

Отчет по лабораторной работе №13

Дисциплина: Операционные системы

Кашкин Иван Евгеньевич

Содержание

Цель работы	5
Задание	6
Теоретическое введение	7
Выполнение лабораторной работы	8
Выводы	14
Контрольные вопросы	15
Список литературы	19

Список иллюстраций

0.1	Создание	8
0.2	calculate.c	9
0.3	calculate.c	9
0.4	calculate.h	10
0.5	main.c	10
0.6	gcc	10
0.7	Makefile	11
0.8	Makefile	11
0.9	Make	12
0.10	Программа №2	12
0.11	Ввод программы	13
0.12	Ввод программы	13

Список таблиц

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.

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`
4. При необходимости исправьте синтаксические ошибки.
5. Создайте `Makefile` со следующим содержанием
6. С помощью `gdb` выполните отладку программы `calcul` (перед использованием `gdb` исправьте `Makefile`)
7. С помощью утилиты `splint` попробуйте проанализировать коды файлов `calculate.c` и `main.c`.

Теоретическое введение

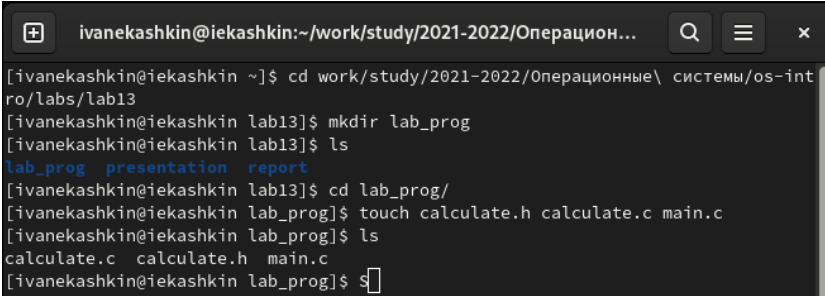
Этапы разработки приложений - Процесс разработки программного обеспечения обычно разделяется на следующие этапы: - планирование, включающее сбор и анализ требований к функционалу и другим характеристикам разрабатываемого приложения; - проектирование, включающее в себя разработку базовых алгоритмов и 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.

Выполнение лабораторной работы

1. Зашёл в каталог лабораторной работы, создал папку для программ и создал сами программы (рис. [-@fig:001])



```
ivaneekashkin@iekashkin:~/work/study/2021-2022/Операционные системы/os-int
ro/labs/lab13$ cd work/study/2021-2022/Операционные системы/os-int
ro/labs/lab13
[ivaneekashkin@iekashkin lab13]$ mkdir lab_prog
[ivaneekashkin@iekashkin lab13]$ ls
lab_prog  presentation  report
[ivaneekashkin@iekashkin lab13]$ cd lab_prog/
[ivaneekashkin@iekashkin lab_prog]$ touch calculate.h calculate.c main.c
[ivaneekashkin@iekashkin lab_prog]$ ls
calculate.c calculate.h main.c
[ivaneekashkin@iekashkin lab_prog]$
```

Рис. 0.1: Создание

2. Написал командный файл для файла calculate.c (рис. [-@fig:002])


```

#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 деления на ноль!");
        }
        else
            return (Numeral/SecondNumeral);
    }
    else if(strncmp(Operation, "pow", 3) == 0)
s {

```

Рис. 0.2: calculate.c

```

        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);
    }
}

```

Рис. 0.3: calculate.c

- Написал командный файл calculate.h (рис. [-@fig:004])

```
#ifndef CALCULATE_H_
#define CALCULATE_H_

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

#endif
```

Рис. 0.4: calculate.h

- Написал командный файл main.c (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("%f",Operation);
    Result = Calculate(Numeral,Operation);
    printf("%6.2f\n",Result);
    return 0;
}
```

Рис. 0.5: main.c

3. Выполните компиляцию программы посредством gcc (рис. [-@fig:006])

