

Отчет по лабораторной работе №14

Тулеева Валерия, НБИбд-01-20

РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук Кафедра прикладной
информатики и теории вероятностей

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

2 Оглавление:

1. Введение:
 - a) Цель работы
2. Описание результатов выполнения задания;
3. Библиография;
4. Вывод;
5. Контрольные вопросы.

3 Введение:

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 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. Источник

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.

Источник

4 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.

5 Описание результатов выполнения

Задания:

1. В домашнем каталоге создайте подкаталог `~/work/lab_prog`(рис 1.1 и 1.2):

```
cd ~/work
```

```
mkdir lab_prog
```

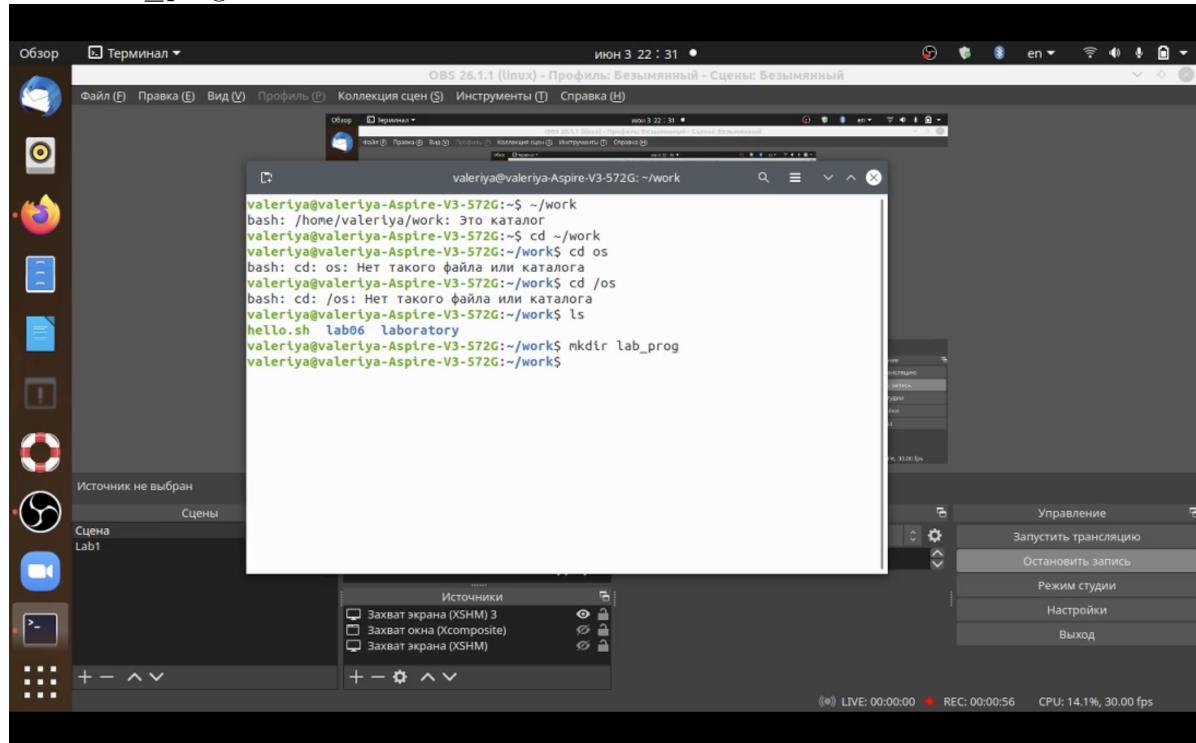


Рис 1.1. Создание подкаталога

2. Создала в нём файлы: calculate.h, calculate.c, main.c (рис 2.1 и 2.2):

```
cd lab_prog
```

```
touch calculate.h
```

```
touch calculate.c
```

```
touch main.c
```

