Лабораторная работа №13

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

Лебедева Ольга Андреевна

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

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

2 Теоретическое введение

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;
- сборка, компиляция и разработка исполняемого модуля;
- тестирование и отладка, сохранение произведённых изменений;
- документирование.

Для создания исходного текста программы разработчик может воспользоваться любым удобным для него редактором текста: vi, vim, mceditor, emacs, geany и др.

3 Ход работы

В домашнем каталоге создали подкаталог ~/work/os/lab_prog. Создали в нём файлы: calculate.h, calculate.c, main.c.(рис. 3.1)

```
oalebedeva@dk8n52 ~/work/os $ mkdir lab_prog
oalebedeva@dk8n52 ~/work/os $ cd lab_prog
oalebedeva@dk8n52 ~/work/os/lab_prog $ touch calculate.h
oalebedeva@dk8n52 ~/work/os/lab_prog $ touch calculate.c
oalebedeva@dk8n52 ~/work/os/lab_prog $ touch main.c
oalebedeva@dk8n52 ~/work/os/lab_prog $ ls
calculate.c  calculate.h  main.c
```

Рис. 3.1: Создание подкаталога

Записали код в три созданных файла. (рис. 3.2) (рис. 3.3) (рис. 3.4)

```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. 3.2: Содержание calculate.c

```
#ifndef CALCULATE_H_
#define CALCULATE_H_

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

#endif /*CALCULATE_H_*/
```

Рис. 3.3: Содержание main.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)
        {
```

Рис. 3.4: Содержание calculate.h

Выполнили компиляцию программы посредством gcc: (рис. 3.5)

```
gcc -c calculate.c
gcc -c main.c
gcc calculate.o main.o -o calcul -lm
```

Рис. 3.5: Компиляция

Проверили все файлы на наличие. (рис. 3.6)

```
oalebedeva@dk8n52 ~/work/os/lab_prog $ ls
calcul  calculate.c  calculate.h  calculate.o  main.c  main.o
```

Рис. 3.6: Все созданные файлы

Исправили синтаксические ошибки в созданном Makefile. (рис. 3.7)

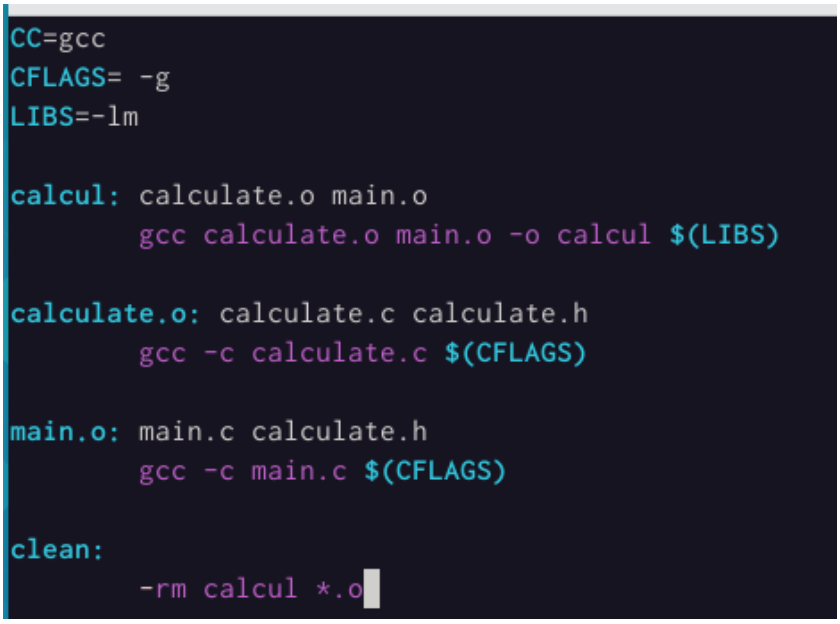

Рис. 3.7: Исправление кода

Запустили отладчик GDB, загрузив в него программу для отладки: (рис. 3.8)

