

Операционные системы

Лабораторная работа №14

Гульдяев Тихон Дмитриевич

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
3	Выводы	15
4	Ответы на контрольные вопросы	16
	Список литературы	20

Список таблиц

Список иллюстраций

2.1	Создание подкаталога ~/work/os/lab_prog	6
2.2	Создание файлов calculate.h, calculate.c, main.c	6
2.3	Исправление ошибки на 14-й строке	6
2.4	Две компиляции и проверка работы	7
2.5	Makefile	7
2.6	Запуск отладчика и команда run	9
2.7	Команда list с параметрами и без	10
2.8	Установка точки останова и информация о точках останова	10
2.9	Запуск программы с точкой останова	11
2.10	Команда backtrace	11
2.11	Команда print и display	11
2.12	Удаление точек останова	12
2.13	Удаление точек останова	13
2.14	Удаление точек останова	14

1. 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.

2. Выполнение лабораторной работы

В домашнем каталоге создаю подкаталог ~/work/os/lab_prog. (рис. 2.1).

```
guldyayev-tikhon@guldyayevtikhon:~/work/os$ mkdir lab_prog
```

Рис. 2.1: Создание подкаталога ~/work/os/lab_prog

Создаю в нём файлы: calculate.h, calculate.c, main.c.(рис. 2.2) И заполняю их согласно приложенному коду.

```
guldyayev-tikhon@guldyayevtikhon:~/work/os$ cd lab_prog/  
guldyayev-tikhon@guldyayevtikhon:~/work/os/lab_prog$ touch calculate.h calculate.c main.c
```

Рис. 2.2: Создание файлов calculate.h, calculate.c, main.c

Выполняю компиляцию программы посредством gcc, с первого раза не получается из-за ошибки, после ее исправления (рис. 2.3) все компилируется и я проверяю работу калькулятора (рис. 2.4)

```
14    scanf("%s", Operation);
```

Рис. 2.3: Исправление ошибки на 14-й строке

```
guldyaev-tikhon@guldyaev-tikhon:~/work/os/lab_prog$ gcc -c calculate.c
guldyaev-tikhon@guldyaev-tikhon:~/work/os/lab_prog$ gcc -c main.c
main.c: In function 'main':
main.c:14:9: warning: format '%s' expects argument of type 'char *', but argument 2 has type 'char (*)[4]' [-Wformat=]
14 | scanf("%s", &operation);
    |         ^
    |         |
    |         char (*)[4]
    |         char *
guldyaev-tikhon@guldyaev-tikhon:~/work/os/lab_prog$ gcc -c main.c
guldyaev-tikhon@guldyaev-tikhon:~/work/os/lab_prog$ gcc calculate.o main.o -o calcul -lm
guldyaev-tikhon@guldyaev-tikhon:~/work/os/lab_prog$ ./calcul
Число: 1
Операция (+, -, *, /, pow, sqrt, sin, cos, tan): +
Второе слагаемое: 2
3.00
```

Рис. 2.4: Две компиляции и проверка работы

Создаю Makefile согласно приложенному коду (рис. 2.5).

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

Рис. 2.5: Makefile

Этот Makefile используется для компиляции и создания исполняемого файла calcul. Давайте рассмотрим его содержимое по шагам:

1. CC = gcc: Эта строка определяет переменную CC, которая устанавливает компилятор C на gcc.
2. CFLAGS =: Эта строка определяет переменную CFLAGS, которая представляет флаги компилятора C. В данном случае, переменная не содержит никаких флагов, поэтому она остается пустой.
3. LIBS = -lm: Эта строка определяет переменную LIBS, которая представляет библиотеки, которые должны быть связаны с программой при компиляции. В данном случае, -lm указывает на библиотеку математических функций.

