Лабораторная работа №13

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

Панченко Денис Дмитриевич

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 Выполнение лабораторной работы

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

```
[ddpanchenko@ddpanchenko ~]$ mkdir ~/work/os/lab_prog
```

Рис. 2.1: Создание подкаталога

Создадим в нём файлы: calculate.h, calculate.c, main.c (рис. 2.2).

```
[ddpanchenko@ddpanchenko ~]$ cd ~/work/os/lab_prog
[ddpanchenko@ddpanchenko lab_prog]$ touch calculate.h
[ddpanchenko@ddpanchenko lab_prog]$ touch calculate.c
[ddpanchenko@ddpanchenko lab_prog]$ touch main.c
```

Рис. 2.2: Создание файлов

Реализуем функцию калькулятора в файле calculate.c (рис. 2.3).

```
////////////////////////////////////////
// calculate.h

#ifndef CALCULATE_H_
#define CALCULATE_H_

float Calculate(float Numeral, char Operation[4]);

#endif /*CALCULATE_H_*/
```

Рис. 2.3: Реализация функции

Реализуем интерфейсный файл calculate.h, описывающий формат вызова функции-калькулятора (рис. 2.4).

```c
//////////////////////////////////////
// calculate.c

#include <stdio.h>
#include <math.h>
#include <string.h>
#include "calculate.h"

float
Calculate(float Numeral, char Operation[4])
{
  float SecondNumeral;
  if(strncmp(Operation, "+", 1) == 0)
    {
      printf("Второе слагаемое: ");
      scanf("%f",&SecondNumeral);
      return(Numeral + SecondNumeral);
    }
  else if(strncmp(Operation, "-", 1) == 0)
    {
      printf("Вычитаемое: ");
      scanf("%f",&SecondNumeral);
      return(Numeral - SecondNumeral);
    }
  else if(strncmp(Operation, "*", 1) == 0)
    {
      printf("Множитель: ");
      scanf("%f",&SecondNumeral);
      return(Numeral * SecondNumeral);
    }
```

Рис. 2.4: Реализация интерфейсного файла

Реализуем основной файл main.c, реализующий интерфейс пользователя к калькулятору (рис. 2.5).

```c
////////////////////////////////////////
// main.c

#include <stdio.h>
#include "calculate.h"

int
main (void)
{
  float Numeral;
  char Operation[4];
  float Result;
  printf("Число: ");
  scanf("%f",&Numeral);
  printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): ");
  scanf("%s",&Operation);
  Result = Calculate(Numeral, Operation);
  printf("%6.2f\n",Result);
  return 0;
}
```

Рис. 2.5: Реализация основного файла

Выполним компиляцию программы посредством gcc (рис. 2.6).

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

Рис. 2.6: Компиляция программы

Создадим Makefile со следующим содержанием (рис. 2.7 - 2.8).

```
[ddpanchenko@ddpanchenko lab_prog]$ touch Makefile
```

Рис. 2.7: Создание файла

```
#
# Makefile
#

CC = gcc
CFLAGS =
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. 2.8: Содержание файла

С помощью gdb выполним отладку программы calcul (рис. 2.9 - 2.19).

```
[ddpanchenko@ddpanchenko lab_prog]$ gdb ./calcul
GNU gdb (GDB) Fedora Linux 12.1-7.fc37
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.
```

Рис. 2.9: Запуск

```
(gdb) run
Starting program: /home/ddpanchenko/work/os/lab_prog/calcul
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".
Число: 12
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): -
Вычитаемое: 10
  2.00
[Inferior 1 (process 5571) exited normally]
```

Рис. 2.10: Запуск программы

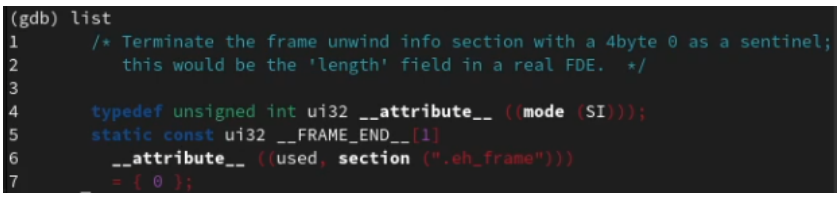

Рис. 2.11: Просмотр кода

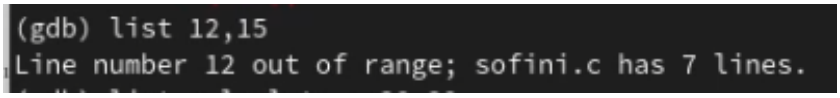

Рис. 2.12: Просмотр строк кода

Рис. 2.13: Просмотр строк неосновного кода

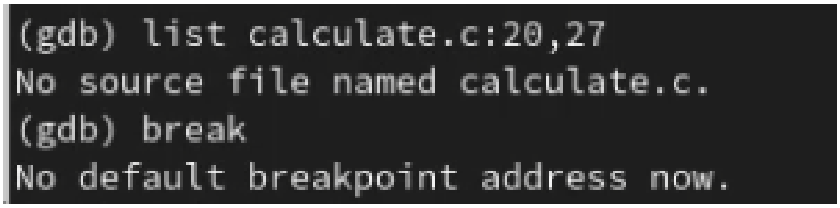

Рис. 2.14: Установка точки останова

Рис. 2.15: Вывод информации о точке

