РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 14

дисциплина: Операционные системы

Студент:

Афтаева Ксения Васильевна

Преподаватель:

Велиева Т.В.

Группа: НПИбд-01-20

МОСКВА 2021 г.

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**.

Задачи:

1. Изучить теоретический материал
2. Выполнить пункты, указанные в описании к лабораторной работе (реализовать калькулятор и тд.)
3. Ответить на контрольные вопросы.

Объект и предмет исследования:

Программирование в оболочке **ОС UNIX/Linux**

Техническое оснащение:

Ноутбук, на котором установлена виртуальная машина с Linux

Теоретические вводные данные: [1]

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 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*.

После завершения написания исходного кода программы (возможно состоящей из нескольких файлов), необходимо её скомпилировать и получить исполняемый модуль.

Компиляция исходного текста и построение исполняемого файла

Стандартным средством для компиляции программ в ОС типа UNIX является **GCC** (*GNU Compiler Collection*). Это набор компиляторов для разного рода языков программирования (C, C++, Java, Фортран и др.). Работа с GCC производится при помощи одноимённой управляющей программы gcc, которая интерпретирует аргументы командной строки, определяет и осуществляет запуск нужного компилятора для входного файла. 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.

Условные обозначения и символы:

Опции при компиляции:

- **-c** компиляция без компоновки — создаются объектные файлы file.o
- **-o** file-name задать имя file-name создаваемому файлу
- **-g** поместить в файл (объектный или исполняемый) отладочную информацию для отладчика gdb
- **-MM** вывести зависимости от заголовочных файлов C и/или C++ программ в формате, подходящем для утилиты make; при этом объектные или исполняемые файлы не будет созданы
- **-Wall** вывод на экран сообщений об ошибках, возникших во время компиляции

Выполнение работы:

1. Создала подкаталог **lab_prog** в каталоге **~/work/os** с помощью команды `mkdir`. Затем перешла в него, введя команду `cd` + полный путь к каталогу (Рис.1)

```
[kvaftaeva@kvaftaeva ~]$ cd ~/work/os/lab_prog
[kvaftaeva@kvaftaeva lab_prog]$
```

Рис.1 Переход в каталог для работы

2. Создала в нём файлы **calculate.h**, **calculate.c**, **main.c** с помощью команды `emacs` (Текстовый редактор, в котором я буду работать). (Рис.2)

```
[kvaftaeva@kvaftaeva lab_prog]$ emacs calculate.h
[kvaftaeva@kvaftaeva lab_prog]$ emacs calculate.c
[kvaftaeva@kvaftaeva lab_prog]$ emacs main.c
```

Рис.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.3-5)

```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);
    }
```

Рис.3 calculate.c ч1

```
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));
```

Рис.4 calculate.c ч2

```
  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.5 calculate.c ч3

Интерфейсный файл **calculate.h**, описывающий формат вызова функции калькулятора (Рис.6)

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

#ifndef CALCULATE_H_
#define CALCULATE_H_

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

#endif /*CALCULATE_H_*/
```

