

Лабораторная работа №13

Дисциплина - операционные системы

Волгин Иван Алексеевич

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	9
5	Выводы	15

Список иллюстраций

4.1	Создание подкаталога	9
4.2	Создание файлов	9
4.3	Код calculate.c	10
4.4	Код calculate.h	10
4.5	Код main.c	11
4.6	Компиляция программы	11
4.7	Создание Makefile	11
4.8	Код Makefile	12
4.9	Запускаем gdb	12
4.10	Запускаем программу внутри отладчика	13
4.11	Анализ кода calculate.c	13
4.12	Анализ кода main.c	14

Список таблиц

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

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.
3. Выполните компиляцию программы посредством `gcc`:
4. При необходимости исправьте синтаксические ошибки.
5. Создайте `Makefile`.
6. С помощью `gdb` выполните отладку программы `calcul` (перед использованием `gdb` исправьте `Makefile`):
7. С помощью утилиты `splint` попробуйте проанализировать коды файлов `calculate.c` и `main.c`.

3 Теоретическое введение

1. 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.
2. Стандартным средством для компиляции программ в ОС типа UNIX является GCC (GNU Compiler Collection). Это набор компиляторов для разного рода языков 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.

Некоторые опции компиляции в gcc Опция | Описание -c | компиляция без компоновки — создаются объектные файлы file.o -o | file-name задать имя file-

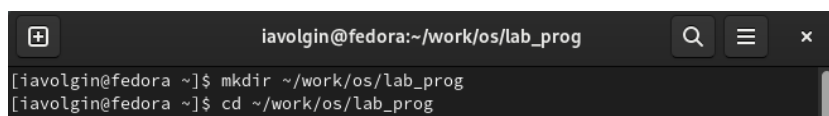
name создаваемому файлу -g | поместить в файл (объектный или исполняемый) отладочную информацию для отладчика gdb -MM | вывести зависимости от заголовочных файлов C и/или C++ программ в формате, подходящем для утилиты make; при этом объектные или исполняемые файлы не будут созданы -Wall | вывод на экран сообщений об ошибках, возникших во время компиляции

3. 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)

Некоторые команды gdb Команда | Описание действия 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

4 Выполнение лабораторной работы

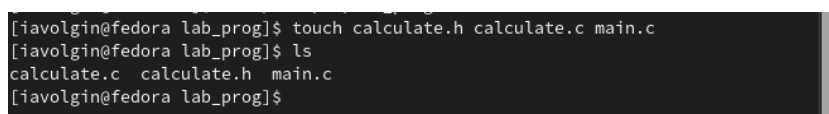
Сперва нужно было создать нужный подкаталог `~/work/os/lab_prog` (рис. 4.1).

A terminal window titled 'iavolgin@fedora:~/work/os/lab_prog'. The prompt is '[iavolgin@fedora ~]\$'. The first command entered is 'mkdir ~/work/os/lab_prog'. The second command is 'cd ~/work/os/lab_prog'.

```
iavolgin@fedora:~/work/os/lab_prog
[iavolgin@fedora ~]$ mkdir ~/work/os/lab_prog
[iavolgin@fedora ~]$ cd ~/work/os/lab_prog
```

