Цель

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

Написать и отладить программу, выполняющую функции калькулятора

Ход работы

1. В каталоге **~/work/2020-2021/OperatingSystems/intro-os** создал подкаталог **lab13**, в котором создал файлы **calculate.h**, **calculate.c**, **main.c.** (Рис.1)

```
[dnbabkov@dnbabkov calc]$ touch calculate.h calculate.c main.c
[dnbabkov@dnbabkov calc]$ ls -l
total 0
-rw-rw-r--. 1 dnbabkov dnbabkov 0 Jun  5 10:13 calculate.c
-rw-rw-r--. 1 dnbabkov dnbabkov 0 Jun  5 10:13 calculate.h
-rw-rw-r--. 1 dnbabkov dnbabkov 0 Jun  5 10:13 main.c
```

Рис.1

2. В файлы написал код, данный в методических материалах (Рис.2, 3, 4)

```
calculate.h         [-M--]  0 L:[  1+ 2   3/  7] *(5
////////////////////////////////////////
// calculate.h

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

Рис.2

```
calculate.c        [-M--]  0 L:[  1+ 2   3/ 54] *(52  /1464b) 0010 0x
////////////////////////////////////
// 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) == 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));
```

Рис.3

```
main.c                [-M--]  9 L:[  1+ 1   2/ 16] *(50  / 403b)
////////////////////////////////////////
// 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;
}
```

Рис.4

3. С помощью **gcc** выполнил компиляцию программы (Рис.5)

```
[dnbabkov@dnbabkov calc]$ gcc -c calculate.c
[dnbabkov@dnbabkov calc]$ gcc -c main.c
[dnbabkov@dnbabkov calc]$ gcc calculate.o main.o -o calcul -lm
[dnbabkov@dnbabkov calc]$ ls -l
total 32
-rwxrwxr-x. 1 dnbabkov dnbabkov 8784 Jun  5 10:21 calcul
-rw-rw-r--. 1 dnbabkov dnbabkov 1411 Jun  5 10:19 calculate.c
-rw-rw-r--. 1 dnbabkov dnbabkov  116 Jun  5 10:18 calculate.h
-rw-rw-r--. 1 dnbabkov dnbabkov 3696 Jun  5 10:21 calculate.o
-rw-rw-r--. 1 dnbabkov dnbabkov  352 Jun  5 10:19 main.c
-rw-rw-r--. 1 dnbabkov dnbabkov 1984 Jun  5 10:21 main.o
[dnbabkov@dnbabkov calc]$ ./calcul
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): pow
Степень: 2
 25.00
```

Рис.5

4. Синтаксических ошибок обнаружено не было

5. Создал Makefile с содержанием, данным в методических материалах (Рис.6)

```
Makefile            [----]  0 L:[  1+ 9  10/ 20] *(114
#
# 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
```

Рис.6

При вызове **make calcul** Makefile делает из объектных файлов **calculate.o** и **main.o** исполняемый файл **calcul** с помощью команды **gcc calculate.o main.o -o calcul -lm**. При вызове make **calculate.o** из файлов **calculate.c** и **calculate.h** создается объектный файл. То же самое происходит при вызове make **main.o**. При вызове make clear удаляется всё, что создает Makefile.

6. Исправил Makefile (Рис.7)

```
Makefile            [-M--] 13 L:[  1+14  15/ 20] *(21
#
# 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
```

Рис.7

После этого сделал с его помощью исполняемый файл **calcul** (Рис.8)

