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

Выполнение лабораторной работы

1. В домашнем каталоге создал подкаталог ~/work/os/lab_prog.

```
$ mkdir work/
$ mkdir work/os/
$ mkdir work/os/lab_prog
$ cd work/os/lab_prog/
```

2. Создал в нём файлы: calculate.h, calculate.c, main.c.

```
 touch calculate.h calculate.c main.c
```

```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;
}
```

```c
#ifndef CALCULATE_H_
#define CALCULAE_H_
float Calculate(float Numeral, char Operation[4]);
#endif
```

```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 слашаемое: ");
                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, "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);
}
```

```c
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
```

3. Выполнил компиляцию программы посредством gcc.

```
gcc -c calculate.c
gcc -c main.c
gcc calculate.o main.o -o calcul -lm
```

4. Исправил синтаксические ошибки.

5. Создал Makefile.

```
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 *~
```

6. С помощью gdb выполните отладку программы calcul.

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

Контрольные вопросы

1. Дополнительную информацию о этих программах можно получить с помощью функций info и man.

2. Unix поддерживает следующие основные этапы разработки приложений:

-создание исходного кода программы;

- представляется в виде файла;

-сохранение различных вариантов исходного текста;

-анализ исходного текста;

Необходимо отслеживать изменения исходного кода, а также при работе более двух программистов над проектом программы нужно, чтобы они не делали изменений кода в одно время.

-компиляция исходного текста и построение исполняемого модуля;

-тестирование и отладка;

-проверка кода на наличие ошибок

-сохранение всех изменений, выполняемых при тестировании и отладке.

3. Использование суффикса ".c" для имени файла с программой на языке Си отражает удобное и полезное соглашение, принятое в ОС UNIX.

Для любого имени входного файла суффикс определяет какая компиляция требуется.

Суффиксы и префиксы указывают тип объекта. Одно из полезных свойств компилятора Си — его способность по суффиксам определять типы файлов.

По суффиксу .c компилятор распознает, что файл abcd.c должен компилироваться, а по суффиксу .o, что файл abcd.o является объектным модулем и для получения исполняемой программы необходимо выполнить редактирование связей.

Простейший пример командной строки для компиляции программы abcd.c и построения исполняемого модуля abcd имеет вид: gcc -o abcd abcd.c.

Некоторые проекты предпочитают показывать префиксы в начале текста изменений для старых (old) и новых (new) файлов.

Опция – prefix может быть использована для установки такого префикса.

Плюс к этому команда bzr diff -p1 выводит префиксы в форме которая подходит для команды patch -p1.

4. Основное назначение компилятора с языка Си заключается в компиляции всей программы в целом и получении исполняемого модуля.

5. 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.

6. makefile для программы abcd.c мог бы иметь вид:

#

#

Makefile

#

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 *~

# End Makefile
```

8. – backtrace – выводит весь путь к текущей точке останова, то есть названия всех функций, начиная от main(); иными словами, выводит весь стек функций;

– break – устанавливает точку останова; параметром может быть номер строки или название

функции;

– clear – удаляет все точки останова на текущем уровне стека (то есть в текущей функции);

– continue – продолжает выполнение программы от текущей точки до конца;

– delete – удаляет точку останова или контрольное выражение;

– display – добавляет выражение в список выражений, значения которых отображаются каждый раз при остановке программы;

– finish – выполняет программу до выхода из текущей функции; отображает возвращаемое значение,если такое имеется;

– info breakpoints – выводит список всех имеющихся точек останова;

– info watchpoints – выводит список всех имеющихся контрольных выражений;

– splist – выводит исходный код; в качестве параметра передаются название файла исходного кода, затем, через двоеточие, номер начальной и конечной строки;

– next – пошаговое выполнение программы, но, в отличие от команды step, не выполняет пошагово вызываемые функции;

– print – выводит значение какого-либо выражения (выражение передаётся в качестве параметра);

– run – запускает программу на выполнение;

– set – устанавливает новое значение переменной

– step – пошаговое выполнение программы;

– watch – устанавливает контрольное выражение, программа остановится, как только значение контрольного выражения изменится;

9. 1) Выполнили компиляцию программы 2)Увидели ошибки в программе 3) Открыли редактор и исправили программу 4) Загрузили программу в отладчик gdb 5) run — отладчик выполнил программу, мы ввели требуемые значения. 6) программа завершена, gdb не видит ошибок.

10. 1 и 2.) Мы действительно забыли закрыть комментарии; 3.) отладчику не понравился формат %s для &Operation, т.к %s — символьный формат, а значит необходим только Operation.

11. 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.

12. 1. Проверка корректности задания аргументов всех ис

функций, а также типов возвращаемых ими значений;

2. Поиск фрагментов исходного текста, корректных с точки зрения синтаксиса языка Си, но малоэффективных с точки зрения их реализации или содержащих в себе семантические ошибки;

3. Общая оценка мобильности пользовательской программы.

Выводы

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