

Отчет по лабораторной работе 13

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

Шалыгин Георгий Эдуардович, НФИбд-02-20

Содержание

1	Цель работы	4
2	Техническое обеспечение:	5
3	Условные обозначения и термины:	6
4	Теоретическое введение:	7
5	Выполнение лабораторной работы	10
6	Выводы	15
7	Библиография	16

List of Figures

5.1	Текст calculate.c	10
5.2	Текст calculate.h	10
5.3	Текст main.c	11
5.4	Компиляция	11
5.5	Запуск	12
5.6	Текст Makefile	12
5.7	Компиляция с использование мэйкфайла	13
5.8	Отладка с помощью gdb	13
5.9	Установка splint	13
5.10	Работа splint	14

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.

Объект исследования: система UNIX.

Предмет исследования: программирование в UNIX.

2 Техническое обеспечение:

- Характеристики техники: AMD Ryzen 5 3500U 2.1 GHz, 8 GB оперативной памяти, 50 GB свободного места на жёстком диске;
- ОС Windows 10 Home
- Git 2.31.1
- Google Chrome 91.0.4472.19
- VirtualBox 2.0
- CentOS 7

3 Условные обозначения и термины:

Текстовым редактором(text editor) называют программу, которая предназначена для редактирования (составления и изменения) файлов, содержащих только текст. [1]

Командный язык - это язык, на котором пользователь взаимодействует с системой в интерактивном режиме.

Командный интерпретатор, интерпретатор командной строки - компьютерная программа, часть операционной системы, обеспечивающая базовые возможности управления компьютером посредством интерактивного ввода команд через интерфейс командной строки или последовательного исполнения пакетных командных файлов.[3]

Подробнее в [2] и [3].

4 Теоретическое введение:

Командный процессор (командная оболочка, интерпретатор команд shell) — это программа, позволяющая пользователю взаимодействовать с операционной системой компьютера. В операционных системах типа UNIX/Linux наиболее часто используются следующие реализации командных оболочек:

- оболочка Борна (Bourne shell или sh) — стандартная командная оболочка UNIX/Linux, содержащая базовый, но при этом полный набор функций;
- C-оболочка (или csh) — надстройка на оболочкой Борна, использующая Сподобный синтаксис команд с возможностью сохранения истории выполнения команд;
- оболочка Корна (или ksh) — напоминает оболочку C, но операторы управления программой совместимы с операторами оболочки Борна;
- BASH — сокращение от Bourne Again Shell (опять оболочка Борна), в основе своей совмещает свойства оболочек C и Корна (разработка компании Free Software Foundation).

POSIX (Portable Operating System Interface for Computer Environments) — набор стандартов описания интерфейсов взаимодействия операционной системы и прикладных программ.

Стандарты POSIX разработаны комитетом IEEE (Institute of Electrical and Electronics Engineers) для обеспечения совместимости различных UNIX/Linuxподобных операционных систем и переносимости прикладных программ на уровне исходного кода. POSIX-совместимые оболочки разработаны на базе оболочки Корна. Рассмотрим основные элементы программирования в оболочке bash. В других оболочках большинство команд будет совпадать с описанными ниже. [3]

Переменные в языке программирования bash

Командный процессор bash обеспечивает возможность использования переменных типа строка символов. Имена переменных могут быть выбраны пользователем. Пользователь имеет возможность присвоить переменной значение некоторой строки символов. Например, команда `mark=/usr/andy/bin` присваивает значение строки символов `/usr/andy/bin` переменной `mark` типа строка символов.

Использование значения, присвоенного некоторой переменной, называется подстановкой. Для того чтобы имя переменной не сливалось с символами, которые могут следовать за ним в командной строке, при подстановке в общем случае используется следующая форма записи: `${имя переменной}`. [3]

Использование арифметических вычислений. Операторы `let` и `read`

Оболочка bash поддерживает встроенные арифметические функции. Команда `let` является показателем того, что последующие аргументы представляют собой выражение, подлежащее вычислению. Простейшее выражение — это единичный терм (`term`), обычно целочисленный.