Рис.6 calculate.h

Основной файл **main.c**, реализующий интерфейс пользователя к калькулятору (Рис.7)

```
////////////////////////////////////
// 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.7 main.c

Данные программы были взяты из описания к лабораторной работе. После написания файлов я сохранила их комбинацией клавиш `C-x C-s` и закрыла.

3. Выполнила компиляцию программы посредством gcc, добавив опцию **-g**, которая нужна для корректной работы с **gdb** (Рис.8)

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

Рис.8 Компиляция программы

4. Синтаксических ошибок не было выявлено

5. Создала **Makefile** с помощью команды `emacs` (Рис.9)

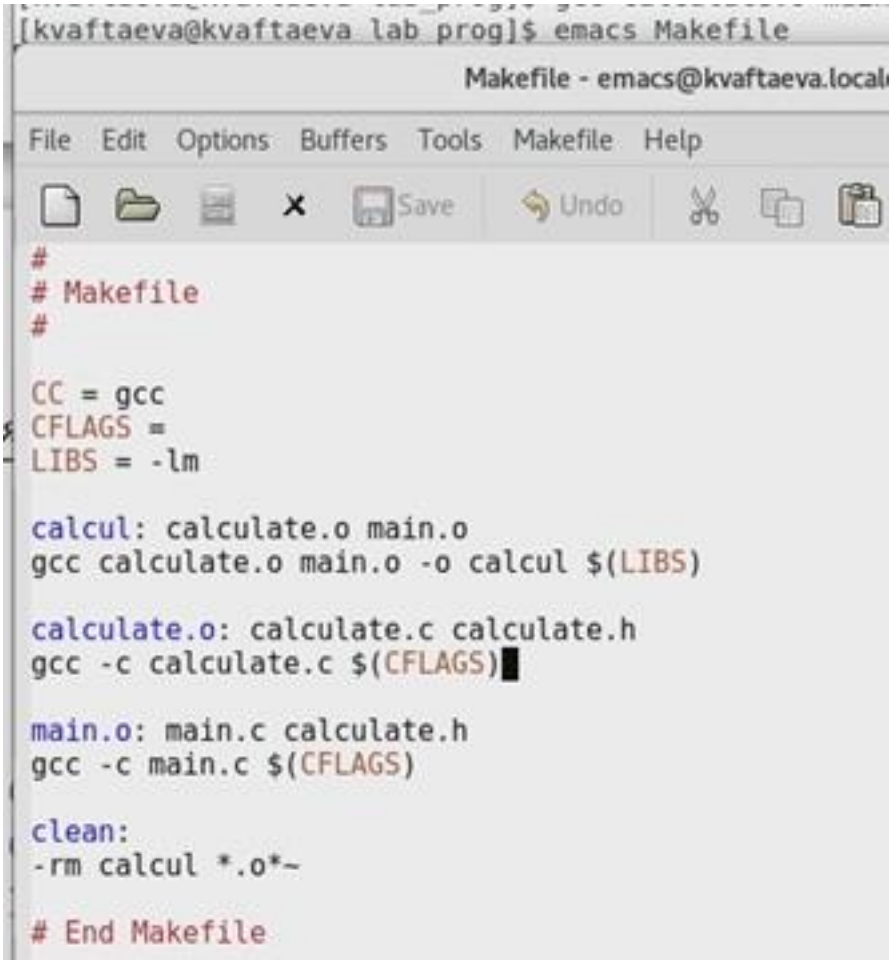

Рис.9 Makefile

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.

6. С помощью **gdb** выполнила отладку программы *calcul*:

- Запустила отладчик **GDB**, загрузив в него программу для отладки `gdb ./calcul` (Рис.10)

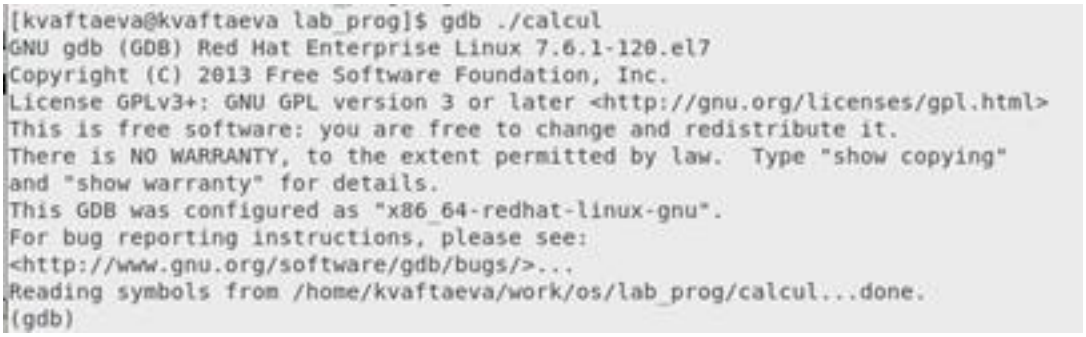

Рис.10 запуск отладчика

Видим, что он был запущен и теперь мы работаем внутри него

- Для запуска программы внутри отладчика ввела команду `run` (Рис.11)

```
(gdb) run
Starting program: /home/kvaftaeva/work/os/lab_prog/./calcul
Число: 1
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): +
Второе слагаемое: 1
  2.00
[Inferior 1 (process 3124) exited normally]
Missing separate debuginfos, use: debuginfo-install glibc-2.17-317.el7.x86_64
```

Рис.11 запуск программы внутри отладчика

Видим, что программа запрашивает первое число, операцию, второе число и успешно выводит результат

- Для постраничного (по 9 строк) просмотра исходного кода использовала команду `list` (Рис 12)
- Для просмотра строк с 12 по 15 основного файла использовала `list 12,15` (Рис.12)
- Для просмотра определённых строк не основного файла использовала `list calculate.c:20,29` (Рис.12)

