

Отчет по лабораторной работе № 14

Тема:

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

Российский Университет Дружбы Народов

Факультет Физико-Математических и Естественных Наук

Дисциплина: Операционные системы

Студент: Довлетмурат Байрамгельдыев

Группа: НФИБд-03-20

Москва, 2021г.

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.

Введение

Разработка программного обеспечения содержит элементы и искусства, и науки, это одна сторона того, что делает ее такой восхищающей и стимулирующей профессией. Данный раздел вводит в тему тестирования программного обеспечения, которая также включает в себя и искусство, и науку; таким образом, это несколько более общий и высокий уровень (читай: «на который можно махнуть рукой»), чем остальная часть данной главы.

Тестирование программ является неотъемлемой частью процесса разработки программного обеспечения. Весьма маловероятно, что программа заработает правильно на 100 процентов при первой компиляции. Программа не несет ответственности за свою правильность; за это отвечает автор программы. Одним из самых важных способов проверки того, что программа работает так, как предполагалось, является ее тестирование.

Ход работы.

1. В домашнем каталоге создал подкаталог ~/work/os/lab_prog.

```
[dowlet@dowlet ~]$ mkdir work
[dowlet@dowlet ~]$ mkdir work/os
[dowlet@dowlet ~]$ mkdir work/os/lab_prog
[dowlet@dowlet ~]$ cd work/os/lab_prog
```

2. Создал в нём файлы: calculate.h, calculate.c, main.c. Это примитивнейший калькулятор, способный складывать, вычитать, умножать, делить, возводить число в степень, вычислять квадратный корень, вычислять sin, cos, tan. При запуске он запрашивает первое число, операцию, второе число. После этого программа выводит результат и останавливается.

```
[dowlet@dowlet lab_prog]$ touch calculate.h calculate.c main.c
```

Открыть

calculate.c

Сохранить

~/work/os/lab_prog

```
#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));
}
```

Стр 5, Стлб 23

```
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));
}
```

Стр 5, Стлб 23


```

}
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);
}
}

```

С ▾ Ширина табуляции: 8 ▾ Стр 5, Стлб 23 ▾ ВС

Реализация функций калькулятора в файле calculate.h:

Открыть ▾  calculate.h
~/work/os/lab_prog Сохранить ≡ - □ ×

```

#ifndef CALCULATE_H_
#define CALCULATE_H_

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

#endif

```

Основной файл main.c, реализующий интерфейс пользователя к калькулятору:

Открыть ▾



main.c

~/work/os/lab_prog

Сохранить



```
#include <stdio.h>
#include "calculate.h"

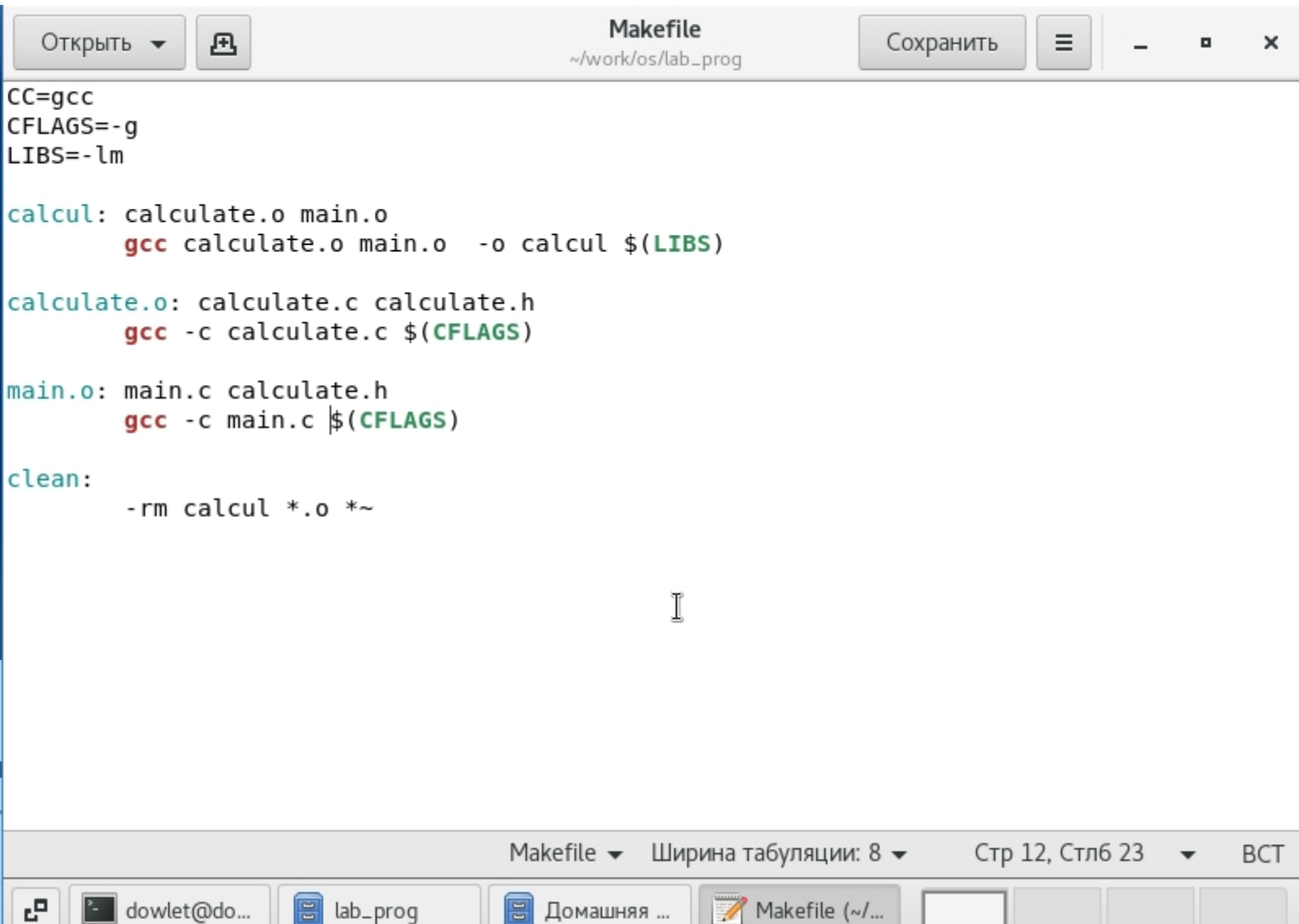
int main(void)
{
    float Numeral;
    char Operation[4];
    float Result;
    printf("Число: ");
    scanf("%f",&Numeral);
    printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): ");
    scanf("%s",&Operation);
    Result = Calculate(Numeral,Operation);
    printf("%6.2f\n",Result);
    return 0;
}
```