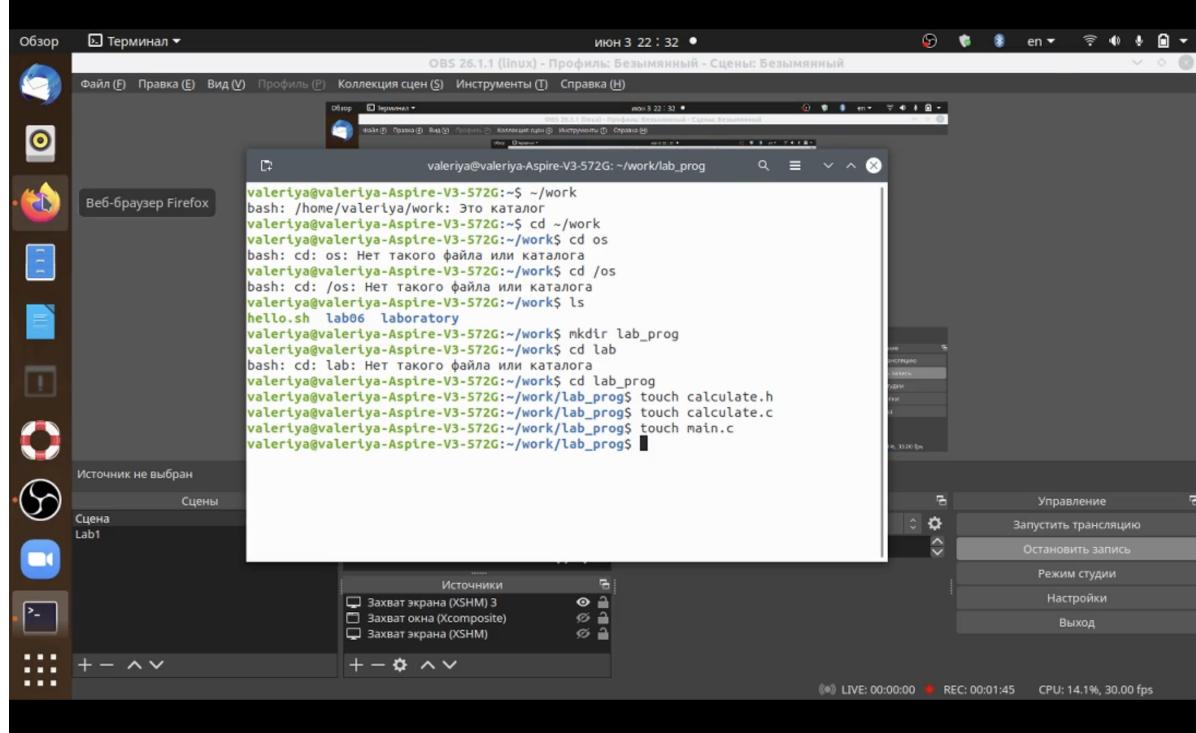


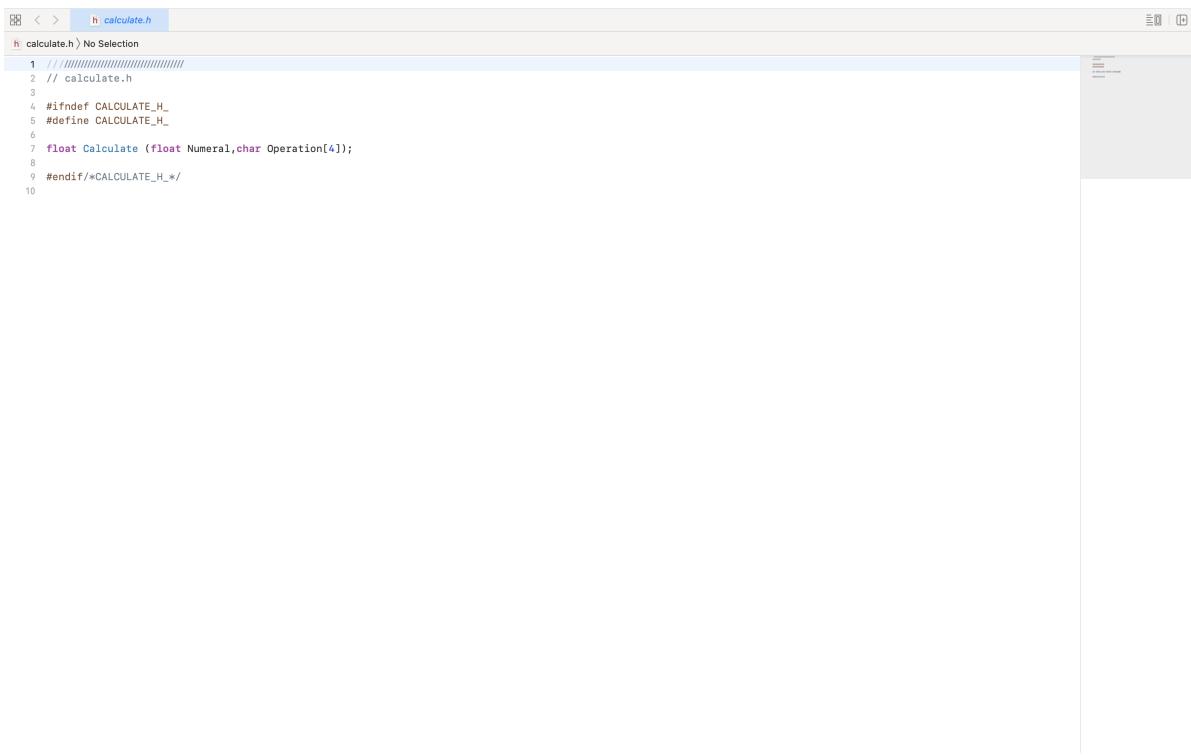
Рис 2.1. Создание файлов

3. Реализовала функций калькулятора в файле calculate.c(рис 3.1)

```
1 //////////////////////////////////////////////////////////////////
2 // calculate.c
3
4 #include <stdio.h>
5 #include <math.h>
6 #include <string.h>
7 #include "calculate.h"
8
9 float
10 Calculate (float Numeral,char Operation[4])
11 {
12     float SecondNumeral;
13     if(strcmp(Operation, "+", 1) == 0)
14     {
15         printf("Второе слагаемое:");
16         scanf("%f",&SecondNumeral);
17         return(Numeral+SecondNumeral);
18     }
19     else if(strcmp(Operation, "-",1) == 0)
20     {
21         printf("Вычитаемое: ");
22         scanf("%f",&SecondNumeral);
23         return(Numeral-SecondNumeral);
24     }
25     else if(strcmp(Operation,"*",1) == 0)
26     {
27         printf("Множитель: ");
28         scanf("%f",&SecondNumeral);
29         return(Numeral*SecondNumeral);
30     }
31     else if(strcmp(Operation,"/",1) == 0)
32     {
33         printf("Делитель: ");
34         scanf("%f",&SecondNumeral);
35         if(SecondNumeral==0)
36         {
37             printf("Ошибка: деление на ноль! ");