```
(gdb) list
4       #include <stdio.h>
5       #include "calculate.h"
6
7       int
8       main (void)
9       {
10        float Numeral;
11        char Operation[4];
12        float Result;
13        printf("Число: ");
(gdb) list 12,15
12        float Result;
13        printf("Число: ");
14        scanf("%f", &Numeral);
15        printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): ");
(gdb) list calculate.c:20,29
20          {
21            printf ("Вычитаемое: ");
22            scanf ("%f", &SecondNumeral);
23            return (Numeral - SecondNumeral);
24          }
25        else if (strncmp(Operation, "*", 1) == 0)
26          {
27            printf("Множитель: ");
28            scanf("%f",&SecondNumeral);
29            return (Numeral * SecondNumeral);
```

Рис.12 Просмотр строк

Видим, что выводятся нужные нам сроки нужных файлов

- Установила точку останова в файле *calculate.c* на строке номер 21 `break 21` (Рис.13)
- Вывела информацию об имеющихся в проекте точка останова `info breakpoints` (Рис.13)

```
(gdb) break 21
Breakpoint 1 at 0x4007d8: file calculate.c, line 21.
(gdb) info breakpoints
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x00000000004007d8 in Calculate at calculate.c:21
(gdb)
```

Рис 13 Точки останова

Видим, что точки останова были успешно установлены

- Запустила программу внутри отладчика и убедилась, что программа остановится в момент прохождения точки останова, введя построчноrun `5 - backtrace` (Рис.14)

```
(gdb) run
Starting program: /home/kvaftaeva/work/os/lab_prog/./calcul
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): -

Breakpoint 1, Calculate (Numeral=5, Operation=0x7fffffffdee0 "-") at calculate.c:21
21              printf ("Вычитаемое: ");
(gdb) backtrace
#0  Calculate (Numeral=5, Operation=0x7fffffffdee0 "-") at calculate.c:21
#1  0x0000000000400a90 in main () at main.c:17
```

Рис.14 Запуск программы с точкой останова

Отладчик выдает следующую информацию:

"#0 Calculate (Numeral=5, Operation=0x7fffffffd280 "-")"

"at calculate.c:21"

"#1 0x0000000000400b2b in main () at main.c:17"

Команда backtrace показал весь стек вызываемых функций от начала программы до текущего места.

- Посмотрела, чему равно на этом этапе значение переменной Numeral, введя `print Numeral` (Рис. 15)
- Сравнилас результатом вывода на экран после использования команды `display Numeral` (Рис.15)

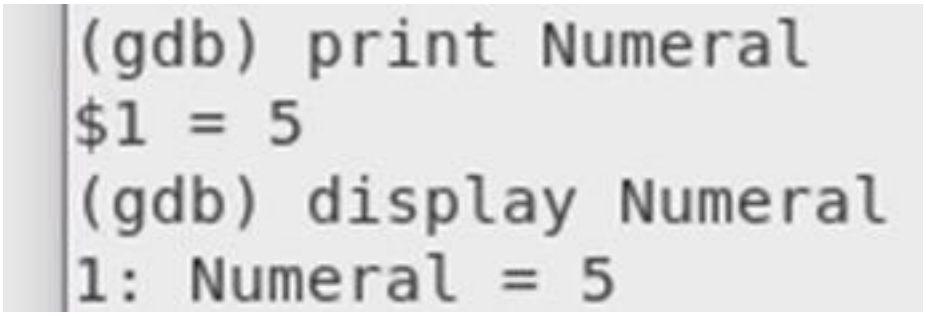

Рис.15 Вывод значения переменной

Оба раза выведено число 5

- Убрала точки останова `delete 1` (Рис.16)

Рис.16 Удаление точки останова

Вывела информацию о точках останова до и после удаления. Видим, что точка была удалена

7. С помощью утилиты **splint** проанализировала коды файлов *calculate.c и main.c.* (Рис.17-18)

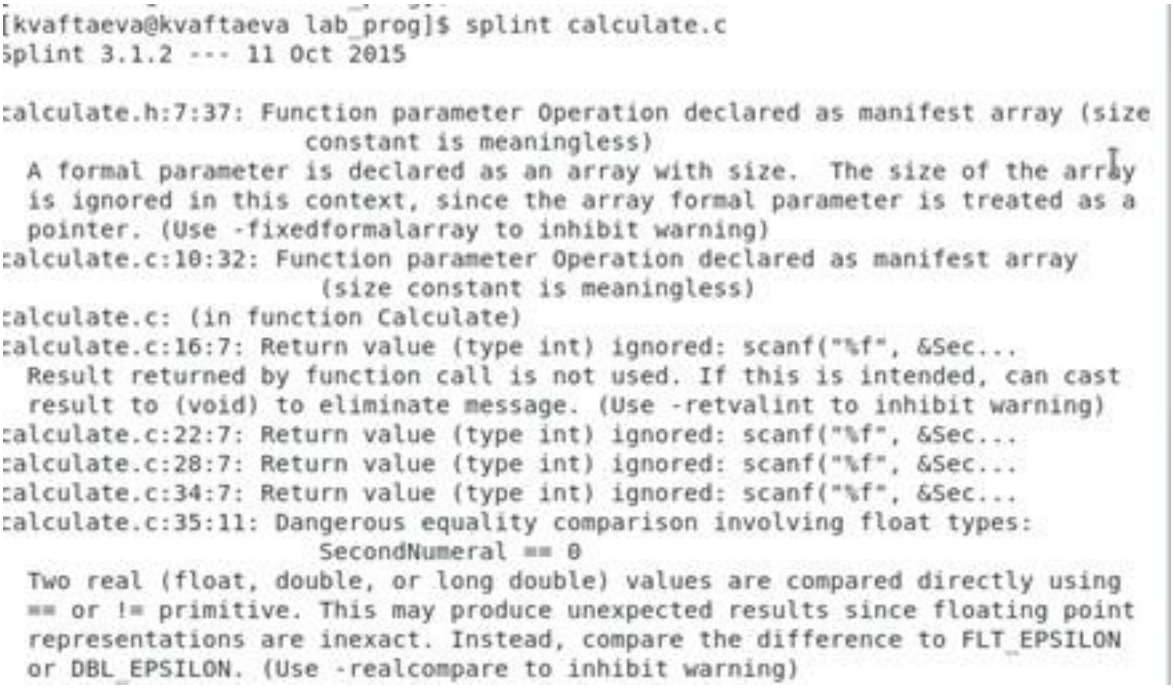

Рис.17 Анализ с помощью утилиты splint 1