```
[dnbabkov@dnbabkov calc]$ make clean
rm calcul *.o *~
rm: cannot remove '*~': No such file or directory
make: [clean] Error 1 (ignored)
[dnbabkov@dnbabkov calc]$ make calculate.o
gcc -c calculate.c -g
[dnbabkov@dnbabkov calc]$ make main.o
gcc -c main.c -g
[dnbabkov@dnbabkov calc]$ make calcul
gcc calculate.o main.o -o calcul -lm
[dnbabkov@dnbabkov calc]$ ls -l
total 40
-rwxrwxr-x. 1 dnbabkov dnbabkov 10512 Jun  5 10:52 calcul
-rw-rw-r--. 1 dnbabkov dnbabkov  1464 Jun  5 10:35 calculate.c
-rw-rw-r--. 1 dnbabkov dnbabkov   172 Jun  5 10:36 calculate.h
-rw-rw-r--. 1 dnbabkov dnbabkov  5808 Jun  5 10:52 calculate.o
-rw-rw-r--. 1 dnbabkov dnbabkov   403 Jun  5 10:35 main.c
-rw-rw-r--. 1 dnbabkov dnbabkov  4000 Jun  5 10:52 main.o
-rw-rw-r--. 1 dnbabkov dnbabkov   286 Jun  5 10:46 Makefile
```

Рис.8

- Вызываю отладчик **gdb** для исполняемого файла с помощью команды **gdb ./calcul** (Рис.9)

```
[dnbabkov@dnbabkov calc]$ gdb ./calcul
GNU gdb (GDB) Red Hat Enterprise Linux 7.6.1-120.el7
Copyright (C) 2013 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-redhat-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/dnbabkov/work/2020-2021/OperatingSystems/laboratory/intro-os/lab14/calc/calcul...(no debugging symbols found)...done.
(gdb)
```

Рис.9

- Запускаю программу в среде отладчика с помощью команды **run** (Рис.10)

```
(gdb) run
Starting program: /home/dnbabkov/work/2020-2021/OperatingSystems/laboratory/intro-os/lab14/calc/./calcul
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): *
Множитель: 5
 25.00
[Inferior 1 (process 5459) exited normally]
Missing separate debuginfos, use: debuginfo-install glibc-2.17-324.el7_9.x86_64
```

Рис.10

- С помощью команды **list** вывожу на экран девять первых строк исходного текста программы (Рис.11)

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

Рис.11

- Вывожу строки с 12 по 15 командой **list12,15** (Рис.12)

```
(gdb) list 12,15
12          scanf("%s",&Operation);
13          Result = Calculate(Numeral, Operation);
14          printf("%6.2f\n",Result);
15      return 0;
```

Рис.12

- Вывожу строки не основного файла (Рис.13)

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

Рис.13

- В файле **calculate.c** устанавливаю точку останова на 21 строке с помощью команды **break 21** (Рис.14)

```
(gdb) list calculate.c:20,27
20      else if(strncmp(Operation, "*", 1) == 0) {
21          printf("Множитель: ");
22          scanf("%f",&SecondNumeral);
23          return(Numeral * SecondNumeral);
24      }
25      else if(strncmp(Operation, "/", 1) == 0) {
26          printf("Делитель: ");
27          scanf("%f",&SecondNumeral);
(gdb) break 21
Breakpoint 1 at 0x400823: file calculate.c, line 21.
```

Рис.14

- Вывожу информацию об имеющихся точках останова с помощью команды **info breakpoints** (Рис.15)

```
(gdb) info breakpoints
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x0000000000400823 in Calculate at calculate.c:21
```

Рис.15

- Запускаю программу внутри отладчика, чтобы убедиться, что точка останова работает (Рис.16)

```
(gdb) run
Starting program: /home/dnbabkov/work/2020-2021/OperatingSystems/laboratory/intro-os/lab14/calc/./calcul
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): *

Breakpoint 1, Calculate (Numeral=5, Operation=0x7fffffffde60 "*") at calculate.c:21
21          printf("Множитель: ");
Missing separate debuginfos, use: debuginfo-install glibc-2.17-324.el7_9.x86_64
```

Рис.16

- Вывожу стек вызываемых функций с помощью команды **backtrace** (Рис.17)

```
(gdb) run
Starting program: /home/dnbabkov/work/2020-2021/OperatingSystems/laboratory/intro-os/lab14/calc/./calcul
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): *