38             return(HUGE_VAL);
39         }
40         else
41             return(Numeral / SecondNumeral);
42     }
43     else if(strcmp(Operation,"pow",3) == 0)
44     {
45         printf("Степень: ");
46         scanf("%f",&SecondNumeral);
47         return(pow(Numeral, SecondNumeral));
48     }
49     else if(strcmp(Operation,"sqrt",4) == 0)
50         return(sqrt(Numeral));
51 }
```

Рис 3.1. Реализация

4. Интерфейсный файл calculate.h, описывающий формат вызова функции- калькулятора (рис 4.1):



The screenshot shows a code editor window with the tab bar at the top labeled "h calculate.h". Below the tab bar, a status bar displays "calculate.h No Selection". The main area of the editor contains the following C header file code:

```
1 //////////////////////////////////////////////////////////////////
2 // calculate.h
3
4 #ifndef CALCULATE_H_
5 #define CALCULATE_H_
6
7 float Calculate (float Numeral,char Operation[4]);
8
9 #endif/*CALCULATE_H_*/
```

Рис 4.1. Интерфейсный файл

5. Основной файл main.c, реализующий интерфейс пользователя к калькулятору:
(рис 5.1)

Рис 5.1. Основной файл

6. Выполнила компиляцию программы посредством gcc(рис 6.1):

```
gcc -c calculate.c
```

```
gcc -c main.c
```

