Отчёт по лабораторной работе №14

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

Николаев Дмитрий Иванович

Содержание

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) Создал в домашнем каталоге подкаталог ~/work/os/lab_prog. А после создал в нём файлы calculate.h, calculate.c, main.c.

```
[dinikolaev@dinikolaev ~]$ cd work
[dinikolaev@dinikolaev work]$ mkdir os
[dinikolaev@dinikolaev work]$ cd os
[dinikolaev@dinikolaev os]$ mkdir lab_prog
[dinikolaev@dinikolaev os]$ cd lab_prog
[dinikolaev@dinikolaev lab_prog]$ touch calculate.h calculate.c main.c
[dinikolaev@dinikolaev lab_prog]$ ls
calculate.c  calculate.h  main.c
[dinikolaev@dinikolaev lab_prog]$
```

- Создание подкаталога и файлов для калькулятора в нём

2) В файле calculate.c, написанном на C, реализовал функции калькулятора (+,-,*,/,pow,sqrt,sin,cos,tan):

```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)
```

Figure 2.1: 2_1

```
        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
```

Figure 2.2: 2_2

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

- Файл calculate.c, реализующий функции калькулятора

3) В файле calculate.h реализовал интерфейс, описывающий формат вызова функции-калькулятора:

```c
#ifndef CALCULATE_H_
#define CALCULATE_H_

float Calculate(float Numeral,char Operation[4]);

#endif
```

- Интерфейсный файл calculate.h

4) В файле main.c, написанном на C, реализовал интерфейс пользователя к калькулятору:

```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;
}
```

- Основной файл main.c, реализующий интерфейс пользователя к калькулятору

5) Выполнил компиляцию программы калькулятора посредством gcc (назвав итоговый файл calcul).

```
[dinikolaev@dinikolaev lab_prog]$ gcc -c calculate.c
[dinikolaev@dinikolaev lab_prog]$ gcc -c main.c
[dinikolaev@dinikolaev lab_prog]$ gcc calculate.o main.o -o calcul -lm
[dinikolaev@dinikolaev lab_prog]$ ls
calcul  calculate.c  calculate.h  calculate.o  main.c  main.o
[dinikolaev@dinikolaev lab_prog]$
```

- Компиляция программы калькулятора

6) Исправил синтаксические ошибки (в “%s, &Operation” знак “&” не нужен).

7) Создал Makefile, реализующий компиляцию программы калькулятора (добавил CFLAGS = -g для дальнейшей отладки gdb).

```
Makefile            [----] 11 L:[  1+ 1   2/ 15] *(
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 *~
```

- Makefile для компиляции программы калькулятора

8) Снова скомпилировал программу с флагом -g (с помощью make) и выполнил с помощью gdb отладку программы calcul.

```
[dinikolaev@dinikolaev lab_prog]$ make
gcc -c calculate.c -g
gcc -c main.c -g
gcc calculate.o main.o  -o calcul -lm
[dinikolaev@dinikolaev lab_prog]$ ls
calcul  calculate.c  calculate.h  calculate.o  main.c  main.o  Makefile
[dinikolaev@dinikolaev lab_prog]$ gdb ./calcul
GNU gdb (GDB) Red Hat Enterprise Linux 8.2-12.el8
Copyright (C) 2018 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".
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...done.
```

- Запустил отладку программы calcul (“gdb ./calcul”)

9) Запустил программу внутри отладчика (“run”), запустил постраничный(по 9 строк) просмотр исходного кода (“list”), запустил просмотр строк с 12 по 15 основного файла (“list 12,15”).

```
(gdb) run
Starting program: /home/dinikolaev/work/os/lab_prog/calcul
Missing separate debuginfos, use: yum debuginfo-install glibc-2.28-127.el8.x86_64
Число: 12
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): /
Делитель: 3
  4.00
[Inferior 1 (process 4360) exited normally]
(gdb) list
1       #include <stdio.h>
2       #include "calculate.h"
3
4       int 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;
```