Команда `let` берет два операнда и присваивает их переменной. Положительным моментом команды `let` можно считать то, что для идентификации переменной ей не нужен знак доллара; вы можете писать команды типа `let sum=x+7`, и `let` будет искать переменную `x` и добавлять к ней 7.

Команда `read` позволяет читать значения переменных со стандартного ввода:

```
echo "Please enter Month and Day of Birth ?"
```

```
read mon day trash [3]
```

Командные файлы и функции

Последовательность команд может быть помещена в текстовый файл. Такой файл называется командным. Далее этот файл можно выполнить по команде: `bash командный_файл [аргументы]`

Чтобы не вводить каждый раз последовательности символов bash, необходимо изменить код защиты этого командного файла, обеспечив доступ к этому файлу по выполнению. Это может быть сделано с помощью команды `chmod +x имя_файла`

Передача параметров в командные файлы и специальные переменные

Пусть к командному файлу `where` имеется доступ по выполнению и этот командный файл содержит следующий конвейер: `who | grep $1`.

В ходе интерпретации файла командным процессором вместо комбинации символов `$1` осуществляется подстановка значения первого и единственного параметра `andy`.

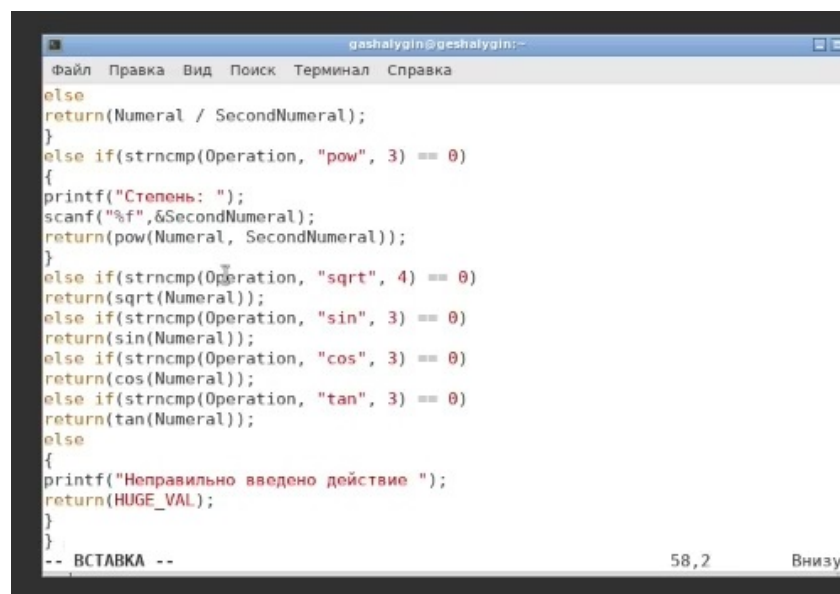
Команда `shift` позволяет удалять первый параметр и сдвигает все остальные на места предыдущих. При использовании в командном файле комбинации символов `$#` вместо неё будет осуществлена подстановка числа параметров, указанных в командной строке при вызове данного командного файла на выполнение.

Оператор цикла `for`

В обобщённой форме оператор цикла `for` выглядит следующим образом: `for имя [in список-значений] do список-команд done`. [3]

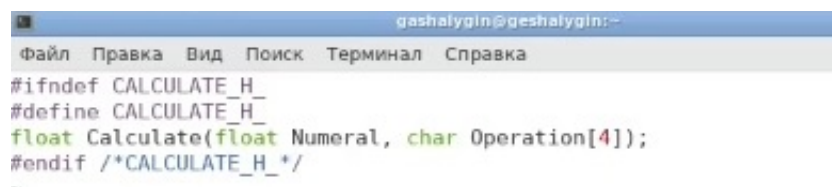
5 Выполнение лабораторной работы

1. Создадим 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. (рис. 5.1, 5.2, 5.3).



