Отчёт по лабораторной работе №13

Дисциплина: Операционные системы

Татьяна Александровна Лебединец

Содержание

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.

Задание

1. В домашнем каталоге создайте подкаталог ~/work/os/lab_prog.
2. Создайте в нём файлы: calculate.h, calculate.c, main.c. 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.
3. Выполните компиляцию программы посредством gcc: gcc -c calculate.c gcc -c main.c gcc calculate.o main.o -o calcul -lm
4. При необходимости исправьте синтаксические ошибки.
5. Создайте Makefile. Поясните в отчёте его содержание.
6. С помощью gdb выполните отладку программы calcul (перед использованием gdb исправьте Makefile):

- Запустите отладчик GDB, загрузив в него программу для отладки
- Для запуска программы внутри отладчика введите команду run
- Для постраничного (по 10 строк) просмотра исходного код используйте команду list
- Для просмотра строк с 12 по 15 основного файла используйте list с параметрами
- Для просмотра определённых строк не основного файла используйте list с параметрами
- Установите точку останова в файле calculate.c на строке номер 21
- Выведите информацию об имеющихся в проекте точка останова

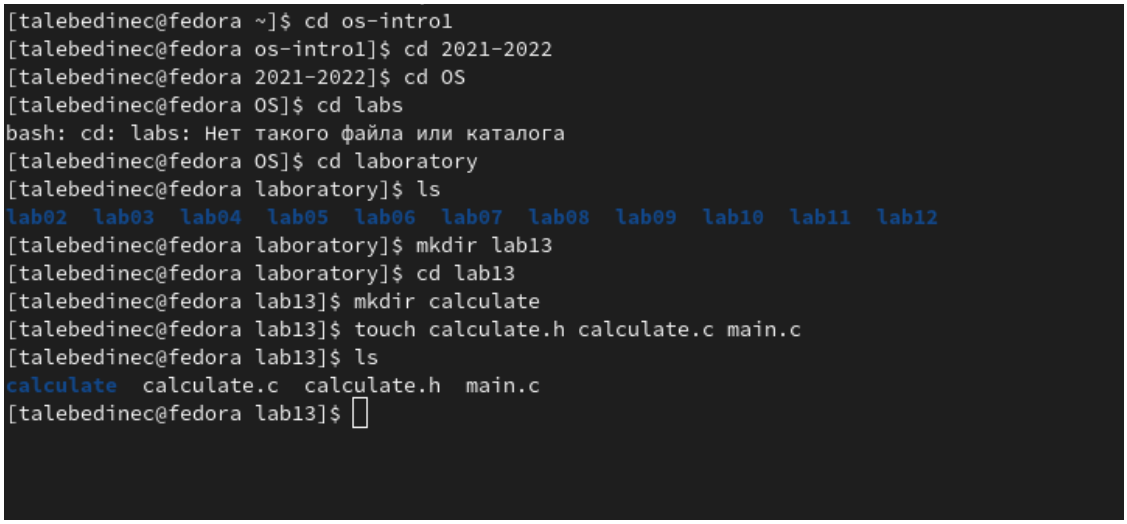
- Запустите программу внутри отладчика и убедитесь, что программа остановится в момент прохождения точки останова
- Отладчик выдаст информацию, а команда backtrace покажет весь стек вызываемых функций от начала программы до текущего места
- Посмотрите, чему равно на этом этапе значение переменной Numeral. На экран должно быть выведено число 5
- Сравните с результатом вывода на экран после использования команды
- Уберите точки останова

7. С помощью утилиты splint попробуйте проанализировать коды файлов calculate.c и main.c.

Выполнение лабораторной работы

1 В домашнем каталоге создаю подкаталог calculate с помощью команды «mkdir calculate».

2 Создаю в каталоге файлы: calculate.h, calculate.c, main.c, используя команды «cd calculate» и «touch calculate.h calculate.c main.c» (рис. -@fig:001).



