Отчёт по лабораторной работе №13

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

Аскеров Александр Эдуардович

Содержание

Список иллюстраций

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 Теоретическое введение

Этапы разработки приложений

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 нескольки

- анализ разработанного кода;

- сборка, компиляция и разработка исполняемого модуля;

- тестирование и отладка, сохранение произведённых изменений;

– документирование.

3 Выполнение лабораторной работы

1. В домашнем каталоге создадим подкаталог ~/work/os/lab_prog.

```
[aeaskerov@fedora os]$ mkdir ~/work/os/lab_prog
```

Рис. 3.1: Создание подкаталога lab_prog

2. Создадим в нём файлы: calculate.h, calculate.c, main.c и запишем в них код, приведённый в инструкции.

```
[aeaskerov@fedora lab_prog]$ touch calculate.h calculate.c main.c
```

Рис. 3.2: Создаём файлы для калькулятора

3. Выполним компиляцию программы посредством gcc.

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

Рис. 3.3: Компиляция программы

4. Исправим ошибки. Допишем –g.

5. Создадим Makefile.

Первые три строки указывают компилятор, флаг и библиотеки для создания файлов в последующих разделах кода. Clean отвечает за удаление объектных файлов.

```
#
# 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
```

Рис. 3.4: Makefile

6. С помощью gdb выполним отладку программы calcul.

- Запустим отладчик GDB, загрузив в него программу для отладки

```
[aeaskerov@fedora lab_prog]$ gdb ./calcul
```

Рис. 3.5: Запуск отладчика

- Для запуска программы внутри отладчика введём команду run

```
(gdb) run
```

Рис. 3.6: Запуск программы внутри отладчика

- Для постраничного (по 9 строк) просмотра исходного код используем команду list

```
(gdb) list
1       ////////////////////
2       // main.c
3
4       #include <stdio.h>
5       #include "calculate.h"
6
7       int
8       main (void)
9       {
10        float Numeral;
```

Рис. 3.7: Постраничный просмотр исходного кода

- Для просмотра строк с 12 по 15 основного файла используем list с параметрами

```
(gdb) list 12,15
12        float Result;
13        printf("Число: ");
14        scanf("%f",&Numeral);
15        printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): ");
(gdb)
```

Рис. 3.8: Просмотр строк с 12 по 15 основного файла

- Для просмотра определённых строк не основного файла используем list с параметрами

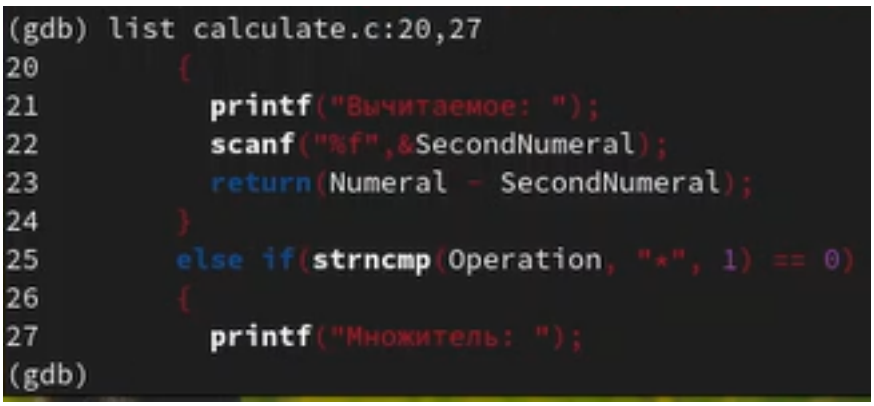

Рис. 3.9: Просмотр определённых строк не основного файла

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

Рис. 3.10: Точка останова на строке 21

- Выведем информацию об имеющихся в проекте точках останова

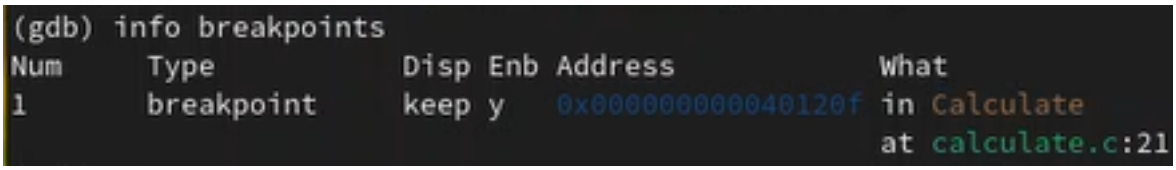

Рис. 3.11: Информация об имеющихся в проекте точках останова

- Запустим программу внутри отладчика и убедимся, что программа остановится в момент прохождения точки останова

```
(gdb) run
Starting program: /home/aeaskerov/work/os/lab_prog/calcul
[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=0x7fffffffdf24 "-") at calculate.c
:21
21          printf("Вычитаемое: ");
(gdb) backtrace
#0  Calculate (Numeral=5, Operation=0x7fffffffdf24 "-") at calculate.c:21
#1  0x00000000004014eb in main () at main.c:17
(gdb)
```

Рис. 3.12: Проверка работы точки останова

- Посмотрим, чему равно на этом этапе значение переменной Numeral.