```
[ivaneekashkin@iekashkin lab_prog]$ gcc -c calculate.c
[1]+  Завершён          emacs
[ivaneekashkin@iekashkin lab_prog]$ gcc -c main.c
[ivaneekashkin@iekashkin lab_prog]$ gcc calculate.o main.o -o calcul -lm
[ivaneekashkin@iekashkin lab_prog]$
```

Рис. 0.6: gcc

4. Создал Makefile (рис. [-@fig:007])

```
CC = gcc
CFLAGS =
LIBS = -lm

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

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

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

clean:
    -rm calcul *.o*-
```

Рис. 0.7: Makefile

- Исправил Makefile (рис. [-@fig:008])

```
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*-
```

Рис. 0.8: Makefile

- С помощью Makefile создал нужные файлы (рис. [-@fig:009])

```

[ivanekashkin@iekashkin lab_prog]$ make clean
rm calcul *.o*~
rm: невозможно удалить '*.o*~': Нет такого файла или каталога
make: [Makefile:15: clean] Ошибка 1 (игнорирование)
[ivanekashkin@iekashkin lab_prog]$ make calculate.o
make: «calculate.o» не требует обновления.
[ivanekashkin@iekashkin lab_prog]$ make calculate.o
gcc -c calculate.c
[ivanekashkin@iekashkin lab_prog]$ make main.o
gcc -c main.c
[ivanekashkin@iekashkin lab_prog]$ make calcul
make: *** Нет правила для сборки цели «calcul». Останов.
[ivanekashkin@iekashkin lab_prog]$ emacs &
[1] 11304
[ivanekashkin@iekashkin lab_prog]$ make calcul
gcc calculate.o main.o -o calcul -lm
[1]+  Завершён          emacs
[ivanekashkin@iekashkin lab_prog]$ █

```

Рис. 0.9: Make

5. С помощью gdb выполните отладку программы calcul (перед использованием gdbm исправьте Makefile) (рис. [-@fig:007])

```

[ivanekashkin@iekashkin lab_prog]$ gdb ./calcul
GNU gdb (GDB) Fedora 11.2-2.fc35
Copyright (C) 2022 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-redhat-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
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...

This GDB supports auto-downloading debuginfo from the following URLs:
https://debuginfod.fedoraproject.org/
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.
Downloading separate debug info for /home/ivanekashkin/work/study/2021-2022/Операционные системы/os-intro/labs/lab13/lab_prog/./calcul...
(No debugging symbols found in ./calcul)
(gdb) █

```

Рис. 0.10: Программа №2

```

(gdb) run
Starting program: /home/ivanekashkin/work/study/2021-2022/Операционные системы
s-intro/labs/lab13/lab_prog/calcul
Downloading separate debug info for /home/ivanekashkin/work/study/2021-2022/Операционные системы/os-intro/labs/lab13/lab_prog/system-supplied DS0 at 0x7ffff7f
000...
Downloading separate debug info for /lib64/libm.so.6...
Downloading separate debug info for /lib64/libc.so.6...
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".
Число: 3
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): -
Вычисляемое: 9
12.00
[Inferior 1 (process 11571) exited normally]
(gdb) 

```

Рис. 0.11: Ввод программы

```

(gdb) list
Downloading source file /usr/src/debug/glibc-2.34-30.fc35.x86_64/elf/sofini.c...
1      /* Terminate the frame unwind info section with a 4byte 0 as a sentinel
2         this would be the 'length' field in a real FDE.  */
3
4      typedef unsigned int ui32 __attribute__((mode(SI)));
5      static const ui32 __FRAME_END__[1]
6          __attribute__((used, section(".eh_frame")))
7          = { 0 };
(gdb) 

```

Рис. 0.12: Ввод программы

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.

Контрольные 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 с 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 можно 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 работе программы, переменные с некорректно заданными значениями и типами и многое другое.

Список литературы

::: {#Лабораторная работа No 13. Средства, применяемые при разработке программного обеспечения в ОС типа UNIX/Linux} :::