# Лабораторная работа №13

## Лабораторная работа № 13. Средства, применяемые при разработке программного обеспечения в ОС типа UNIX/Linux

Старовойтов Егор Сергеевич

## Содержание

## 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.

## 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:

```
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:

```
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 *~
```

Поясните в отчёте его содержание. 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.

# Теоретическое введение

## 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.

# Ход работы

## Шаг 1

В домашнем каталоге создан подкаталог ~/work/os/lab_prog

```
[liveuser@localhost-live ~]$ mkdir work
[liveuser@localhost-live ~]$ cd work
[liveuser@localhost-live work]$ mkdir os
[liveuser@localhost-live work]$ mkdir lab_prog
[liveuser@localhost-live work]$ cd lab_prog
[liveuser@localhost-live lab_prog]$
```

*Создание подкаталога*

## Шаг 2

В каталоге ~/work/os/lab_prog созданы файлы calculate.h, calculate.c, main.c.

```
[liveuser@localhost-live ~]$ mkdir work
[liveuser@localhost-live ~]$ cd work
[liveuser@localhost-live work]$ mkdir os
[liveuser@localhost-live work]$ mkdir lab_prog
[liveuser@localhost-live work]$ cd lab_prog
[liveuser@localhost-live lab_prog]$ touch calculate.h
[liveuser@localhost-live lab_prog]$ touch calculate.c
[liveuser@localhost-live lab_prog]$ touch main.c
[liveuser@localhost-live lab_prog]$
```

*Создание файлов*

```
[liveuser@localhost-live lab_prog]$ cat calculate.h
#ifndef CALCULATE_H_
#define CALCULATE_H_

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

#endif /*CALCULATE_H_*/
[liveuser@localhost-live lab_prog]$
```

*calculate.h*

```
[liveuser@localhost-live lab_prog]$ cat 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("Vtoroe slagaemoe");
        scanf("%f", &SecondNumeral);
        return Numeral + SecondNumeral;
    } else if (strncmp(Operation, "-", 1) == 0) {
        printf("Vichitaemoe");
        scanf("%f", &SecondNumeral);
        return Numeral - SecondNumeral;
    } else if (strncmp(Operation, "*", 1) == 0) {
        printf("Mnojitel");
        scanf("%f", &SecondNumeral);
        return Numeral * SecondNumeral;
    } else if (strncmp(Operation, "/", 1) == 0) {
        printf("Delitel");
        scanf("%f", &SecondNumeral);
        if (SecondNumeral == 0) {
            printf("Oshibka: delenie na 0");
            return HUGE_VAL;
        } else {
            return Numeral / SecondNumeral;
        }
        return Numeral - SecondNumeral;
    } else if (strncmp(Operation, "pow", 3) == 0) {
        printf("Stepen: ");
        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("Nepravilno vvedeno deystvie");
```

*calculate.c*

```
[liveuser@localhost-live lab_prog]$ cat main.c
#include <stdio.h>
#include "calculate.h"

int main(void) {
    float Numeral;
    char Operation[4];
    float Result;
    printf("Chislo: ");
    scanf("%f", &Numeral);
    printf("Operaciya (+, -, *, /, pow, sqrt, sin, cos, tan):");
    scanf("%s", &Operation);
    Result = Calculate(Numeral, Operation);
    printf("%6.2f\n", Result);
    return 0;
}
[liveuser@localhost-live lab_prog]$
```

*main.c*

## Шаг 3

Я выполнил компиляцию программы.

```
[liveuser@localhost-live lab_prog]$ gcc -c calculate.c
[liveuser@localhost-live lab_prog]$ gcc -c main.c
[liveuser@localhost-live lab_prog]$ gcc calculate.o main.o -o calcul -lm
```

*Компиляция программы*

## Шаг 4

Исправил синтаксические ошибки (опечатки)

## Шаг 5

Создал Makefile

```
[liveuser@localhost-live lab_prog]$ cat 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 *~
[liveuser@localhost-live lab_prog]$
```

*Makefile*

## Шаг 6

С помощью gdb выполнил отладку.

```
[liveuser@localhost-live lab_prog]$ touch Makefile
[liveuser@localhost-live lab_prog]$ vi Makefile
[liveuser@localhost-live lab_prog]$ gdb ./calcul
GNU gdb (GDB) Fedora 10.2-9.fc35
Copyright (C) 2021 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...
(No debugging symbols found in ./calcul)
(gdb) run
Starting program: /home/liveuser/work/lab_prog/calcul

Downloading separate debug info for /home/liveuser/work/lab_prog/system-supplied DSO at 0x7ffff7fc9000...
Downloading separate debug info for /lib64/libm.so.6...
Downloading separate debug info for /lib64/libc.so.6...
list
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".
Chislo: Operaciya (+, -, *, /, pow, sqrt, sin, cos, tan):Nepravilno vvedeno deystvie   inf
[Inferior 1 (process 31875) exited normally]
```

*Отладка*

## Шаг 7

С помощью утилиты splint проанализировал исходный код файлов calculate.c и main.c, увидел 16 предупреждений связанных с преобразованием типов в calculate.c.

