

Лабораторная работа №13. Средства, применяемые при разработке программного обеспечения в ОС типа UNIX/Linux.

Парфенова Елизавета Евгеньевна

RUDN University, Moscow, Russian Federation

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.

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.
3. Выполните компиляцию программы посредством `gcc`

4. При необходимости исправьте синтаксические ошибки.
5. Создайте Makefile со следующим содержанием.
6. С помощью gdb выполните отладку программы calcul (перед использованием gdb исправьте Makefile)
7. С помощью утилиты splint попробуйте проанализировать коды файлов calculate.c и main.c.

Первым шагом создаем каталог `~/work/os/lab_prog` с помощью ***mkdir*** ***-p*** и файлы `calculate.h`, `calculate.c`, `main.c` с помощью ***touch***. (рис. 1)

```
[eeparfenova@fedora ~]$ mkdir -p ~/work/os/lab_prog
[eeparfenova@fedora ~]$ cd ~/work/os/lab_prog
[eeparfenova@fedora lab_prog]$ touch calculate.h
[eeparfenova@fedora lab_prog]$ touch calculate.c
[eeparfenova@fedora lab_prog]$ touch main.c
[eeparfenova@fedora lab_prog]$ ls
calculate.c calculate.h main.c
```

Figure 1: Создание каталогов и файлов

Далее записываем программы в эти файлы. Затем выполняем компиляцию программы посредством gcc, используя команды: **gcc -c calculate.c**, **gcc -c -g main.c**, **gcc calculate.o main.o -o calcul -lm*** (рис. 2)

```
[eeparfenova@fedora lab_prog]$ gcc -c calculate.c
[eeparfenova@fedora lab_prog]$ gcc -c -g main.c
[eeparfenova@fedora lab_prog]$ gcc calculate.o main.o -o calcul -lm
[eeparfenova@fedora lab_prog]$ ls
calcul calculate.c calculate.h calculate.h~ calculate.o main.c main.o
```

Figure 2: Компиляция программы

Далее создаем Makefile с помощью *touch* и записываем туда код из файла Лабораторной работы, внося некоторые изменения в 6 и 19 строках. (рис. 3)

```
#  
# 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
```

Figure 3: Makefile

После запускаем отладчик GDB командой ***gdb ./calcul***, загрузив в него программу для отладки.(рис. 4)

```
[eeparfenova@fedora lab_prog]$ gdb ./calcul
GNU gdb (GDB) Fedora 11.2-2.fc35
Copyright (C) 2022 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...
```

Figure 4: Запуск отладчика

Запускаем программу внутри отладчика, используя команду **run**. (рис. 5)

```
(gdb) run
Starting program: /home/eeperfenova/work/os/lab_prog/calcul

This GDB supports auto-downloading debuginfo from the following URLs:
https://debuginfod.fedoraproject.org/
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.
Downloading separate debug info for /home/eeperfenova/work/os/lab_prog/system-suppl
lib64/libc.so.6...
Downloading separate debug info for /lib64/libm.so.6...
Downloading separate debug info for /lib64/libg.so.6...
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): pow
Степень: 2
25.00
[Inferior 1 (process 11015) exited normally]
```

Figure 5: Запуск программы

Постранично смотрим код, используя *list*, а после смотрим исходный код с 12 строки по 15 строку командой *list 12,15* (рис. 6)

```
(gdb) list
1  //////////////////////////////////////////////////
2  // main.c
3
4  #include <stdio.h>
5  #include "calculate.h"
6  int
7  main (void)
8  {
9      float Numeral;
10     char Operation[4];
(gdb) list 12,15
12     printf("Введите: ");
13     scanf("%f",&Numeral);
14     printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): ");
15     scanf("%s",&Operation);
```

Figure 6: Просмотр исходного кода

Затем устанавливаем точку останова в файле `calculate.c` на строке номер 14, сразу после ввода числа. Делаем это, используя команду ***break 14***. Смотрим информацию об имеющихся в проекте точка останова командой ***info breakpoints***. Запускаем программу и проверяем, когда она остановится. Она остановилась правильно, перед вводом операции. (рис. 7)

```
(gdb) break 14
Breakpoint 1 at 0x4010b0: file main.c, line 14.
(gdb) info breakpoints
Num     Type             Disp Enb Address            What
1       breakpoint      keep y   0x00000000004010b1 in main at main.c:14
(gdb) run
Starting program: /home/eeeparfenova/work/os/lab_prog/calcul
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".
Число: 5

Breakpoint 1, main () at main.c:14
14      printf("Введите 1 - '+', 2 - '-', 3 - '*', 4 - ':': ");
```

Figure 7: Точки останова

Смотрим, чему равно на этом этапе значение переменной `Numeral`, введя ***print Numeral***. Оно равно пяти, так как ввели мы именно это число. Затем сравниваем с результатом вывода на экран, который делаем командой ***display Numeral***. Значения совпали. (рис. 8)

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

Figure 8: Значение переменной

После просто убираем точки останова, вызвав вначале информацию о них через *info breakpoints*, а после удалив, используя *delete 1* (точка останова номер 1) (рис. 9)

```
(gdb) info breakpoints
Num   Type             Disp Enb Address            What
1      breakpoint      keep y   0x00000000004014b3 in main at main.c:14
breakpoint already hit 1 time
(gdb) delete 1
```

Figure 9: Удаление точки останова

Последним шагом с помощью утилиты splint анализируем коды файлов calculate.c и main.c. (рис. 10) (рис. 11)

```
insp@fennofedora lab_prog$ splint calculate.c
Splint 3.1.2 --- 23 Jul 2021

calculate.h:7:37: 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 -fixedformarray to inhibit warning)
calculate.c:18:31: Function parameter Operation declared as manifest array
        (size constant is meaningless)
calculate.c: (in function calculate)
calculate.c:18:3: Return value (type int) ignored: scanf("%d", &dec...
Result returned by function call is not used. If this is intended, can cast
result to void() to eliminate message. (Use -retvalue to inhibit warning)
calculate.c:18:3: Return value (type int) ignored: scanf("%d", &dec...
calculate.c:18:3: Return value (type int) ignored: scanf("%d", &dec...
calculate.c:18:3: Return value (type int) ignored: scanf("%d", &dec...
calculate.c:18:3: Return value (type int) ignored: scanf("%d", &dec...
calculate.c:18:3: Dangerous equality comparison involving float types:
        SuccessNumber == 0
Two real (float, double, or long double) values are compared directly using
== or != in pre-11.0. This may produce unexpected results since floating point
representations are inexact. Instead, compare the difference to FLT_EPSILON
or DBL_EPSILON. (Use -realcompare to inhibit warning)
calculate.c:18:3: Return value type double does not match declared type float:
        &dec * 100
```

Figure 10: Анализ кода файла calculate.c

```
insp@fennofedora lab_prog$ splint main.c
Splint 3.1.2 --- 23 Jul 2021

calculate.h:7:37: 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 -fixedformarray to inhibit warning)
main.c: (in function main)
main.c:18:2: Return value (type int) ignored: scanf("%d", &dec...
Result returned by function call is not used. If this is intended, can cast
result to void() to eliminate message. (Use -retvalue to inhibit warning)
main.c:19:13: Format argument 1 to mixed %d() expects char * path char [%d] =
        Operation
Type of parameter is not consistent with corresponding code to format string.
(Use -formattype to inhibit warning)
main.c:19:13: Converting format code
main.c:19:13: Return value (type int) ignored: scanf("%d", &dec...
Finished checking --- a code warnings
insp@fennofedora lab_prog$
```

Figure 11: Анализ кода файла main.c

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