3.Выполнил компиляцию программы посредством gcc:

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

4. Исправил синтаксические ошибки.

5. Создал Makefile.



В содержании файла указаны флаги компиляции, тип компилятора и файлы, которые должен собрать сборщик. 6. С помощью gdb выполнил отладку программы calcul (перед использованием gdb исправил Makefile): – запустите отладчик GDB, загрузив в него программу для отладки: `gdb ./calcul` – для запуска программы внутри отладчика ввел команду `run`.

```
[dowlet@dowlet lab_prog]$ 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/dowlet/work/os/lab_prog/calcul...(no debugging symbols found)
...done.
(gdb) run
Starting program: /home/dowlet/work/os/lab_prog/./calcul
Число: 9
Операция (+, -, *, /, pow, sqrt, sin, cos, tan): +
Второе слагаемое: 4
13.00
[Inferior 1 (process 3170) exited normally]
Missing separate debuginfos, use: debuginfo-install glibc-2.17-317.el7.x86_64
(gdb)
```

– для постраничного (по 9 строк) просмотра исходного код использовал команду `list`. – для просмотра строк с 12 по 15 основного файла использовал `list` с параметрами: `list 12,15`.

```

(gdb) list
1      #include <stdio.h>
2      #include "calculate.h"
3
4      int main(void)
5      {
6          float Numeral;
7          char Operation[4];
8          float Result;
9          printf("Число: ");
10         scanf("%f",Numeral);
(gdb) list 12,15
12         scanf("%s",Operation);
13         Result = Calculate(Numeral,Operation);
14         printf("%6.2f\n",Result);
15         return 0;
(gdb) █

```

– для просмотра определённых строк не основного файла использовала list с параметрами: list calculate.c:20,29 – установил точку останова в файле calculate.c на строке номер 21: list calculate.c:20,27 break 21 – вывел информацию об имеющихся в проекте точка останова: info breakpoints

```

(gdb) list calculate.c:20,29
20         scanf("%f",&SecondNumeral);
21         return(Numeral*SecondNumeral);
22     }
23     else if(strncmp(Operation,"/",1)==0){
24         printf("Делитель: ");
25         scanf("%f",&SecondNumeral);
26         if(SecondNumeral==0){
27             printf("Ошибка: деление на ноль! ");
28             return(HUGE_VAL);
29         }
(gdb) list calculate.c:20,27
20         scanf("%f",&SecondNumeral);
21         return(Numeral*SecondNumeral);
22     }
23     else if(strncmp(Operation,"/",1)==0){
24         printf("Делитель: ");
25         scanf("%f",&SecondNumeral);
26         if(SecondNumeral==0){
27             printf("Ошибка: деление на ноль! ");
(gdb) █

```

```

(gdb) break 21
Breakpoint 1 at 0x400848: file calculate.c, line 21.
(gdb) info breakpoints
Num      Type           Disp Enb Address                  What
1        breakpoint     keep y   0x0000000000400848 in Calculate at calculate.c:21
(gdb)

```

Вывод:

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

Библиография.

<https://it.wikireading.ru/1194>. <https://testmatick.com/ru/luchshie-instrumenty-dlya-testirovaniya-proizvoditelnosti-v-linux/>.