4. `calcul: calculate.o main.o`: Эта строка определяет цель `calcul`, которая зависит от файлов `calculate.o` и `main.o`. Если эти файлы изменились, цель `calcul` будет пересобрана.
5. `gcc calculate.o main.o -o calcul $(LIBS)`: Эта строка определяет правило для создания исполняемого файла `calcul`. Она использует компилятор `gcc` для компиляции `calculate.o` и `main.o` вместе с библиотеками, определенными в переменной `LIBS`, и создает исполняемый файл `calcul`.
6. `calculate.o: calculate.c calculate.h`: Эта строка определяет правило для создания файла `calculate.o`. Она указывает, что `calculate.o` зависит от файлов `calculate.c` и `calculate.h`. Если эти файлы изменились, `calculate.o` будет пересобран.
7. `gcc -g -c calculate.c $(CFLAGS)`: Эта строка определяет команду компиляции для файла `calculate.c`. Она использует компилятор `gcc` с флагом `-g` для включения отладочной информации и флагами, определенными в переменной `CFLAGS`, и создает объектный файл `calculate.o`.
8. `main.o: main.c calculate.h`: Эта строка определяет правило для создания файла `main.o`. Она указывает, что `main.o` зависит от файлов `main.c` и `calculate.h`. Если эти файлы изменились, `main.o` будет пересобран.
9. `gcc -g -c main.c $(CFLAGS)`: Эта строка определяет команду компиляции для файла `main.c`. Она использует компилятор `gcc` с флагом `-g` для включения отладочной информации и флагами, определенными в переменной `CFLAGS`, и создает объектный файл `main.o`.
10. `clean: rm -f calcul .o`: Эта строка определяет цель `clean`, которая используется для удаления всех объектных файлов (`.o`) и исполняемого файла `calcul`. При запуске команды `make clean` все эти файлы будут удалены.

С помощью `gdb` выполняю отладку программы `calcul`.

Запускаю отладчик `GDB`, загрузив в него программу для отладки и ввожу команду `run` (рис.

2.6)


```

guldyayev-tikhon@guldyayev-tikhon:~/work/os/lab_prog$ gdb ./calcul
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04.1) 9.2
Copyright (C) 2020 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-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://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...
(gdb) run
Starting program: /home/guldyayev-tikhon/work/os/lab_prog/calcul
Число: 1
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): +
Второе слагаемое: 3
4.00
[Inferior 1 (process 38916) exited normally]

```

Рис. 2.6: Запуск отладчика и команда run

Для постраничного (по 9 строк) просмотра исходного кода использую команду list, а для просмотра строк с 12 по 15 основного файла использую list с параметрами(list calculate.c:20,29) и для просмотра определённых строк не основного файла использую list с параметрами(list calculate.c:20,29) (рис. 2.7).

```

(gdb) list
1      ///////////////////////////////////////////////////
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;
(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);
(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)
(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);

```

Рис. 2.7: Команда list с параметрами и без

Устанавливаю точку останова в файле calculate.c на строке номер 21 и вывожу информацию об имеющихся в проекте точка останова (рис. 2.8).

```

(gdb) break 21
Breakpoint 1 at 0x55555555306: file calculate.c, line 21.
(gdb) info breakpoints
Num   Type             Disp Enb Address                  What
1     breakpoint      keep y   0x000055555555306  in Calculate
                                     at calculate.c:21

```

Рис. 2.8: Установка точки останова и информация о точках останова

Запускаю программу внутри отладчика и вижу, что программа останавливается в момент

прохождения точки останова (рис. 2.9).

```
(gdb) run
Starting program: /home/guldyayev-tikhon/work/os/lab_prog/calcul
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): -
Вычитаемое: 4

Breakpoint 1, Calculate (Numeral=5, Operation=0x7fffffffde04 "-")
    at calculate.c:21
21      return(Numeral - SecondNumeral);
```

Рис. 2.9: Запуск программы с точкой останова

Команда backtrace показывает весь стек вызываемых функций от начала программы до текущего места (рис. 2.10).

```
(gdb) backtrace
#0 Calculate (Numeral=5, Operation=0x7fffffffde04 "-") at calculate.c:21
#1 0x00005555555555bd in main () at main.c:15
```

Рис. 2.10: Команда backtrace

Смотрю, чему равно на этом этапе значение переменной Numeral (print Numeral) и сравниваю с результатом вывода на экран после использования команды display Numeral, оба вывода равны 5 (рис. 2.11).