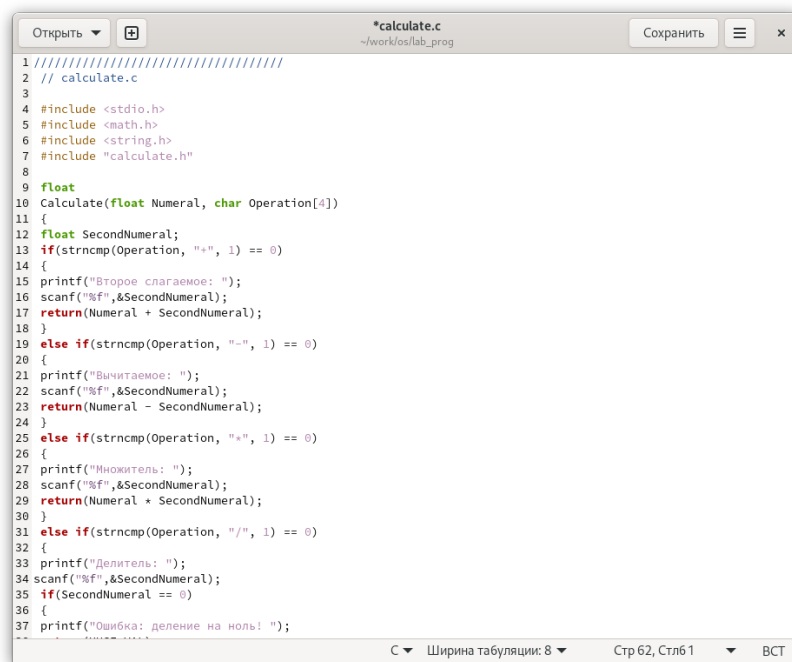
Рис. 4.1: Создание подкаталога

Затем создать в нем файлы `calculate.h`, `calculate.c`, `main.c` (рис. 4.2) и ввести код в эти файлы: `calculate.c` (рис. 4.3), `calculate.p` (рис. 4.4), `main.c` (рис. 4.5)

A terminal window showing the creation of files. The prompt is '[iavolgin@fedora lab_prog]\$'. The command entered is 'touch calculate.h calculate.c main.c'. The next prompt is '[iavolgin@fedora lab_prog]\$' followed by 'ls', which outputs 'calculate.c calculate.h main.c'.

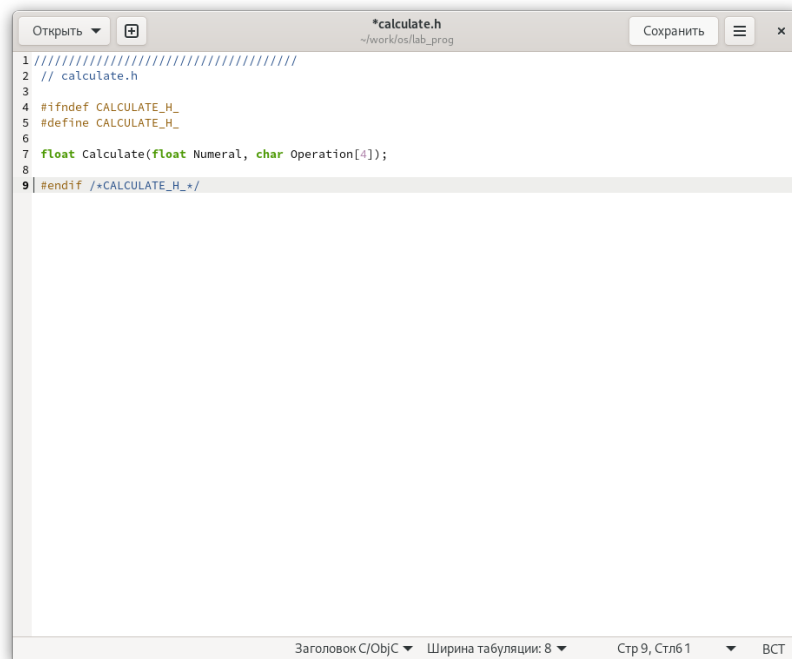
```
[iavolgin@fedora lab_prog]$ touch calculate.h calculate.c main.c
[iavolgin@fedora lab_prog]$ ls
calculate.c calculate.h main.c
[iavolgin@fedora lab_prog]$
```