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

Рис. 3.13: Значение переменной Numeral

- Сравним с результатом вывода на экран после использования команды

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

Рис. 3.14: Другой способ отображения

- Уберём точки останова

```
(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. 3.15: Удаление точек останова

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

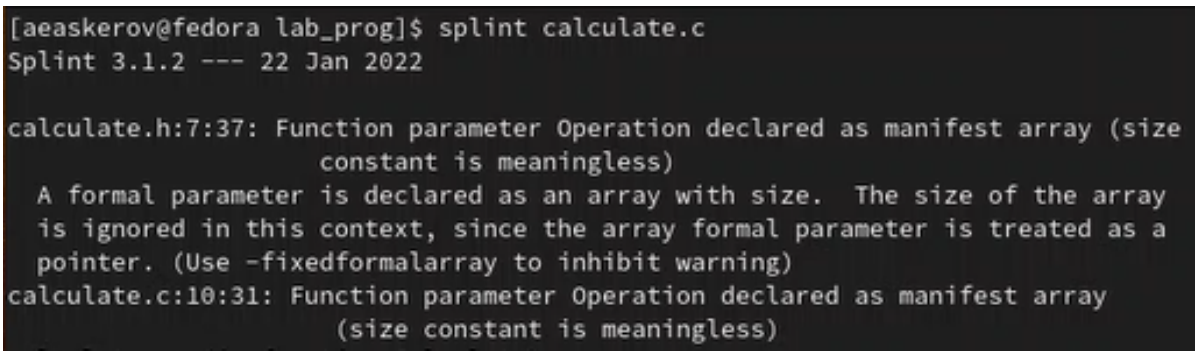

Рис. 3.16: Анализ программы calculate.c

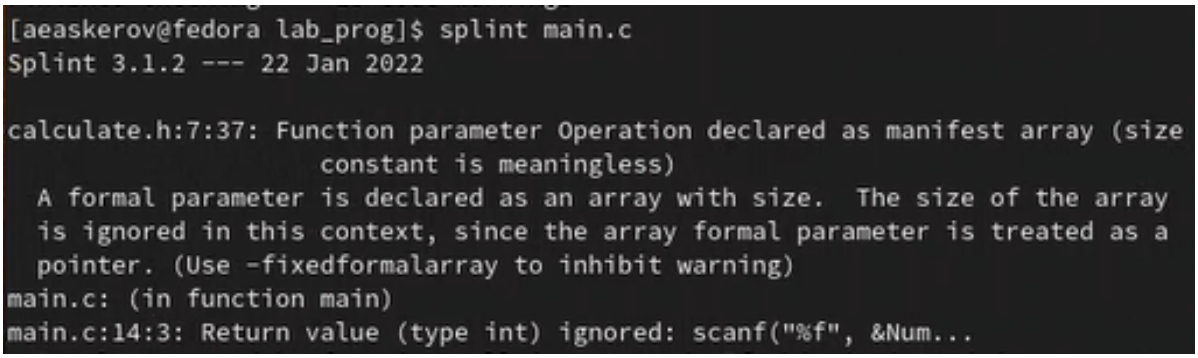

Рис. 3.17: Анализ программы main.c

4 Контрольные вопросы

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

Чтобы получить информацию о возможностях программ gcc, make, gdb и др. нужно воспользоваться командой man или опцией –help (-h) для каждой команды.

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

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; осборка, компиляция и разработка исполняемого модуля; отестирование и отладка, сохранение произведённых изменений;
- документирование. Для создания исходного текста программы разработчик может воспользоваться любым удобным для него редактором текста: vi, vim, mceditor, emacs, geanyи др. После завершения написания исходного кода программы (возможно состоящей из нескольких файлов), необходимо её скомпилировать и получить исполняемый модуль.

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

Для имени входного файла суффикс определяет какая компиляция требуется. Суффиксы указывают на тип объекта. Файлы с расширением (суффиксом) .c воспринимаются gcc как программы на языке C, файлы с расширением .cc или .C как файлы на языке C++, а файлы срасширением .o считаются объектными. Например, в команде «gcc -c main.c»: gcc по расширению (суффиксу) .c распознает тип файла для компиляции и формирует объектный модуль файл с расширением .o. Если требуется получить исполняемый файл с определённым именем (например, hello), то требуется воспользоваться опцией o и в качестве параметра задать имя создаваемого файла

4. Каково основное назначение компилятора языка C в UNIX?

Основное назначение компилятора языка Си в UNIX заключается в компиляции всей программы и получении исполняемого файла/модуля.

5. Для чего предназначена утилита make?

Для сборки разрабатываемого приложения и собственно компиляции полезно воспользоваться утилитой make. Она позволяет автоматизировать процесс преобразования файлов программы из одной формы в другую, отслеживает взаимосвязи между файлами.

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

Для работы с утилитой make необходимо в корне рабочего каталога с проектом создать файл с названием makefileили Makefile, в котором будут описаны правила обработки файлов программного комплекса. В самом простом случае Makefile имеет следующий синтаксис: … : …<команда 1>… Сначала задаётся список целей, разделённых пробелами, за которым идёт двоеточие и список

зависимостей. Затем в следующих строках указываются команды. Строки с командами обязательно должны начинаться с табуляции. В качестве цели в Makefile может выступать имя файла или название какого-то действия. Зависимость задаёт исходные параметры (условия) для достижения указанной цели. Зависимость также может быть названием какого-то действия. Команды собственно действия, которые необходимо выполнить для достижения цели. Общий синтаксис Makefile имеет вид: target1 [target2…]:[[dependment1…][(tab)commands] [#commentary][(tab)commands] [#commentary]. Здесь знак # определяет начало комментария (содержимое от знака # и до конца строки не будет обрабатываться. Одинарное двоеточие указывает на то, что последовательность команд должна содержаться в одной строке. Для переноса можно в длинной строке команд можно использовать обратный слэш (). Двойное двоеточие указывает на то, что последовательность команд может содержаться в нескольких последовательных строках.