- запуск программы, просмотр с 1 по 9 и с 12 по 15 строки

10) Запустил просмотр строк файл calculate.c с 20 по 29 строки (“list calculate.c:20,29”), Установил точку останова в файле calculate.c на 21-ой строке (“list calculate.c:20,27” “break 21”) и вывел информацию об имеющихся точках останова (“info breakpoints”).

```
(gdb) list calculate.c: 20,29
20              }
21              else if(strncmp(Operation,"*",1)==0)
22              {
23                      printf("Множитель: ");
24                      scanf("%f",&SecondNumeral);
25                      return(Numeral*SecondNumeral);
26              }
27              else if(strncmp(Operation,"/",1)==0)
28              {
29                      printf("Делитель: ");
(gdb) list calculate.c: 20,27
20              }
21              else if(strncmp(Operation,"*",1)==0)
22              {
23                      printf("Множитель: ");
24                      scanf("%f",&SecondNumeral);
25                      return(Numeral*SecondNumeral);
26              }
27              else if(strncmp(Operation,"/",1)==0)
(gdb) break 21
Breakpoint 1 at 0x40087b: file calculate.c, line 21.
(gdb) info breakpoints
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x000000000040087b in Calculate at calculate.c:21
```

- просмотр строк файла calculate.c с 20 по 29 и установка точки останова на 21-ой строке

11) Запустил программу внутри отладчика и убедился в остановке в момент прохождения точки останова, с помощью команды backtrace просмотрел стек вызываемых функций от начала программы до текущего момента, посмотрел значение переменной Numeral (“print Numeral”) и сравнил с выводом после команды “display Numeral”, убрал точки останова (“delete 1”).

```
(gdb) run
Starting program: /home/dinikolaev/work/os/lab_prog/calcul
5Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): *

Breakpoint 1, Calculate (Numeral=5, Operation=0x7fffffffdf54 "*") at calculate.c:21
21              else if(strncmp(Operation,"*",1)==0)
(gdb) backtrace
#0  Calculate (Numeral=5, Operation=0x7fffffffdf54 "*") at calculate.c:21
#1  0x0000000000400ad4 in main () at main.c:13
(gdb) print Numeral
$1 = 5
(gdb) display Numeral
1: Numeral = 5
(gdb) info breakpoints
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x000000000040087b in Calculate at calculate.c:21
        breakpoint already hit 1 time
(gdb) delete 1
```

- Запуск программы и проверка значений на точке останова с последующим её удалением

12) Проанализировал коды файлов calculate.c и main.c с помощью утилиты splint.

```
calculate.h:3:38: 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:9:2: 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:11:14: 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:11:10: Corresponding format code
main.c:11:2: Return value (type int) ignored: scanf("%s", &Ope...

Finished checking --- 4 code warnings
```

- splint main.c

- splint calculate.c

2.1 Контрольные вопросы

1. С помощью функций info и man.
2. Процесс разработки программного обеспечения обычно разделяется на следующие этапы:
    a. Планирование, включающее сбор и анализ требований к функционалу и другим характеристикам разрабатываемого приложения;
    b. Проектирование, включающее в себя разработку базовых алгоритмов и спецификаций, определение языка программирования;

c. Непосредственная разработка приложения:

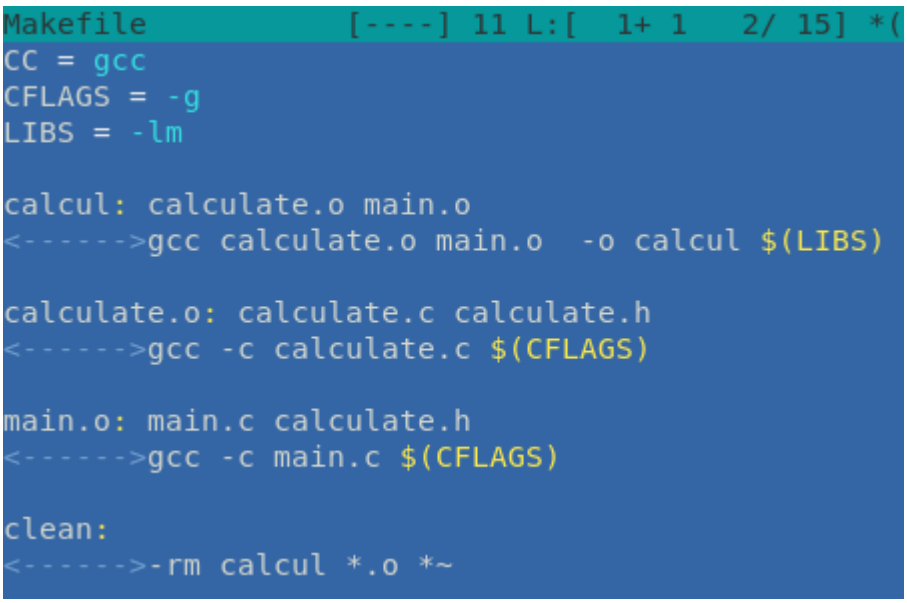
– Кодирование — по сути создание исходного текста программы; – Анализ разработанного кода; – Сборка, компиляция и разработка исполняемого модуля; – Тестирование и отладка, сохранение произведённых изменений; – Документирование