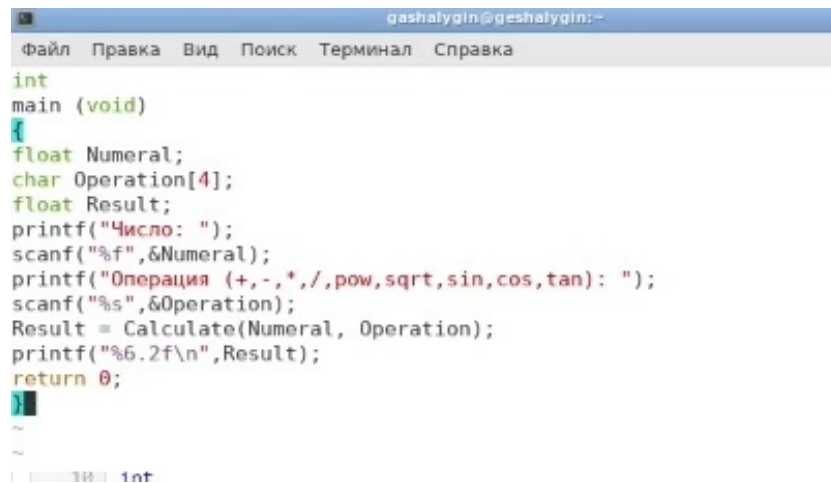
```
gashalygin@gashalygin:~  
Файл Правка Вид Поиск Терминал Справка  
else  
return(Numeral / SecondNumeral);  
}  
else if(strncmp(Operation, "pow", 3) == 0)  
{  
printf("Степень: ");  
scanf("%f",&SecondNumeral);  
return(pow(Numeral, SecondNumeral));  
}  
else if(strncmp(Operation, "sqrt", 4) == 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);  
}  
}  
-- ВСТАВКА -- 58,2 Внизу
```

Figure 5.1: Текст calculate.c



```
gashalygin@gashalygin:~  
Файл Правка Вид Поиск Терминал Справка  
#ifndef CALCULATE_H_  
#define CALCULATE_H_  
float Calculate(float Numeral, char Operation[4]);  
#endif /*CALCULATE_H_*/  
~
```

Figure 5.2: Текст calculate.h



```

gashalygin@geshalygin:~
Файл Правка Вид Поиск Терминал Справка

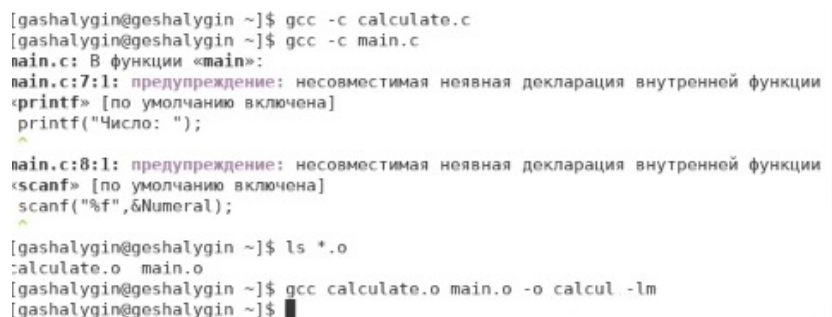
int
main (void)
{
    float Numeral;
    char Operation[4];
    float Result;
    printf("Число: ");
    scanf("%f",&Numeral);
    printf("Операция (+, -, *, /, pow, sqrt, sin, cos, tan): ");
    scanf("%s",&Operation);
    Result = Calculate(Numeral, Operation);
    printf("%.2f\n",Result);
    return 0;
}

```

Figure 5.3: Текст main.c

2. Выполните компиляцию программы посредством gcc. (рис. 5.4).

Исправим указанные синтаксические ошибки, и скомпилируем заново, результат работы готовой программы на рис. 5.5.



```

[gashalygin@geshalygin ~]$ gcc -c calculate.c
[gashalygin@geshalygin ~]$ gcc -c main.c
main.c: В функции «main»:
main.c:7:1: предупреждение: несовместимая неявная декларация внутренней функции
«printf» [по умолчанию включена]
printf("Число: ");
^
main.c:8:1: предупреждение: несовместимая неявная декларация внутренней функции
«scanf» [по умолчанию включена]
scanf("%f",&Numeral);
^
[gashalygin@geshalygin ~]$ ls *.o
calculate.o main.o
[gashalygin@geshalygin ~]$ gcc calculate.o main.o -o calcul -lm
[gashalygin@geshalygin ~]$

