Российский университет дружбы народов

факультет физико-математических и естественых наук

Отчет по лабараторной рабете № 14

дисциплина : Операционыые системы

студент Блохин александр НКН

Москва

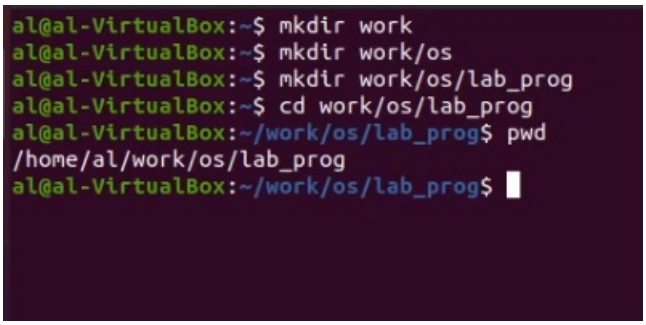
2021

Цель работы

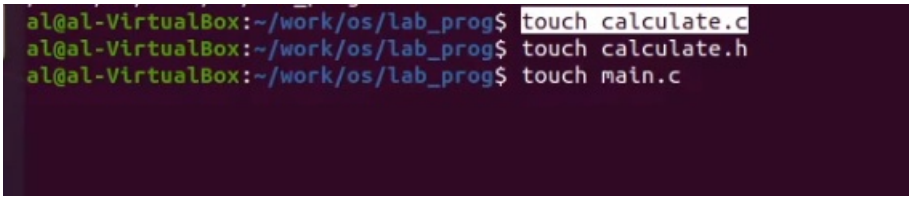
Приобрести простейшие навыки разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linux на примере создания на языке программирования С калькулятора с простейшими функциями.

Ход работы

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.

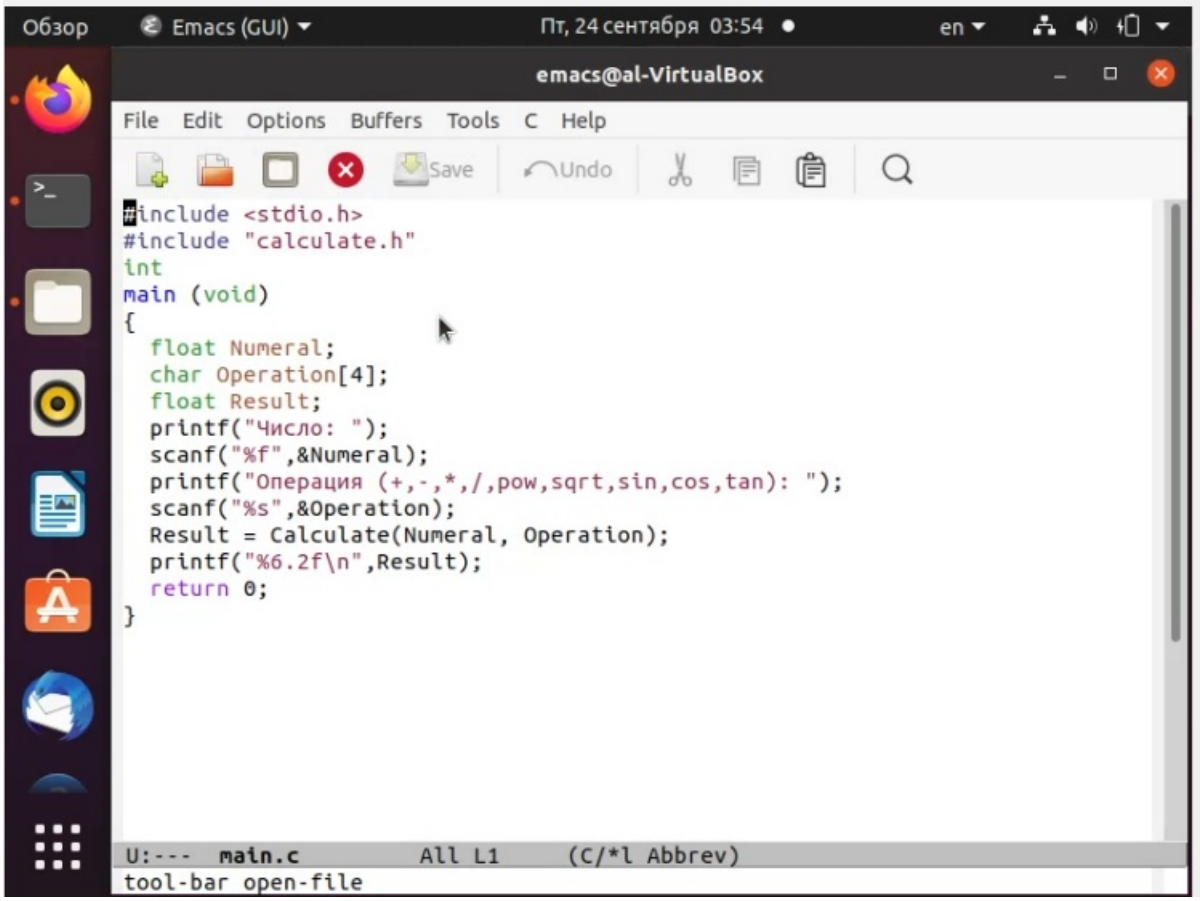



```
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "calculate.h"
float
Calculate(float Numeral, char Operation[4])
{
  float SecondNumeral;
  if(strncmp(Operation, "+", 1) == 0)
    {
      printf("Второе слагаемое: ");
      scanf("%f",&SecondNumeral);
      return(Numeral + SecondNumeral);
    }
  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);
U:---  calculate.c    Top L1     (C/*l Abbrev)
tool-bar open-file
```

```
#ifndef CALCULATE_H_
#define CALCULATE_H_

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

#endif /*CALCULATE_H_*/
```



