РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 14

дисциплина: Операционные системы

Студент:

Гаглоев Олег Мелорович

Преподаватель:

Велиева Т.В.

Группа: НПИбд-01-20

```
                        МОСКВА 2021 г.
```

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 работы:

1) Создал каталог и три необходимых файла. Проделал это с помощью команды mkdir(каталог) и emacs(файлы)

```
om-gagloev@omgagloev:~$ cd work
om-gagloev@omgagloev:~/work$ cd 2020-2021
om-gagloev@omgagloev:~/work/2020-2021$ ls
'Операционные системы'
om-gagloev@omgagloev:~/work/2020-2021$ cd 'Операционные системы'
om-gagloev@omgagloev:~/work/2020-2021/Операционные системы$ mkdir lab_prog
om-gagloev@omgagloev:~/work/2020-2021/Операционные системы$ cd lab_prog
om-gagloev@omgagloev:~/work/2020-2021/Операционные системы/lab_prog$ emacs calculate.h
om-gagloev@omgagloev:~/work/2020-2021/Операционные системы/lab_prog$ emacs calculate.c
om-gagloev@omgagloev:~/work/2020-2021/Операционные системы/lab_prog$ emacs main.c
```

2) Скопировал и изменил код, предоставленный в задании

```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)
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);}}
```

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

```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;
}
```

3) Исправил кучу синтаксических ошибок, после чего всё заработало и выполнил компиляцию программы посредством gcc, добавив опцию -g в противном случае были проблемы с работой gdb

```
om-gagloev@omgagloev:~/work/2020-2021/Операционные системы/lab_prog$ emacs main.c
om-gagloev@omgagloev:~/work/2020-2021/Операционные системы/lab_prog$ gcc -c -g main.c
om-gagloev@omgagloev:~/work/2020-2021/Операционные системы/lab_prog$ emacs main.c
om-gagloev@omgagloev:~/work/2020-2021/Операционные системы/lab_prog$ gcc -c -g calculate.com-gagloev@omgagl
ev:~/work/2020-2021/Операционные системы/lab_prog$ gcc -c -g main.c
om-gagloev@omgagloev:~/work/2020-2021/Операционные системы/lab_prog$ gcc calculate.o main.o -o calcul -lm
```

emacs@omgagloev

File Edit Options Buffers Tools Makefile Help

Save Undo

```
# 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
```

4) Создал Makefile 5) Далее с помощью gdb выполнил отладку программы ./calcul с помощью команды gdb ./calcul и для запуска программы внутри отладчика ввел команду run

```
om-gagloev@omgagloev:~/work/2020-2021/Операционные системы/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/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-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
Starting program: /home/om-gagloev/work/2020-2021/Операционные системы/lab_prog/calcul
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): -
Вычитаемое: 3
  2.00
[Inferior 1 (process 23122) exited normally]
(gdb)
```

6) Для постраничного (по 9 строк) просмотра исходного кода использовал команду list Для просмотра строк с 12 по 15 основного файла использовал list 12,15 Для просмотра определённых строк не основного файла использовал list calculate.c:20,29

```
(gdb) list
1       #include<stdio.h>
2       #include"calculate.h"
3       int
4       main (void)
5       {
6         float Numeral;
7         char Operation[4];
8         float Result;
9         printf("Число: ");
10        scanf("%f",&Numeral);
(gdb) list 12,15
12        scanf("%s", Operation);
13        Result = Calculate(Numeral, Operation);
14        printf("%6.2f\n",Result);
15        return 0;
(gdb) list calculate.c:20,29
20          {printf("Делитель: ");
21            scanf("%f",&SecondNumeral);
22            if(SecondNumeral==0)
23              {printf("Ошибка: деление на ноль! ");
24                return(HUGE_VAL);}
25            else return(Numeral/SecondNumeral);}
26        else if(strncmp(Operation,"pow",3)==0)
27          {printf("Степень: ");
28            scanf("%f",&SecondNumeral);
29            return(pow(Numeral, SecondNumeral));}
(gdb) list calculate.c:20,27
20          {printf("Делитель: ");
21            scanf("%f",&SecondNumeral);
22            if(SecondNumeral==0)
23              {printf("Ошибка: деление на ноль! ");
24                return(HUGE_VAL);}
25            else return(Numeral/SecondNumeral);}
26        else if(strncmp(Operation,"pow",3)==0)
27          {printf("Степень: ");
(gdb)
```

7) Установил точку останова в файле calculate.c на строке номер 13 break 13 и убедился в их наличии , выполнив программы и использовав команду info breakpoints

```
(gdb) break 13
Note: breakpoint 5 also set at pc 0x5555555552ee.
Breakpoint 6 at 0x5555555552ee: file calculate.c, line 13.
(gdb) info breakpoints
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x000055555555539a in Calculate at calculate.c:21
2       breakpoint     keep y   0x0000555555555389 in Calculate at calculate.c:20
3       breakpoint     keep y   0x000055555555536d in Calculate at calculate.c:19
4       breakpoint     keep y   0x000055555555539a in Calculate at calculate.c:21
5       breakpoint     keep y   0x00005555555552ee in Calculate at calculate.c:13
        breakpoint already hit 1 time
6       breakpoint     keep y   0x00005555555552ee in Calculate at calculate.c:13
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/om-gagloev/work/2020-2021/Операционные системы/lab_prog/calcul
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): -

Breakpoint 5, Calculate (Numeral=5, Operation=0x7fffffffdef4 "-") at calculate.c:13
13            scanf("%f",&SecondNumeral);
(gdb)
```

