

РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ № 14

дисциплина: Операционные системы

Студент: Юрченко Артём Алексеевич Группа: НФИбд-02-20

МОСКВА

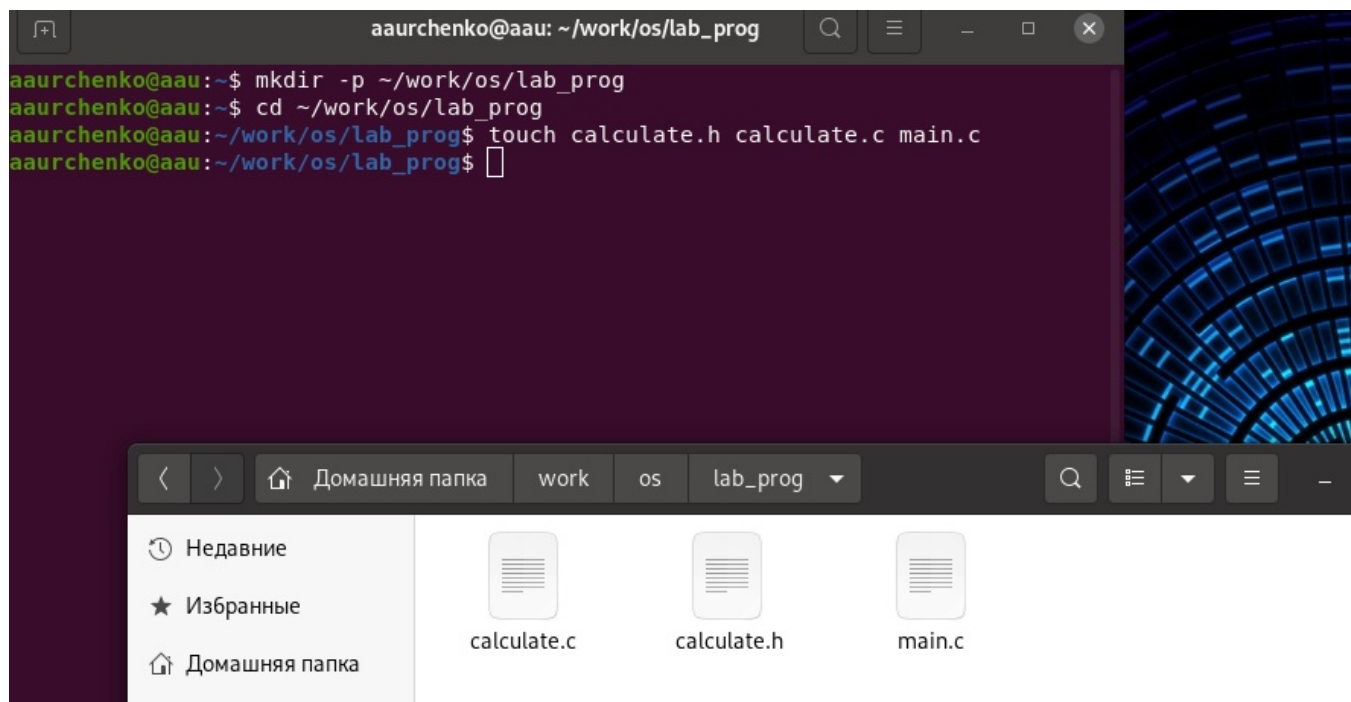
2021 г.

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.

Выполнение лабораторной работы.

1. В домашнем каталоге создаём 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.

Текстовый редактор ▾

Пт, 29 октября 00:59

11,1 °C

en ▾

Открыть ▾

main.c

Сохранить

~/work/os/lab_prog

```
1 //////////////////////////////////////////////////
2 // main.c
3 #include <stdio.h>
4 #include "calculate.h"
5 int
6 main (void)
7 {
8     float Numeral;
9     char Operation[4];
10    float Result;
11    printf("Число: ");
12    scanf("%f",&Numeral);
13    printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): ");
14    scanf("%s",&Operation);
15    Result = Calculate(Numeral, Operation);
16    printf("%.2f\n",Result);
17    return 0;
18 }
```