{#fig:001 width=70%}

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. Открыв редактор Emacs, приступил к редактированию созданных файлов. (рис. -@fig:004) (рис. -@fig:003). Интерфейсный файл calculate.h (рис. -@fig:002).

```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!");
```

{#fig:003 width=70%}

```
          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));
  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);
    }
}
```

{#fig:004 width=70%}

```
#ifndef CALCULATE_H_
#define CALCULATE_H_

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

#endif
```

{#fig:002 width=70%}

Основной файл main.c, реализующий интерфейс пользователя к калькулятору (рис. -@fig:005).

```
#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;
}
```

{#fig:005 width=70%}

Выполнила компиляцию программы посредством gcc, используя команды «gcc -c calculate.c», «gcc -c main.c» и «gcc calculate.o main.o -o calcul -lm» (рис. -@fig:007).

```
[talebedinec@fedora lab13]$ emacs &
[3] 18678
[2]   Завершён        emacs
[talebedinec@fedora lab13]$ -c calculate.c
bash: -c: command not found...
[talebedinec@fedora lab13]$ gcc -c calculate.c
[talebedinec@fedora lab13]$ gcc -c main.c
main.c: В функции «main»:
main.c:14:12: предупреждение: неявная декларация функции «Calculate»; имел
 виду «calculate»? [-Wimplicit-function-declaration]
   14 |    Result = Calculate(Numeral,Operation);
      |             ^~~~~~~~~
      |             calculate
[talebedinec@fedora lab13]$ gcc calculate.o main.o -o calcul -lm
[talebedinec@fedora lab13]$ touch Makefile
[talebedinec@fedora lab13]$
```

{#fig:007 width=70%} { #fig:006 }

4 В ходе компиляции программы никаких ошибок выявлено не было.

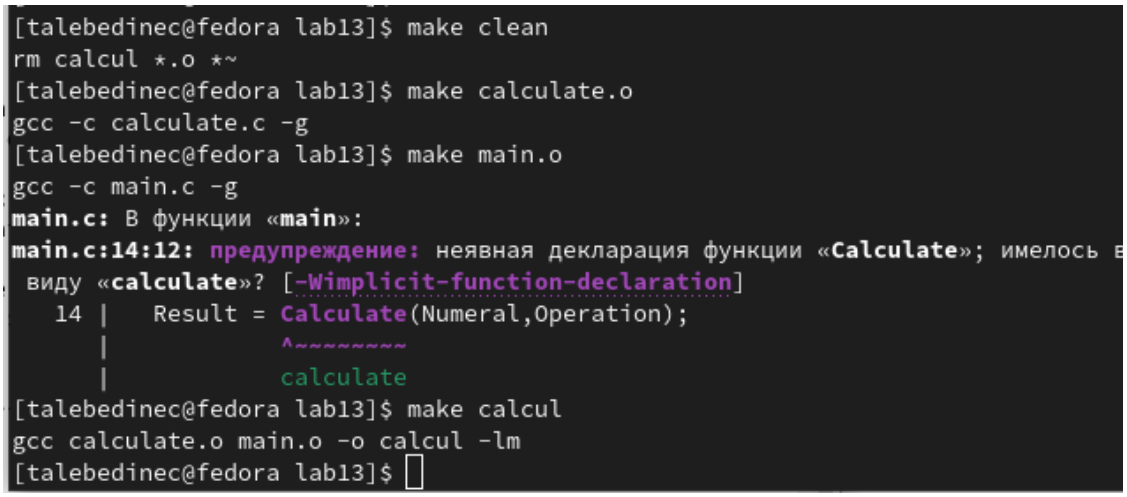
5 Создала Makefile с необходимым содержанием (рис. -@fig:006). Данный файл необходим для автоматической компиляции файлов calculate.c (цель calculate.o), main.c (цель main.o), а также их объединения в один исполняемый файл calcul (цель calcul). Цель clean нужна для автоматического удаления файлов. Переменная 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.

```
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 *~
```