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

```
valeriya@valeriya-Aspire-V3-572G:~/work/lab_prog$ gcc -c calculate.c
valeriya@valeriya-Aspire-V3-572G:~/work/lab_prog$ gcc -c main.c
main.c: In function 'main':
main.c:16:11: warning: format '%s' expects argument of type 'char *', but argument
ment 2 has type 'char (*)[4]' [-Wformat=]
  16 |     scanf("%s", &operation);
      |           ~^ ~~~~~
      |           |
      |           | char (*[4])
      |           char *
valeriya@valeriya-Aspire-V3-572G:~/work/lab_prog$ gcc calculate.o main.o -o calc
ul -lm
valeriya@valeriya-Aspire-V3-572G:~/work/lab_prog$ █
```

Рис 6.1. Компиляция

7. Создала Makefile: (рис 7.1)

```
touch Makefile
ul -lm
valeriya@valeriya-Aspire-V3-572G:~/work/lab_prog$ touch Makefile
valeriya@valeriya-Aspire-V3-572G:~/work/lab_prog$
```

```
1 #
2 # Makefile
3 #
4
5 CC = gcc
6 CFLAGS =
7 LIBS = -lm
8
9 calcul: calculate.o main.o
10      gcc calculate.o main.o -o calcul $(LIBS)
11
12 calculate.o: calculate.c calculate.h
13      gcc -c calculate.c $(CFLAGS)
14
15 main.o: main.c calculate.h
16      gcc -c main.c $(CFLAGS)
17
18 clean:
19      -rm calcul *.o *~
20
21 # End Makefile
```

Рис 7.1. Makefile

Код:

#Makefile

CC = gcc (Переменные)

CFLAGS =

LIBS = -lm

calcul: calculate.o main.o (calcul - цель, calculate.o & main.o - названия файлов, которые мы хотим скомпилировать)

gcc calculate.o main.o -o calcul \$(LIBS) (команда компиляции gcc с опциями)

calculate.o: calculate.c calculate.h (calculate.o - цель, calculate.c & calculate.h - названия файлов, которые мы хотим скомпилировать)

gcc -c calculate.c \$(CFLAGS) (команда компиляции gcc с опциями)

main.o: main.c calculate.h (main.o - цель, main.c & calculate.h - названия файлов,

которые мы хотим скомпилировать)

gcc -c main.c \$(CFLAGS) (команда компиляции gcc с опциями)

clean: (Цель с именем clean производит очистку каталога от файлов, полученных в результате компиляции)

```
-rm calcul *.o *
```

```
#End Makefile
```

8. С помощью gdb выполните отладку программы calcul:

- Запустила отладчик GDB, загрузив в него программу для отладки (рис 8.1):

```
gdb ./calcul
```

