

Титульный лист

РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Лабораторная работа 14

По дисциплине "Операционные системы"

Выполнил:

Студент группы НПВбм-01-19

Студенческий билет №: 1032193844

Саидов Ахият Магомадович

Руководитель: Валиева Татьяна Рефатовна

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.

Начало работы

1. В домашнем каталоге создадим подкаталог *~/work/os/lab_prog*.

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

Рисунок 1

2. Создадим в нём файлы: *calculate.h*, *calculate.c*, *main.c*.

```
[AMSaidov@amsaidov lab_prog]$ touch calculate.h calculate.c main.c  
[AMSaidov@amsaidov lab_prog]$ gedit calculate.c  
[AMSaidov@amsaidov lab_prog]$ gedit calculate.h  
[AMSaidov@amsaidov lab_prog]$ gedit main.c
```

Рисунок 2

Реализация функций калькулятора в файле *calculate.c*:

```

////////////////////////////////////
// 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("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));
    }
}

```

Рисунок 3

Интерфейсный файл *calculate.h*, описывающий формат вызова функции калькулятора:

```

////////////////////////////////////
// calculate.h
#ifndef CALCULATE_H_
#define CALCULATE_H_
float Calculate(float Numeral, char Operation[4]);
#endif /*CALCULATE_H_*/

```

Рисунок 4

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

```

////////////////////////////////////
// main.c
#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("%.2f\n",Result);
    return 0;
}

```

Рисунок 5

3. Выполним компиляцию программы посредством `gcc`.

```

#
# Makefile
#
CC = gcc
CFLAGS = -g
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 *~
# End Makefile

```

Рисунок 6

4. При необходимости исправим синтаксические ошибки.
5. Создадим *Makefile* со следующим содержанием:

```

[AMSaidov@amsaidov lab_prog]$ gcc -c calculate.c
[AMSaidov@amsaidov lab_prog]$ gcc -c main.c
[AMSaidov@amsaidov lab_prog]$ gcc calculate.o main.o -o calcul -lm

```

Рисунок 7

6. С помощью `gdb` выполните отладку программы `calcul` (перед использованием `gdb` исправьте *Makefile*):

- Запустите отладчик *GDB*, загрузив в него программу для отладки:

```
gdb ./calcul
```

```
[AMSaidov@amsaidov 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/AMSaidov/work/os/lab_prog/calcul...done.
e.
```

Рисунок 8

- Для запуска программы внутри отладчика введите команду *run*:

```
run
(gdb) run
Starting program: /home/AMSaidov/work/os/lab_prog/./calcul
Число: 12
Операция (+, -, *, /, pow, sqrt, sin, cos, tan): *
Умножитель: 3
36.00
[Inferior 1 (process 10192) exited normally]
Missing separate debuginfos, use: debuginfo-install glibc-2.17-32
5.el7 9.x86_64
```

Рисунок 9

- Для постраничного (по 9 строк) просмотра исходного код используйте команду *list*:

```
list
(gdb) list
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: ");
```

Рисунок 10

- Для просмотра строк с 12 по 15 основного файла используйте *list* с параметрами:

```
list 12,15
(gdb) list 12,15
12     scanf("%f",&Numeral);
13     printf("Операция (+, -, *, /, pow, sqrt, sin, cos, tan): ");
14     scanf("%s",&Operation);
15     Result = Calculate(Numeral, Operation);
```

Рисунок 11

- Для просмотра определённых строк не основного файла используйте *list* с параметрами:

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

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

Рисунок 12

- Установите точку останова в файле *calculate.c* на строке номер 21:

```
list calculate.c:20,27
```

```
break 21
```

```
(gdb) list calculate.c:20,27
20     scanf("%f",&SecondNumeral);
21     return(Numeral - SecondNumeral);
22 }
23     else if(strncmp(Operation, "*", 1) == 0)
24     {
25         printf("Множитель: ");
26         scanf("%f",&SecondNumeral);
27         return(Numeral * SecondNumeral);
(gdb) break 21
Breakpoint 1 at 0x4007fd: file calculate.c, line 21.
```

Рисунок 13

- Выведите информацию об имеющихся в проекте точка останова:

```
info breakpoints
```

```
(gdb) break 21
```

```
Breakpoint 1 at 0x4007fd: file calculate.c, line 21.
```

```
(gdb) info breakpoints
```

Num	Type	Disp	Enb	Address	What
1	breakpoint	keep	y	0x00000000004007fd	in Calculate at calculate.c

```
:21
```

Рисунок 14

- Запустите программу внутри отладчика и убедитесь, что программа остановится в момент прохождения точки останова:

```
run
```

```
5
```

```
-
```

```
backtrace
```



```
(gdb) run
Starting program: /home/AMSaidov/work/os/lab_prog/./calcul
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): -
Вычитаемое: backtrace
```

```
Breakpoint 1, Calculate (Numeral=5,
    Operation=0x7fffffffde30 "-") at calculate.c:21
21      return(Numeral - SecondNumeral);
```

Рисунок 15

- О отладчик выдаст следующую информацию:

```
#0 Calculate (Numeral=5, Operation=0x7fffffffde30 "-")
```

```
at calculate.c:21
```

```
#1 0x0000000000400b2b in main () at main.c:17
```

а команда *backtrace* покажет весь стек вызываемых функций от начала программы до текущего места.

