

# Средства, применяемые при разработке программного обеспечения в ОС типа Unix/Linux

Roman P.Li

NEC–2022, 28 May – 4 June, 2022 Moscow, the Russian Federation

## Цель работы

Приобрести простейшие навыки разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linux на примере создания на языке программирования C 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.
3. Выполните компиляцию программы посредством `gcc`
4. При необходимости исправьте синтаксические ошибки.
5. Создайте `Makefile` со следующим содержанием:

```
1 #
2 # Makefile
3 #
4
5 CC = gcc
6 CFLAGS =
7 LIBS = -lm
8
9 calcul: calculate.o main.o
10 gcc calculate.o main.o -o calcul $(LIBS)
```

```

11
12 calculate.o: calculate.c calculate.h
13 gcc -c calculate.c $(CFLAGS)
14
15 main.o: main.c calculate.h
16 gcc -c main.c $(CFLAGS)
17
18 clean:
19 -rm calcul *.o *~
20
21 # End Makefile

```

Поясните в отчёте его содержание.

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

- Запустите отладчик GDB, загрузив в него программу для отладки.
- Для запуска программы внутри отладчика введите команду run
- Для постраничного (по 9 строк) просмотра исходного код используйте команду list:
- Для просмотра строк с 12 по 15 основного файла используйте list с параметрами
- Для просмотра определённых строк не основного файла используйте list с параметрами
- Установите точку останова в файле calculate.c на строке номер 21
- Выведите информацию об имеющихся в проекте точка останова
- Запустите программу внутри отладчика и убедитесь, что программа остановится в момент прохождения точки останова
- Отладчик выдаст следующую информацию

```

1 #0 Calculate (Numeral=5, Operation=0x7ffffffd280 "-")
2 at calculate.c:21
3 #1 0x000000000400b2b 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:

```
1 print Numeral
```

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

```
1 display Numeral
```

- Уберите точки останова:

```
1 info breakpoints
2 delete 1

```

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

## Выполнение лабораторной работы

Описываются проведённые действия, в качестве иллюстрации даётся ссылка на иллюстрацию (рис. [-@fig:001])

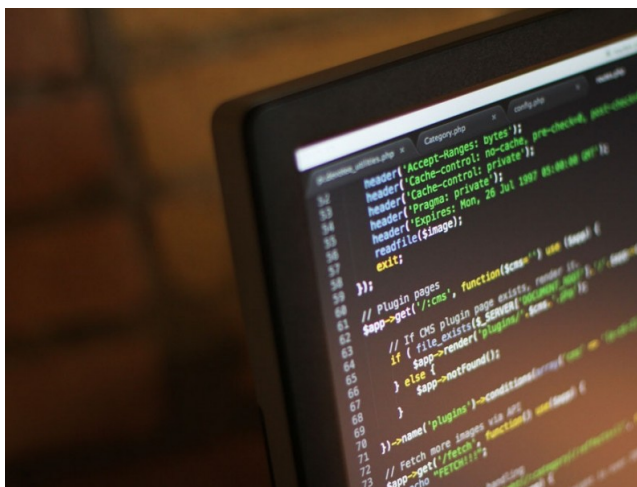


Рис. 1: Название рисунка

### Задание 1.

- Создадим в домашнем каталоге подкаталог '~/work/os/lab\_prog

`mkdir ~/work/os/lab_prog`

### Задание 2.

- Создадим в новом каталоге файлы calculate.h, calculate.c, main.c

`cd ~/work/os/lab_prog`

`touch calculate.h calculate.c main.c`

- Скопируем текст программ из лабораторной работы в эти файлы.

`emacs &`

### Задание 3.

-Выполним компиляцию программу посредством gcc(рис. [-@fig:001]):

```
gcc -c calculate.c
gcc -c main.c
gcc calculate.o main.o -o calcul -lm
```

```
[rpli@fedora lab_prog]$ gcc -c calculate.c
[rpli@fedora lab_prog]$ gcc -c main.c
[rpli@fedora lab_prog]$ gcc calculate.o main.o -o calcul -lm
```

Рис. 2: Компиляцию посредством gcc

#### Задание 4.

- Синтаксических ошибок не обнаружено

#### Задание 5.

- Создадим makefile с требуемым содержанием(см. лабораторную работу н. 13) в каталоге tech\_prog

touch Makefile  
emacs Makefile

Пояснение содержания makefile:

5-7 строки-локальные переменные. 9, 15, 18(синим)-названия процессов. 9, 10, 12, 13, 15, 16, 19-команды для терминала. Таким образом, наш makefile выполняет следующие действия: 1. Компилирует объектные файлы в executable 2. Компилирует программу calculate.c 3. Компилирует программу main.c 4. Удаляет оставшиеся объектные файлы.

#### Задание 6.

- В моем случае в makefile не хватало знаком табуляции(без них никак) и значение переменной должно быть установлено как -g, чтобы в будущем работать с отладчиком(рис. [-@fig:002]):

```
[aspetro@fedora lab_prog]$ make
Makefile:5: *** missing separator. Stop.
[aspetro@fedora lab_prog]$ make
make: 'calcul' is up to date.
[aspetro@fedora lab_prog]$ make clean
rm calcul *.o *~
[aspetro@fedora lab_prog]$ ls
calculate.c calculate.h main.c Makefile
[aspetro@fedora lab_prog]$ gcc -c calculate.c
gcc -c main.c
gcc calculate.o main.o -o calcul $
/usr/bin/ld: cannot find $: No such file or directory
collect2: error: ld returned 1 exit status
make: *** [Makefile:9: calcul] Error 1
[aspetro@fedora lab_prog]$ make
gcc calculate.o main.o -o calcul -lm
[aspetro@fedora lab_prog]$ ls
calcul calculate.c calculate.h calculate.o main.c main.o Makefile
[aspetro@fedora lab_prog]$ ./calcul
Hello: 12
[aspetro@fedora lab_prog]$
```

```
1 #
2 # Makefile
3 #
4 CC = gcc
5 CFLAGS =
6 LIBS = -lm
7
8 calcul: calculate.o main.o
9 gcc calculate.o main.o -o calcul $(LIBS)
10
11 calculate.o: calculate.c calculate.h
12 gcc -c calculate.c $(CFLAGS)
13
14 main.o: main.c calculate.h
15 gcc -c main.c $(CFLAGS)
16
17 clean:
18 rm calcul *.o *~
19
20 # End Makefile
```

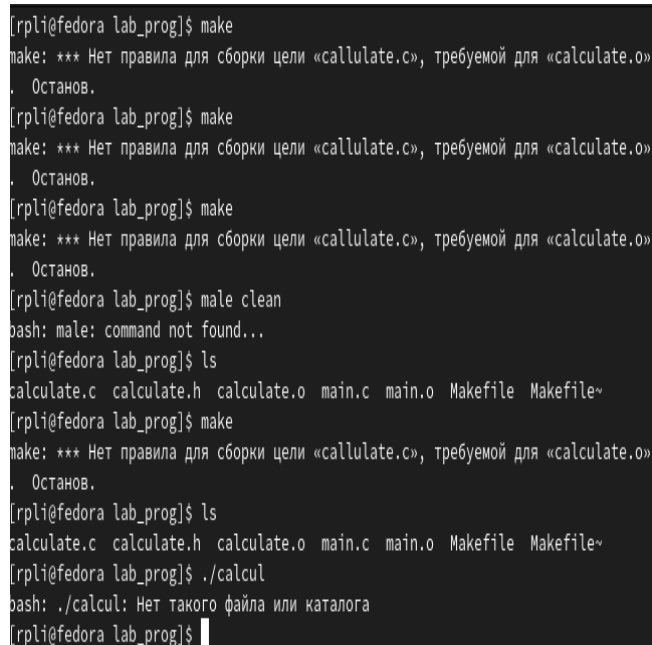
Рис. 3: Проверка makefile

1. Запустим отладчик нашего приложения с помощью команды(прежде обязательно пропишите make)(рис. [-@fig:003])

make // Если не создали еще исполняемый файл(в папке tech\_prog).  
gdb ./calcul

## 2. Запустим исполняемый файл внутри

run



```
[rpli@fedora lab_prog]$ make
make: *** Нет правила для сборки цели «calculate.c», требуемой для «calculate.o».
. Останов.
[rpli@fedora lab_prog]$ make
make: *** Нет правила для сборки цели «calculate.c», требуемой для «calculate.o».
. Останов.
[rpli@fedora lab_prog]$ make
make: *** Нет правила для сборки цели «calculate.c», требуемой для «calculate.o».
. Останов.
[rpli@fedora lab_prog]$ make clean
bash: make: command not found...
[rpli@fedora lab_prog]$ ls
calculate.c calculate.h calculate.o main.c main.o Makefile Makefile~
[rpli@fedora lab_prog]$ make
make: *** Нет правила для сборки цели «calculate.c», требуемой для «calculate.o».
. Останов.
[rpli@fedora lab_prog]$ ls
calculate.c calculate.h calculate.o main.c main.o Makefile Makefile~
[rpli@fedora lab_prog]$ ./calcul
bash: ./calcul: Нет такого файла или каталога
[rpli@fedora lab_prog]$
```

Рис. 4: Запуск и проверка работоспособности

## 3. Установим точку остановки в файле calculate.c на строке 21(рис. [-@fig:004]) и проверим значение переменной Numeral в точке остановки и в конце программы(рис. [-@fig:004]) :

```
list calculate.c:20, 27
break 21
info breakpoints
run
5
-
backtrace
print Numeral
display Numeral
info breakpoints
```

delete 1

```
(gdb) list calculate.c:20, 27
20      {
21          printf("Вычитаемое: ");
22          scanf("%f",&SecondNumeral);
23          return (Numeral - SecondNumeral);
24      }
25      else if (strcmp(Operation, "+") == 0)
26      {
27          printf("Множитель: ");
(gdb) break 21
Breakpoint 1 at 0x40120f: file calculate.c, line 21.
(gdb) info breakpoints
Num Type             Disp Enb Address            What
1 breakpoint keep y  0x000000000040120f in Calculate at calculate.c:21
(gdb) run
Starting program: /home/aepetrov/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.
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): -
Breakpoint 1, Calculate (Numeral=5, Operation=0x7fffffffdb74 "-") at calculate.c:21
21      printf("Вычитаемое: ");
(gdb) backtrace
#0 Calculate (Numeral=5, Operation=0x7fffffffdb74 "-") at calculate.c:21
#1 0x00000000004014eb in main () at main.c:17
(gdb) display numeral
No symbol "numeral" in current context.
(gdb) display Numeral
1: Numeral = 5
(gdb) info breakpoints
Num Type             Disp Enb Address            What
1 breakpoint keep y  0x000000000040120f in Calculate at calculate.c:21
1 breakpoint already hit 1 time
(gdb) delete 1
```

Рис. 5: Установка точки останова и проверка переменной в разных частях программы с последующим удалением

## Задание 7

-См. рис. [-@fig:005] и рис. [-@fig:006]

splint calculate.c

splint main.c

## Выводы

С помощью данной лабораторной работы я научился работать с компилятором, компилировать программные файлы языка C/C++, создавать сценарии работы в makefile с помощью утилиты make и так же заниматься отладкой программы.

```

splint 3.1.2 --- 22 Jan 2022

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 -fixedformalarray to inhibit warning)
calculate.c:10:31: Function parameter Operation declared as manifest array
(size constant is meaningless)
calculate.c: (in function Calculate)
calculate.c:16:2: 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:22:2: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:28:2: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:34:2: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:35:5: Dangerous equality comparison involving float types:
SecondNumeral == 0
Two real (float, double, or long double) values are compared directly using
== or != primitive. 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:38:8: Return value type double does not match declared type float:
(HUGE_VAL)
To allow all numeric types to match, use +relaxtypes.
calculate.c:46:2: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:47:9: Return value type double does not match declared type float:
(pow(Numeral, SecondNumeral))
calculate.c:50:8: Return value type double does not match declared type float:
(sqrt(Numeral))
calculate.c:52:9: Return value type double does not match declared type float:
(sin(Numeral))
calculate.c:54:9: Return value type double does not match declared type float:
(cos(Numeral))
calculate.c:56:10: Return value type double does not match declared type float:
(tan(Numeral))
calculate.c:60:8: Return value type double does not match declared type float:
(HUGE_VAL)

finished checking --- 15 code warnings

```

Рис. 6: splint calculate.c

```

splint 3.1.2 --- 22 Jan 2022

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 -fixedformalarray to inhibit warning)
main.c: (in function main)
main.c:14:4: 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:16:15: Format argument 1 to scanf (%s) 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:16:12: Corresponding format code
main.c:16:4: Return value (type int) ignored: scanf("%s", &Ope...

finished checking --- 4 code warnings

```

Рис. 7: splint main.c