Рис. 4.2: Создание файлов



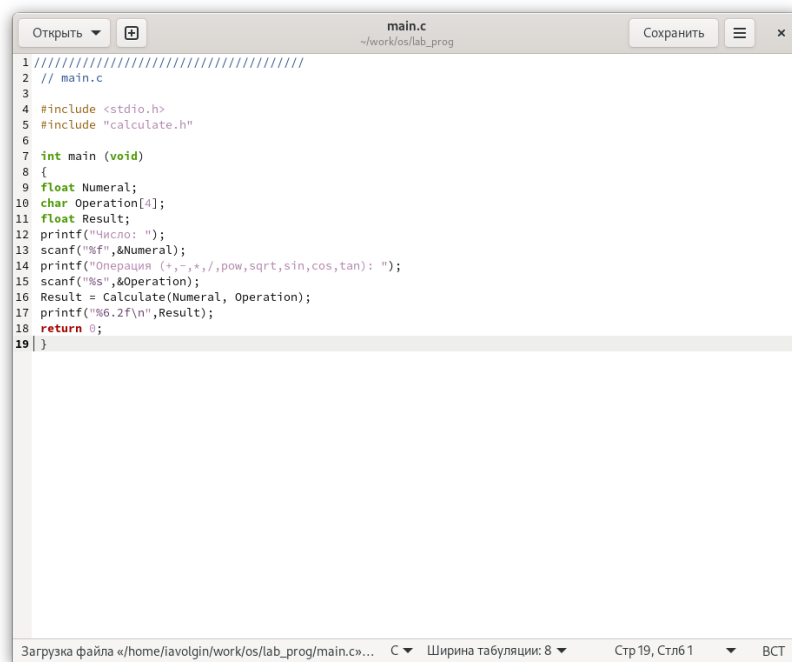
```
1 //////////////////////////////////////////////////
2 // calculate.c
3
4 #include <stdio.h>
5 #include <math.h>
6 #include <string.h>
7 #include "calculate.h"
8
9 float
10 Calculate(float Numeral, char Operation[4])
11 {
12     float SecondNumeral;
13     if(strncmp(Operation, "+", 1) == 0)
14     {
15         printf("Второе слагаемое: ");
16         scanf("%f",&SecondNumeral);
17         return(Numeral + SecondNumeral);
18     }
19     else if(strncmp(Operation, "-", 1) == 0)
20     {
21         printf("Вычитаемое: ");
22         scanf("%f",&SecondNumeral);
23         return(Numeral - SecondNumeral);
24     }
25     else if(strncmp(Operation, "*", 1) == 0)
26     {
27         printf("Имножитель: ");
28         scanf("%f",&SecondNumeral);
29         return(Numeral * SecondNumeral);
30     }
31     else if(strncmp(Operation, "/", 1) == 0)
32     {
33         printf("Делитель: ");
34         scanf("%f",&SecondNumeral);
35         if(SecondNumeral == 0)
36         {
37             printf("Ошибка: деление на ноль! ");
38         }
39     }
40 }
```

Рис. 4.3: Код calculate.c



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

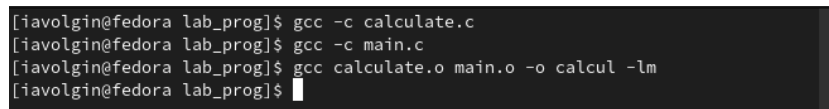
Рис. 4.4: Код calculate.h



```
1 //////////////////////////////////////////////////
2 // main.c
3
4 #include <stdio.h>
5 #include "calculate.h"
6
7 int main (void)
8 {
9     float Numeral;
10    char Operation[4];
11    float Result;
12    printf("Число: ");
13    scanf("%f",&Numeral);
14    printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): ");
15    scanf("%s",&Operation);
16    Result = Calculate(Numeral, Operation);
17    printf("%6.2f\n",Result);
18    return 0;
19 }
```

Рис. 4.5: Код main.c

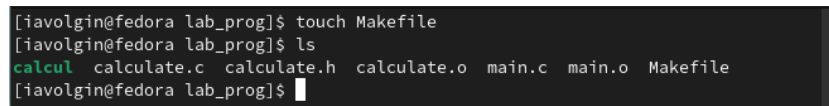
Далее нужно было выполнить компиляцию программы посредством gcc (рис. 4.6).



```
[iavolgin@fedora lab_prog]$ gcc -c calculate.c
[iavolgin@fedora lab_prog]$ gcc -c main.c
[iavolgin@fedora lab_prog]$ gcc calculate.o main.o -o calcul -lm
[iavolgin@fedora lab_prog]$
```

Рис. 4.6: Компиляция программы

Затем нужно было проверить код на синтаксические ошибки, я их не нашел. После этого создать Makefile (рис. 4.7) и написать там следующий код (рис. 4.8).



```
[iavolgin@fedora lab_prog]$ touch Makefile
[iavolgin@fedora lab_prog]$ ls
calcul calculate.c calculate.h calculate.o main.c main.o Makefile
[iavolgin@fedora lab_prog]$
```