Breakpoint 1, Calculate (Numeral=5, Operation=0x7fffffffde60 "*") at calculate.c:21
21          printf("Множитель: ");
(gdb) backtrace
#0  Calculate (Numeral=5, Operation=0x7fffffffde60 "*") at calculate.c:21
#1  0x0000000000400a90 in main () at main.c:13
```

Рис.17

- Вывожу значение переменной **Numeral** командой **print Numeral** (Рис.18)

```
(gdb) print Numeral
$1 = 5
```

Рис.18

- Вывожу переменную **Numeral** с помощью команды **display Numeral** (Рис.19)

```
(gdb) display Numeral
1: Numeral = 5
```

Рис.19

- Удаляю точки останова командой **delete** (Рис.20)

```
(gdb) info breakpoints
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x0000000000400823 in Calculate at calculate.c:21
        breakpoint already hit 1 time
(gdb) delete 1
(gdb) info breakpoints
No breakpoints or watchpoints.
```

Рис.20

7. С помощью утилиты **splint** анализирую файлы **main.c** и **calculate.c** (Рис.21, 22)

```
[dnbabkov@dnbabkov calc]$ splint main.c
Splint 3.1.2 --- 11 Oct 2015

calculate.h:6:37: 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:10:5: 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:12:16: 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:12:13: Corresponding format code
main.c:12:5: Return value (type int) ignored: scanf("%s", &Ope...

Finished checking --- 4 code warnings
```

Рис.22

```
[dnbabkov@dnbabkov calc]$ splint calculate.c
Splint 3.1.2 --- 11 Oct 2015

calculate.h:6:37: 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:8:37: Function parameter Operation declared as manifest array (size
                    constant is meaningless)
calculate.c: (in function Calculate)
calculate.c:12:5: 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:17:5: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:22:5: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:27:5: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:28:8: 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:30:8: Return value type double does not match declared type float:
                    (HUGE_VAL)
  To allow all numeric types to match, use +relaxtypes.
calculate.c:37:5: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:38:11: Return value type double does not match declared type float:
                     (pow(Numeral, SecondNumeral))
calculate.c:41:11: Return value type double does not match declared type float:
                     (sqrt(Numeral))
calculate.c:43:11: Return value type double does not match declared type float:
                     (sin(Numeral))
calculate.c:45:11: Return value type double does not match declared type float:
                     (cos(Numeral))
calculate.c:47:11: Return value type double does not match declared type float:
                     (tan(Numeral))
calculate.c:51:11: Return value type double does not match declared type float:
                     (HUGE_VAL)

Finished checking --- 15 code warnings
```

Рис.21

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:
    - 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.

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:

```
<цель_1> <цель_2> ... : <зависимость_1> <зависимость_2> ...
<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. 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:

```
#
# Makefile for abcd.c
#
CC = gcc
CFLAGS =
# Compile abcd.c normaly
abcd: abcd.c
$(CC) -o abcd $(CFLAGS) abcd.c
clean:
-rm abcd *.o *~
# End Makefile for abcd.c
```

В этом примере в начале файла заданы три переменные: CC и CFLAGS. Затем указаны цели, их зависимости и соответствующие команды. В командах происходит обращение к значениям переменных. Цель с именем clean производит очистку каталога от файлов, полученных в результате компиляции. Для её описания использованы регулярные выражения. 7. Во время работы над кодом программы программист неизбежно сталкивается с появлением ошибок в ней. Использование отладчика для поиска и устранения ошибок в программе существенно облегчает жизнь программиста. В комплект программ GNU для ОС типа UNIX входит отладчик GDB (GNU Debugger). 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. Основные команды отладчика gdb:

- 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. Схема отладки программы показана в 6 пункте лабораторной работы.
10. При первом запуске компилятор не выдал никаких ошибок, но в коде программы main.c допущена ошибка, которую компилятор мог пропустить (возможно, из-за версии 8.3.0-19): в строке scanf("%s", &Operation); нужно убрать знак &, потому что имя массива символов уже является указателем на первый элемент этого массива.
11. 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. Утилита splint 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.

Вывод

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