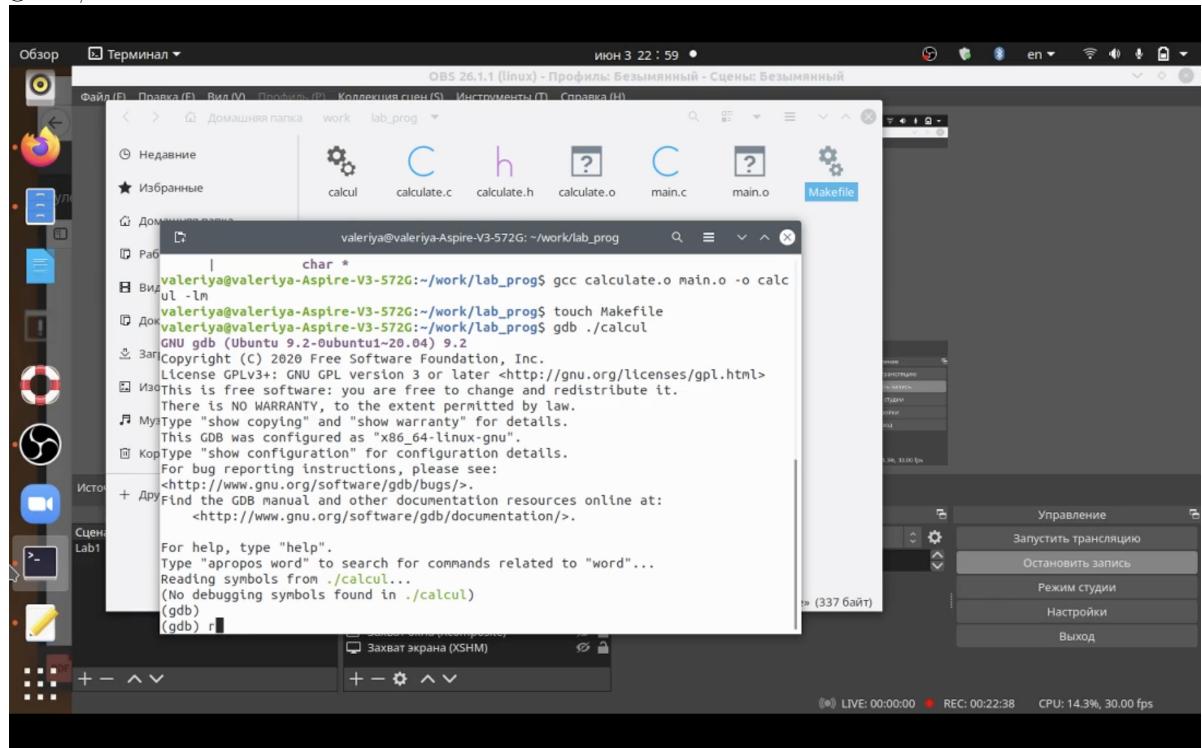


Рис 8.1. Запуск отладчика

- Для запуска программы внутри отладчика ввела команду run (рис 8.2):

```
run
```

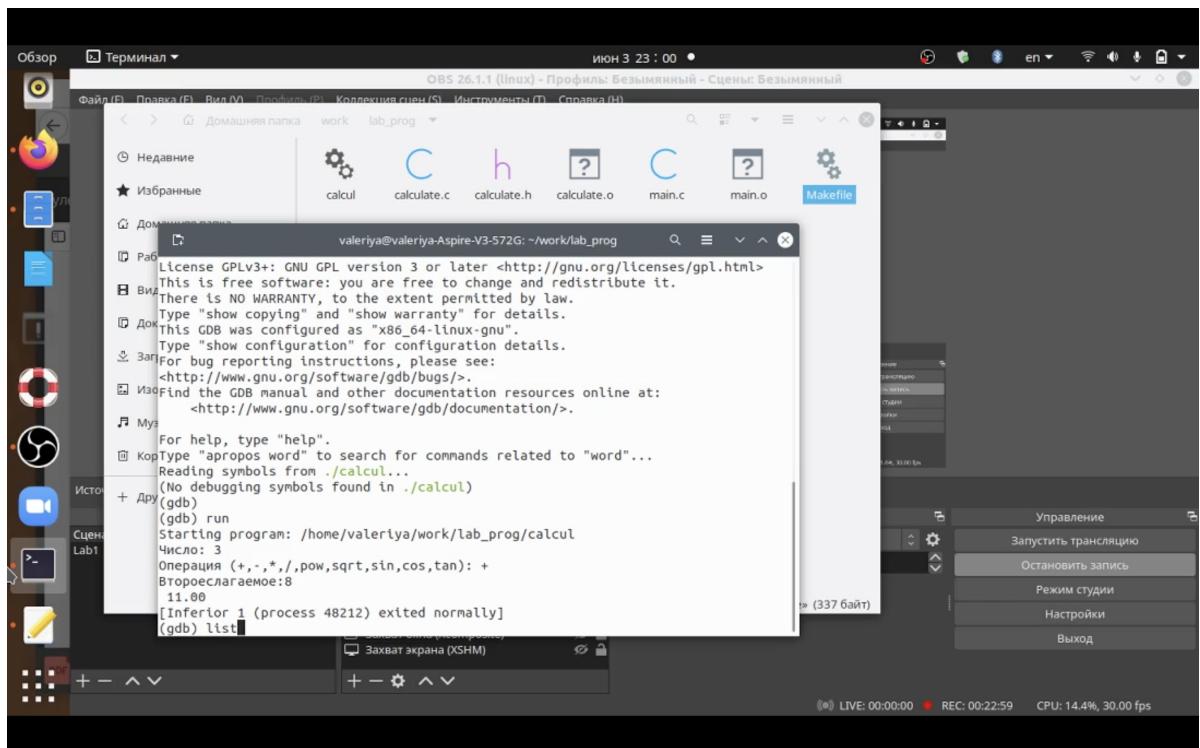


Рис 8.2. Запуск программы

- Для постраничного (по 9 строк) просмотра исходного кода использовала команду list (рис 8.3):

```
list
(gdb) list
Д1      in init-first.c
```

