

Отчет по лабораторной работе №13

Средства для создания приложений в ОС UNIX

Кочкарев “sakochkarev” Станислав

Содержание

1	Цель работы	3
2	Задание	4
3	Выполнение лабораторной работы	5
4	Выводы	12
5	Контрольные вопросы	13

1 Цель работы

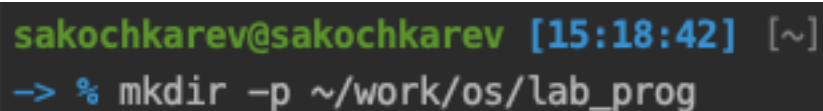
Приобрести простейшие навыки разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linux на примере создания на языке программирования С калькулятора с простейшими функциями.

2 Задание

Написать программу и проанализировать ее код и выполнение.

3 Выполнение лабораторной работы

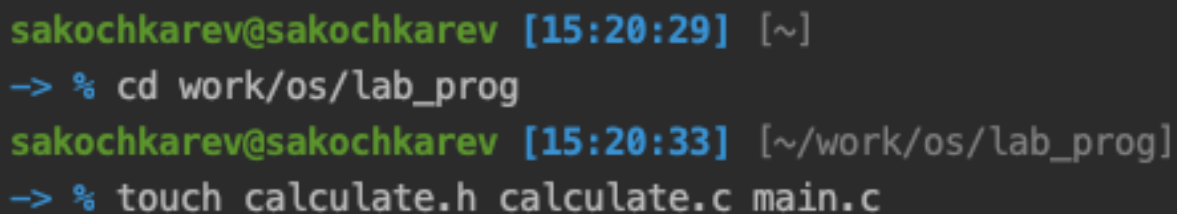
Первым делом мы создали в домашнем каталоге подкаталог `~/work/os/lab_prog` командой `mkdir -p ~/work/os/lab_prog` (рис. 3.1).



```
sakochkarev@sakochkarev [15:18:42] [~]  
-> % mkdir -p ~/work/os/lab_prog
```

Рис. 3.1: Создание директории

Далее в нем были созданы три файла `calculate.h`, `calculate.c`, `main.c` (рис. 3.2).



```
sakochkarev@sakochkarev [15:20:29] [~]  
-> % cd work/os/lab_prog  
sakochkarev@sakochkarev [15:20:33] [~/work/os/lab_prog]  
-> % touch calculate.h calculate.c main.c
```

Рис. 3.2: Создание файлов

В каждый из этих файлов были написаны соответствующие коды (рис. 3.3, 3.4, 3.5).

```

1 //////////////////////////////////////////////////
2 // calculate.c
3 #include <stdio.h>
4 #include <math.h>
5 #include <string.h>
6 #include "calculate.h"
7 float Calculate(float Numeral, char Operation[4]) {
8     float SecondNumeral;
9     if(strncmp(Operation, "+", 1) == 0) {
10         printf("Второе слагаемое: ");
11         scanf("%f", &SecondNumeral);
12         return(Numeral + SecondNumeral);
13     }
14     else if(strncmp(Operation, "-", 1) == 0) {
15         printf("Вычитаемое: "); scanf("%f", &SecondNumeral); return(Numeral - SecondNumeral);
16     }
17     else if(strncmp(Operation, "*", 1) == 0) {
18         printf("Множитель: "); scanf("%f", &SecondNumeral); return(Numeral * SecondNumeral);
19     }
20     else if(strncmp(Operation, "/", 1) == 0)
21     {
22         printf("Делитель: ");
23         scanf("%f", &SecondNumeral); if(SecondNumeral == 0)
24         {
25             printf("Ошибка: деление на ноль! "); return(HUGE_VAL);
26         }
27         else
28             return(Numeral / SecondNumeral);

```

Рис. 3.3: Код файла calculate.c

```

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

```

Рис. 3.4: Код файла calculate.h

```

1 //////////////////////////////////////////////////
2 // main.c
3 #include <stdio.h>
4 #include "calculate.h"
5
6 int
7 main (void) {
8     float Numeral;
9     char Operation[4];
10    float Result;
11    printf("Число: ");
12    scanf("%f",&Numeral);
13    printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): "); scanf("%s",&Operation);
14    Result = Calculate(Numeral, Operation); printf("%.2f\n",Result);
15    return 0;
16 }

```

Рис. 3.5: Код файла main.c

После этого была выполнена компиляция данных файлов (рис. 3.6).

```

sakochkarev@sakochkarev [15:23:17] [~/work/os/lab_prog]
-> % gcc -c calculate.c
gcc -c main.c
gcc calculate.o main.o -o calcul -lm
main.c:13:73: warning: format specifies type 'char *' but the argument has type 'char (*)[4]' [-Wformat]
printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): "); scanf("%s",&Operation);
~ ~ ~ ~ ~ ^~~~~~
1 warning generated.

```

Рис. 3.6: Компиляция файлов

Далее был создан Makefile (рис. 3.7), где был написан соответствующий код (рис. 3.8).

```

sakochkarev@sakochkarev [15:20:36] [~/work/os/lab_prog]
-> % touch Makefile

```

Рис. 3.7: Создание Makefile

```

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

```

Рис. 3.8: Содержание Makefile

Код содержит в себе переменные с используемым компилятором, а также фла-гами для компиляции. В нем находится четыре цели, все из которых, кроме `clean` выполняют сам процесс компиляции. Цель `clean` позволяет удалить результаты компиляции.