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

Рис. 2.11: Команда print и display

Уберию точки останова (рис. 2.12).

```
(gdb) info breakpoints
Num      Type          Disp Enb Address            What
1        breakpoint  keep y  0x0000555555555306 in calculate
                                at calculate.c:21
breakpoint already hit 1 time
(gdb) delete 1
(gdb) info breakpoints
No breakpoints or watchpoints.
```

Рис. 2.12: Удаление точек останова

С помощью утилиты splint анализирую код файла calculate.c (рис. 2.13).

1. calculate.h:5:37: Предупреждение указывает на то, что параметр функции Operation объявлен как манифестный массив (с указанием размера), но размер игнорируется, так как параметр массива обрабатывается как указатель. Предлагается использовать флаг `-fixedformalarray`, чтобы подавить это предупреждение.
2. calculate.c:8:31: То же предупреждение, как и в предыдущем случае, указывает на то, что параметр функции Operation объявлен как манифестный массив, но размер игнорируется.
3. calculate.c:14:1, calculate.c:20:1, calculate.c:26:1, calculate.c:32:1, calculate.c:44:1: Предупреждения указывают на игнорирование возвращаемого значения функции `scanf`. Результат функции `scanf` не используется. Можно привести результат к типу `(void)`, чтобы устранить предупреждение. Использование флага `-retvalint` подавляет это предупреждение.
4. calculate.c:33:4: Предупреждение указывает на опасное сравнение на равенство (`==`) между числами с плавающей запятой (`float`). Рекомендуется сравнивать разницу между числами с плавающей запятой и использовать `FLT_EPSILON` или `DBL_EPSILON`. Использование флага `-realcompare` подавляет это предупреждение.
5. calculate.c:36:7, calculate.c:45:7, calculate.c:48:7, calculate.c:50:7, calculate.c:52:7, calculate.c:54:7, calculate.c:58:7: Предупреждения указывают на несоответствие типов возвращаемого значения функций. Возвращаемое значение указано как `double`, но ожидается тип `float`. Использование флага `-relaxtypes` позволяет сопоставлять все числовые типы.

```

guldyaev-tikhon@guldyaev-tikhon:~/work/os/lab_prog$ splint calculate.c
Splint 3.1.2 --- 20 Feb 2018

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
  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:36:7: Return value type double does not match declared type float:
    (HUGE_VAL)
  To allow all numeric types to match, use +relaxtypes.
calculate.c:44:1: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:45:7: Return value type double does not match declared type float:
    (pow(Numeral, SecondNumeral))
calculate.c:48:7: Return value type double does not match declared type float:
    (sqrt(Numeral))
calculate.c:50:7: Return value type double does not match declared type float:
    (sin(Numeral))
calculate.c:52:7: Return value type double does not match declared type float:
    (cos(Numeral))
calculate.c:54:7: Return value type double does not match declared type float:
    (tan(Numeral))
calculate.c:58:7: Return value type double does not match declared type float:
    (HUGE_VAL)
Finished checking --- 15 code warnings

```

Рис. 2.13: Удаление точек останова

С помощью утилиты splint анализирую код файла main.c (рис. 2.14).

1. calculate.h:5:37: Предупреждение указывает на то, что параметр функции Operation объявлен как манифестный массив (с указанием размера), но размер игнорируется, так как параметр массива обрабатывается как указатель. Предлагается использовать флаг -fixedformalarray, чтобы подавить это предупреждение.
2. main.c:12:1, main.c:14:1: Предупреждения указывают на игнорирование возвращаемого значения функции scanf. Результат функции scanf не используется. Можно привести результат к типу (void), чтобы устранить предупреждение. Использование флага -retvalint подавляет это предупреждение.

```
guldyaev-tikhon@guldyaevtikhon:~/work/os/lab_prog$ splint main.c
Splint 3.1.2 --- 20 Feb 2018

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:14:1: Return value (type int) ignored: scanf("%s", Oper...

Finished checking --- 3 code warnings
guldyaev-tikhon@guldyaevtikhon:~/work/os/lab_prog$
```

Рис. 2.14: Удаление точек останова

3. Выводы

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

4. Ответы на контрольные вопросы

1. Как получить информацию о возможностях программ gcc, make, gdb и др.?