3. В данном контексте суффикс позволяет определить какая компиляция требуется для программы, он указывает тип объекта. Например по суффиксу ".c" компилятор распознаёт, что файл abc.c должен компилироваться (Язык C), а по суффиксу ".o", что файл abc.o является объектным. Так, для компиляции программы abc.c и построения исполняемого файла abc нужно ввести: "gcc -o abc abc.c".

4. Назначение состоит в компиляции всей программы в целом и получении исполняемого файла.

5. Утилита make освобождает пользователя от различной рутинной работы (например перекомпиляция файлов после внесённых изменений) и служит для документирования взаимосвязей между указанными файлами. Описание этого хранится в специальном make-файле (makefile или Makefile).

6. Подобный пример можно увидеть в основной части работы (п.7, Скриншот 6), где Makefile производит компиляцию файлов calculate.c, main.c (с флагом -g для отладки), создаёт исполняемый файл calcul на основе объектных файлов calculate.o и main.o (с флагом -lm) и позволяет удалить созданный исполняемый файл и объектные файлы.



- Пример Makefile

7. Практически все отладчики поддерживают возможность пошаговой отладки программы (а также выполнение до курсора и выход из подпрограммы), сделать это можно посредством установки точке останова.

8. Характеристики основных команд отладчика gdb:

   - backtrace - вывод на экран пути к текущей точке останова;
   - break - установка точки останова (аргумент - номер строки или название функции);
   - clear - удаление всех точек останова в функции;
   - continue - продолжение выполнения программы;
   - delete - удаление точки останова;
   - display - добавление выражения в список выражений, значения которых отображаются при достижении точки останова;
   - finish - выполнение программы до момента выхода из функции;
   - info breakpoints - вывод на экран списка используемых точек останова;
   - info watchpoints - вывод на экран списка используемых контрольных выражений;

- list - вывод на экран исходного кода (аргумент - название файла и через “:” номер начальной и конечной строк);
- next - выполнение программы пошагово, но без выполнения вызываемых в программе функций;
- print - вывод значения указываемого в качестве параметра выражения;
- run - запуск программы на выполнение;
- set - установка нового значения переменной;
- step - пошаговое выполнение программы;
- watch - установка контрольного выражения, при изменении значения которого программа будет остановлена.

9. Схема отладки программы calcul:

    1) Выполнил компиляцию программы (с флагом -g).
    2) Просмотрел ошибки в программе.
    3) Исправил ошибки.
    4) Загрузил программу в отладчике gdb.
    5) Выполнил программу в отладчике (“run”).
    6) Ввёл некоторые значения.
    7) Установил точку останова.
    8) Проверил значения на момент остановки.
    9) Программа завершена -> отладчик gdb ошибок не видит.

10. Отладчик указал на неверный формат “%s” для &Operation, “%s” - символьный формат, так что ссылка не нужна (“&”), таким образом нужен только Operation.

11. Помогают понять исходный код программы такие средства как: cscope - исследует функции, содержащиеся в программе; splint - осуществляет критическую проверку программ, которые написаны на языке C.

12. Утилита splint осуществляет такие задачи как:

a. Проверка корректности задания аргументов всех использованных в программе функций и типов возвращаемых ими значений;
b. Поиск фрагментов исходного текста, корректных с точки зрения синтаксиса языка C, но малоэффективных с точки зрения их реализации или содержащих в себе семантические ошибки;
c. Общая оценка мобильности пользовательской программы.

3 Вывод

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