

ОТЧЕТ О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ №13

дисциплина: Операционные системы

Шилоносов Данил Вячеславович

Содержание

1	Цель работы	4
2	Задачи	5
3	Теоретическое введение	12
3.1	Этапы разработки приложений	12
3.2	Компиляция исходного текста и построение исполняемого файла	12
4	Выполнение лабораторной работы	14
4.1	Редактирование, компиляция и линковка исходных файлов . . .	14
4.2	Работа с отладчиком gdb	15
4.3	Утилита splint	18
5	Выводы	20

Список иллюстраций

4.1	Исходные файлы	14
4.2	Изменение Makefile	15
4.3	Компиляция и линковка	15
4.4	Запуск gdb	16
4.5	Запуск программы в отладчике gdb	16
4.6	Отображение calculate.c с 20 по 29 строку и установка точки останова на 21 строке	17
4.7	Отображение отладочной информации	18
4.8	splint calculate.c	19

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 Задачи

1. В домашнем каталоге создайте подкаталог ~/work/os/lab_prog.
2. Создайте 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. Реализация функций калькулятора в файле calculate.h:

```
////////////////////////////////////  
// 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);
}
}

```

Интерфейсный файл calculate.h, описывающий формат вызова функции-калькулятора:

```

////////////////////////////////////
// calculate.h

#ifndef CALCULATE_H_
#define CALCULATE_H_

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

#endif /*CALCULATE_H_*/

```

Основной файл main.c, реализующий 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("%.2f\n",Result);  
    return 0;  
}
```

3. Выполните компиляцию программы посредством gcc:

```
gcc -c calculate.c  
gcc -c main.c  
gcc calculate.o main.o -o calcul -lm
```

4. При необходимости исправьте синтаксические ошибки.

5. Создайте Makefile со следующим содержанием:


```

#
# 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

```

Поясните в отчёте его содержание.

6. С помощью gdb выполните отладку программы calcul (перед использованием gdb исправьте Makefile):

- Запустите отладчик GDB, загрузив в него программу для отладки:

```
gdb ./calcul
```

– Для запуска программы внутри отладчика введите команду run:

run

- Для постраничного (по 9 строк) просмотра исходного код используйте команду list:

list

- Для просмотра строк с 12 по 15 основного файла используйте list с параметрами:

list 12,15

- Для просмотра определённых строк не основного файла используйте list с параметрами:

list calculate.c:20,29

- Установите точку останова в файле calculate.c на строке номер 21:

list calculate.c:20,27

break 21

- Выведите информацию об имеющихся в проекте точка останова:

info breakpoints

- Запустите программу внутри отладчика и убедитесь, что программа останавливается в момент прохождения точки останова:

run

5

-

backtrace

- Отладчик выдаст следующую информацию:

```
#0 Calculate (Numeral=5, Operation=0x7fffffff280 "-")
at calculate.c:21
#1 0x00000000400b2b in main () at main.c:17
```

а команда `backtrace` покажет весь стек вызываемых функций от начала программы до текущего места.

- Посмотрите, чему равно на этом этапе значение переменной `Numeral`, введя:

```
print Numeral
```

На экран должно быть выведено число 5. - Сравните с результатом вывода на экран после использования команды:

```
display Numeral
```

- Уберите точки останова

```
info breakpoints
delete 1
```

7. С помощью утилиты `splint` попробуйте проанализировать коды файлов `calculate.c` и `main.c`.

3 Теоретическое введение

3.1 Этапы разработки приложений

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.

3.2 Компиляция исходного текста и построение исполняемого файла

Стандартным средством для компиляции программ в ОС типа UNIX является GCC (GNU Compiler Collection). Это набор компиляторов для разного рода язы-

ков программирования (C, C++, Java, Фортран и др.). Работа с GCC производится при помощи одноимённой управляющей программы gcc, которая интерпретирует аргументы командной строки, определяет и осуществляет запуск нужного компилятора для входного файла. 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. Для компиляции файла main.c, содержащего написанную на языке C простейшую программу:

```
/*  
* main.c  
*/  
#include <stdio.h>  
int main()  
{  
printf("Hello World!\n");  
return 0;  
}
```

достаточно в командной строке ввести:

```
gcc -c main.c
```

4 Выполнение лабораторной работы

4.1 Редактирование, компиляция и линковка исходных файлов

Создадим файлы `calculate.c`, `calculate.h`, `main.c` и `Makefile`: (рис. [4.1])

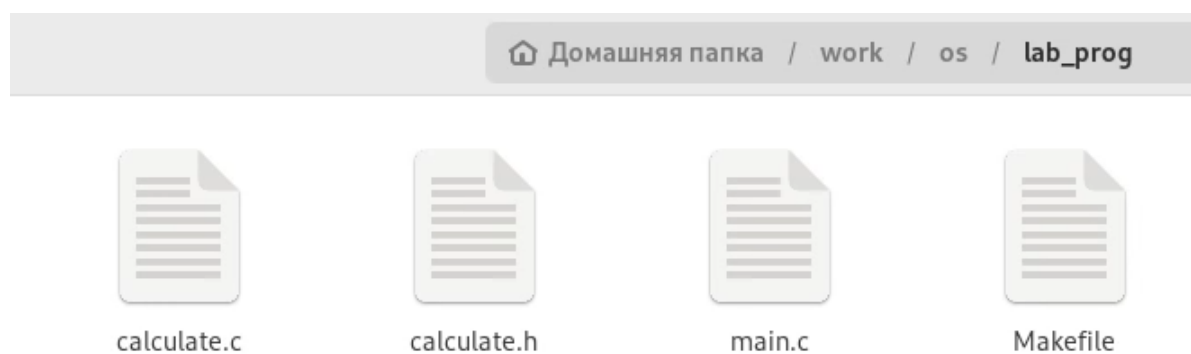


Рис. 4.1: Исходные файлы

Далее, вставим в эти файлы соответствующий код из условия задания.

Исправим описание `Makefile` (добавим опцию `-g` для того, чтобы сохранялась отладочная информация): (рис. [4.2])


<pre> 1 # 2 # Makefile 3 # 4 5 CC = gcc 6 CFLAGS = 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 </pre>		<pre> 1 # 2 # Makefile 3 # 4 5 CC = gcc 6 CFLAGS = -g 7 LIBS = -lm 8 9 calcul: calculate.o main.o 10 \$(CC) calculate.o main.o -o calcul \$(LIBS) 11 12 calculate.o: calculate.c calculate.h 13 \$(CC) -c calculate.c \$(CFLAGS) 14 15 main.o: main.c calculate.h 16 \$(CC) -c main.c \$(CFLAGS) 17 18 clean: 19 -rm calcul *.o *~ 20 21 # End Makefile 22 </pre>
--	---	---

Рис. 4.2: Изменение Makefile

С помощью компилятора GCC скомпилируем объектных файлы, затем слинкуем их в исполняемую программу: (рис. [4.3])

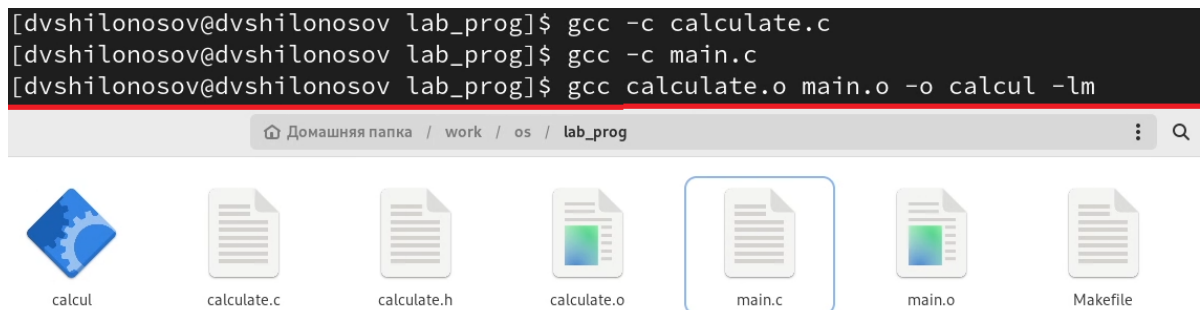


Рис. 4.3: Компиляция и линковка

4.2 Работа с отладчиком gdb

Запустим отладчик gdb: (рис. [4.4])