Рис 8.3. Просмотр кода

- Установила точку останова на строке номер 21 (рис 8.4):

```
break 21
(gdb) break 21
Breakpoint 1 at 0x7ffff7c8ef20: file init-first.c, line 44.
```

Рис 8.4. Точка останова

- Выведите информацию об имеющихся в проекте точка останова (рис 8.5):

info breakpoints

```
(gdb) info breakpoints
Num      Type            Disp Enb Address          What
1       breakpoint     keep y  0x00007ffff7c8ef20 in __libc_init_first
                                         at init-first.c:44
```

Рис 8.5. Вывод информации

- Посмотрела, чему равно на этом этапе значение переменной Numeral, введя:

```
print Numeral
```

На экране было выведено число 5.

- Убрала точки останова (рис 8.6):

```
info breakpoints
```

```
delete 1
```

```
(gdb) info breakpoints
Num      Type            Disp Enb Address          What
1       breakpoint     keep y  0x00007ffff7c8ef20 in __libc_init_first
                                         at init-first.c:44
(gdb) delete 1
(gdb) break 21
No line 21 in the current file.
Make breakpoint pending on future shared library load? (y or [n]) y
Breakpoint 2 (21) pending.
```

Рис 8.6. Удаление точки отсanova

9. С помощью утилиты `splint` попробуйте проанализировать коды файлов `calculate.c` и `main.c` (рис 9.1 и 9.2):

```
sudo apt install splint (установка утилиты splint) (рис 9.0):
```