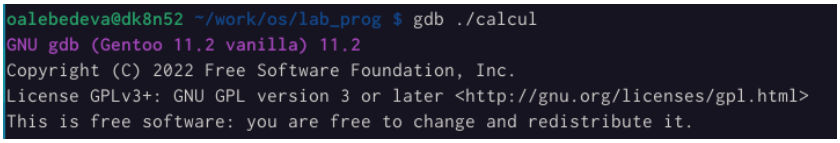

Рис. 3.8: Запуск GDB

Для запуска программы внутри отладчика ввели команду run: (рис. 3.9)

Рис. 3.9: Запуск программы

Для постраничного (по 9 строк) просмотра исходного код использовали команду list. (рис. 3.10)

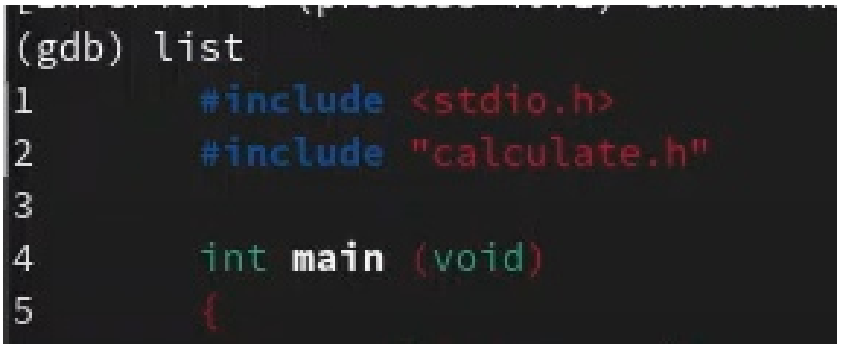

Рис. 3.10: Команда list

Для просмотра строк с 12 по 15 основного файла использовали list с параметрами: (рис. 3.11)

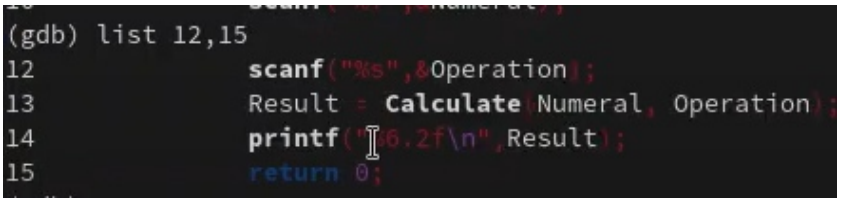

Рис. 3.11: Команда list с параметрами

Для просмотра определённых строк не основного файла использовали list с параметрами: (рис. 3.12)

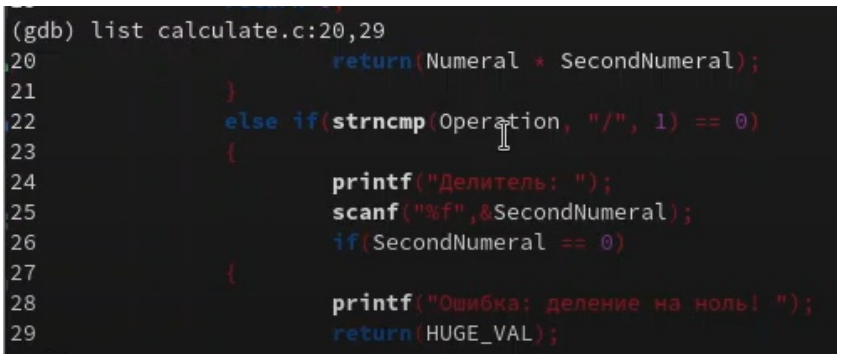

Рис. 3.12: Команда list с параметрами

Установили точку останова в файле calculate.c на строке номер 21: (рис. 3.13)

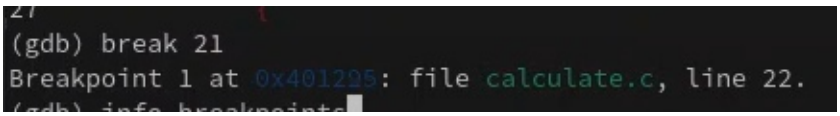

Рис. 3.13: Точка останова

Вывели информацию об имеющихся в проекте точка останова. Запустили программу. (рис. 3.14)

