Отчет по лабораторной работе №13

Операционные системы

Ничипорова Елена Дмитриевна

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 Выполнение лабораторной работы

1. Создаю новый подкаталог ~/work/os/lab_prog(рис. 2.1)

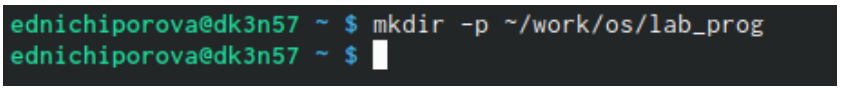

Рис. 2.1: Создание нового подкаталога

- Создала в каталоге файлы: calculate.h, calculate.c, main.c.(рис. 2.2)

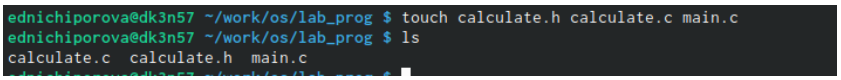

Рис. 2.2: Создание файлов

- 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. 2.3)(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);
    }
  else if(strncmp(Operation,"/", 1) == 0)
    {printf("Делитель: ");
      scanf("%f",&SecondNumeral);
      if(SecondNumeral== 0)
        {printf("Ошибка: деление на ноль! ");
          return(HUGE_VAL);
          }
      else
        return(Numeral/ SecondNumeral);
    }
  else if(strncmp(Operation,"pow", 3) == 0)
    {printf("Степень: ");
     scanf("%f",&SecondNumeral);
     return(pow(Numeral,SecondNumeral));
    }
  else if(strncmp(Operation,"sqrt", 4) == 0)
    return(sqrt(Numeral));
  else if(strncmp(Operation,"sin", 3) == 0)
    return(sin(Numeral));
```

Рис. 2.3: текст программы

```
    return(sqrt(Numeral));
  else if(strncmp(Operation,"sin", 3) == 0)
    return(sin(Numeral));
  else if(strncmp(Operation,"cos", 3) == 0)
    return(cos(Numeral));
  else if(strncmp(Operation,"tan", 3) == 0)
    return(tan(Numeral));
  else
    {
      printf("Неправильно введено действие ");
      return(HUGE_VAL);
    }
}
```

Рис. 2.4: текст программы

- Интерфейсный файлcalculate.h, описывающий формат вызова функции-калькулятора:(рис. 2.5)

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

#ifndef CALCULATE_H_
#define CALCULATE_H_

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

#endif /*CALCULATE_H_*/
```

Рис. 2.5: текст программы

- Основной файл main.c,реализующий интерфейс пользователя к калькулятору:(рис. 2.6)

```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. 2.6: текст программы

- выполнила компиляцию программы посредством gcc(рис. 2.7). Обнаружила предупреждение в файле main.c и исправила программу(рис. 2.8). Заново скомпилировала, после этого ошибок не возникло.(рис. 2.9)

```
ednichiporova@dk3n57 ~/work/os/lab_prog $ gcc -c -g calculate.c
[1]   Завершён        emacs
[2]-  Завершён        emacs
[3]+  Завершён        emacs
ednichiporova@dk3n57 ~/work/os/lab_prog $ gcc -c -g  main.c
main.c: В функции «main»:
main.c:16:12: предупреждение: формат «%s» ожидает аргумент типа «char *», но аргумент 2 имеет тип «char (*)[4]» [-Wformat=]
   16 |   scanf ("%s",&Operation);
      |           ~^   ~~~~~~~~~
      |            |   |
      |            |   char (*)[4]
      |            char *
ednichiporova@dk3n57 ~/work/os/lab_prog $ gcc -g calculate.o main.o -o calcul -lm
```

Рис. 2.7: компиляция программы

```
ednichiporova@dk3n57 ~/work/os/lab_prog $ gcc -c -g calculate.c
ednichiporova@dk3n57 ~/work/os/lab_prog $ gcc -c -g  main.c
ednichiporova@dk3n57 ~/work/os/lab_prog $ gcc -g calculate.o main.o -o calcul -lm
```

Рис. 2.8: исправленный текст программы

```
////////////////////////////////////////
// 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. 2.9: компиляция программы

- Создала Makefile с необходимым содержанием и исправила его(рис. 2.10) Данный файл необходим для автоматической компиляции файлов calculate.c (цель calculate.o), main.c (цель main.o), а также их объединения в один исполняемый файл calcul (цель calcul). Цель clean нужна для автоматического удаления файлов. Переменная CC отвечает за утилиту для компиляции. Переменная CFLAGS отвечает за опции в данной утилите. Переменная LIBS отвечает за опции для объединения объектных файлов в один исполняемый файл.

```
#
# Makefile
#
CC = gcc
CFLAGS = -g
LIBS = -lm
calcul: calculate.o main.o
	$(CC) calculate.o main.o  -o calcul $(LIBS)
calculate.o: calculate.c calculate.h
	$(CC) -c calculate.c $(CFLAGS)
main.o: main.c calculate.h
	$(CC) -c main.c $(CFLAGS)
clean:
	-rm calcul *.o *~
# End Makefile
```