```
[liveuser@localhost-live lab_prog]$ splint calculate.c
Splint 3.1.2 --- 23 Jul 2021

calculate.h:4:37: 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:6:37: Function parameter Operation declared as manifest array (size
                    constant is meaningless)
calculate.c: (in function Calculate)
calculate.c:10:9: 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:14:9: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:18:9: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:22:9: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:23:13: 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:25:20: Return value type double does not match declared type float:
                     HUGE_VAL
  To allow all numeric types to match, use +relaxtypes.
calculate.c:29:16: Unreachable code: return Numeral -...
  This code will never be reached on any possible execution. (Use -unreachable
  to inhibit warning)
calculate.c:32:9: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:33:16: Return value type double does not match declared type float:
                     pow(Numeral, SecondNumeral)
calculate.c:35:16: Return value type double does not match declared type float:
                     sqrt(Numeral)
calculate.c:37:16: Return value type double does not match declared type float:
                     sin(Numeral)
calculate.c:39:16: Return value type double does not match declared type float:
                     cos(Numeral)
calculate.c:41:16: Return value type double does not match declared type float:
                     tan(Numeral)
calculate.c:44:16: Return value type double does not match declared type float:
                     HUGE_VAL

Finished checking --- 16 code warnings
[liveuser@localhost-live lab_prog]$ split main.c
[liveuser@localhost-live lab_prog]$
```

*Анализ исходного кода*

## Вывод

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

## Контрольные вопросы

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

Спросить в интернете или использовать утилиту man, также можно использовать опцию -h у gcc для получения дополнительной информации.

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

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

Суффикс - это составная часть имени файла, например его расширение.

## 4. Каково основное назначение компилятора языка C в UNIX?

Преобразование исходного кода программ в объектные файлы.

## 5. Для чего предназначена утилита make?

Утилита make позволяет автоматизировать процесс преобразования файлов программы из одной формы в другую, отслеживает взаимосвязи между файлами.

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

В самом простом случае Makefile имеет следующий синтаксис:

```
<цель_1> <цель_2> ... : <зависимость_1> <зависимость_2> ...
<команда 1>
...
<команда n>
```

Сначала задаётся список целей, разделённых пробелами, за которым идёт двоеточие и список зависимостей. Затем в следующих строках указываются команды. Строки с командами обязательно должны начинаться с табуляции.

В качестве цели в Makefile может выступать имя файла или название какого-то действия. Зависимость задаёт исходные параметры (условия) для достижения указанной цели. Зависимость также может быть названием какого-то действия. Команды — собственно действия, которые необходимо выполнить для достижения цели. Рассмотрим пример Makefile для написанной выше простейшей программы, выводящей на экран приветствие ’Hello World!’:

```
hello: main.c
gcc -o hello main.c
```

Здесь в первой строке hello — цель, main.c — название файла, который мы хотим скомпилировать; во второй строке, начиная с табуляции, задана команда компиляции gcc с опциями. Для запуска программы необходимо в командной строке набрать команду make:

```
make
```

```
Общий синтаксис Makefile имеет вид:
1 target1 [target2...]:[:] [dependment1...]
2 [(tab)commands] [#commentary]
3 [(tab)commands] [#commentary]
```

Здесь знак # определяет начало комментария (содержимое от знака # и до конца строки не будет обрабатываться. Одинарное двоеточие указывает на то, что последовательность команд должна содержаться в одной строке. Для переноса можно в длинной строке команд можно использовать обратный слэш (\). Двойное двоеточие указывает на то, что последовательность команд может содержаться в нескольких последовательных строках

## 7. Назовите основное свойство, присущее всем программам отладки. Что необходимо сделать, чтобы его можно было использовать?

Возможность останавливать выполнение программы на определенных строчках кода. Для этого нужно установить так называемые брейкпоинты.

## 8. Назовите и дайте основную характеристику основным командам отладчика gdb.

Установка брейкопинтов, пошаговое выполнение отлаживаемой программы, вывод исходного кода постранично или построчно, возможность узнать значение переменных.

## 9. Опишите по шагам схему отладки программы, которую Вы использовали при выполнении лабораторной работы.

1. Установка брейкпоинтов
2. Пошаговое исполнение программы и вывод значения переменных

## 10. Прокомментируйте реакцию компилятора на синтаксические ошибки в программе при его первом запуске.

Процесс компиляции аварийно завершается, указывая на ошибки.

## 11. Назовите основные средства, повышающие понимание исходного кода программы.

Комментарии, единый стиль кода, линтеры.

## 12. Каковы основные задачи, решаемые программой splint?

Увидеть ошибки и предупреждения, указывающие на различные проблемы в коде программы.