РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

ОТЧЕТ

по лабораторной работе № 13

дисциплина: операценные системы

Студент: Ван Яо

Группа: НПмбд-02-21

МОСКВА

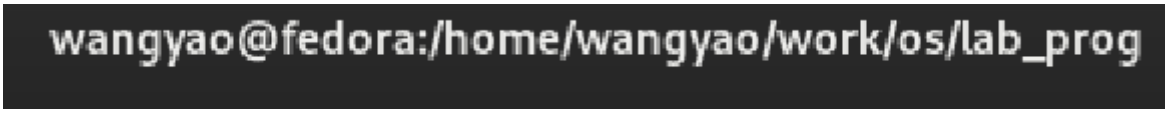
20 г.

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.

Последовательность выполнения работы

\1. В домашнем каталоге создайте подкаталог ~/work/os/lab_prog



2Создайте 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. Реализация функций калькулятора в файле calculate.h:

```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("second term: ");
scanf("%f",&SecondNumeral);
return(Numeral+SecondNumeral);
}
else if(strncmp(Operation,"-", 1)==0)
{
printf("subtrahend: ");
scanf("%f",&SecondNumeral);
return(Numeral-SecondNumeral);
}

else if(strncmp(Operation,"*", 1)==0)
{
```

```c
#ifndef CALCULATE_H_
#define CALCULATE_H_

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

~
~
```

```
#include <stdio.h>
#include "calculate.h"
int
main (void)
{
float Numeral;
char Operation[4];
float Result;
printf("number: ");
scanf("%f",&Numeral);
printf("Operation (+,-,*,/,pow,sqrt,sin,cos,tan): ");
scanf("%s",&Operation);
Result=Calculate(Numeral,Operation);
printf("%6.2f\n",Result);
return 0;
}
```

3

Выполните компиляцию программы посредством gcc:

```
[root@fedora lab_prog]# gcc -c calculate.c
[root@fedora lab_prog]# gcc -c main.c
[root@fedora lab_prog]# gcc calculate.o main.o -o calcul -lm
```

\5. Создайте Makefile со следующим содержанием:

```
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*~
```

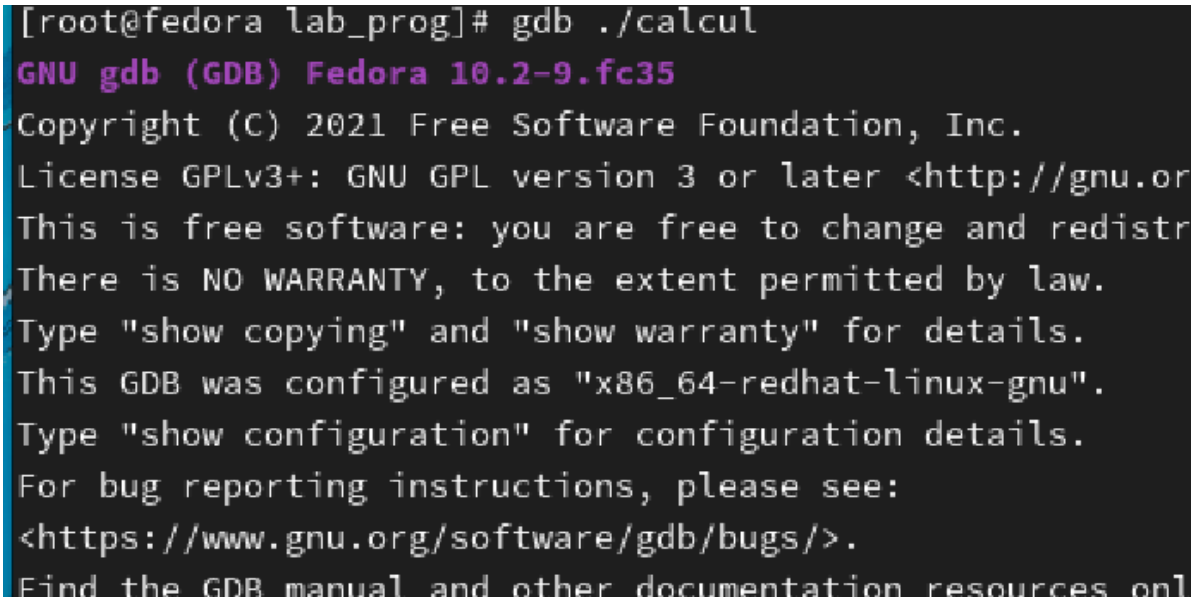
в начале файла заданы три переменные: CC и CFLAGS