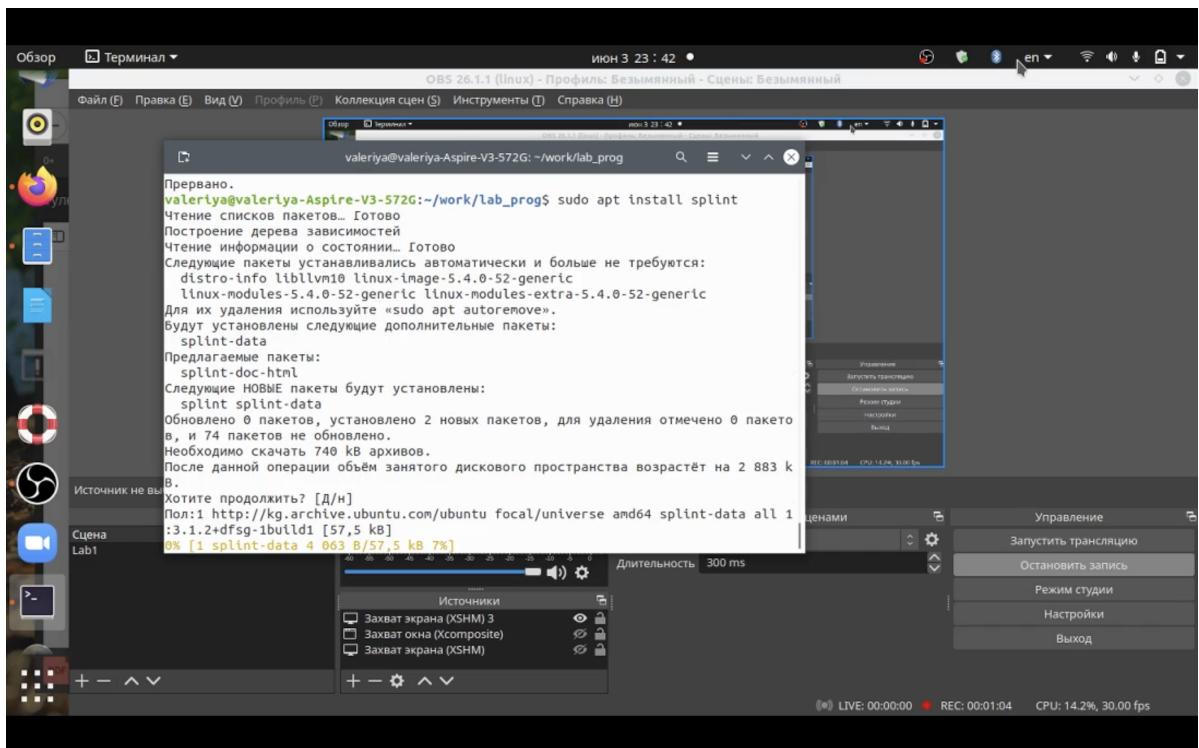
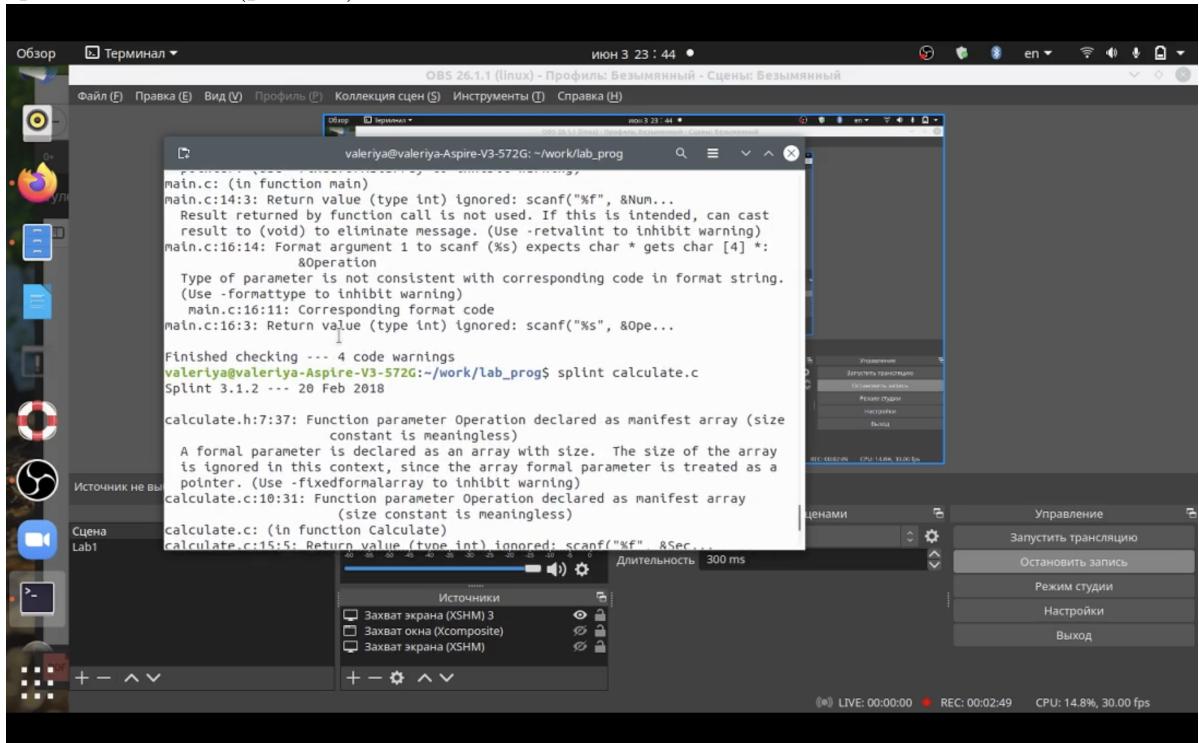


Рис 9.0. Установка

splint calculate.c(рис 9.1):



```

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:37:15: Return value type double does not match declared type float:
(HUGE_VAL)
To allow all numeric types to match, use +relaxtypes.
calculate.c:45:5: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:46:11: Return value type double does not match declared type float:
(pow(Numeral, SecondNumeral))
calculate.c:49:11: Return value type double does not match declared type float:
(sqrt(Numeral))
calculate.c:51:11: Return value type double does not match declared type float:
(sin(Numeral))
calculate.c:53:11: Return value type double does not match declared type float:
(cos(Numeral))
calculate.c:55:11: Return value type double does not match declared type float:
(tan(Numeral))
calculate.c:59:13: Return value type double does not match declared type float:
(HUGE_VAL)

Finished checking --- 15 code warnings