```
(gdb) run
Starting program: /home/ddpanchenko/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): -
Вычитаемое: backtrace
  5.00
[Inferior 1 (process 5600) exited normally]
```

Рис. 2.16: Запуск программы

```
(gdb) print Numeral
```

Рис. 2.17: Значение переменной

```
(gdb) display Numeral
```

Рис. 2.18: Сравнение

```
(gdb) info breakpoints
No breakpoints or watchpoints.
(gdb) delete 1
No breakpoint number 1.
```

Рис. 2.19: Удаление точки останова

3 Вывод

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

4 Контрольные вопросы

1. Информацию о возможностях программ gcc, make, gdb и др. можно получить из их официальной документации, а также из различных руководств и учебников по программированию на языке C и UNIX.

2. Основными этапами разработки приложений в UNIX являются:

- проектирование архитектуры приложения и выбор используемых технологий;
- написание и отладка исходного кода на языке C;
- компиляция исходного кода в исполняемый файл с помощью компилятора gcc;
- тестирование и отладка приложения с помощью отладчика gdb;
- сборка приложения и создание пакета для установки с помощью утилиты make.

3. Суффикс в контексте языка программирования - это часть имени файла, которая указывает на его тип или формат. Например, суффикс .c обозначает исходный код на языке C, а .o - объектный файл, полученный в результате компиляции исходного кода.

4. Основное назначение компилятора языка C в UNIX - преобразование исходного кода на языке C в машинный код, который может быть исполнен на компьютере.

5. Утилита make предназначена для автоматизации процесса сборки программы. Она позволяет определить зависимости между файлами и компонентами программы и автоматически проводить перекомпиляцию только тех файлов, которые изменились.

6. Пример структуры Makefile: CC=gcc CFLAGS=-Wall -Werror LDFLAGS=-lm

main: main.o utils.o $(CC) $(LDFLAGS) -o main main.o utils.o
main.o: main.c $(CC) $(CFLAGS) -c main.c
utils.o: utils.c utils.h $(CC) $(CFLAGS) -c utils.c

Основные элементы: - переменные, например CC, CFLAGS, LDFLAGS, которые определяют используемый компилятор, флаги компиляции и линковки; - цели, например main, которые соответствуют именам файлов, которые нужно собрать; - зависимости, например main.o: main.c, которые определяют зависимости между целями и файлами; - команды, например $(CC) $(LDFLAGS) -o main main.o utils.o, которые определяют действия, необходимые для сборки файла.

7. Основное свойство, присущее всем программам отладки, - возможность остановки выполнения программы в определенной точке и пошагового ее выполнения. Чтобы его можно было использовать, необходимо включить в исходный код программы отладочную информацию, которую компилятор добавляет в объектный файл при использовании опции -g.

8. Основные команды отладчика gdb:

- run - запустить программу;
- break - установить точку останова;
- next - выполнить следующую строку кода, не заходя внутрь функций;
- step - выполнить следующую строку кода, заходя внутрь функций;
- print - вывести значение переменной;
- watch - установить точку останова при изменении значения переменной;
- quit - выйти из отладчика.

9. Схема отладки программы, которую я использовал при выполнении лабораторной работы:

- компиляция исходного кода с опцией -g;
- запуск отладчика gdb;
- установка точек останова в соответствующих местах кода;
- выполнение программы пошагово с помощью команд next и step;
- вывод значений переменных с помощью команды print;
- исправление ошибок в исходном коде и повторная компиляция.

10. Компилятор при первом запуске реагирует на синтаксические ошибки в программе и сообщает о них, указывая номер строки и тип ошибки. Это позволяет исправить ошибки и повторно скомпилировать программу.

11. Основные средства, повышающие понимание исходного кода программы:

- комментарии в коде, поясняющие его структуру и логику;
- правильное именование переменных, функций и классов;
- форматирование кода для улучшения его читаемости;
- использование отладочных сообщений, которые выводят информацию о ходе выполнения программы.

12. Основные задачи, решаемые программой splint - проверка кода на соответствие стандартам и рекомендациям по программированию на языке C. Она позволяет выявлять потенциальные ошибки и уязвимости в коде, а также улучшать его читаемость и поддерживаемость.