Затем указаны цели, их зависимости и соответствующие команды

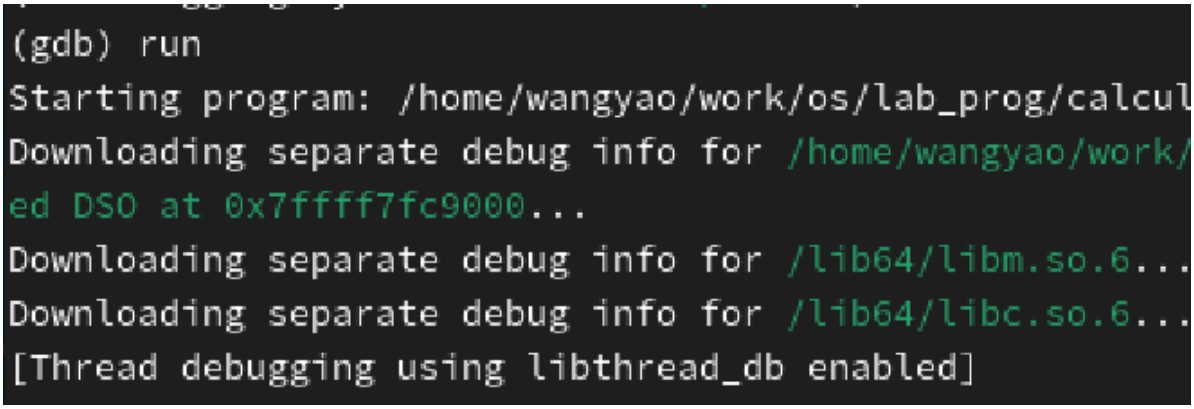
В командах происходит обращение к значениям переменных.

\6. С помощью gdb выполните отладку программы calcul (перед использованием gdb исправьте Makefile): – Запустите отладчик GDB, загрузив в него программу для отладки:

Запустите отладчик GDB, загрузив в него программу для отладки

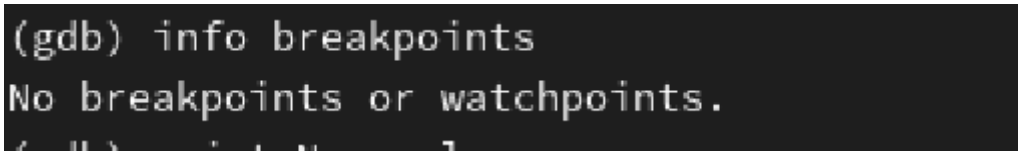


Для запуска программы внутри отладчика введите команду run

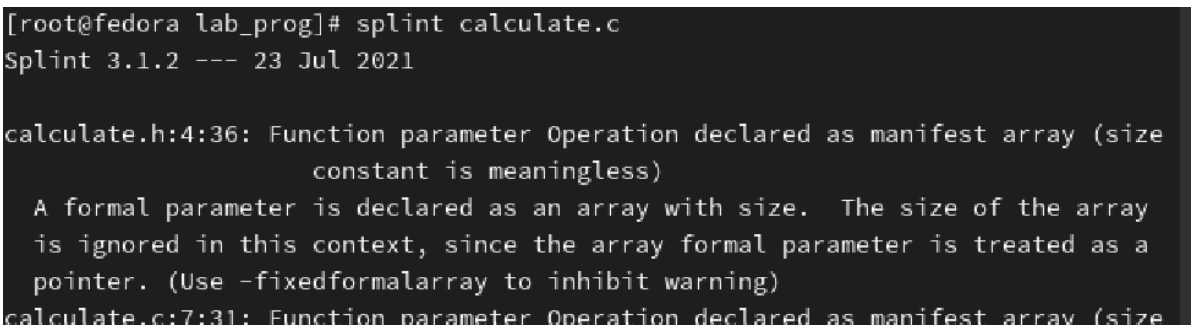


– Для постраничного (по 9 строк) просмотра исходного код используйте команду list:

Выведите информацию об имеющихся в проекте точка останова:



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



```
[root@fedora lab_prog]# splint main.c
Splint 3.1.2 --- 23 Jul 2021

calculate.h:4:36: 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:10:1: 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:12:12: Format argument 1 to scanf (%s) expects char * gets char [4] *:
                &Operation
  Type of parameter is not consistent with corresponding code in format string.
```

Выводы

получил 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

1info man

2– 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.

3

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

4

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

5Для 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

6

В самом простом случае Makefile имеет следующий синтаксис: 1 <цель*1*> <*цель*2> ... : <зависимость*1*> <*зависимость*2> ... 2 <команда 1> 3 ... 4 <команда n> 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.

7В 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.

8

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

9

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: 1 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: 1 gdb file.o

11splint

12

Эта утилита анализирует программный код, проверяет корректность задания аргументов использованных в программе функций и типов возвращаемых значений, обнаруживает синтаксические и семантические ошибки.