

Отчет по лабораторной работе №13

Операционные системы

Ничипорова Елена Дмитриевна

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
3	Выводы	19
	Список литературы	20

Список иллюстраций

2.1	Создание нового подкаталога	6
2.2	Создание файлов	6
2.3	текст программы	7
2.4	текст программы	8
2.5	текст программы	8
2.6	текст программы	9
2.7	компиляция программы	9
2.8	исправленный текст программы	9
2.9	компиляция программы	10
2.10	текст программы Makefile	10
2.11	удаление файлов	11
2.12	компиляция файлов	11
2.13	Работа с gdb	11
2.14	run - запуск программы	12
2.15	list	12
2.16	list 12,15	12
2.17	list calculate:20,29	13
2.18	точка останова	13
2.19	info breakpoint	13
2.20	проверка работы программы	14
2.21	delet 1	14
2.22	результат работы splint calculate.c	15
2.23	splint main.c	15

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)

```
ednichiporova@dk3n57 ~ $ mkdir -p ~/work/os/lab_prog
ednichiporova@dk3n57 ~ $
```

Рис. 2.1: Создание нового подкаталога

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

```
ednichiporova@dk3n57 ~/work/os/lab_prog $ touch calculate.h calculate.c main.c
ednichiporova@dk3n57 ~/work/os/lab_prog $ ls
calculate.c calculate.h main.c
ednichiporova@dk3n57 ~/work/os/lab_prog $
```

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)

```

////////////////////////////////////
// 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 файл calculate.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)


```

////////////////////////////////////
// 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("%.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.

```

1 #
2 # Makefile
3 #
4 CC = gcc
5 CFLAGS = -g
6 LIBS = -lm
7 calcul: calculate.o main.o
8     $(CC) calculate.o main.o -o calcul $(LIBS)
9 calculate.o: calculate.c calculate.h
10     $(CC) -c calculate.c $(CFLAGS)
11 main.o: main.c calculate.h
12     $(CC) -c main.c $(CFLAGS)
13 clean:
14     -rm calcul *.o *~
15 # End Makefile

```

Рис. 2.10: текст программы Makefile

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

```
ednichiporova@dk3n57 ~/work/os/lab_prog $ make clean
rm calcul *.o *
```

Рис. 2.11: удаление файлов

- Выполнила компиляцию файлов (рис. 2.12)

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

Рис. 2.12: компиляция файлов

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

```
ednichiporova@dk3n57 ~/work/os/lab_prog $ gdb ./calcul
GNU gdb (Gentoo 10.2 vanilla) 10.2
Copyright (C) 2021 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://bugs.gentoo.org/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./calcul...
(gdb)
```

Рис. 2.13: Работа с gdb

- Запустила программу внутри отладчика(рис. 2.14)

```

ednichiporova@kns7: /work/os/lab_prog $ gdb ./calcul
GNU gdb (Gentoo 10.2 vanilla) 10.2
Copyright (C) 2021 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://bugs.gentoo.org/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./calcul...
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/e/d/ednichiporova/work/os/lab_prog/calcul
Число: 6
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): *
Множитель: 5
30.00
[Inferior 1 (process 11267) exited normally]
(gdb)

```

Рис. 2.14: run - запуск программы

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

```

(gdb) list
1  //////////////////////////////////////////
2  // main.c
3
4  #include<stdio.h>
5  #include"calculate.h"
6
7  int
8  main (void)
9  {
10     float Numeral;

```

Рис. 2.15: list

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

```

(gdb) list 12,15
12     float Result;
13     printf ("Число: ");
14     scanf ("%f",&Numeral);
15     printf ("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): ");
(gdb)

```

Рис. 2.16: list 12,15

- Просмотрела определенные строки(рис. 2.17)


```
(gdb) print Numeral
$1 = 5
(gdb) display Numeral
1: Numeral = 5
```

Рис. 2.20: проверка работы программы

- Убрала точки останова(рис. 2.21)

```
(gdb) info breakpoint
Num  Type      Disp Enb Address      What
1    breakpoint keep y  0x00005555555527d in Calculate at calculate.c:22
(gdb) delete 1
(gdb) info breakpoint
Undefined info command: "breakpoint". Try "help info".
```

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 сохраняются. 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 потере данных.

```

ednichiporova@dk3n57 ~/work/os/lab_prog $ splint calculate.c
Splint 3.1.2 --- 13 Jan 2021

calculate.h:7:37: 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:10:33: Function parameter Operation declared as manifest array
        (size constant is meaningless)
calculate.c: (in function Calculate)
calculate.c:14:6: 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:19:7: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:24:7: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:29:7: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:30:10: 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:32:10: Return value type double does not match declared type float:
        (HUGE_VAL)
    To allow all numeric types to match, use +relaxtypes.
calculate.c:39:6: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:40:12: Return value type double does not match declared type float:
        (pow(Numeral, SecondNumeral))
calculate.c:43:11: Return value type double does not match declared type float:
        (sqrt(Numeral))
calculate.c:45:11: Return value type double does not match declared type float:
        (sin(Numeral))
calculate.c:47:11: Return value type double does not match declared type float:
        (cos(Numeral))
calculate.c:49:11: Return value type double does not match declared type float:
        (tan(Numeral))
calculate.c:53:13: Return value type double does not match declared type float:
        (HUGE_VAL)

Finished checking --- 15 code warnings

```

Рис. 2.22: результат работы splint calculate.c

```

ednichiporova@dk3n57 ~/work/os/lab_prog $ splint main.c
Splint 3.1.2 --- 13 Jan 2021

calculate.h:7:37: 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)
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:16:3: Return value (type int) ignored: scanf("%s", Oper...

Finished checking --- 3 code warnings

```

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 в компиляции 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 в одной 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 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.

3 Выводы

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

Список литературы