{#fig:006 width=70%}

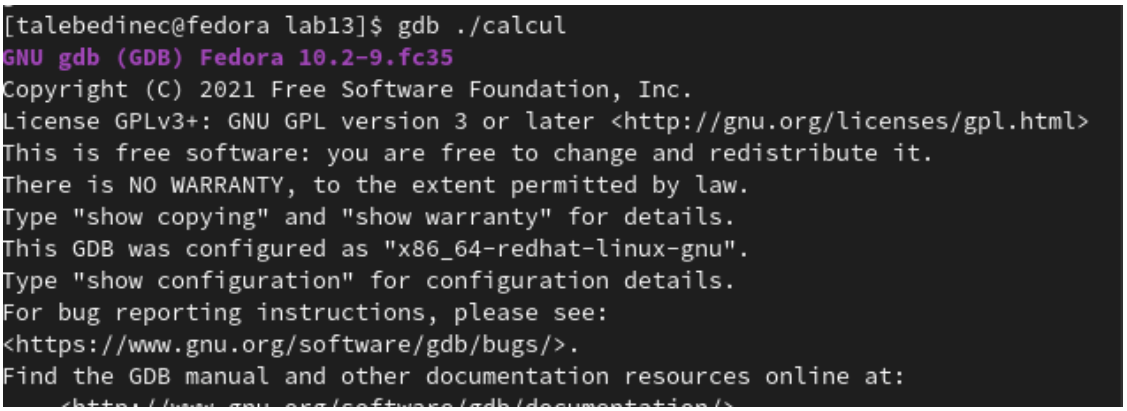
6 Далее исправила Makefile (рис. -@fig:008). В переменную CFLAGS добавила опцию -g, необходимую для компиляции объектных файлов и их использования в программе отладчика GDB. Сделала так, что утилита компиляции выбирается с помощью переменной CC.

После этого я удалила исполняемые и объектные файлы из каталога с помощью команды «make clean». Выполнила компиляцию файлов, используя команды «make calculate.o», «make main.o», «male calcul» (рис. -@fig:008).



{#fig:008 width=70%}

Далее с помощью gdb выполнила отладку программы calcul. Запустила отладчик GDB, загрузив в него программу для отладки, используя команду: «gdb ./calcul» (рис. -@fig:009).



{#fig:009 width=70%}

Для запуска программы внутри отладчика ввела команду «run» (рис. -@fig:010).

{#fig:010 width=70%}

Для постраничного (по 10 строк) просмотра исходного кода использовала команду «list».

Для просмотра строк с 12 по 15 основного файла использовала команду «list 12,15».

Для просмотра определённых строк не основного файла использовала команду «list calculate.c:20,29».

Установила точку останова в файле calculate.c на строке номер 18, используя команды «list calculate.c:15,22» и «break 18».

Вывела информацию об имеющихся в проекте точках останова с помощью команды «info breakpoints».

Запустила программу внутри отладчика и убедилась, что программа остановилась в момент прохождения точки останова. Использовал команды «run», «5», «–» и «backtrace».

Посмотрела, чему равно на этом этапе значение переменной Numeral, введя команду «print Numeral».

Сравнила с результатом вывода на экран после использования команды «display Numeral». Значения совпадают.

Убрала точку останова с помощью команд «info breakpoints» и «delete 3».

С помощью утилиты splint проанализировала коды файлов calculate.c и main.c. Воспользовалась командами «splint calculate.c» и «splint main.c» (рис. -@fig:011).
![Рис 11 - splint](image/11.png) {#fig:011 width=70%}
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.

Проанализировала код файла calculate.c. Проанализировала код файла main.c.

#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 этапы:

планирование, включающее сбор и анализ требований кфункционалу 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.

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:

<цель_1> <цель_2> ... : <зависимость_1> <зависимость_2> ...
<команда 1>
...
<команда n>
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 – выполнить 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.

Выводы

В ходе выполнения данной лабораторной работы я приобрёл простейшие навыки разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linux на примере создания на языке программирования С калькулятора с простейшими функциями.

Список литературы