Открыть ▾

*calculate.h

Сохранить

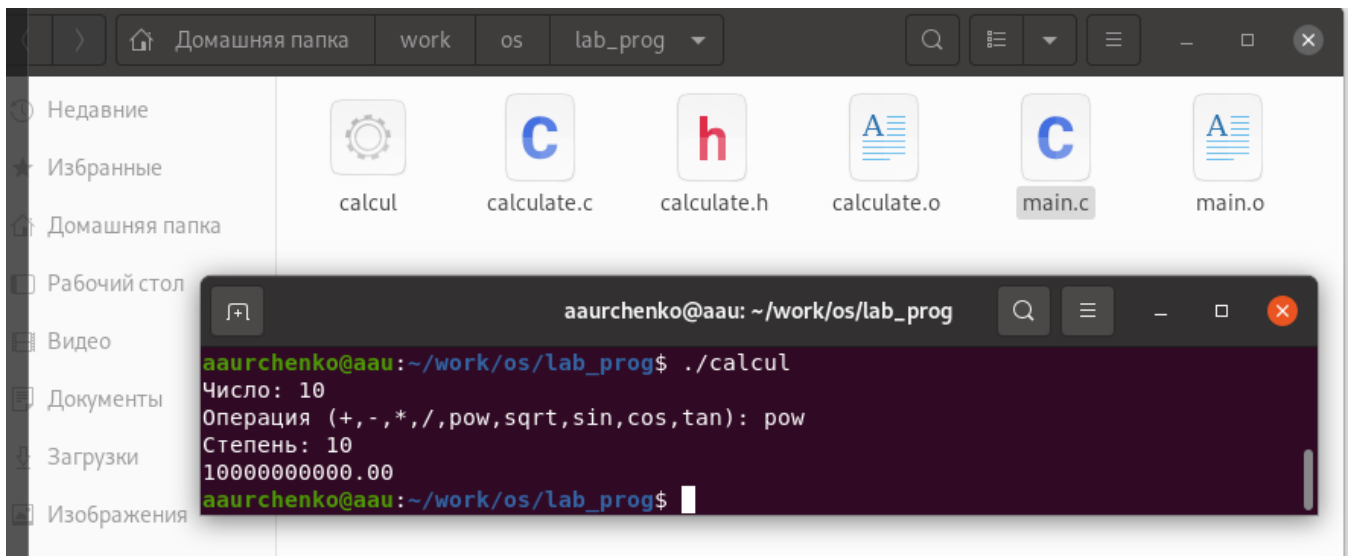
~/work/os/lab_prog

```
1 //////////////////////////////////////////////////
2 // calculate.c
3 #include <stdio.h>
4 #include <math.h>
5 #include <string.h>
6 #include "calculate.h"
7
8 float
9 Calculate(float Numeral, char Operation[4])
10 {
11     float SecondNumeral;
12     if(strncmp(Operation, "+", 1) == 0)
13     {
14         printf("Второе слагаемое: ");
15         scanf("%f",&SecondNumeral);
16         return(Numeral + SecondNumeral);
17     }
18     else if(strncmp(Operation, "-", 1) == 0)
19     {
20         printf("Вычитаемое: ");
21         scanf("%f",&SecondNumeral);
22         return(Numeral - SecondNumeral);
23     }
24     else if(strncmp(Operation, "*", 1) == 0)
25     {
26         printf("Множитель: ");
27         scanf("%f",&SecondNumeral);
28         return(Numeral * SecondNumeral);
29     }
30     else if(strncmp(Operation, "/", 1) == 0)
31     {
32         printf("Делитель: ");
33         scanf("%f",&SecondNumeral);
34         if(SecondNumeral == 0)
35         {
36             printf("Ошибка: деление на ноль! ");
37             return(HUGE_VAL);
38         }
39     }
40 }
```