Рис. 4.7: Создание Makefile

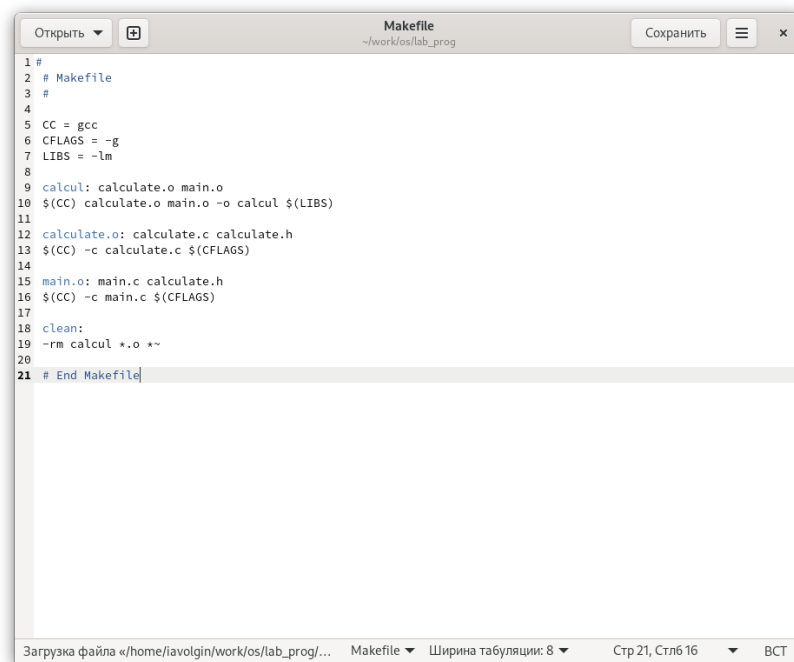


Рис. 4.8: Код Makefile

Далее я запустил gdb, загрузив в него программ отладки (рис. 4.9) и затем запустил саму программу уже внутри отладчика (рис. 4.10).

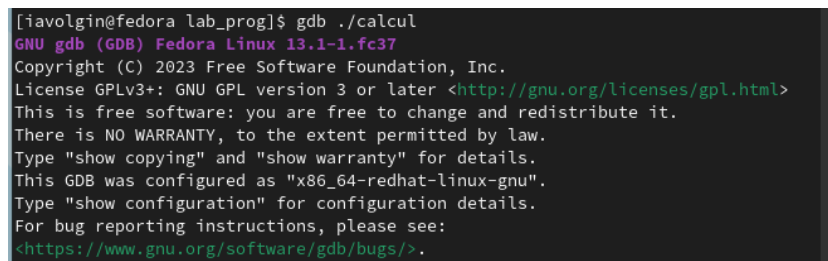


Рис. 4.9: Запускаем gdb

```
(gdb) run
Starting program: /home/iavolgin/work/os/lab_prog/calcul
Downloading separate debug info for system-supplied DSO at 0x7ffff7fc6000
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".
Число: 3
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): *
Множитель: 5
15.00
[Inferior 1 (process 4534) exited normally]
(gdb)
```

Рис. 4.10: Запускаем программу внутри отладчика

После этого я попытался выполнить следующие задания и у меня никак не получалось и я не мог найти в чем проблема, поэтому я перешел к последнему пункту, где с помощью утилиты splint нужно было проанализировать коды файлов calculate.c (рис. 4.11) и main.c (рис. 4.12).

```
iavolgin@fedora:~/work/os/lab_prog
[iavolgin@fedora lab_prog]$ splint calculate.c
Splint 3.1.2 --- 23 Jul 2022

calculate.h:7:38: 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)
calculate.c:10:32: Function parameter Operation declared as manifest array
(size constant is meaningless)
calculate.c: (in function Calculate)
calculate.c:16:2: 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:22:2: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:28:2: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:34:1: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:35:5: 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:38:8: Return value type double does not match declared type float:
```

Рис. 4.11: Анализ кода calculate.c

Рис. 4.12: Анализ кода main.c

5 Выводы

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