Рис. 2.10: текст программы Makefile

- Удалила исполняемые и объектные файлы из каталога(рис. 2.11)

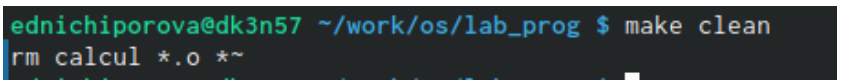

Рис. 2.11: удаление файлов

- Выполнила компиляцию файлов (рис. 2.12)

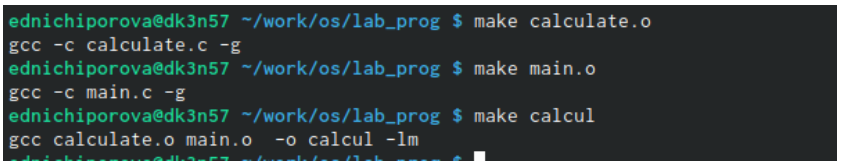

Рис. 2.12: компиляция файлов

- С помощью gdb выполнила отладку программы calcul. Запустила отладчик GDB(рис. 2.13)

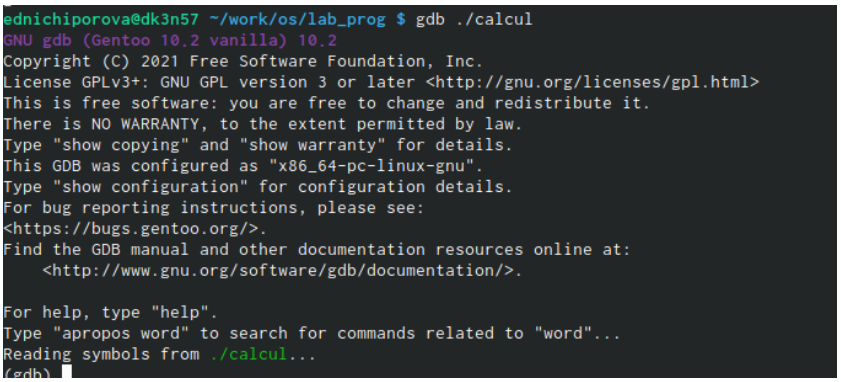

Рис. 2.13: Работа с gdb

- Запустила программу внутри отладчика(рис. 2.14)

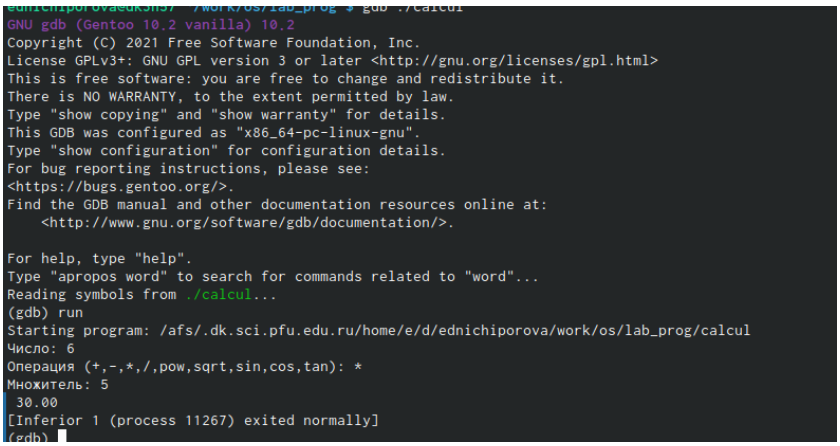

Рис. 2.14: run - запуск программы

- Постранично просмотрела исходный код с помощью команды “list”(рис. 2.15)

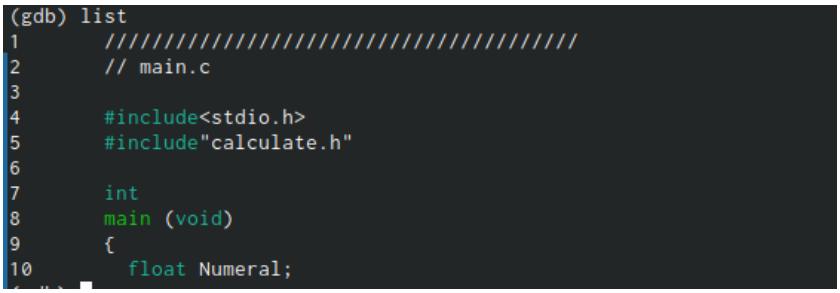

Рис. 2.15: list

- просмотрела строки 12-15 с помощью команды “list 12,15”(рис. 2.16)

