Лабораторная работа №13

Дисцилина: Операционные системы

Маслова Анастасия Сергеевна

Содержание

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.

Задание

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. Реализация функций калькулятора в файле calculate.h:

```
////////////////////////////////////
// calculate.c

#include <stdio.h>
#include <math.h>
#include <string.h>
#include "calculate.h"

float
1Calculate(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));
else
{
printf("Неправильно введено действие ");
return(HUGE_VAL);
}
}
```

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_*/
```

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("%6.2f\n",Result);
return 0;
}
```

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.
5. Создайте Makefile со следующим содержанием:

```
#
# Makefile
#

CC = 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 *~

# End Makefile
```

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

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

```
gdb ./calcul
```

- Для запуска программы внутри отладчика введите команду run:

```
run
```

- Для постраничного (по 9 строк) просмотра исходного код используйте команду ``bash list

```
- Для просмотра строк с 12 по 15 основного файла используйте list с параметрами:
```bash
list 12,15
```

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

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

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

```
list calculate.c:20,27
break 21
```

- Выведите информацию об имеющихся в проекте точка останова:

```
info breakpoints
```

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

```
run
5
-
backtrace
```

- Отладчик выдаст следующую информацию:

```
#0 Calculate (Numeral=5, Operation=0x7fffffffd280 "-")
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
```
```

На экран должно быть выведено число 5. - Сравните с результатом вывода на экран после использования команды:

```
display Numeral
```

- Уберите точки останова:

```
info breakpoints
delete 1
```

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

Теоретическое введение

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.

Более подробно - здесь.

Выполнение лабораторной работы

1. В домашнем каталоге я создала подкаталог ~/work/os/lab_prog.
2. Создала в нём файлы: calculate.h, calculate.c, main.c и в них написала код по примеру лабораторной работы (рис.1).

```
[asmaslova@fedora ~]$ cd work
[asmaslova@fedora work]$ cd os
bash: cd: os: No such file or directory
[asmaslova@fedora work]$ ls
study
[asmaslova@fedora work]$ cd study
[asmaslova@fedora study]$ ls
2021-2022
[asmaslova@fedora study]$ cd ,,
bash: cd: ,,: No such file or directory
[asmaslova@fedora study]$ cd ,,
bash: cd: ,,: No such file or directory
[asmaslova@fedora study]$ cd ..
[asmaslova@fedora work]$ ls
study
[asmaslova@fedora work]$ pwd
/home/asmaslova/work
[asmaslova@fedora work]$ mkdir os
[asmaslova@fedora work]$ cd os
[asmaslova@fedora os]$ mkdir lab_prog
[asmaslova@fedora os]$ cd lab_prog/
[asmaslova@fedora lab_prog]$ ls
[asmaslova@fedora lab_prog]$ pwd
/home/asmaslova/work/os/lab_prog
[asmaslova@fedora lab_prog]$ touch calculate.h calculate.c main.c
[asmaslova@fedora lab_prog]$ ls
calculate.c  calculate.h  main.c
```

рис.1 Создание подкаталога и необходимых файлов

3. Далее я выполнила компиляцию программы посредством gcc (рис.2).

```
[asmaslova@fedora lab_prog]$ gcc -c calculate.c
[asmaslova@fedora lab_prog]$ gcc -c main.c
[asmaslova@fedora lab_prog]$ gcc calculate.o main.o -o calcul -lm
[asmaslova@fedora lab_prog]$ ./calcul
Число: 15
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): sin
  0.65
[asmaslova@fedora lab_prog]$ ./calcul
Число: 16
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): -
Вычитаемое: 24
 -8.00
[asmaslova@fedora lab_prog]$ ./calcul
Число: 91
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): pow
Степень: 3
753571.00
[asmaslova@fedora lab_prog]$ ./calcul
Число: 25
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): tan
 -0.13
[asmaslova@fedora lab_prog]$ touch Makefile
[asmaslova@fedora lab_prog]$ emacs Makefile &
[1] 4750
```

рис.2 Компиляция программы посредством gcc

4. Синтаксических ошибок компилятор мне не выдал, поэтому я ничего не исправляла, вместо этого я проверила работу программы и убедилась, что она работает.
5. После этого я создала Makefile с необходимым содержанием (рис.2).
6. С помощью gdb выполните отладку программы calcul (рис.3-4)

```
[asmaslova@fedora lab_prog]$ gdb ./calcul
GNU gdb (GDB) Fedora 10.2-9.fc35
Copyright (C) 2021 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".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./calcul...
(No debugging symbols found in ./calcul)
(gdb) run
Starting program: /home/asmaslova/work/os/lab_prog/calcul
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".
Число: 15
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): -
Вычитаемое: 3
 12.00
[Inferior 1 (process 5688) exited normally]
(gdb) list
1       /usr/src/debug/glibc-2.34-7.fc35.x86_64/elf/<built-in>: Success.
(gdb) list 12,15
12      /usr/src/debug/glibc-2.34-7.fc35.x86_64/elf/<built-in>: Success.
(gdb) list calculate.c:20,29
No source file named calculate.c.
(gdb) break 21
No line 21 in the current file.
Make breakpoint pending on future shared library load? (y or [n]) y
Breakpoint 1 (21) pending.
(gdb) list
12      in /usr/src/debug/glibc-2.34-7.fc35.x86_64/elf/<built-in>
(gdb) list 12,21
12      in /usr/src/debug/glibc-2.34-7.fc35.x86_64/elf/<built-in>
(gdb) info breakpoints
Num     Type           Disp Enb Address    What
1       breakpoint     keep y   <PENDING>  21
```

```
(gdb) info breakpoints
Num     Type           Disp Enb Address    What
1       breakpoint     keep y   <PENDING>  21
(gdb) run
Starting program: /home/asmaslova/work/os/lab_prog/calcul
Downloading source file /usr/src/debug/glibc-2.34-7.fc35.x86_64/elf/rtld.c...

Breakpoint 1, _dl_start (arg=0x7fffffffe070) at rtld.c:71
71        HP_TIMING_NOW (*var);
(gdb) backtrace
#0  _dl_start (arg=0x7fffffffe070) at rtld.c:71
#1  0x00007ffff7fcc008 in _start () from /lib64/ld-linux-x86-64.so.2
#2  0x0000000000000001 in ?? ()
#3  0x00007fffffffe384 in ?? ()
#4  0x0000000000000000 in ?? ()
(gdb) print Numeral
No symbol "Numeral" in current context.
(gdb) delete 1
(gdb) info breakpoints
No breakpoints or watchpoints.
```