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.(Добавил -g для отладки)

```
al@al-VirtualBox:~/work/os/lab_prog$ gcc -c calculate.c
al@al-VirtualBox:~/work/os/lab_prog$ gcc -c main.c
al@al-VirtualBox:~/work/os/lab_prog$ gcc calculate.o main.o -o calcul -lm
al@al-VirtualBox:~/work/os/lab_prog$
```

5. Создайте Makefile со следующим содержанием

6. С помощью gdb выполните отладку программы calcul

```
(gdb) run
Starting program: /home/al/work/os/lab_prog/calcul
Число: 1
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): +
Второе слагаемое: 3
  4.00
[Inferior 1 (process 8007) exited normally]
(gdb)
```

```
(gdb) list
1       #include <stdio.h>
2       #include "calculate.h"
3       int
4       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)
```

```
(gdb) list calculate.c:20,29
20          }
21        else if(strncmp(Operation, "*", 1) == 0)
22          {
23            printf("Множитель: ");
24            scanf("%f",&SecondNumeral);
25            return(Numeral * SecondNumeral);
26          }
27        else if(strncmp(Operation, "/", 1) == 0)
28          {
29            printf("Делитель: ");
(gdb) break 21
Breakpoint 1 at 0x1319: file calculate.c, line 21.
(gdb)
```

```
(gdb) info breakpoints
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x0000000000001319 in Calculate
                                                   at calculate.c:21
(gdb)
```

```
(gdb) run
Starting program: /home/al/work/os/lab_prog/calcul
Число: 1
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): +

Breakpoint 3, Calculate (Numeral=4.59163468e-41,
    Operation=0xc2 <error: Cannot access memory at address 0xc2>)
    at calculate.c:7
7       {
(gdb) display Numeral
1: Numeral = 4.59163468e-41
(gdb) print Numeral
$1 = 4.59163468e-41
(gdb) delete 1
(gdb) delete 2
(gdb) delete 3
(gdb) info breakpoints
No breakpoints or watchpoints.
(gdb)
```

Вывод

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.

Контрольные вопросы

1. Дополнительную информацию о этих программах можно получить с помощью функций info и man и help

2. 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. Суффикс - расширение. Например c.

4. gcc по расширению (суффиксу) .c распознает тип файла для компиляции и формирует объектный модуль — файл с расширением .o.

5. Для сборки разрабатываемого приложения и собственно компиляции полезно воспользоваться утилитой make. Она позволяет автоматизировать процесс преобразования файлов программы из одной формы в другую, отслеживает взаимосвязи между файлами.

6. target1 [target2...]:[:] [dependment1...] [(tab)commands] [#commentary] [(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 вдлинной строке команд можно использовать обратный слэш (). Двойное двоеточие указывает на то, что последовательность команд может содержаться в нескольких последовательных строках. Пример более сложного синтаксиса Makefile:

CC = gcc CFLAGS = abcd: abcd.c $(CC) -o abcd $(CFLAGS) abcd.c clean: -rm abcd *.o *~ В этом примере в начале файла заданы три переменные: CC и CFLAGS. Затем указаны цели, их зависимости и соответствующие команды. В командах происходит обращение к значениям переменных. Цель с именем clean производит очисткукаталога от файлов, полученных в результате компиляции. Для её описания использованы регулярные выражения.

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 сутивывод 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. Выполнили компиляцию программы 2)Увидели ошибки в программе Открыли редактор и исправили программу Загрузили программу в отладчик gdb run — отладчик выполнил программу, мы ввели требуемые значения. программа завершена, gdb не видит ошибок.

10. в сообшении указывался файл и строка с ошибкой, а ак же ее характер.

11. cscope - исследование функций, содержащихся в программе; splint — критическая проверка программ, написанных на языке Си.

12. Ещё одним средством проверки исходных кодов программ, написанных на языке C, является утилита splint. Эта утилита анализирует программный код, проверяеткорректность 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.