Операционные системы

Лабораторная работа №13

Матюшкин Денис Владимирович (НПИбд-02-21)

Содержание

1 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.

2 Ход работы

1. В домашнем каталоге создадим подкаталог ~/work/os/lab_prog. Создадим в нём файлы: calculate.h, calculate.c, main.c (рис. 2.1).

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

Рис. 2.1: Создание файлов

2. 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. Реализация функций калькулятора в файле calculate.c (рис. 2.2). Интерфейсный файл calculate.h, описывающий формат вызова функции-калькулятора (рис. 2.3). Основной файл main.c, реализующий интерфейс пользователя к калькулятору (рис. 2.4).

```
////////////////////////////////////
// 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: ");
U:**-  calculate.c     Top L19     (C/*l Abbrev)
```

Рис. 2.2: Редактирование calculate.c

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

#ifndef CALCULATE_H_
#define CALCULATE_H_

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

#endif /*CALCULATE_H_*/
```

Рис. 2.3: Редактирование calculate.h

```
////////////////////////////////////////
// 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;
}
U:**-  main.c          All L17     (C/*l Abbrev)
```

Рис. 2.4: Редактирование main.c

3. Выполним компиляцию программы посредством gcc (рис. 2.5).

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

Рис. 2.5: Компиляция файлов

4. Исправим некоторые синтаксические ошибки.

5. Создадим Makefile со следующим содержанием (рис. 2.6). В этом файле мы создаем переменные CC, CFLAGS, LIBS. Инициализируем их. Создаем блоки, в которых прописываем какие команды будут выполняться. При этом подставляя значение нами созданных переменных.

```
#
# 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 *~
```

Рис. 2.6: Создание Makefile

6. С помощью gdb выполним отладку программы calcul (перед использованием gdb исправим Makefile) (рис. 2.7).

```
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 *~
```

Рис. 2.7: Редактирование Makefile

6.1. Запустим отладчик GDB, загрузив в него программу для отладки (рис. 2.8).

```
[dvmatyushkin@dvmatyushkin lab_prog]$ make
gcc -c calculate.c -g
gcc -c main.c -g
gcc calculate.o main.o -o calcul -lm
[dvmatyushkin@dvmatyushkin lab_prog]$ gdb ./calcul
GNU gdb (GDB) Fedora 11.2-2.fc35
Copyright (C) 2022 Free Software Foundation, Inc.
```

Рис. 2.8: Запуск отладки

6.2. Для запуска программы внутри отладчика введем команду run. Для постраничного (по 9 строк) просмотра исходного код используем команду list (рис. 2.9).

```
Using host libthread_db library "/lib64/libthread_db.so.1".
Число: 16
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): sqrt
  4.00
[Inferior 1 (process 10051) exited normally]
(gdb) list
1       ////////////////////////////////////////
2       // main.c
3
4       #include <stdio.h>
5       #include "calculate.h"
6
7       int main (void)
8       {
9         float Numeral;
10        char Operation[4];
```

Рис. 2.9: Работа с отладкой

6.3. Для просмотра строк с 12 по 15 основного файла используем list с параметрами. Для просмотра определённых строк не основного файла используем list с параметрами (рис. 2.10).

```
(gdb) list 12,15
12        printf("Число: ");
13        scanf("%f",&Numeral);
14        printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): ");
15        scanf("%s",&Operation);
(gdb) list calculate.c:20,29
20            printf("Вычитаемое: ");
21            scanf("%f",&SecondNumeral);
22            return(Numeral - SecondNumeral);
23          }
24        else if(strncmp(Operation, "*", 1) == 0)
25          {
26            printf("Множитель: ");
27            scanf("%f",&SecondNumeral);
28            return(Numeral * SecondNumeral);
29          }
```

Рис. 2.10: Работа с отладкой

6.4. Установим точку останова в файле calculate.c на строке номер 21. Выведем информацию об имеющихся в проекте точка останова (рис. 2.11).

```
(gdb) list calculate.c:20,27
20            printf("Вычитаемое: ");
21            scanf("%f",&SecondNumeral);
22            return(Numeral - SecondNumeral);
23          }
24        else if(strncmp(Operation, "*", 1) == 0)
25          {
26            printf("Множитель: ");
27            scanf("%f",&SecondNumeral);
(gdb) break 21
Breakpoint 1 at 0x40121e: file calculate.c, line 21.
(gdb) info breakpoints
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x000000000040121e in Calculate at calculate.c:21
```

Рис. 2.11: Установка точки останова

6.5. Запустим программу внутри отладчика и убедимся, что программа остановится в момент прохождения точки останова (рис. 2.12). Отладчик выдаст следующую информацию:

```
#0 Calculate (Numeral=5, Operation=0x7fffffffd280 "-")
at calculate.c:21
#1 0x0000000000400b2b in main () at main.c:17
```

```
(gdb) run
Starting program: /home/dvmatyushkin/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=0x7fffffffdea4 "-") at calculate.c:21
21            scanf("%f",&SecondNumeral);
(gdb) backtrace
#0  Calculate (Numeral=5, Operation=0x7fffffffdea4 "-") at calculate.c:21
#1  0x00000000004014eb in main () at main.c:16
```

Рис. 2.12: Работа с точкой останова

6.6. Посмотрим, чему равно на этом этапе значение переменной Numeral. Сравним с результатом вывода на экран после использования команды. Уберем точки останова (рис. 2.13).