Чтобы получить информацию о возможностях программ gcc, make, gdb и других утилит в UNIX, вы можете использовать следующие команды:

- gcc –help - выведет информацию о доступных опциях компилятора gcc.
- make –help - покажет справку по использованию утилиты make.
- gdb –help - выведет информацию о командах и опциях отладчика gdb.

2. Назовите и дайте краткую характеристику основным этапам разработки приложений в UNIX.

Основные этапы разработки приложений в UNIX:

- Написание исходного кода на выбранном языке программирования.
- Компиляция и сборка программы с использованием компилятора и утилиты make.
- Запуск и отладка программы с помощью отладчика gdb.
- Тестирование и исправление ошибок.
- Развертывание и использование программы.

3. Что такое суффикс в контексте языка программирования? Приведите примеры использования.

В контексте языка программирования суффикс - это часть имени файла, которая указывает на его тип или назначение. Например, в имени файла program.cc суффикс .cc указывает на то, что это файл с исходным кодом на языке программирования C++.

4. Каково основное назначение компилятора языка C в UNIX?

Основное назначение компилятора языка C в UNIX - это преобразование исходного кода на языке C в машинный код, который может быть исполнен компьютером. Компилятор выполняет проверку синтаксиса и семантики программы, а затем создает исполняемый файл.

5. Для чего предназначена утилита make?

Утилита make предназначена для автоматизации процесса компиляции программного проекта. Она позволяет определить зависимости между файлами и указать правила для их компиляции. make проверяет, какие файлы были изменены, и компилирует только необходимые файлы для создания исполняемого файла или библиотеки.

6. Приведите пример структуры Makefile. Дайте характеристику основным элементам этого файла.

```
target: dependencies
    command1
    command2
    . . .
```

- target - цель, которую нужно построить (например, имя исполняемого файла).
- dependencies - зависимости, файлы, от которых зависит цель.
- command1, command2, ... - команды, которые нужно выполнить для построения

7. Назовите основное свойство, присущее всем программам отладки. Что необходимо сделать, чтобы его можно было использовать?

Основное свойство, присущее всем программам отладки, - это возможность просмотра и изменения состояния программы во время выполнения. Чтобы использовать отладчик, необходимо компилировать программу с опцией -g, чтобы включить отладочную информацию в исполняемый файл.

8. Назовите и дайте основную характеристику основным командам отладчика gdb.

Основные команды отладчика gdb:

- run - запустить программу или перезапустить её после остановки.
- break - установить точку останова на указанной строке кода.
- step - выполнить следующую строку кода и остановиться.
- print - вывести значение переменной или выражения.
- continue - продолжить выполнение программы до следующей точки останова.
- quit - выйти из отладчика gdb.

9. Опишите по шагам схему отладки программы, которую Вы использовали при выполнении лабораторной работы.

- Запуск gdb
- Запуск программы run
- Просмотр исходного кода командой list
- Установка точки останова на нужной строке
- Запуск программы run
- Нахождение и исправление ошибки
- Повторение процесса отладки

10. Прокомментируйте реакцию компилятора на синтаксические ошибки в программе при его первом запуске. При первом запуске компилятор обычно проверяет синтаксис программы и выдает сообщения об ошибках, таких как пропущенные точки с запятой, неправильное использование ключевых слов или несоответствие типов данных. Компилятор указывает на строки, содержащие ошибки, и предоставляет информацию о характере ошибок, чтобы помочь в их исправлении.

11. Назовите основные средства, повышающие понимание исходного кода программы.

Основные средства, повышающие понимание исходного кода программы:

- Комментарии в коде, которые поясняют назначение и логику работы различных частей программы.
- Описательные имена переменных, функций и классов, которые отражают их назначение и функциональность.

- Структурирование кода с использованием отступов и блоков, чтобы легче читать и понимать его структуру.
- Документация и руководства пользователя, которые описывают функциональность и использование программы.

12. Каковы основные задачи, решаемые программой splint? Программа splint предназначена для статического анализа и проверки исходного кода на наличие ошибок, несоответствий и потенциальных проблем. Она обнаруживает проблемы, связанные с типами данных, потенциальными ошибками в памяти, неиспользуемыми переменными и другими подобными проблемами. splint помогает улучшить качество и безопасность программного кода.

Список литературы

<https://www.google.ru>

<https://chat.openai.com/chat>