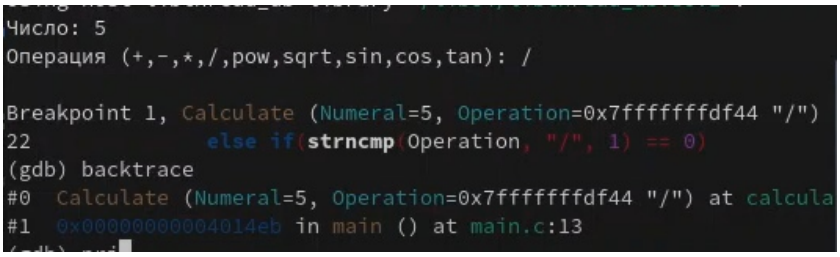

Рис. 3.14: Запуск программы

Посмотрели, чему равно на этом этапе значение переменной Numeral. (рис. 3.15)

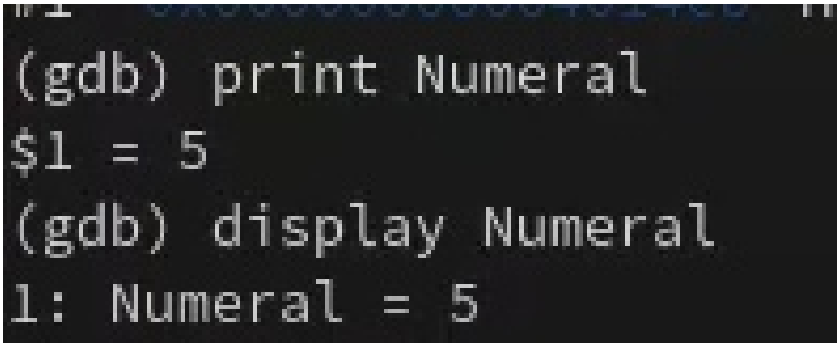

Рис. 3.15: Numeral

Удалили точку останова. (рис. 3.16)

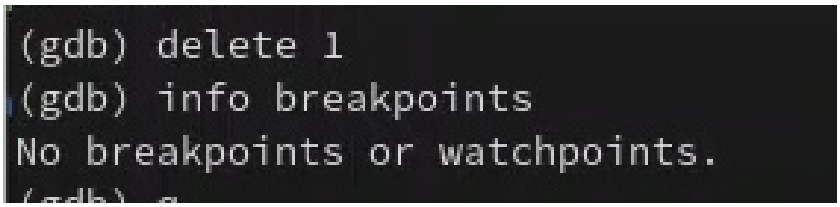

Рис. 3.16: Удаление точки останова

С помощью утилиты splint проанализировали коды файлов calculate.c и main.c. (рис. 3.17) (рис. 3.18)

```
oalebedeva@dk8n52 ~/work/os/lab_prog $ splint calculate.c
Splint 3.1.2 --- 13 Jan 2021

calculate.h:4:37: 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:6:31: Function parameter Operation declared as manifest array (size
                    constant is meaningless)
calculate.c: (in function Calculate)
calculate.c:12:3: 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:18:3: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:24:3: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:30:3: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:31:5: 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:34:9: Return value type double does not match declared type float:
                    (HUGE_VAL)
  To allow all numeric types to match, use +relaxtypes.
calculate.c:42:3: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:43:9: Return value type double does not match declared type float:
                    (pow(Numeral, SecondNumeral))
calculate.c:46:9: Return value type double does not match declared type float:
                    (sqrt(Numeral))
calculate.c:48:9: Return value type double does not match declared type float:
                    (sin(Numeral))
calculate.c:50:9: Return value type double does not match declared type float:
                    (cos(Numeral))
calculate.c:52:9: Return value type double does not match declared type float:
                    (tan(Numeral))
calculate.c:56:9: Return value type double does not match declared type float:
                    (HUGE_VAL)
```