7. С помощью утилиты splint я попробовала проанализировать коды файлов calculate.c и main.c и увидела множество синтаксических ошибок, которые не выдал компилятор (рис.5).

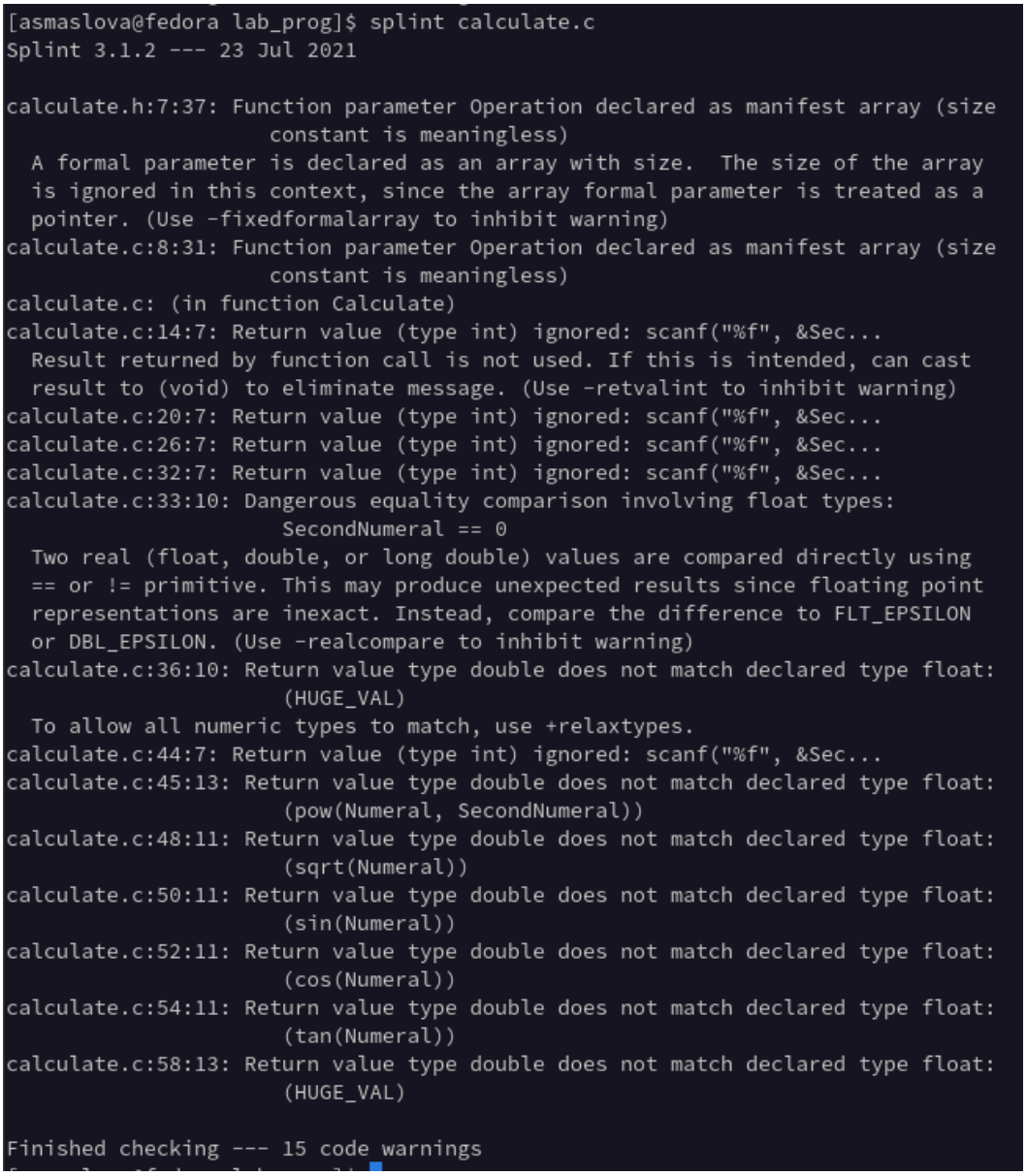

рис.5 Результат работы утилиты splint

Выводы

В результате лабораторной работы я приобрела простейшие навыки разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linux на примере создания на языке программирования С калькулятора с простейшими функциями.

Список литературы