Рис. 2.16: list 12,15

- Просмотрела определенные строки(рис. 2.17)

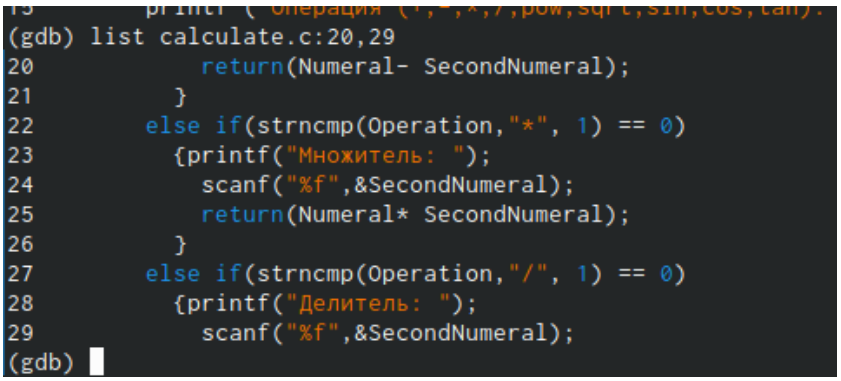

Рис. 2.17: list calculate:20,29

- Поставила точку останова в файле calculate.c на 21 строке(рис. 2.18)

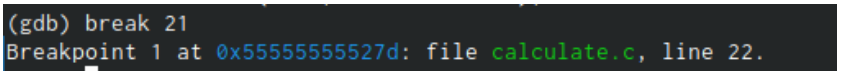

Рис. 2.18: точка останова

- Вывела информацию о имеющихся точках останова (рис. 2.19)

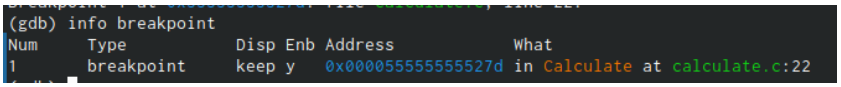

Рис. 2.19: info breakpoint

- Запустила программу внутри отладчика и убедилась, что программа остановилась в момент прохождения точки останова. Посмотрела, чему равно на этом этапе значениие переменной Numeral, сравнила с результатом вывода на экран после инспользования команды “display Numeral”. Значения совпадают(рис. 2.20)

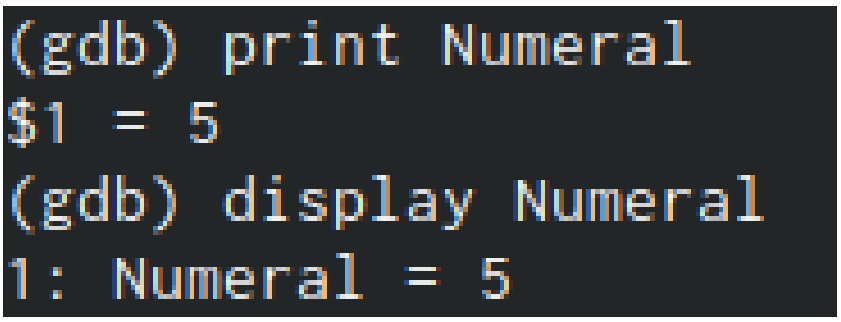

Рис. 2.20: проверка работы программы

- Убрала точки останова(рис. 2.21)

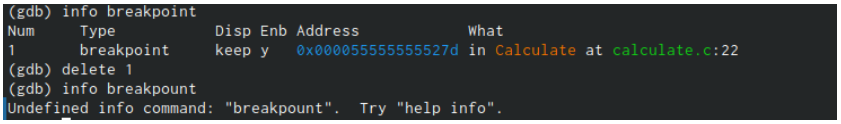

Рис. 2.21: delet 1

- С помощью утилиты splint проанализируем коды файлов calculate.c и main.c. Воспользуемся командами splint calculate.c и splint main.c (рис. **??**) (рис. **??**). С помощью утилиты splint выяснилось, что в файлах calculate.c и main.c присутствует функция чтения scanf, возвращающая целое число (тип int), но эти числа не используются и нигде не сохранятся. Утилита вывела предупреждение о том, что в файле calculate.c происходит сравнение вещественного числа с нулем. Также возвращаемые значения (тип double) в функциях pow, sqrt, sin, cos и tan записываются в переменную типа float, что свидетельствует о потери данных.