```
(gdb) print Numeral
$1 = 5
(gdb) display Numreal
No symbol "Numreal" in current context.
(gdb) display Numeral
1: Numeral = 5
(gdb) delete 1
(gdb) info breakpoints
No breakpoints or watchpoints.
```

Рис. 2.13: Работа с отладкой

7. С помощью утилиты splint проанализируем коды файлов calculate.c и main.c (рис. 2.14 и рис. 2.15). Здесь мы можем увидеть что утилита splint выводит информацию о коде программы. Например то, что возвращаемое значение функции scanf() игнорируется .

```
[dvmatyushkin@dvmatyushkin lab_prog]$ splint calculate.c
Splint 3.1.2 --- 23 Jul 2021

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:9:37: Function parameter Operation declared as manifest array (size
                    constant is meaningless)
calculate.c: (in function Calculate)
calculate.c:15:7: 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:21:7: Return value (type int) ignored: scanf("%f", &Sec...
```

Рис. 2.14: splint calculate.c

```
[dvmatyushkin@dvmatyushkin lab_prog]$ splint main.c
Splint 3.1.2 --- 23 Jul 2021

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:13:3: Return value (type int) ignored: scanf("%f", &Num...
```

Рис. 2.15: splint main.c

3 Контрольные вопросы

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

Воспользоваться интернетом, воспользоваться командой man, info

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

Unix поддерживает следующие основные этапы разработки приложений:

- создание исходного кода программы;
- охранение различных вариантов исходного текста;
- анализ исходного текста; Необходимо отслеживать изменения исходного кода, а также при работе более двух программистов над проектом программы нужно, чтобы они не делали изменений кода в одно время;
- компиляция исходного текста и построение исполняемого модуля;
- тестирование и отладка;
- сохранение всех изменений, выполняемых при тестировании и отладке.

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

Использование суффикса “.c” для имени файла с программой на языке Си отражает удобное и полезное соглашение, принятое в ОС UNIX. Для любого имени входного файла суффикс определяет какая компиляция требуется. Суффиксы и префиксы указывают тип объекта. Одно из полезных свойств компилятора Си — его способность по суффиксам определять типы файлов. По суффиксу .c компилятор распознает, что файл abcd.c должен компилироваться, а по суффиксу

.o, что файл abcd.o является объектным модулем и для получения исполняемой программы необходимо выполнить редактирование связей. Простейший пример командной строки для компиляции программы abcd.c и построения исполняемого модуля abcd имеет вид: gcc -o abcd abcd.c.

4. Каково основное назначение компилятора языка С в UNIX?

В компиляции всей программы в целом и получении исполняемого модуля.

5. Для чего предназначена утилита make?

Для упрощения и автоматизации работы пользователя с командной строкой

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

Текст, следующий за точкой с запятой, и все последующие строки, начинающиеся с литеры табуляции, являются командами ОС UNIX, которые необходимо выполнить для обновления целевого файла. Таким образом, спецификация взаимосвязей имеет формат:

```
target1 [ target2...]: [:] [dependment1...]
[(tab)commands]
[#commentary]
[(tab)commands]
[#commentary],
```

где # — специфицирует начало комментария; : — последовательность команд ОС UNIX должна содержаться в одной строке make-файла (файла описаний), есть возможность переноса команд (), но она считается как одна строка; :: — последовательность команд ОС UNIX может содержаться в нескольких последовательных строках файла описаний.

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

Все программы отладки позволяют отслеживать состояние программы на любом из этапов ее исполнения. Для того чтобы эту возможность использовать необ-

ходимо изучить документацию по использованию определенного отладчика. Понять общие принципы отладки.

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

- backtrace – выводит весь путь к текущей точке останова, то есть названия всех функций, начиная от main(); иными словами, выводит весь стек функций;
- break – устанавливает точку останова; параметром может быть номер строки или название функции;
- clear – удаляет все точки останова на текущем уровне стека (то есть в текущей функции);
- continue – продолжает выполнение программы от текущей точки до конца;
- delete – удаляет точку останова или контрольное выражение;
- display – добавляет выражение в список выражений, значения которых отображаются каждый раз при остановке программы;
- finish – выполняет программу до выхода из текущей функции; отображает возвращаемое значение,если такое имеется;
- info breakpoints – выводит список всех имеющихся точек останова;
- info watchpoints – выводит список всех имеющихся контрольных выражений;
- list – выводит исходный код; в качестве параметра передаются название файла исходного кода, затем, через двоеточие, номер начальной и конечной строки;
- next – пошаговое выполнение программы, но, в отличие от команды step, не выполняет пошагово вызываемые функции;

- print – выводит значение какого-либо выражения (выражение передаётся в качестве параметра);
- run – запускает программу на выполнение;
- set – устанавливает новое значение переменной;
- step – пошаговое выполнение программы;
- watch – устанавливает контрольное выражение, программа остановится, как только значение контрольного выражения изменится;

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

Сначала я запустил отладчик для моей программы. Установил интересующую меня точку остановки. Запустил программу ожидая, что программа остановится на точке остановки. Узнал необходимые данные моей программы на текущем этапе ее исполнения путем ввода команд. Отобразил данные. Завершил программу, снял точки остановки.

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

В моем случае не было синтаксических ошибок. Были ошибки семантические. Компилятор начал жаловаться на то, что программа по смыслу принимает указатель на char массив. В то время как я вводил не указатель, а прямое значение. Ошибка была исправлена.

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

- cscope - исследование функций, содержащихся в программе;
- lint - критическая проверка программ, написанных на языке Си.

12. Каковы основные задачи, решаемые программой splint?

Splint - инструмент для статической проверки C-программ на наличие уязвимостей и ошибок

4 Вывод

- В ходе этой лабораторной работы мы приобрели простейшие навыки разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linux на примере создания на языке программирования C калькулятора с простейшими функциями.