec...
    Result returned by function call is not used. If this is intended, can cast
    result to (void) to eliminate message. (Use -retvalint to inhibit warning)
calculate.c:21:7: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:27:7: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:33:7: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:34:10: Dangerous equality comparison involving float types:
    SecondNumeral == 0
    Two real (float, double, or long double) values are compared directly using
    == or != primitive. This may produce unexpected results since floating point
    representations are inexact. Instead, compare the difference to FLT_EPSILON
    or DBL_EPSILON. (Use -realcompare to inhibit warning)
calculate.c:37:17: Return value type double does not match declared type float:
```

рис.14

```
lab_prog : bash — Konsole
Файл Правка Вид Закладки Настройка Справка
(tan(Numeral))
calculate.c:59:13: Return value type double does not match declared type float:
    (HUGE_VAL)

Finished checking --- 15 code warnings
voshishuk@dk8n78 ~/work/os/lab_prog $ splint main.c
Splint 3.1.2 --- 13 Jan 2021

calculate.h:7:37: Function parameter Operation declared as manifest array (size
        constant is meaningless)
    A formal parameter is declared as an array with size. The size of the array
    is ignored in this context, since the array formal parameter is treated as a
    pointer. (Use -fixedformalarray to inhibit warning)
main.c: (in function main)
main.c:13:3: Return value (type int) ignored: scanf("%f", &Num...
    Result returned by function call is not used. If this is intended, can cast
    result to (void) to eliminate message. (Use -retvalint to inhibit warning)
main.c:15:14: Format argument 1 to scanf (%s) expects char * gets char [4] *:
    &Operation
    Type of parameter is not consistent with corresponding code in format string.
    (Use -formattype to inhibit warning)
main.c:15:11: Corresponding format code
main.c:15:3: Return value (type int) ignored: scanf("%s", &Op...)

Finished checking --- 4 code warnings
voshishuk@dk8n78 ~/work/os/lab_prog $
```

рис.15

Выводы

Я приобрел простейшие навыки разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linux на примере создания калькулятора с простейшими функциями.

Контрольные вопросы

1. Чтобы получить информацию о возможностях программ `gcc`, `make`, `gdb` и др. нужно воспользоваться командой `man` или опцией `-help (-h)` для каждой команды.
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: о кодирование – по сути создание исходного текста программы (возможно в нескольких вариантах); – анализ разработанного кода; о сборка, компиляция и разработка исполняемого модуля; о тестирование и отладка, сохранение произведённых изменений;
 - документирование. Для создания исходного текста программы разработчик может воспользоваться любым удобным для него редактором текста: `vi`, `vim`, `msceditor`, `emacs`, `geany` и др. После завершения написания исходного кода программы (возможно состоящей из нескольких файлов), необходимо её скомпилировать и получить исполняемый модуль.
3. Для имени входного файла суффикс определяет какая компиляция требуется. Суффиксы указывают на тип объекта. 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. Например, в команде «`gcc -c main.c`»: `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`».
4. Основное назначение компилятора языка Си в UNIX заключается в компиляции всей программы и получении исполняемого файла/модуля.
5. 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.
6. 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. В самом простом случае `Makefile` имеет следующий синтаксис: `... : ... <команда>`
`1> ...` 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. Общий синтаксис Makefile имеет вид: target1 [target2...]:[:] [dependment1...] [(tab)commands] [#commentary] [(tab)commands] [#commentary] Здесь знак # определяет начало комментария (содержимое от знака # и до конца строки не будет обрабатываться. Одинарное двоеточие указывает на то, что последовательность команд должна содержаться в одной строке. Для переноса можно в длинной строке команд можно использовать обратный слэш \. Двойное двоеточие указывает на то, что последовательность команд может содержаться в нескольких последовательных строках. Пример более сложного синтаксиса Makefile: