

Лабораторная работа н. 13

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

Петров Артем Евгеньвич

Содержание

Цель работы	4
Задание	5
Выполнение лабораторной работы	8
Задание 1.	8
Задание 2.	8
Задание 3.	8
Задание 4.	9
Задание 5.	9
Задание 6.	9
Задание 7	12
Выводы	15
Контрольные вопросы	15

Список иллюстраций

0.1	Компиляцию посредством gcc	9
0.2	Проверка makefile	10
0.3	Запуск и проверка работоспособности	11
0.4	Установка точки останова и проверка переменной в разных частях программы с последующим удалением	12
0.5	splint calculate.c	13
0.6	splint main.c	14

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.
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 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:

```
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

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
```



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

Рис. 0.1: 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]):

```

[aeetrov@fedora lab_prog]$ make
Makefile:1: *** missing separator. Stop.
[aeetrov@fedora lab_prog]$ make
make: 'calcul' is up to date.
[aeetrov@fedora lab_prog]$ make clean
rm calcul *.o *~
[aeetrov@fedora lab_prog]$ ls
calculate.c calculate.h main.c Makefile
[aeetrov@fedora lab_prog]$ make
gcc -c calculate.c
gcc -c main.c
gcc calculate.o main.o -o calcul $
/usr/bin/ld: cannot find $: No such file or directory
collect2: error: ld returned 1 exit status
make: *** [Makefile:9: calcul] Error 1
[aeetrov@fedora lab_prog]$ make
gcc calculate.o main.o -o calcul -lm
[aeetrov@fedora lab_prog]$ ls
calcul calculate.c calculate.h calculate.o main.c main.o Makefile
[aeetrov@fedora lab_prog]$ ./calcul
Число: 12
Операция (*,-,+,/,pow,sqrt,sin,cos,tan): sqrt

```

```

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. 0.2: 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

```

[aepetrov@fedora lab_prog]$ gdb ./calcul
GNU gdb (GDB) Fedora 12.1-1.fc36
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...

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.
(No debugging symbols found in ./calcul)
(gdb) run
Starting program: /home/aepetrov/work/os/lab_prog/calcul
Downloading 0.03 MB separate debug info for system-supplied DSO at 0x7ffff7fc4000
Downloading 2.25 MB separate debug info for /lib64/libm.so.6
Downloading 7.39 MB separate debug info for /lib64/libc.so.6
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".
Число: 25
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): +
Второе слагаемое: 34
59.00
[Inferior 1 (process 284015) exited normally]
(gdb) list
Downloading 0.00 MB source file /usr/src/debug/glibc-2.35-11.fc36.x86_64/elf/sofini.c
1      /* Terminate the frame unwind info section with a 4byte 0 as a sentinel;
2         this would be the 'length' field in a real FDE.  */
3
4      typedef unsigned int ui32 __attribute__((mode(SI)));
5      static const ui32 __FRAME_END__[1]
6          __attribute__((used, section(".eh_frame")))
7          = { 0 };

```

Рис. 0.3: 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(strncmp(Operation, "*", 1) == 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
breakpoint already hit 1 time
(gdb) delete 1

```

Рис. 0.4: Установка точки остановки 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

```
[aepetrov@fedora lab_prog]$ 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)
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. 0.5: splint calculate.c

```

[ae Petrov@fedora lab_prog]$ splint main.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. 0.6: 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.

Контрольные вопросы

1. Можно использовать стандартные команды для получения справки - `man gcc`, `man make`, `man gdb` ну или вместо утилиты `man` использовать команду `--help`
2. Основные этапы разработки приложений в Unix: Создание исходного кода (написание в IDE) -> Сохранение промежуточных файлов или альтернативных веток разработки исходного кода -> Компиляция исходных файлов или их интерпритация в зависимости от выбранного языка программирования и/или системы сборки проектов -> Тестирование проекта который был собран -> Запись в соответствующую ветку разработки Git (main или dev, по-умолчанию)
3. Суффикс - нужен для определения расширения в контексте файловой системы или компилятора с помощью которого будет производиться компиляция или интерпретация исходного кода в работающую программу (например `hello1.py` компилируется только `ipython`, а вот `hello2.c` компилируется только `gcc`, `Cmake`)
4. Компилятор Си предназначен для компиляции внутренних файлов системы без полного скачивания программ, а просто скачав исходный код системных утилит и произвести с помощью встроенного компилятора компиляцию системных утилит

5. Утилита make - предназначена для упрощения разработки приложений, путем написания файла конфигурации который описывает пути компиляции для компилятора языка программирования
6. Можно использовать пример из лабораторной работы
7. Пошаговая отладка программ (трассировка) - её суть заключается в пошаговом выполнении каждой строки кода
8. Основные команды отладчика gdb:
 - backtrace - вывод на экран путь к текущей точке останова.
 - break - установить точку останова (строка или функция)
 - clear - удалить все точки останова в функции
 - continue - продолжить выполнение программы
 - delete (n) - удалить точку останова
 - display - добавить выражение в список выражений, значения которых отображаются при достижении точки останова программы
 - finish - выполнить программу до момента выхода из функции
 - info breakpoints - вывести на экран список используемых точек останова
 - info watchpoints - вывести на экран список используемых контрольных выражений
 - list - вывести на экран исходный код (в качестве параметра может быть указано название файла и через двоеточие номера начальной и конечной строк)
 - next - выполнить программу пошагово, но без выполнения вызываемых в программе функций
 - print - вывести значение указываемого в качестве параметра выражения
 - run - запуск программы на выполнение
 - set[variable] - установить новое значение переменной
 - step - пошаговое выполнение программы
 - watch - установить контрольное выражение, при изменении значения которого программа будет остановлена
9. Мои действия при отладке программ: Запустил Makefile -> Начал отлад-

ку (run) -> Вывел содержимое main файла -> Установил точку останова в main файле -> Продолжил выполнение (run) -> Использовал команды print & display для вывод промежуточных данных -> Удалил точку останова -> Закончил отладку

10. Нейтральная реакция компилятора, т.е. программных ошибок обнаружено не было
11. cppcheck, splint, cscope и другие
12. Проверка корректности аргументов и поиск ошибок и значений в программе которые могут быть улучшены, а также оценка всей программы