

Отчёт по лабораторной работе №13

Дисциплина: Операционные системы

Дупленских Василий Викторович

Содержание

Цель работы:	1
Выполнение лабораторной работы:	2
1. Создаю новый подкаталог calculate в home и в нём уже создаю 3 файла:.....	2
2.1. Реализую калькулятор в файле calculate.c:.....	2
2.2. Описываю формат вызова функции калькулятора в файле calculate.h:.....	3
2.3. Реализую интерфейс пользователя к калькулятору в файле main.c:	3
3. Выполняю компиляцию программы посредством gcc:	3
4. Синтаксических ошибок нет, поэтому исправлять я ничего не собираюсь!	4
5. Создаю Makefile для компиляции в gdb:	4
6.1. Используя Makefile заново все компилирую:	4
6.2. Произвожу отладку через gdb:	5
7. Анализирую коды при помощи утилиты splint:	6
Ответы на вопросы:.....	6
Вывод:.....	9

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.

Выполнение лабораторной работы:

1. Создаю новый подкаталог calculate в home и в нём уже создаю 3 файла:

```
[vvduplenskikh@vvduplenskikh ~]$ mkdir calculate
[vvduplenskikh@vvduplenskikh ~]$ ls
28      '#new3#'      random.sh~
backup  '#new4#'      semafor.sh
backup.sh prog11.1.sh   semafor.sh~
backup.sh~ prog11.1.sh~   text
bin      prog11.2      vvduplenskikh.tar
calculate prog11.2.c    work
cplus    prog11.2.c~   Видео
format.sh prog11.2.sh   Документы
format.sh~ prog11.2.sh~  Загрузки
lab07.sh prog11.3.sh   Изображения
lab07.sh~ prog11.3.sh~  Музыка
ls.sh     prog11.4.sh   Общеизвестные
ls.sh~    prog11.4.sh~  'Рабочий стол'
man.sh    prog2.sh      'Снимок экрана от 2022-04-23 20-47-49.png'
man.sh~   prog2.sh~     Шаблоны
'#new2#'  random.sh

[vvduplenskikh@vvduplenskikh ~]$ cd calculate
bash: cd: calculate: Нет такого файла или каталога
[vvduplenskikh@vvduplenskikh ~]$ cd calculate
[vvduplenskikh@vvduplenskikh calculate]$ touch calculate.h calculate.c main.c
[vvduplenskikh@vvduplenskikh calculate]$ ls
calculate.c calculate.h main.c
[vvduplenskikh@vvduplenskikh calculate]$
```

Создание каталога и текстовых файлов

2.1. Реализую калькулятор в файле calculate.c:

```
File Edit Options Buffers Tools C Help
// 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("Вчитаемое: ");
        scanf("%f", &SecondNumeral);
        return(Numeral - SecondNumeral);
    }
    else if(strncmp(Operation, "*", 1) == 0)
    {
        printf("Умножитель: ");
        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));
    }
}

U: --- calculate.c Top LI (C/1 Abbrev)
Welcome to GNU Emacs, one component of the GNU/Linux operating system.
Bnace Tutorial Learn basic keystroke commands (Учебник Emacs)
```

calculate.c

2.2. Описываю формат вызова функции калькулятора в файле calculate.h:

```
calculate.h
~/calculate

1 //////////////////////////////////////////////////
2 // calculate.h
3
4 #ifndef CALCULATE_H_
5 #define CALCULATE_H_
6 float Calculate(float Numeral, char Operation[4]);
7
8 #endif /*CALCULATE_H_*/
```

calculate.h

2.3. Реализую интерфейс пользователя к калькулятору в файле main.c:

```
1 //////////////////////////////////////////////////
2 // main.c
3 #include <stdio.h>
4 #include "calculate.h"
5
6 int
7 main (void)
8 {
9     float Numeral;
10    char Operation[4];
11    float Result;
12    printf("Число: ");
13    scanf("%f",&Numeral);
14    printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): ");
15    scanf("%s",&Operation);
16    Result = Calculate(Numeral, Operation);
17    printf("%6.2f\n",Result);
18    return 0;
19 }
20
21
```

main.c

3. Выполняю компиляцию программы посредством gcc:

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

gcc

4. Синтаксических ошибок нет, поэтому исправлять я ничего не собираюсь!