Ответы на контрольные вопросы:

1. Информацию об этих программах можно получить с помощью функций `info` и `man`.
2. Unix поддерживает следующие основные этапы разработки приложений: -создание исходного кода программы; - представляется в виде файла -сохранение различных вариантов исходного текста; -анализ исходного текста; необходимо отслеживать изменения исходного кода, а также при работе более двух программистов над проектом программы нужно, чтобы они не делали изменений кода в одно время. - компиляция исходного текста и построение исполняемого модуля; -тестирование и отладка; - проверка кода на наличие ошибок -сохранение всех изменений, выполняемых при тестировании и отладке.
3. 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. Одно из полезных свойств компилятора Си — его способность по суффиксам определять типы файлов. По суффиксу .c компилятор распознает, что файл `abcd.c` должен компилироваться, а по суффиксу .o, что файл `abcd.o` является объектным модулем и для получения исполняемой программы необходимо выполнить редактирование связей. Простейший пример командной строки для компиляции программы `abcd.c` и построения исполняемого модуля `abcd` имеет вид: `gcc -o abcd abcd.c`. Некоторые проекты предпочитают показывать префиксы в начале текста изменений для старых (old) и новых (new) файлов. Опция — `prefix` может быть использована для установки такого префикса. Плюс к этому команда `bzr diff -p1` выводит префиксы в форме которая подходит для команды `patch -p1`.
4. Основное назначение компилятора с языка Си заключается в компиляции всей программы в целом и получении исполняемого модуля.
5. При разработке большой программы, состоящей из нескольких исходных файлов заголовков, приходится постоянно следить за файлами, которые требуют перекомпиляции после внесения изменений. Программа `make` освобождает пользователя от такой рутинной работы и 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. В общем случае `make-файл` содержит последовательность записей (строк), определяющих зависимости между файлами. Первая строка записи представляет собой список целевых (зависимых) файлов, разделенных пробелами, за которыми следует двоеточие и список файлов, от которых зависят целевые. Текст, следующий за точкой с запятой, и все последующие строки, начинающиеся с литеры табуляции, являются командами ОС UNIX, которые необходимо выполнить для обновления целевого файла. Таким образом, спецификация взаимосвязей имеет формат: `target1 [target2...]: [:] [dependment1...] [(tab)commands] [#commentary]` `[(tab)commands] [#commentary]`, где `#` — специфицирует начало комментария, так как содержимое строки, начиная с `#` и до конца строки, не будет обрабатываться командой `make`; `:` — последовательность команд ОС UNIX должна содержаться в одной строке `make-файла` (файла описаний), есть возможность переноса команд `()`, но она считается как одна строка; `::` — последовательность команд ОС UNIX может содержаться в нескольких последовательных строках файла описаний. Приведённый выше `make-файл` для программы `abcd.c` включает два способа компиляции и построения исполняемого модуля. Первый способ предусматривает обычную компиляцию с построением исполняемого модуля с именем `abcd`. Второй способ позволяет включать в исполняемый модуль `testabcd` возможность выполнить процесс отладки на уровне исходного текста. Пример можно найти в задании 5.
7. Пошаговая отладка программ заключается в том, что выполняется один оператор программы и, затем контролируются те переменные, на которые должен был воздействовать данный оператор. Если в программе имеются уже отлаженные подпрограммы, то подпрограмму можно рассматривать, как один оператор программы и воспользоваться вторым способом отладки программ. Если в программе существует достаточно большой участок программы, уже отлаженный ранее, то его можно выполнить, не контролируя переменные, на которые он воздействует. Использование точек останова позволяет пропускать уже отлаженную часть программы. Точка останова устанавливается в местах, где необходимо проверить содержимое переменных или просто проконтролировать, передаётся ли управление данному оператору. Практически во всех отладчиках поддерживается это свойство (а также выполнение программы до курсора и выход из подпрограммы). Затем отладка программы продолжается в пошаговом режиме с контролем локальных и глобальных переменных, а также внутренних регистров микроконтроллера и напряжений на выводах этой микросхемы.
8. `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
9. 1. Выполнил компиляцию программы 2) Увидел ошибки в программе 3) Открыл редактор и исправил программу 4) Загрузил программу в отладчик `gdb` 5) `run` — отладчик выполнил программу, ввел требуемые значения. 6) Использовал другие команды отладчика и проверил работу программы.
 10. . Отладчику не понравился формат `%s` для `&Operation`, т.к. `%s` — символьный формат, а значит необходим только `Operation`.
 11. Если вы работаете с исходным кодом, который не вами разрабатывался, то назначение различных конструкций может быть не совсем понятным. Система разработки приложений UNIX предоставляет различные средства, повышающие понимание исходного кода. К ним относятся: — `cscope` - исследование функций, содержащихся в программе; — `splint` — критическая проверка программ, написанных на языке Си.
 12. 1. Проверка корректности задания аргументов всех использованных в программе функций, а также типов возвращаемых ими значений;
 13. Поиск фрагментов исходного текста, корректных с точки зрения синтаксиса языка Си, но малоэффективных с точки зрения их реализации или содержащих в себе семантические ошибки;
 14. Общая оценка мобильности пользовательской программы.