

Лабораторная работа №14

Операционные системы

Александрова Ульяна

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	9
5	Выводы	13
6	Контрольные вопросы	14

Список иллюстраций

4.1	Создание файлов	9
4.2	calculate.h	9
4.3	calculate.c	10
4.4	main.c	11
4.5	Компиляция	11
4.6	Makefile	11
4.7	Отладка	12
4.8	Просмотр кода	12

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` со следующим содержанием.
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

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.

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.

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).

Ещё одним средством проверки исходных кодов программ, написанных на языке C, является утилита splint. Эта утилита анализирует программный код, проверяет корректность задания аргументов использованных в программе функций и типов возвращаемых значений, обнаруживает синтаксические и семантические ошибки. В отличие от компилятора C анализатор splint генерирует комментарии с описанием разбора кода программы и осуществляет общий контроль, обнаруживая такие ошибки, как одинаковые объекты, определённые в разных файлах, или объекты, чьи значения не используются в работе программы, переменные с некорректно заданными значениями и типами и многое другое.

4 Выполнение лабораторной работы

Создала требуемый репозиторий и файлы, необходимые для выполнения работы (рис. fig. 4.1).

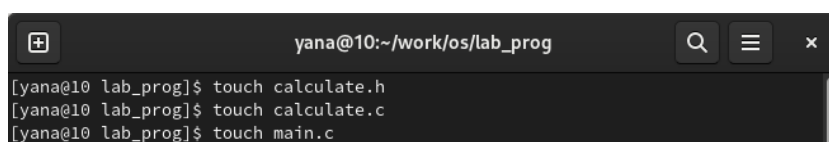
A terminal window with a dark background. The title bar shows 'yana@10:~/work/os/lab_prog'. The terminal contains three lines of commands: 'touch calculate.h', 'touch calculate.c', and 'touch main.c', each preceded by a prompt '[yana@10 lab_prog]\$'.

Рис. 4.1: Создание файлов

Записываю реализацию функций калькулятора сначала в файле calculate.h (рис. fig. 4.2).

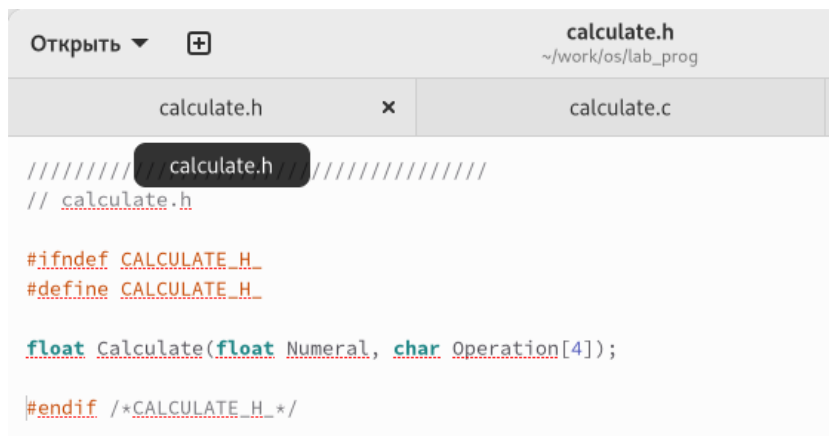
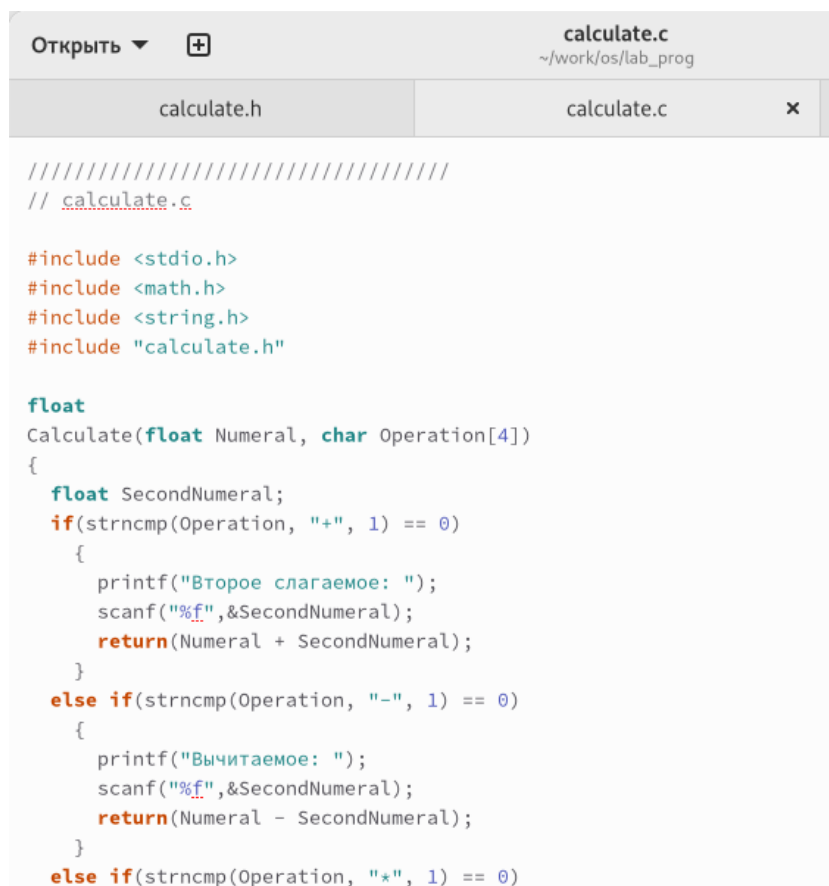
A code editor window with a light gray background. The title bar shows 'calculate.h' and the path '~/work/os/lab_prog'. Below the title bar, there are two tabs: 'calculate.h' and 'calculate.c'. The 'calculate.h' tab is active, showing the following code: '////////// calculate.h //////////', '// calculate.h', '#ifndef CALCULATE_H', '#define CALCULATE_H', 'float Calculate(float Numeral, char Operation[4]);', and '#endif /*CALCULATE_H_*/'. The code is color-coded: keywords are in red, types in blue, and identifiers in black.