Далее, используя `gdb` была выполнена отладка программы `calcul` (рис. 3.9).


```

sakochkarev@sakochkarev [15:26:41] [~/work/os/lab_prog]
-> % sudo gdb ./calcul
Password:
GNU gdb (GDB) 12.1
Copyright (C) 2022 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-apple-darwin21.3.0".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
  <http://www.gnu.org/software/gdb/documentation/>.

--Type <RET> for more, q to quit, c to continue without paging--
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./calcul...
(gdb) list
1      //////////////////////////////////////////
2      // main.c
3      #include <stdio.h>
4      #include "calculate.h"
5
6      int
7      main (void) {
8          float Numeral;
9          char Operation[4];
10         float Result;
(gdb)

```

Рис. 3.9: Отладка программы calcul

В конце, используя утилиту `splint`, были проанализированы коды файлов `calculate.c` (рис. 3.10) и `main.c` (рис. 3.11).

```

sakochkarev@sakochkarev [15:05:05] [~/work/os/lab_prog]
-> % splint calculate.c
Splint 3.1.2 --- 30 Oct 2021

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:7:37: Function parameter Operation declared as manifest array (size
        constant is meaningless)
calculate.c: (in function Calculate)
calculate.c:11:9: 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:15:43: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:18:37: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:23:1: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:23:32: 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:25:63: Return value type double does not match declared type float:
        (HUGE_VAL)
    To allow all numeric types to match, use +relaxtypes.
calculate.c:32:29: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:32:63: Return value type double does not match declared type float:
        (pow(Numeral, SecondNumeral))
calculate.c:35:7: Return value type double does not match declared type float:
        (sqrt(Numeral))
calculate.c:37:7: Return value type double does not match declared type float:
        (sin(Numeral))
calculate.c:39:7: Return value type double does not match declared type float:
        (cos(Numeral))
calculate.c:41:7: Return value type double does not match declared type float:
        (tan(Numeral))
calculate.c:43:74: Return value type double does not match declared type float:
        (HUGE_VAL)

Finished checking --- 15 code warnings

```

Рис. 3.10: Анализирование calculate.c

```
sakochkarev@sakochkarev [15:05:09] [~/work/os/lab_prog]
-> % splint main.c
Splint 3.1.2 --- 30 Oct 2021

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:12:1: 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:13:73: 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:13:70: Corresponding format code
main.c:13:62: Return value (type int) ignored: scanf("%s", &Op...

Finished checking --- 4 code warnings
```

Рис. 3.11: Анализирование main.c

4 Выводы

По выполнении данной лабораторной работы мы приобрели 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 Контрольные вопросы

1. Используя команду `man`. Например `man gdb`.
2. Придумать приложение, написать код, выпустить приложение.
3. В контексте языка программирования, суффикс – это явный указатель типа литерала. Например:

```
float f1 = 1.0L;  
float f2 = 1.0;  
float f3 = 1.0f;  
float f4 = 1;  
float f5 = 1ULL;  
float f6 = (char)1;  
float f7 = true;
```

4. Для компилирования текстовых файлов с кодом программы в исполняемые файлы.
5. Для сборки разрабатываемого приложения и собственно компиляции.
6. Пример структуры:

```
#  
# 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
```

где *abcd* – цель, строка ниже – команда выполняемая при выполнении цели. *clean* – тоже цель, однако не входящая в общий список целей при выполнении make. Она позволяет удалить результаты компиляции. 7. Программы отладки частично декомпилируют исполняемый файл. Для того, что декомпиляция была корректной, необходимо использовать флаг *-g* при компиляции.

8.

Команда	Описание действия
<hr/>	
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	вывести на экран список используемых точек останова
breakpoints	
info	вывести на экран список используемых контрольных выражений
watchpoints	

Команда	Описание действия
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

9.
 - Запускаем отладчик GDB, загрузив в него программу для отладки
 - Для запуска программы внутри отладчика вводим команду run
 - Для постраничного (по 9 строк) просмотра исходного код используем команду list
 - Для просмотра строк с 12 по 15 основного файла используем list с параметрами
 - Для просмотра определённых строк не основного файла используем list с параметрами
 - Устанавливаем точку останова в файле calculate.c на строке номер 21
 - Выводим информацию об имеющихся в проекте точка останова
 - Запускаем программу внутри отладчика и убеждаемся, что программа остановится в момент прохождения точки останова
 - Отладчик выдаст информацию, а команда backtrace покажет весь стек вызываемых функций от начала программы до текущего места.
 - Посмотрим, чему равно на этом этапе значение переменной Numeral, введя 110. На экран должно быть выведено число 5.

- Сравниваем с результатом вывода на экран после использования команды
- Убираем точки останова

10. Классная реакция. Показывает синтаксические ошибки в программе.

11. Визуализатор кода может быть полезен для понимания кода программы.

12. Анализ программного кода, проверка корректности задания аргументов использованных в программе функций и типов возвращаемых значений, а также обнаруживание синтаксических и семантические ошибки.