Отчёт о выполнении лабораторной работы №14

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

Российский Университет Дружбы Народов
Факультет Физико-Математических и Естественных Наук

Дисциплина: *Операционные системы*

Студент: *Евсельев Дмитрий*

Группа: *НКНбд-01-20*

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.

Задание

1. В домашнем каталоге создать подкаталог ~/work/os/lab_prog .
2. Создайть в нём файлы: calculate.h , calculate.c , main.c .
3. Выполнить компиляцию программы посредством gcc
4. При необходимости исправить синтаксические ошибки.
5. Создать Makefile
6. С помощью gdb выполнить отладку программы calcul (перед использованием gdb исправить Makefile)
7. С помощью утилиты splint проанализировать коды файлов calculate.c и main.c .

Выполнение работы

1. В домашнем каталоге создал подкаталог ~/work/os/lab_prog
2. Создайл в нём файлы: calculate.h , calculate.c , main.c

   Рис.1 Содержимое каталога

```
dimon@vostro:~/work/os/lab_prog$ ls -l
итого 16
-rw-rw-r-- 1 dimon dimon 1650 июн  3 00:28 calculate.c
-rw-rw-r-- 1 dimon dimon  171 июн  5 13:54 calculate.h
-rw-rw-r-- 1 dimon dimon  431 июн  5 14:09 main.c
```

3. Выполнить компиляцию программы посредством gcc

Рис.2 Компиляция файлов gcc

```
dimon@vostro:~/work/os/lab_prog$ gcc -c calculate.c -g
dimon@vostro:~/work/os/lab_prog$ gcc -c main.c -g
dimon@vostro:~/work/os/lab_prog$ gcc calculate.o main.o -o calcul -g  -lm
dimon@vostro:~/work/os/lab_prog$ ll
итого 68
drwxrwxr-x 2 dimon dimon  4096 июн  5 21:55 ./
drwxrwxr-x 3 dimon dimon  4096 июн  3 00:24 ../
-rwxrwxr-x 1 dimon dimon 21352 июн  5 21:55 calcul*
-rw-rw-r-- 1 dimon dimon  1650 июн  3 00:28 calculate.c
-rw-rw-r-- 1 dimon dimon   171 июн  5 13:54 calculate.h
-rw-rw-r-- 1 dimon dimon  8720 июн  5 21:54 calculate.o
-rw-rw-r-- 1 dimon dimon   431 июн  5 14:09 main.c
-rw-rw-r-- 1 dimon dimon  6720 июн  5 21:55 main.o
```

4. Создал Makefile

Рис.3 Makefile

```
1  Создать Makefile#
2  # Makefile
3  #
4
5  CC = gcc
6  CFLAGS = -g
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
```

5. С помощью gdb выполнил отладку программы calcul (перед использованием gdb исправить Makefile)

Рис.3 Makefile

```
Создать Makefile#
# 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
```

6. С помощью gdb выполнил отладку программы calcul

Рис.4 Запуск отладчика

```
dimon@vostro:~/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/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...
```

Рис.5 Запуск программы внутри отладчика

```
(gdb) run
Starting program: /home/dimon/work/os/lab_prog/calcul
Число: 4
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): sqrt
  2.00
[Inferior 1 (process 12667) exited normally]
(gdb) run
Starting program: /home/dimon/work/os/lab_prog/calcul
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): +
Второе слагаемое: 7
 12.00
[Inferior 1 (process 12702) exited normally]
```

Рис.6 Команда list

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

Рис.7 Просмотр строк с 12 по 15

```
(gdb) list 12,15
12          printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): ");
13          scanf("%s",Operation);       // error was fixed
14          Result = Calculate(Numeral, Operation);
15          printf("%6.2f\n",Result);
```

Рис.8 Просмотр строк неосновного файла calculate.c

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

Рис.9 Установка, проверка и удаление точке останова

```
(gdb) list calculate.c:15,20
15          }
16          else if(strncmp(Operation, "-", 1) == 0){
17              printf("Вычитаемое: ");
18              scanf("%f",&SecondNumeral);
19              return(Numeral - SecondNumeral);
20          }
(gdb) break 17
Breakpoint 1 at 0x5555555552dd: file calculate.c, line 17.
(gdb) run
Starting program: /home/dimon/work/os/lab_prog/calcul
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): -

Breakpoint 1, Calculate (Numeral=5, Operation=0x7fffffffe024 "-")
    at calculate.c:17
17              printf("Вычитаемое: ");
(gdb) backtrace
#0  Calculate (Numeral=5, Operation=0x7fffffffe024 "-") at calculate.c:17
#1  0x00005555555555bd in main () at main.c:14
(gdb) print Numeral
$1 = 5
(gdb) display Numeral
1: Numeral = 5
(gdb) info brealpoints
Undefined info command: "brealpoints".  Try "help info".
(gdb) info breakpoints
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x00005555555552dd in Calculate
                                                   at calculate.c:17
        breakpoint already hit 1 time
(gdb) delete 1
(gdb) info breakpoints
```

Рис.10 Анализ splint calculate.c

```
dimon@vostro:~/work/os/lab_prog$ splint calculate.c
Splint 3.1.2 --- 20 Feb 2018

calculate.h:5:37: 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:13:9: 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:18:9: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:23:9: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:28:9: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:29:12: 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:31:19: Return value type double does not match declared type float:
                     (HUGE_VAL)
  To allow all numeric types to match, use +relaxtypes.
calculate.c:38:9: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:39:15: Return value type double does not match declared type float:
                     (pow(Numeral, SecondNumeral))
calculate.c:42:15: Return value type double does not match declared type float:
                     (sqrt(Numeral))
calculate.c:44:15: Return value type double does not match declared type float:
                     (sin(Numeral))
calculate.c:46:15: Return value type double does not match declared type float:
                     (cos(Numeral))
calculate.c:48:15: Return value type double does not match declared type float:
```

Рис.11 Анализ splint main.c

```
dimon@vostro:~/work/os/lab_prog$ splint main.c
Splint 3.1.2 --- 20 Feb 2018

calculate.h:5:37: 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:11:5: 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:13:5: Return value (type int) ignored: scanf("%s", Oper...

Finished checking --- 3 code warnings
```

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.

Контрольные вопросы

1. Как получить информацию о возможностях программ gcc, make, gdb и др.? С помощью функций info и man.

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

- создание исходного кода программы, которая 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. 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. 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. Каково основное назначение компилятора языка С в UNIX? Компиляция всей программы в целом и получении исполняемого модуля.

5. Для чего предназначена утилита make? 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.

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, 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. Назовите основное свойство, присущее всем программам отладки. Что необходимо сделать, чтобы его можно было использовать? 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. 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 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. Опишите по шагам схему отладки программы, которую Вы использовали при выполнении лабораторной работы.

1. Выполнили компиляцию программы 2)Увидели ошибки в программе
2. Открыли редактор и исправили программу

3. Загрузили программу в отладчик gdb
4. run — отладчик выполнил программу, мы ввели требуемые значения.
5. программа завершена, gdb не видит ошибок.

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

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

12. Каковы основные задачи, решаемые программой splint?

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.