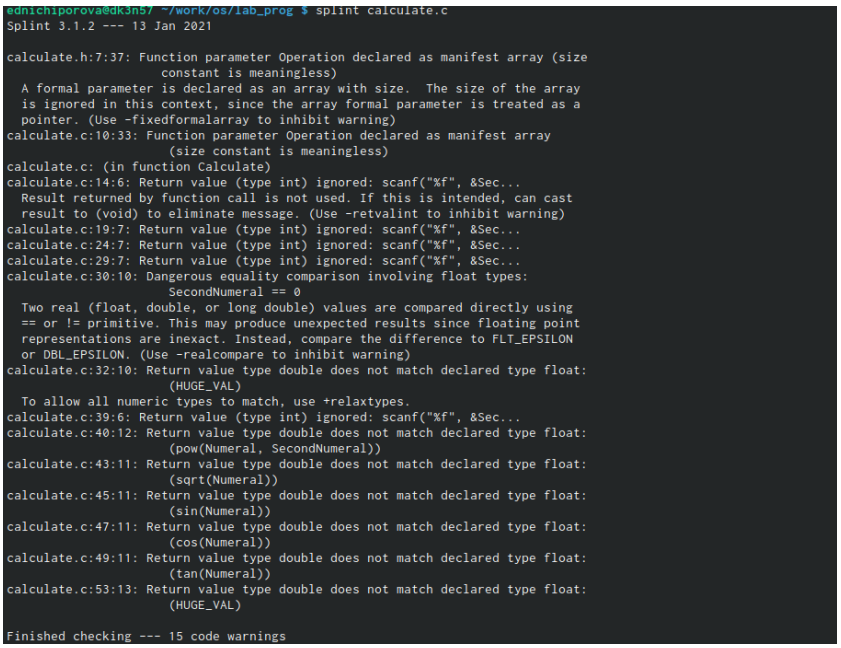

Рис. 2.22: результат работы splint calculate.c

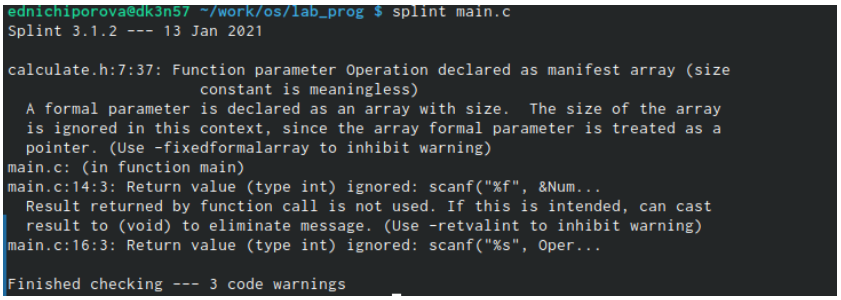

Рис. 2.23: splint main.c

2. Контрольные вопросы

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

- Процесс разработки программного обеспечения обычно разделяется на следующие

  – 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 с расширением .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: «gcc -o hello main.c».

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

- 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.

- Для работы с утилитой make необходимо в корне рабочего каталога с Вашим проектом создать файл с названием makefile или Makefile, в котором будут описаны правила обработки файлов Вашего 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 водной 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.

- 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 GNU для 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

- Основные команды отладчика gdb: backtrace – вывод на экран пути к текущей точке останова (по сути вывод – названий всех функций) break – установить точку останова (в качестве параметра может быть указан номер строки или название функции) clear – удалить все точки останова в функции continue – продолжить выполнение программы delete – удалить точку останова display – добавить выражение в список выражений, значения которых отображаются при достижении точки останова программы finish – выполнить программу до момента выхода из функции info breakpoints – вывести на экран список используемых точек останова info watchpoints – вывести на экран список используемых контрольных выражений list – вывести на экран исходный код (в качестве параметра может быть указано название файла и через двоеточие номера начальной и конечной строк) next – выпол-

нить программу пошагово, но без выполнения вызываемых в программе функций print – вывести значение указываемого в качестве параметра выражения run – запуск программы на выполнение set – установить новое значение переменной step – пошаговое выполнение программы 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.

- Схема отладки программы показана в 6 пункте лабораторной работы.
- 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.
- 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.
- 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 С анализатор splint генерирует 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 работепрограммы, переменные с некорректно заданными значениями и типами и многое другое.

3 Выводы

Приобрела простейшие навыки разработки,анализа,тестирования и отладки при-ложений в ОС типа UNIX/Linux на примере создания на языке программированияС калькулятора с простейшими функциями.

Список литературы