

Отчёт по лабораторной работе №13

Средства для создания приложений в ОС UNIX

Магамадов Асхаб Ахиатович

Содержание

1	Цель работы	4
2	Выполнение лабораторной работы	5
3	Вывод	13
4	Контрольные вопросы	14

Список иллюстраций

2.1	Компиляция	8
2.2	Использование make	10
2.3	Использование отладчика	10
2.4	Использование отладчика	11
2.5	Использование отладчика	11
2.6	Использование splint	12

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 Выполнение лабораторной работы

1. Создали подкаталог для файлов лаб работы
2. Создал в нём файлы: calculate.h , calculate.c , main.c . Это будет примитивнейший калькулятор, способный складывать, вычитать, умножать и делить, возводить число в степень, брать квадратный корень, вычислять sin, cos, tan. При запуске он будет запрашивать первое число, операцию, второе число. После этого программа выведет результат и остановится.

Код файла calculate.c (реализует функции калькулятора)

```
////////////////////////////////////  
// 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));
    else if(strncmp(Operation, "tan", 3) == 0)
        return(tan(Numeral));
    else
    {
        printf("Неправильно введено действие ");
        return(HUGE_VAL);
    }
}

```

Код файла calculate.h (описывает формат вызова функции калькулятора)

```

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

```

Код файла main.c (реализует интерфейс пользователя к калькулятору)

```

////////////////////////////////////

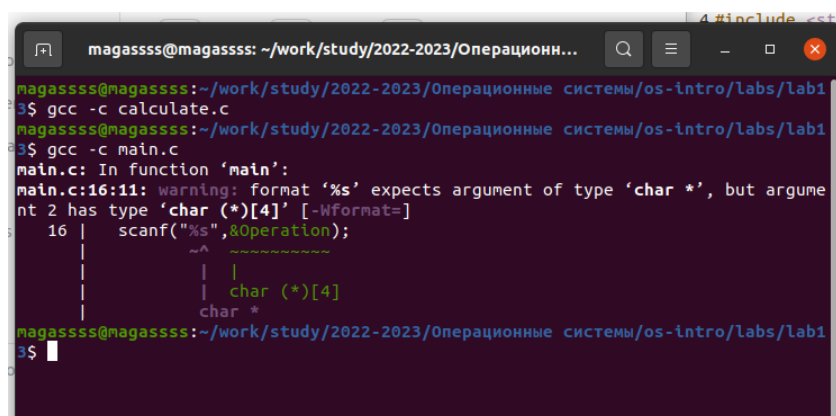
```

```
// 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;
}
```

3. Выполнили компиляцию программы посредством gcc :



```
magassss@magassss: ~/work/study/2022-2023/Операционн...
3$ gcc -c calculate.c
magassss@magassss:~/work/study/2022-2023/Операционные системы/os-intro/labs/lab1
3$ gcc -c main.c
main.c: In function 'main':
main.c:16:11: warning: format '%s' expects argument of type 'char *', but argume
nt 2 has type 'char (*)[4]' [-Wformat=]
   16 |     scanf("%s",&Operation);
      |           ~^  ~~~~~
      |           |   |
      |           |   char (*)[4]
      |           |   char *
magassss@magassss:~/work/study/2022-2023/Операционные системы/os-intro/labs/lab1
3$
```

Рис. 2.1: Компиляция

4. При необходимости исправили синтаксические ошибки.

5. Создали Makefile со следующим содержанием:

```
#  
# 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
```

С помощью программы make получаем различные варианты построения исполняемого модуля.

```
magasssss@magasssss: ~/work/study/2022-2023/Операционн...
|         |         | char (*)[4]
|         |         | char *
magasssss@magasssss:~/work/study/2022-2023/Операционные системы/os-intro/labs/lab1
3$ make clean
rm calcul *.o *~
rm: cannot remove 'calcul': No such file or directory
rm: cannot remove '*~': No such file or directory
make: [Makefile:14: clean] Error 1 (ignored)
magasssss@magasssss:~/work/study/2022-2023/Операционные системы/os-intro/labs/lab1
3$ make
gcc -c calculate.c -g
gcc -c main.c -g
main.c: In function 'main':
main.c:16:11: warning: format '%s' expects argument of type 'char *', but argume
nt 2 has type 'char (*)[4]' [-Wformat=]
   16 |     scanf("%s",&operation);
      |           ^~
      |           |
      |           | char (*)[4]
      |           | char *
gcc calcul.o main.o -o calcul -lm
magasssss@magasssss:~/work/study/2022-2023/Операционные системы/os-intro/labs/lab1
3$
```

Рис. 2.2: Использование make

4. С помощью gdb выполнил отладку программы calcul

```
magasssss@magasssss: ~/work/study/2022-2023/Операционн...
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) r
Starting program: /home/magasssss/work/study/2022-2023/Операционные системы/os-in
tro/labs/lab13/calcul
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): *
Множитель: 5
25.00
[Inferior 1 (process 2435) exited normally]
(gdb)
```