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

Во время работы над кодом программы программист неизбежно сталкивается с появлением ошибок в ней. Использование отладчика для поиска и устранения ошибок в программе существенно облегчает жизнь программиста. В комплект программ GNUдля ОС типа UNIX входит отладчик GDB(GNU Debugger). Для использования GDB необходимо скомпилировать анализируемый код программы таким образом, чтобы отладочная информация содержалась в результирующем бинарном файле. Для этого следует воспользоваться опцией –g компилятора gcc: gcc –c file.c -g. После этого для начала работы с gdb необходимо в командной строке ввести одноимённую команду, указав в качестве аргумента анализируемый бинарный файл: gdb file.o

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

Основные команды отладчика gdb: 1. backtrace вывод на экран пути к текущей точке останова (по сути, вывод названий всех функций); 2. break установить точку останова (в качестве параметра может быть указан номер строки или название функции); 3. clear удалить все точки останова в функции; 4. continue продолжить выполнение программы; 5. delete удалить точку останова; 6. display добавить выражение в список выражений, значения которых отображаются при достижении точки останова программы; 7. finish выполнить программу до момента выхода из функции; 8. info breakpoints вывести на экран список используемых точек останова; 9. info watchpoints вывести на экран список используемых контрольных выражений; 10. list вывести на экран исходный 11. next выполнить программу пошагово, но без выполнения вызываемых в программе функций; 12. print вывести значение указываемого в качестве параметра выражения; 13. run запуск программы на выполнение; 14. set установить новое значение переменной; 15. step пошаговое выполнение программы; 16. watch установить контрольное выражение, при изменении значения которого программа будет остановлена. Для выхода из gdb можно воспользоваться командой quit (или её сокращённым вариантом q) или комбинацией клавиш Ctrl d. Более подробную информацию по работе с gdb можно получить с помощью команд gdb h и man gdb.

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

Схема отладки программы показана в 6 пункте лабораторной работы.

10. Прокомментируйте реакцию компилятора на синтаксические ошибки в программе при его первом запуске.

При первом запуске компилятор не выдал никаких ошибок, но в коде программы main.c допущена ошибка, которую компилятор мог пропустить (возможно, из-за версии 8.3.0-19): в строке scanf(“%s”, &Operation); нужно убрать знак &,

потому что имя массива символов уже является указателем на первый элемент этого массива.

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

Система разработки приложений UNIX предоставляет различные средства, повышающие понимание исходного кода. К ним относятся: cscope исследование функций, содержащихся в программе, lint критическая проверка программ, написанных на языке Си.

12. Каковы основные задачи, решаемые программой splint?

Утилита splint анализирует программный код, проверяет корректность задания аргументов использованных в программе функций и типов возвращаемых значений, обнаруживает синтаксические и семантические ошибки. В отличие от компилятора Санализатор splintгенерирует комментарии с описанием разбора кода программы и осуществляет общий контроль, обнаруживая такие ошибки, как одинаковые объекты, определённые в разных файлах, или объекты, чьи значения не используются в работе программы, переменные с некорректно заданными значениями и типами и многое другое.

5 Выводы

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