```

Figure 5.4: Компиляция

```

[gashalygin@geshalygin ~]$ ./calcul
Число: 0
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): +
Второе слагаемое: 0
0.00
[gashalygin@geshalygin ~]$ ./calcul
Число: 10
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): /
Делитель: 4
0.00
[gashalygin@geshalygin ~]$ ./calcul
Число: 10
Операция (+,-,*,/,pow,sqrt,sin,cos,tan):

```

Figure 5.5: Запуск

3. Создадим Makefile. (рис. 5.6). В нем прописаны те же команды для компиляции программы, вызываемые ранее вручную с флагами в виде переменных, которые можно менять при необходимости. Компиляция с его использованием на рис. 5.7

```

gashalygin@geshalygin:~
Файл  Правка  Вид  Поиск  Терминал  Справка

C = 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 *~

```

Figure 5.6: Текст Makefile

```
[gashalygin@geshalygin ~]$ make
make: `calcul' не требует обновления.
[gashalygin@geshalygin ~]$ rm calcul
[gashalygin@geshalygin ~]$ make
gcc calculate.o main.o -o calcul -lm
[gashalygin@geshalygin ~]$ rm calcul
[gashalygin@geshalygin ~]$ make
gcc calculate.o main.o -o calcul -lm
[gashalygin@geshalygin ~]$ ./calcul
Число: 1
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): sin
0.00
[gashalygin@geshalygin ~]$
```

Figure 5.7: Компиляция с использованием мейкфайла

4. С помощью gdb выполним отладку программы calcul (перед использованием gdb исправим Makefile) (рис. 5.8).

```
[gashalygin@geshalygin ~]$ gdb ./calcul
GNU gdb (GDB) Red Hat Enterprise Linux 7.6.1-120.el7
Copyright (C) 2013 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".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/gashalygin/calcul...(no debugging symbols found)...done.
(gdb) run
Starting program: /home/gashalygin/./calcul
Число: 0
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): 0
Неправильно введено действие: 2139095040.00
[Inferior 1 (process 10892) exited normally]
Missing separate debuginfos, use: debuginfo-install glibc-2.17-317.el7.x86_64
(gdb)
```

Figure 5.8: Отладка с помощью gdb

5. Затем установим утилиту splint и с ее помощью проанализируем коды файлов calculate.c и main.c. (рис. 5.9, 5.10).

```
[root@geshalygin geshalygin]# yum install splint
Загружены модули: fastestmirror, langpacks
Loading mirror speeds from cached hostfile
```

Figure 5.9: Установка splint

```

        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:34:7: Return value type double does not match declared type float:
(HUGE_VAL)
To allow all numeric types to match, use +relaxtypes.
calculate.c:42:1: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:43:7: Return value type double does not match declared type float:
(pow(Numeral, SecondNumeral))
calculate.c:46:7: Return value type double does not match declared type float:
(sqrt(Numeral))
calculate.c:48:7: Return value type double does not match declared type float:
(sin(Numeral))
calculate.c:50:7: Return value type double does not match declared type float:
(cos(Numeral))
calculate.c:52:7: Return value type double does not match declared type float:
(tan(Numeral))
calculate.c:56:7: Return value type double does not match declared type float:
(HUGE_VAL)

Finished checking --- 15 code warnings
[root@geshalygin gashalygin]#

```

Figure 5.10: Работa splint

6 Выводы

В процессе работы над лабораторной работы были изучены основы программирования в оболочке ОС UNIX, получен опыт разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linux на примере создания на языке программирования С калькулятора с простейшими функциями.

7 Библиография

1. <https://docs.altlinux.org/ru-RU/archive/2.3/html-single/junior/alt-docs-extras-linuxnovice/ch02s10.html>
2. <http://bourabai.kz/os/shells.htm>
3. Д.С. Кулябов, А.В. Королькова / Администрирование локальных систем. Лабораторные работы. — М.: Российский университет дружбы народов, 2017. — 119 с.