```

Рис 9.1. Код calculate.c

splint main.c(рис 9.2):

```

(HUGE_VAL)

Finished checking --- 15 code warnings
valeriya@valeriya-Aspire-V3-572G:~/work/lab_prog$ splint main.c
Splint 3.1.2 --- 20 Feb 2018

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:14:3: 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:16:14: 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:16:ii: Corresponding format code
main.c:16:3: Return value (type int) ignored: scanf("%s", &Op...
    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)

Finished checking --- 4 code warnings

```

Рис 9.2. Код main.c

6 Библиография:

1. Источник 1

7 Вывод:

В данной лабораторной работе, я приобрела простейшие навыки разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linux на примере создания на языке программирования С калькулятора с простейшими функциями.

8 Контрольные вопросы (лабораторная работа №14)

1. Как получить информацию о возможностях программ gcc, make, gdb и др.?

Дополнительную информацию о этих программах можно получить с помощью функций info и man.

2. Назовите и дайте краткую характеристику основным этапам разработки приложений в UNIX.

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 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. Что такое суффикс в контексте языка программирования? Приведите примеры

использования.

Файлы с расширением (суффиксом) .с воспринимаются gcc как программы на языке С, файлы с расширением .cc или .C — как файлы на языке C++, а файлы с расширением .o считаются объектными.

Таким образом, gcc по расширению (суффиксу) .c распознает тип файла для компиляции и формирует объектный модуль — файл с расширением .o.

4. Каково основное назначение компилятора языка С в UNIX?

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.

5. Для чего предназначена утилита make?

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.

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

В самом простом случае Makefile имеет следующий синтаксис:

... : ...

<команда 1>

...

<команда n>

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.

7. Назовите основное свойство, присущее всем программам отладки. Что необходимо сделать, чтобы его можно было использовать?

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:

```
gcc -c file.c -g
```

После этого для начала работы с gdb необходимо в командной строке ввести одноимённую команду, указав в качестве аргумента анализируемый бинарный файл: `gdb file.o`

8. Назовите и дайте основную характеристику основным командам отладчика gdb.

Некоторые команды gdb

Команда	Описание действия
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

Для выхода из gdb можно воспользоваться командой `quit` (или её сокращённым вариантом `q`) или комбинацией клавиш `Ctrl-d`. Более подробную информацию по работе с gdb можно получить с помощью команд `gdb -h` и `man gdb`.

9. Опишите по шагам схему отладки программы, которую Вы использовали при выполнении лабораторной работы.

- Выполнила компиляцию программы посредством gcc:

```
gcc -c calculate.c
```

```
gcc -c main.c
```

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

- При необходимости исправила синтаксические ошибки.

- Создала Makefile со следующим содержанием:

```
#Makefile
CC = gcc
CFLAGS =
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
```

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

Выводит информацию об ошибках.

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

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;
- lint — критическая проверка программ, написанных на языке Си.

12. Каковы основные задачи, решаемые программой splint?

Ещё одним средством проверки исходных кодов программ, написанных на языке С, 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 С 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.

Для анализа кода программы следует выполнить следующую команду:

```
splint example.c
В результате на экран будут выведены следующие пять предупреждений Splint 3.1.2 — 03 May 2009 example.c: (in function main) example.c:10:18:
Function sum expects arg 1 to be float gets int: x To allow all numeric types to match,
use +relaxtypes. example.c:10:21: Function sum expects arg 2 to be float gets int: y
example.c:10:2: Assignment of int to float: z = x + y + sum(x, y) example.c:11:9: Return
value type float does not match declared type int: z Finished checking — 4 code warnings
```

Первые два предупреждения относятся к функции `sum`, которая определена для двух аргументов вещественного типа, но при вызове ей передаются два аргумента `x` и `y` целого типа. Третье предупреждение относится к присвоению вещественной переменной `z` значения целого типа, которое получается в результате суммирования `x + y + sum(x, y)`. Далее вещественное число `z` возвращается в качестве результата работы функции `main`, хотя данная функция объявлена как функция, возвращающая целое значение.