Пришлось немного понастаивть точек останова, не сразу дошло, что вычитание стоит на пару строк выше 8) Использовал команду backtrace и увидел, что она выводит

```
(gdb) backtrace
#0  Calculate (Numeral=5, Operation=0x7fffffffdef4 "-") at calculate.c:13
#1  0x00005555555555bd in main () at main.c:13
(gdb) print Numeral
$1 = 5
(gdb) display Numeral
1: Numeral = 5
(gdb) info breackpoints
Undefined info command: "breackpoints".  Try "help info".
(gdb) info breakpoints
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x000055555555539a in Calculate at calculate.c:21
2       breakpoint     keep y   0x0000555555555389 in Calculate at calculate.c:20
3       breakpoint     keep y   0x000055555555536d in Calculate at calculate.c:19
4       breakpoint     keep y   0x000055555555539a in Calculate at calculate.c:21
5       breakpoint     keep y   0x00005555555552ee in Calculate at calculate.c:13
        breakpoint already hit 1 time
6       breakpoint     keep y   0x00005555555552ee in Calculate at calculate.c:13
        breakpoint already hit 1 time
(gdb) delete 1
(gdb) delete 2
(gdb) delete 3
(gdb) delete 4
(gdb) delete 5
(gdb) delete 6
(gdb) info breakpoints
No breakpoints or watchpoints.
(gdb)
```

.Команда backtrace показал весь стек вызываемых функций от начала программы до текущего места. Далее посмотрел значение Numeral с помощью команды print Numeral после этого сравнил вывод с командой display . Видим, что значения одинаковы. Далее отображаю все точки и удаляю их с помощью команды delete n 9) Скачал утилиты splint

```
om-gagloev@omgagloev:~/work/2020-2021/Операционные системы/lab_prog$ sudo apt install spli
nt
Чтение списков пакетов... Готово
```

10) Посмотрел и проанализировал информацию, которую он выдаёт:

```
    declared type float: (cos(Numeral))
../../\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\3
    77\377\377\377\377\377 \377\377\377\377\377\377\377\377\377\377\377\377\377\
    377/lab_prog/calculate.c:38:11: Return value type double does not match
    declared type float: (tan(Numeral))
../../\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\3
    77\377\377\377\377\377 \377\377\377\377\377\377\377\377\377\377\377\377\377\
    377/lab_prog/calculate.c:40:11: Return value type double does not match
    declared type float: (HUGE_VAL)

Finished checking --- 15 code warnings
```

```
om-gagloev@omgagloev:~/work/2020-2021/Операционные системы/lab_prog$ splint main.c
Splint 3.1.2 --- 20 Feb 2018

../../\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\3
    77\377\377\377\377\377 \377\377\377\377\377\377\377\377\377\377\377\377\377\
    377/lab_prog/calculate.h:3:36: 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)
../../\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377
\377\377\377 \377\377\377\377\377\377\377\377\377\377\377\377\377\377\377/lab_prog/main.c: (in
 function main)
../../\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\3
    77\377\377\377\377\377 \377\377\377\377\377\377\377\377\377\377\377\377\377\
    377/lab_prog/main.c:10:3: 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)
../../\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\3
    77\377\377\377\377\377 \377\377\377\377\377\377\377\377\377\377\377\377\377\
    377/lab_prog/main.c:12:3: Return value (type int) ignored:
    scanf("%s", Oper...

Finished checking --- 3 code warnings
```

Как пример: видим, что в обоих случаях у нас игнорируется значение функции scanf()

Вывод :

Я 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.

Контрольные вопросы :

1) С помощью функций info и man. 2)Создание исходного кода программы, которая представляется в виде файла сохранение различных вариантов исходного текста; 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. [1] 4) Компиляция всей программы в целом и получении исполняемого модуля. 5) 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. [1] 6) 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. 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 – 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 передаётся 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) 1) Выполнили компиляцию программы 2) Увидели ошибки в программе 3) Открыли редактор и исправили программу 4) Загрузили программу в отладчик gdb 5) run — отладчик выполнил программу, мы ввели требуемые значения. 6) программа завершена, gdb не видит ошибок. 10) Не возникло – cscope - исследование функций, содержащихся в программе; – splint — критическая проверка программ, написанных на языке Си. 11) 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 синтаксиса языка Си, но малоэффективных с точки зрения их реализации или содержащих в себе семантические ошибки; 3) Общая оценка мобильности пользовательской программы.