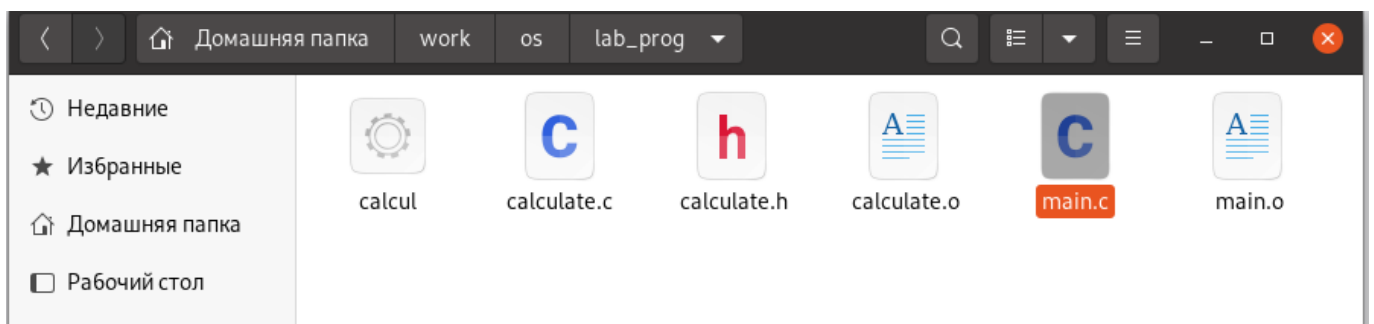
```
Текстовый редактор  Пт, 29 октября 00:58
*calculate.h
~/work/os/lab_prog

1 //////////////////////////////////////////////////
2 // calculate.h
3 #ifndef CALCULATE_H_
4 #define CALCULATE_H_
5 float Calculate(float Numeral, char Operation[4]);
6 #endif /*CALCULATE_H_*/
```

3. Реализуем функций калькулятора в файле calculate.c.



```
aaurchenko@aaau: ~/work/os/lab_prog
aaurchenko@aaau:~/work/os/lab_prog$ ./calcul
Число: 10
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): pow
Степень: 10
10000000000.00
aaurchenko@aaau:~/work/os/lab_prog$
```



4. Реализуем заголовочный файл calculate.h, описывающий формат вызова функции калькулятора.

5. Пишем основной файл main.c, реализующий интерфейс пользователя к калькулятору.

6. Выполняем компиляцию программы посредством gcc и проверяем работу калькулятора.

7. Создаём Makefile. Для создания используем образец из работы. Попутно исправляем ошибки в образце, которые препятствуют оптимальной работе отладчика.

```
aaurchenko@aaau:~/work/os/lab_prog$ touch make
aaurchenko@aaau:~/work/os/lab_prog$ make calculate.o
```

8. Используем Makefile.

9. Запускаем отладчик GDB, загрузив в него программу для отладки.

```
gdb ./calcul
```

10. Для запуска программы внутри отладчика вводим команду run.

```
run
```

11. Для постраничного (по 9 строк) просмотра исходного код используем команду list.

```
list
```

12. Для просмотра строк с 12 по 15 основного файла используем list с параметрами.

```
list 12,15
```

13. Для просмотра определённых строк не основного файла используем list с параметрами.

```
list calculate.c:20,29
```

```
(gdb) list calculate.c:20,27
20      return (Numeral - SecondNumeral);
21    }
22    else if (strcmp(Operation,"**",1) == 0)
23    {
24      printf("Множитель: ");
25      scanf("%f",&SecondNumeral);
26      return (Numeral * SecondNumeral);
27    }
(gdb) break 21
Breakpoint 1 at 0x555555555319: file calculate.c, line 22.
(gdb) list calculate.c:20,27
20      return (Numeral - SecondNumeral);
21    }
22    else if (strcmp(Operation,"**",1) == 0)
23    {
24      printf("Множитель: ");
25      scanf("%f",&SecondNumeral);
26      return (Numeral * SecondNumeral);
27    }
(gdb) info breakpoints
Num   Type             Disp Enb Address            What
1     breakpoint       keep y   0x0000555555555319 in calculate
      at calculate.c:22
(gdb) run
Starting program: /home/aviljin/work/os/lab_prog/calcul
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): -
Вычитаемое: 2
3.00
[Inferior 1 (process 86082) exited normally]
(gdb)
```

14. Устанавливаем точку остановки в файле calculate.c на строке номер 22, чтобы программа остановилась, после ввода операции вычитания.

```
list calculate.c:15,25
break 18
```

15. Выводим информацию об имеющихся в проекте точке останова.

```
info breakpoints
```

16. Запускаем программу внутри отладчика и убеждаемся, что программа остановилась в момент прохождения точки останова.

```
run
5
-
```

17. Используем команду backtrace, которая показывает весь стек вызываемых функций от начала программы до текущего места.

```
backtrace
```