Рис. 4.2: calculate.h

Затем в calculate.c (рис. fig. 4.3).



```
////////////////////////////////////
// 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)
```

Рис. 4.3: calculate.c

И в main.c (рис. fig. 4.4).



```
main.c
~/work/os/lab_prog

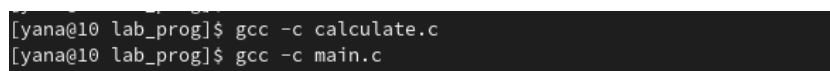
calculate.h  calculate.c

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

Рис. 4.4: main.c

Выполняю компиляцию программы посредством gcc (рис. fig. 4.5).



```
[yana@10 lab_prog]$ gcc -c calculate.c
[yana@10 lab_prog]$ gcc -c main.c
```

Рис. 4.5: Компиляция

Создаю и заполняю файл Makefile (рис. fig. 4.6).



```
Makefile
~/work/os/lab_prog

calculate.h  calculate.c  main.c  Makefile

#
# 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 *~
```

Рис. 4.6: Makefile

Запустила отладчик gdb, программа работает (рис. fig. 4.7).

```
yana@10:~/work/os/lab_prog — g... x yana@10:~/work/os/lab_prog x
(gdb) n
The program is not being run.
(gdb) run
Starting program: /home/yana/work/os/lab_prog/calcul
Downloading separate debug info for /home/yana/work/os/lab_prog/system-sup
DSO at 0x7ffff7fc4000...

Downloading separate debug info for /lib64/libm.so.6...
Downloading separate debug info for /lib64/libc.so.6...
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".
Число: 12
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): -
Вычитаемое: 3
9.00
[Inferior 1 (process 3080) exited normally]
```

Рис. 4.7: Отладка

Для постраничного просмотра исходного код использовала команду list (рис. fig. 4.8).

```
yana@10:~/work/os/lab_prog — g... x yana@10:~/work/os/lab_prog x
(gdb) list
Downloading source file /usr/src/debug/glibc-2.35-4.fc36.x86_64/elf/sofin
1      /* Terminate the frame unwind info section with a 4byte 0 as a ser
2      this would be the 'length' field in a real FDE. */
3
4      typedef unsigned int ui32 __attribute__ ((mode (SI)));
5      static const ui32 __FRAME_END__[1]
6      __attribute__ ((used, section (".eh_frame")))
7      = { 0 };
```

Рис. 4.8: Просмотр кода

5 Выводы

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

6 Контрольные вопросы

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

info, man.

2. Назовите и дайте краткую характеристику основным этапам разработки приложений в UNIX.

- создание исходного кода программы;
- представляется в виде файла;
- сохранение различных вариантов исходного текста;
- анализ исходного текста;
- компиляция исходного текста и построение исполняемого модуля;
- тестирование и отладка;
- сохранение всех изменений, выполняемых при тестировании и отладке.

3. Что такое суффикс в контексте языка программирования? Приведите примеры использования.

Использование суффикса “.c” для имени файла с программой на языке Си отражает удобное и полезное соглашение, принятое в ОС UNIX. Для любого имени входного файла суффикс определяет какая компиляция требуется. Суффиксы и префиксы указывают тип объекта.

4. Каково основное назначение компилятора языка C в UNIX?

Основное назначение компилятора с языка Си заключается в компиляции всей программы в целом.

5. Для чего предназначена утилита make?

Программа make служит для документирования взаимосвязей между файлами. Описание взаимосвязей и соответствующих действий хранится в так называемом make-файле, который по умолчанию имеет имя makefile или Makefile.

6. Приведите пример структуры Makefile. Дайте характеристику основным элементам этого файла.
7. Назовите основное свойство, присущее всем программам отладки. Что необходимо сделать, чтобы его можно было использовать?
8. Назовите и дайте основную характеристику основным командам отладчика gdb.
9. Опишите по шагам схему отладки программы, которую Вы использовали при выполнении лабораторной работы.
10. Прокомментируйте реакцию компилятора на синтаксические ошибки в программе при его первом запуске.
11. Назовите основные средства, повышающие понимание исходного кода программы.
12. Каковы основные задачи, решаемые программой splint