```
[dvshilonosov@dvshilonosov lab_prog]$ gdb ./calcul
GNU gdb (GDB) Fedora Linux 13.1-3.fc37
Copyright (C) 2023 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:
<https://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) █
```

Рис. 4.4: Запуск gdb

В отладчик запустим программу и введем требуемые значения (рис. [4.5])

```
(gdb) run
Starting program: /home/dvshilonosov/work/os/lab_prog/calcul

This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.fedoraproject.org/>
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): pow
Степень: 3
125.00
[Inferior 1 (process 6075) exited normally]
```

Рис. 4.5: Запуск программы в отладчике gdb

С помощью команды

list

отобразим нужную для просмотра часть кода, а затем, с помощью команды break поставим точку останова: (рис. [4.6])

```
(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) list calculate.c:20,27
20     {
21     printf("Вычитаемое: ");
22     scanf("%f",&SecondNumeral);
23     return(Numeral - SecondNumeral);
24     }
25     else if(strncmp(Operation, "*", 1) == 0)
26     {
27     printf("Множитель: ");
(gdb) break 21
Breakpoint 1 at 0x40120f: file calculate.c, line 21.
(gdb) info breakpoints
Num      Type          Disp Enb Address                  What
1        breakpoint    keep y  0x000000000040120f in Calculate at calculate.c:21
```

Рис. 4.6: Отображение calculate.c с 20 по 29 строку и установка точки останова на 21 строке

Вновь запустив программу, видим дополнительную информацию, касающуюся ее выполнения: (рис. [4.7])

```

(gdb) run
Starting program: /home/dvshilonosov/work/os/lab_prog/calcul
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): -

Breakpoint 1, Calculate (Numeral=5, Operation=0x7fffffffdf24 "-") at calculate.c:21
21      printf("Вычитаемое: ");
(gdb) backtrace
#0 Calculate (Numeral=5, Operation=0x7fffffffdf24 "-") at calculate.c:21
#1 0x00000000004014eb in main () at main.c:16
(gdb) print Numeral
$1 = 5
(gdb) display Numeral
1: Numeral = 5

```

Рис. 4.7: Отображение отладочной информации

4.3 Утилита splint

Запустим с помощью утилиты splint файл calculate.c и увидим информацию, которые содержит предупреждения о программе: (рис. [4.8])

```
[dvshilonosov@dvshilonosov lab_prog]$ splint calculate.c

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:22:1: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:28:1: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:34:1: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:35:4: 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:38:7: Return value type double does not match declared type float:
    (HUGE_VAL)
To allow all numeric types to match, use +relaxtypes.
calculate.c:46:1: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:47:7: Return value type double does not match declared type float:
    (pow(Numeral, SecondNumeral))
calculate.c:50:7: Return value type double does not match declared type float:
    (sqrt(Numeral))
calculate.c:52:7: Return value type double does not match declared type float:
    (sin(Numeral))
calculate.c:54:7: Return value type double does not match declared type float:
    (cos(Numeral))
calculate.c:56:7: Return value type double does not match declared type float:
    (tan(Numeral))
calculate.c:60:7: Return value type double does not match declared type float:
    (HUGE_VAL)
```

Рис. 4.8: splint calculate.c

5 Выводы

В процессе выполнения лабораторной работы были получены простейшие навыки разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linux на примере создания на языке программирования C калькулятора с простейшими функциями.