```
[kvaftaeva@kvaftaeva lab_prog]$ splint main.c
Splint 3.1.2 --- 11 Oct 2015

calculate.h:7:37: Function parameter Operation declared as manifest array (siz
                    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:14:3: Return value (type int) ignored: scanf("%f", &Num...
  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)
main.c:16:14: Format argument 1 to scanf (%s) expects char * gets char [4] *:
                &Operation
  Type of parameter is not consistent with corresponding code in format string
  (Use -formattype to inhibit warning)
   main.c:16:11: Corresponding format code
main.c:16:3: Return value (type int) ignored: scanf("%s", &Ope...

Finished checking --- 4 code warnings
```

Рис.18 Анализ с помощью утилиты splint 2

Здесь мы можем увидеть что утилита splint выводит информацию о коде программы. Например то, что возвращаемое значение функции scanf() игнорируется

Контрольные вопросы:

1. С помощью функций info и man.

   – создание исходного кода программы, которая представляется 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. [1]

3. 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. [1]

4. Компиляция всей программы в целом и получении исполняемого модуля.

5. Make-файл содержит последовательность записей (строк), определяющих зависимости между файлами. Первая строка записи представляет собой список целевых (зависимых) файлов, разделенных пробелами, за которыми следует двоеточие и список файлов, от которых зависят целевые. [1]

6. 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, так как содержимое строки, начиная с # и до конца строки, не будет обрабатываться командой make; : — 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. 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.

  - 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; [1]

1) Выполнили компиляцию программы

2) Увидели ошибки в программе

3) Открыли редактор и исправили программу

4) Загрузили программу в отладчик gdb

5) run — отладчик выполнил программу, мы ввели требуемые значения.

6) программа завершена, gdb не видит ошибок.

10. Не возникло

  - cscope - исследование функций, содержащихся в программе;

– splint — критическая проверка программ, написанных на языке Си.

1) Проверка корректности задания аргументов всех использованных в программе функций, а также типов возвращаемых ими значений;
2) Поиск фрагментов исходного текста, корректных с точки зрения синтаксиса языка Си, но малоэффективных с точки зрения их реализации или содержащих в себе семантические ошибки;
3) Общая оценка мобильности пользовательской программы. [2]

Заключение:

Теоретический материал изучен и пригодится в дальнейшей работе. Все цели и задачи выполнены.

Вывод:

Я 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**.

Библиографический список:

[1]: Описание к лабораторной работе №14

[2]: Splint - программа проверки АС