Рис. 3.17: Анализ кода calculate.c

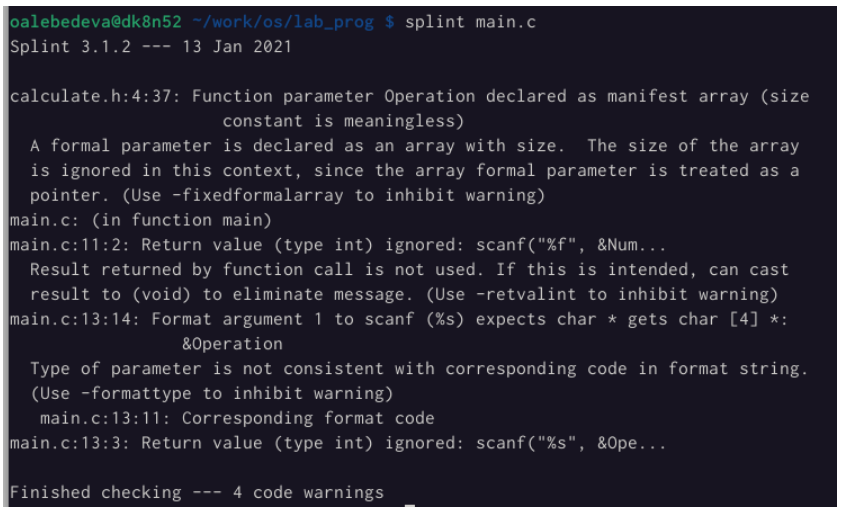

Рис. 3.18: Анализ кода main.c

4 Вывод

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

5 Ответы на вопросы

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

Дополнительную информацию о этих программах можно получить с помощью функций info и man.

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

Unix поддерживает следующие основные этапы разработки приложений:

- создание исходного кода программы;
- редставляется в виде файла;
- сохранение 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;
- проверка кода на наличие ошибок
- 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. Некоторые проекты предпочитают показывать префиксы в начале текста изменений для старых (old) и новых (new) файлов. Опция – prefix может быть использована для установки такого префикса. Плюс к этому команда bzr diff -p1 выводит префиксы в форме которая подходит для команды patch -p1.

4. Каково основное назначение компилятора языка С в UNIX?

Основное назначение компилятора с языка Си заключается в компиляции всей программы в целом и получении исполняемого модуля.

5. Для чего предназначена утилита make?

При разработке большой программы, состоящей из нескольких исходных файлов заголовков, приходится постоянно следить за файлами, которые требуют перекомпиляции после внесения изменений. Программа make освобождает пользователя от такой рутинной работы и служит для документирования взаимосвязей между файлами. Описание взаимосвязей и соответствующих действий хранится в так называемом make-файле, который по умолчанию имеет имя makefile или Makefile.

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

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

Пошаговая отладка программ заключается в том, что выполняется один оператор программы и, затем контролируются те переменные, на которые должен был воздействовать данный оператор. Если в программе имеются уже отлаженные подпрограммы, то подпрограмму можно рассматривать, как один оператор программы и воспользоваться вторым способом отладки программ. Если в программе существует достаточно большой участок программы, уже отлаженный ранее, то его можно выполнить, не контролируя переменные, на которые он воздействует. Использование точек останова позволяет пропускать уже отлаженную часть программы. Точка останова устанавливается в местах, где необходимо проверить содержимое переменных или просто проконтролировать, передаётся ли управление данному оператору. Практически во всех отладчиках поддерживается это свойство (а также выполнение программы до курсора и выход из подпрограммы). Затем отладка программы продолжается в пошаговом режиме с контролем локальных и глобальных переменных, а также внутренних регистров микроконтроллера и напряжений на выводах этой микросхемы. 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;
- splist – 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;

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

- break или b - создание точки останова;
- info или i - вывести информацию, доступные значения: break, registers, frame, locals, args;

- run или r - запустить программу;
- continue или c - продолжить выполнение программы после точки останова;

9. Опишите по шагам схему отладки программы, которую Вы использовали при выполнении лабораторной работы.
10. Прокомментируйте реакцию компилятора на синтаксические ошибки в программе при его первом запуске.

Отладчику не понравился формат %s для &Operation, т.к %s — символьный формат, а значит необходим только Operation.