5. Создаю Makefile для компиляции в gdb:

```
#
# Makefile
#

CC = gcc
CFLAGS = -g
LIBS = -lm

calcul: calculate.o main.o
    $(CC) gcc calculate.o main.o -o calcul $(LIBS)

calculate.o: calculate.c calculate.h
    $(CC) gcc -c calculate.c $(CFLAGS)

main.o: main.c calculate.h
    $(CC) gcc -c main.c $(CFLAGS)

clean:
    -rm calcul *.o *~

# End Makefile
```

```
;sqrt,sin,cos,tan): ");
Operation);
```

Make gdb

6.1. Используя Makefile заново все компилирую:

```
[vvduplenskikh@vvduplenskikh calculate]$ make clean
Makefile:6: *** пропущен разделитель (возможно нужен TAB вместо восьми пробелов?)
). Останов.
[vvduplenskikh@vvduplenskikh calculate]$ make calculate.o
make: «calculate.o» не требует обновления.
[vvduplenskikh@vvduplenskikh calculate]$ make clean
rm calcul *.o *~
[vvduplenskikh@vvduplenskikh calculate]$ make calculate.o
gcc gcc -c calculate.c -g
gcc: предупреждение: gcc: входные файлы компоновки не использованы, поскольку ко
мпоновка не выполнялась
gcc: ошибка: gcc: linker input file not found: Нет такого файла или каталога
make: *** [Makefile:9: calculate.o] Ошибка 1
[vvduplenskikh@vvduplenskikh calculate]$ make main.o
gcc gcc -c main.c -g
gcc: предупреждение: gcc: входные файлы компоновки не использованы, поскольку ко
мпоновка не выполнялась
gcc: ошибка: gcc: linker input file not found: Нет такого файла или каталога
make: *** [Makefile:12: main.o] Ошибка 1
[vvduplenskikh@vvduplenskikh calculate]$ make calcul
gcc gcc calculate.o main.o -o calcul -lm
/usr/bin/ld: невозможно найти gcc: Нет такого файла или каталога
collect2: ошибка: выполнение ld завершилось с кодом возврата 1
make: *** [Makefile:6: calcul] Ошибка 1
[vvduplenskikh@vvduplenskikh calculate]$
```

reboot

6.2. Произвожу отладку через gdb:

```
Съ Using host libthread_db library "/lib64/libthread_db.so.1".
Число: list 12, 15
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): Неправильно введено действие inf
[Inferior 1 (process 4543) exited normally]
(гdb) 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     }
(гdb) 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);
(гdb) break 21
Breakpoint 1 at 0x40121e: file calculate.c, line 21.
(гdb) info breakpoints
Num Type Disp Enb Address What
1 breakpoint keep y 0x000000000040121e in Calculate
at calculate.c:21
(гdb) run
Starting program: /home/vvduplenskikh/calculate/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=0x7fffffffdf14 "-")
at calculate.c:21
21         scanf("%f",&SecondNumeral);
(гdb) backtrace
#0 Calculate (Numeral=5, Operation=0x7fffffffdf14 "-") at calculate.c:21
#1 0x00000000004014eb in main () at main.c:16
(гdb) print Numeral
$1 = 5
(гdb) display Numeral
1: Numeral = 5
(гdb) info breakpoints
Num Type Disp Enb Address What
1 breakpoint keep y 0x000000000040121e in Calculate
at calculate.c:21
breakpoint already hit 1 time
(гdb) delete 1
(гdb) q
A debugging session is active.

Inferior 1 [process 4578] will be killed.

Quit anyway? (y or n) ☐
```

gdb

7. Анализирую коды при помощи утилиты splint:

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

calculate.h:6:37: 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:9:31: 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:21:8: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:27:8: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:33:8: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:34:11: 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:37:18: Return value type double does not match declared type float:
        (HUGE_VAL)
    To allow all numeric types to match, use +relaxtypes.
calculate.c:45:8: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:46:14: Return value type double does not match declared type float:
        (pow(Numeral, SecondNumeral))
calculate.c:49:12: Return value type double does not match declared type float:
        (sqrt(Numeral))
calculate.c:51:12: Return value type double does not match declared type float:
        (sin(Numeral))
calculate.c:53:12: Return value type double does not match declared type float:
        (cos(Numeral))
calculate.c:55:12: Return value type double does not match declared type float:
        (tan(Numeral))
calculate.c:59:15: Return value type double does not match declared type float:
        (HUGE_VAL)

Finished checking --- 15 code warnings
[vvduplenskikh@vvduplenskikh calculate]$
```

splint

Ответы на вопросы:

1. Чтобы получить информацию о возможностях программ gcc, make, gdb и др. нужно воспользоваться командой man или опцией -help (-h) для каждой команды.
2. 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.
3. 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».
 4. Основное назначение компилятора языка Си в UNIX заключается в компиляции всей программы и получении исполняемого файла/модуля.
 5. 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.
 6. 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 в одной строке. Для переноса можно в длинной строке команд использовать обратный слэш \. Двойное двоеточие указывает на то, что последовательность команд может содержаться в нескольких последовательных строках.

7. 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

8. 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.

9. Схема отладки программы показана в 6 пункте лабораторной работы.
10. 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.
11. 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.
12. 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 C 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.

Вывод:

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