18. Просматриваем, чему равно на этом этапе значение переменной Numeral.

```
print Numeral
```

19. Сравниваем с результатом вывода на экран после использования команды display.

```
display Numeral
```

20. Убираем точки останова.

```
info breakpoints
delete 2
```

21. С помощью утилиты splint пробуем проанализировать коды файлов calculate.c и main.c. Найдены незначительные ошибки, не влияющие на работоспособность кода. Например, утилита предлагает посылать не весь массив Operation в функцию Calculate, а только указатель на него. Это не является ошибкой, но оптимизирует код. Также splint предупреждает, что опасно однозначно сравнивать числа с плавающей точкой и целочисленные числа, опять же хорошее замечание, но на работу сильно не влияет. В общем, splint не нашел серьезных ошибок, что, в целом, логично, поскольку код работает.

```
splint calculate.c
splint main.c
```

```
Splint 3.1.2 --- 20 Feb 2018

calculate.h:5:37: Function parameter Operation declared as manifest ar
ray (size
        constant is meaningless)
  A formal parameter is declared as an array with size.  The size of t
he array
  is ignored in this context, since the array formal parameter is trea
ted as a
  pointer. (Use -fixedformalarray to inhibit warning)
main.c: (In function main)
main.c:10:2: Return value (type int) ignored: scanf("%f", &Num...
  Result returned by function call is not used.  If this is intended, c
an cast
  result to (void) to eliminate message. (Use -retvalint to inhibit wa
rning)
main.c:12:2: Return value (type int) ignored: scanf("%s", &Ope...

Finished checking --- 3 code warnings
```

Выводы

Мы приобрели простейшие навыки разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linux. Закрепили знания, полученные в прошлых работах. Создали на языке программирования C калькулятор с простейшими функциями и разобрали на нем основные навыки отладки.

Термины

- GCC (GNU Compiler Collection) - это набор компиляторов для разного рода языков программирования (C, C++, Java, Фортран и др.).
- GDB (GNU Debugger) - отладчик для поиска и устранения ошибок в программе. Входит в комплект программ GNU для ОС типа UNIX.
- Утилита make позволяет автоматизировать процесс преобразования файлов программы из одной формы в другую, отслеживает взаимосвязи между файлами.
- Утилита splint анализирует программный код, проверяет корректность задания аргументов использованных в программе функций и типов возвращаемых значений, обнаруживает синтаксические и семантические ошибки.
- Командный процессор (командная оболочка, интерпретатор команд shell) — это программа, позволяющая пользователю взаимодействовать с операционной системой компьютера.
- POSIX (Portable Operating System Interface for Computer Environments) — набор стандартов описания интерфейсов взаимодействия операционной системы и прикладных программ.
- Последовательность команд может быть помещена в текстовый файл. Такой файл называется командным.
- Каталог, он же директория, (от английского Directory) — это объект в ФС (файловой системе), необходимый для того, чтобы упростить работу с файлами.
- Домашний каталог - каталог, предназначенный для хранения собственных данных пользователя Linux. Как правило, является текущим непосредственно после регистрации пользователя в системе.
- Команда - записанный по специальным правилам текст (возможно с аргументами), представляющий собой указание на выполнение какой-либо функций (или действий) в операционной системе.

Выводы

Мы изучили основы программирования в оболочке ОС UNIX. Закрепили знания, полученные в прошлых работах. Научились писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

Термины

- Командный процессор (командная оболочка, интерпретатор команд shell) — это программа, позволяющая пользователю взаимодействовать с операционной системой компьютера.
- POSIX (Portable Operating System Interface for Computer Environments) — набор стандартов описания интерфейсов взаимодействия операционной системы и прикладных программ.
- Последовательность команд может быть помещена в текстовый файл. Такой файл называется командным.
- Флаги — это опции командной строки, обычно помеченные знаком минус; Например, для команды ls флагом может являться -F.
- Каталог, он же директория, (от английского Directory) — это объект в ФС (файловой системе), необходимый для того, чтобы упростить работу с файлами.
- Домашний каталог - каталог, предназначенный для хранения собственных данных пользователя Linux. Как правило, является текущим непосредственно после регистрации пользователя в системе.
- Команда - записанный по специальным правилам текст (возможно с аргументами), представляющий собой указание на выполнение какой-либо функций (или действий) в операционной системе.