Рис. 2.3: Использование отладчика

```

magassss@magassss: ~/work/study/2022-2023/Операционн...
15     printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): ");
(gdb) list calculate.c:20,29
20     {
21         printf("Вычитаемое: ");
22         scanf("%f",&SecondNumeral);
23         return(Numeral - SecondNumeral);
24     }
25     else if(strncmp(Operation, "*", 1) == 0)
26     {
27         printf("Множитель: ");
28         scanf("%f",&SecondNumeral);
29         return(Numeral * SecondNumeral);
(gdb) break 21
Breakpoint 1 at 0x555555552dd: file calculate.c, line 21.
(gdb) r
Starting program: /home/magassss/work/study/2022-2023/Операционные системы/os-in
tro/labs/lab13/calcul
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): -

Breakpoint 1, Calculate (Numeral=5, Operation=0x7fffffffdeb4 "-")
at calculate.c:21
21     printf("Вычитаемое: ");
(gdb)

```

Рис. 2.4: Использование отладчика

```

magassss@magassss: ~/work/study/2022-2023/Операционн...
Breakpoint 1 at 0x555555552dd: file calculate.c, line 21.
(gdb) r
Starting program: /home/magassss/work/study/2022-2023/Операционные системы/os-in
tro/labs/lab13/calcul
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): -

Breakpoint 1, Calculate (Numeral=5, Operation=0x7fffffffdeb4 "-")
at calculate.c:21
21     printf("Вычитаемое: ");
(gdb) backtrace
#0 Calculate (Numeral=5, Operation=0x7fffffffdeb4 "-") at calculate.c:21
#1 0x00005555555555bd in main () at main.c:17
(gdb) print Numeral
$1 = 5
(gdb) disp Numeral
1: Numeral = 5
(gdb) i b
Num Type Disp Enb Address What
1 breakpoint keep y 0x000055555555552dd in Calculate
at calculate.c:21
breakpoint already hit 1 time
(gdb) dele 1
(gdb)

```

Рис. 2.5: Использование отладчика

5. С помощью утилиты `splint` попробовали проанализировать коды файлов

3 Вывод

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.

4 Контрольные вопросы

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

Ответ: Для этого есть команда man и предлагающиеся к ней файлы.

2. Назовите и дайте краткую характеристику основным этапам разработки приложений в UNIX. Ответ: Кодировка, Компиляция, Тест.

3. Что такое суффикс в контексте языка программирования? Приведите примеры использования. Ответ: Это расширения файлов.

4. Каково основное назначение компилятора языка C в UNIX? Ответ: Программа gcc, которая интерпретирует к определенному языку программирования аргументы командной строки и определяет запуск нужного компилятора для нужного файла.

5. Для чего предназначена утилита make? Ответ: Для компиляции группы файлов. Собрания из них программы, и последующего удаления.

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

```
program: main.o lib.o
cc -o program main.o lib.o
main.o lib.o: defines.h
```

В имени второй цели указаны два файла и для этой же цели не указана команда компиляции. Кроме того, нигде явно не указана зависимость объектных

файлов от «*.c»-файлов. Дело в том, что программа make имеет predetermined правила для получения файлов с определёнными расширениями. Так, для цели-объектного файла (расширение «.o») при обнаружении соответствующего файла с расширением «.c» будет вызван компилятор «cc -c» с указанием в параметрах этого «.c»-файла и всех файлов-зависимостей.

7. Назовите основное свойство, присущее всем программам отладки. Что необходимо сделать, чтобы его можно было использовать? Ответ: Программы для отладки нужны для нахождения ошибок в программе. Для их использования надо скомпилировать программу таким образом, чтобы отладочная информация содержалась в конечном бинарном файле.

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

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. Ответ:

10. `gdb -silent ./calcul`

11. `run`

12. `list`

13. `backtrace`

14. `breakpoints`

15. `print Numeral`

16. `Splint` (Не использовался по причине отсутствия команды в консоли).

17. Прокомментируйте реакцию компилятора на синтаксические ошибки в программе при его первом запуске. Ответ: Консоль выводит ошибку с номером строки и ошибочным сегментом, но при этом есть возможность выполнить программу сразу.

18. Назовите основные средства, повышающие понимание исходного кода программы. Ответ:

- a) Правильный синтаксис
- b) Наличие комментариев
- c) Разбиение большой сложной программы на несколько сегментов попроще.

12. Каковы основные задачи, решаемые программой splint? Ответ: split – разбиение файла на меньшие, определённого размера. Может разбивать текстовые файлы по строкам и любые – по байтам. По умолчанию читает со стандартного ввода и создает файлы с именами вида хаа, хаб и т.д. По умолчанию разбиение идёт по 1000 строк в файле.