- Посмотрите, чему равно на этом этапе значение переменной *Numeral*, введя:

```
print Numeral
```

```
(gdb) print Numeral
$1 = 5
```

Рисунок 16

На экран должно быть выведено число 5.

- Сравните с результатом вывода на экран после использования команды:

```
display Numeral
```

```
(gdb) display Numeral
1: Numeral = 5
```

Рисунок 17

- Уберите точки останова:

```
info breakpoints
```

```
delete 1
```

```
(gdb) info breakpoints
```

Num	Type	Disp	Enb	Address	What
1	breakpoint	keep	y	0x00000000004007fd	in Calculate at calculate.c:21

```
breakpoint already hit 1 time
```

```
(gdb) delete 1
```

Рисунок 18

7. С помощью утилиты *splint* попробуем проанализировать коды файлов *calculate.c* и *main.c*.

```

Finished checking --- 15 code warnings
[AMSaidov@amsaidov lab_prog]$ clear

[AMSaidov@amsaidov lab_prog]$ splint calculate.c
Splint 3.1.2 --- 11 Oct 2015

calculate.h:5:37: 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:8:31: Function parameter Operation declared as manifest array (size
                    constant is meaningless)
calculate.c: (in function Calculate)
calculate.c:14:1: 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:20:1: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:26:1: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:32:1: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:33:4: Dangerous equality comparison involving float types:
                    SecondNumeral == 0

```

Рисунок 19

```

[AMSaidov@amsaidov lab_prog]$ splint main.c
Splint 3.1.2 --- 11 Oct 2015

calculate.h:5:37: 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)
main.c: (in function main)
main.c:12:1: Return value (type int) ignored: scanf("%f", &Num...
    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)
main.c:14:12: Format argument 1 to scanf (%) expects char * gets char [4] *:
                    &Operation
    Type of parameter is not consistent with corresponding code in format string.
    (Use -formattype to inhibit warning)
    main.c:14:9: Corresponding format code
main.c:14:1: Return value (type int) ignored: scanf("%s", &Ope...

```

Рисунок 20

Вывод

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

Контрольные вопросы

1. Для этого есть команда `man` и предлагающиеся к ней файлы.
2. Кодировка, Компиляция, Тест.
3. Это расширения файлов.
4. Программа `gcc`, которая интерпретирует к определенному языку программирования аргументы командной строки и определяет запуск нужного компилятора для нужного файла
5. Для компиляции группы файлов. Собрания из них программы, и последующего удаления.
6. `program: main.o lib.o`
`cc -o program main.o lib.o`

main.o lib.o: defines.h

В имени второй цели указаны два файла и для этой же цели не указана команда компиляции. Кроме того, нигде явно не указана зависимость объектных 14 файлов от «*.c»-файлов. Дело в том, что программа *make* имеет predetermined правила для получения файлов с определёнными расширениями. Так, для цели-объектного файла (расширение «.o») при обнаружении соответствующего файла с расширением «.c» будет вызван компилятор «cc -c» с указанием в параметрах этого «.c»-файла и всех файлов-зависимостей.

7. Программы для отладки нужны для нахождения ошибок в программе. Для их использования надо скомпилировать программу таким образом, чтобы отладочная информация содержалась в конечном бинарном файле.

8.
 - *backtrace* – выводит весь путь к текущей точке останова, то есть названия всех функций, начиная от *main()*; иными словами, выводит весь стек функций;
 - *break* – устанавливает точку останова; параметром может быть номер строки или название функции;
 - *clear* – удаляет все точки останова на текущем уровне стека (то есть в текущей функции);
 - *continue* – продолжает выполнение программы от текущей точки до конца;
 - *delete* – удаляет точку останова или контрольное выражение;
 - *display* – добавляет выражение в список выражений, значения которых отображаются каждый раз при остановке программы;
 - *finish* – выполняет программу до выхода из текущей функции; отображает возвращаемое значение, если такое имеется;
 - *info breakpoints* – выводит список всех имеющихся точек останова;
 - *info watchpoints* – выводит список всех имеющихся контрольных выражений;
 - *list* – выводит исходный код; в качестве параметра передаются название файла исходного кода, затем, через двоеточие, номер начальной и конечной строки;
 - *next* – пошаговое выполнение программы, но, в отличие от команды *step*, не выполняет пошагово вызываемые функции;
 - *print* – выводит значение какого-либо выражения (выражение передаётся в качестве параметра);
 - *run* – запускает программу на выполнение;
 - *set* – устанавливает новое значение переменной *step* – пошаговое выполнение программы;
 - *watch* – устанавливает контрольное выражение, программа остановится, как только значение контрольного выражения изменится;
9.
 - *gdb -silent ./calcul*
 - *run 12. list*
 - *backtrace*
 - *breakpoints*
 - *print Numeral*
 - *Splint*

10. Консоль выводит ошибку с номером строки и ошибочным сегментом, но при этом есть возможность выполнить программу сразу.

11.
 - Правильный синтаксис
 - Наличие комментариев
 - Разбиение большой сложной программы на несколько сегментов попроще.
12. *Split* – разбиение файла на меньшие, определённого размера. Может разбивать текстовые файлы по строкам и любые – по байтам. По умолчанию читает со стандартного ввода и создает файлы с именами вида *хаа*, *хаб* и т.д. По умолчанию разбиение идёт по 1000 строк в файле.