

Отчёт по лабораторной работе

Лабораторная работа № 13

Мурзаев Замир Зейнадинович

Содержание

1 Цель работы	5
2 Задание	6
3 Выполнение лабораторной работы	7
4 Ответы на вопросы	11

Список иллюстраций

3.1	Создание подкаталога	7
3.2	Создание файлов	7
3.3	Программа на С	8
3.4	Программа на С	8
3.5	Программа на С	9
3.6	gcc	9
3.7	Makefile	10

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 С калькулятора с простейшими функциями.

2 Задание

1. В домашнем каталоге создайте подкаталог ~/work/os/lab_prog.
2. Создайте в нём файлы: calculate.h, calculate.c, main.c.
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):

3 Выполнение лабораторной работы

В домашнем каталоге создаем подкаталог (рис. 3.1).

```
zzmurzaev@dk5n51 ~ $ cd work/os
zzmurzaev@dk5n51 ~/work/os $ mkdir lab_prog
zzmurzaev@dk5n51 ~/work/os $ █
```

Рис. 3.1: Создание подкаталога

Создаем в нем файлы (рис. 3.2).

```
zzmurzaev@dk5n51 ~/work/os $ touch calculate.h
zzmurzaev@dk5n51 ~/work/os $ touch calculate.c
zzmurzaev@dk5n51 ~/work/os $ touch main.c
```

Рис. 3.2: Создание файлов

Заполняем calculate.c (рис. 3.3).

```
calculate.c
~/work/os/lab_prog

calculate.c                               calculate.h

1 #include <stdio.h>
2 #include <math.h>
3 #include <string.h>
4 #include "calculate.h"
5
6 float Calculate(float Numeral, char Operation[4])
7 {
8     float SecondNumeral;
9     if(strncmp(Operation, "+", 1) == 0)
10    {
11        printf("Второе слагаемое: ");
12        scanf("%f", &SecondNumeral);
13        return(Numeral + SecondNumeral);
14    }
15    else if(strncmp(Operation, "-", 1) == 0)
16    {
17        printf("Вычитаемое: ");
18        scanf("%f", &SecondNumeral);
19        return(Numeral - SecondNumeral);
20    }
21    else if(strncmp(Operation, "*", 1) == 0)
22    {
23        printf("Множитель: ");
24        scanf("%f", &SecondNumeral);
25        return(Numeral * SecondNumeral);
26    }
27    else if(strncmp(Operation, "/", 1) == 0)
28    {
29        printf("Делитель: ");
30        scanf("%f", &SecondNumeral);
31        return(Numeral / SecondNumeral);
32    }
33}
```

Рис. 3.3: Программа на С

Заполняем calculate.h (рис. 3.4).

```
calculate.h
~/work/os/lab_prog

calculate.c                               calculate.h

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

Рис. 3.4: Программа на С

Заполняем main.c (рис. 3.5).

```
main.c
~/work/os/lab_prog

calculate.c          calculate.h

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

Рис. 3.5: Программа на С

Компилируем файлы (рис. 3.6).

```
zzmurzaev@dk5n51 ~/work/os/lab_prog $ gcc -c main.c
zzmurzaev@dk5n51 ~/work/os/lab_prog $ gcc calculate.o main.o -o calcul -lm
```

Рис. 3.6: gcc

Создаем Makefile (рис. 3.7).

В содержании файла указаны флаги компиляции, тип компилятора и файлы, которые должен собрать сборщик.

The screenshot shows a terminal window with the title "Makefile" and the path "~/work/os/lab_prog". The window contains the following Makefile code:

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

Рис. 3.7: Makefile

Проверяем работу калькулятора (рис. ??).

```
zzmurzaev@dk5n51 ~/work/os/lab_prog $ ./calcul
Число: 3
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): -
Вычитаемое: 2
1.00 # Выводы
```

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

4 Ответы на вопросы

Как получить информацию о возможностях программ `gcc`, `make`, `gdb` и др.?

Ответ: при помощи программы `man`.

Назовите и дайте краткую характеристику основным этапам разработки приложений в СУБД.

Ответ: 1. Выбор названия 2. Выбор языка программирования 3. Попытка выполнить работу всю зараз 4. Отрицание 5. Гнев 6. Торг 7. Депрессия 8. Принятие

Что такое суффикс в контексте языка программирования? Приведите примеры использования.

Ответ: финальная часть названия программы, обычно отделяемая точкой.

Каково основное назначение компилятора языка С в UNIX?

Ответ: компилятор языка С в UNIX в основном компилирует программы языка С в UNIX, написанные на языке С в UNIX.

Для чего предназначена утилита `make`?

Ответ: При разработке большой программы, состоящей из нескольких исходных файлов заголовков, приходится постоянно следить за файлами, которые требуют перекомпиляции после внесения изменений. Программа `make` освобождает пользователя от такой рутинной работы и служит для документирования взаимосвязей между файлами. Описание взаимосвязей и соответствующих действий хранится в так называемом `make`-файле, который по умолчанию имеет имя `makefile` или `Makefile`.

Приведите пример структуры Makefile. Дайте характеристику основным элементам этого файла.

Ответ: В общем случае 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. 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. Приведённый выше make-файл для программы abcd.c включает два способа компиляции и построения исполняемого модуля. Первый способ предусматривает обычную компиляцию с построением исполняемого модуля с именем abcd. Второй способ позволяет включать в исполняемый модуль testabcd возможность выполнить процесс отладки на уровне исходного текста. Пример можно найти в задании 5.

Назовите основное свойство, присущее всем программам отладки. Что необходимо сделать для его реализации?

Ответ: свойство - анализ кода; для анализа необходимо скомпилировать программу.

Назовите и дайте основную характеристику основным командам отладчика gdb.

Ответ: см. ответ к вопросу 6.

Опишите по шагам схему отладки программы, которую вы использовали при выполнении

Ответ: 1. Вначале я запустил gdb 2. Затем я его закрыл

Прокомментируйте реакцию компилятора на синтаксические ошибки в программе при его

Ответ: когда я увидел реакцию компилятора на синтаксические ошибки в программе при его первом запуске, я был возмущён, поражён, обескуражен, ошеломлён, фruстрирован и изумлён. Но использовал совершенно другие выражения.

Назовите основные средства, повышающие понимание исходного кода программы.

Ответ: Если вы работаете с исходным кодом, который не вами разрабатывался, то назначение различных конструкций может быть не совсем понятным. Система разработки приложений UNIX предоставляет различные средства, повышающие понимание исходного кода. К ним относятся: – cscope - исследование функций, содержащихся в программе; – splint — критическая проверка программ, написанных на языке Си.

Каковы основные задачи, решаемые программой splint?

Ответ: анализ кода. # Список литературы{.unnumbered}