Отчёт по лабораторной работе 13”

Средства, применяемые при разработке программного обеспечения в ОС типа UNIX/Linux

Ангелина Дмитриевна Чванова

Содержание

Список иллюстраций

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. 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.
6. С помощью gdb выполните отладку программы calcul (перед использованием gdb исправьте Makefile).

3 Теоретическое введение

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

4 Выполнение лабораторной работы

1. В домашнем каталоге создайте подкаталог ~/work/os/lab_prog.(рис. 4.1)

```
[adchvanova@fedora ~]$ mkdir -p ~/work/os/lab_prog
```

Рис. 4.1: подкаталог ~/work/os/lab_prog

2. Создайте в нём файлы: calculate.h, calculate.c, main.c.(рис. 4.2) Это будет примитивнейший калькулятор, способный складывать, вычитать, умножать и делить, возводить число в степень, брать квадратный корень, вычислять sin, cos, tan. (рис. 4.3,4.4,4.5,4.6) При запуске он будет запрашивать первое число, операцию, второе число. После этого программа выведет результат и остановится.

```
[adchvanova@fedora ~]$ cd ~/work/os/lab_prog
[adchvanova@fedora lab_prog]$ touch calculate.h calculate.c main.c
[adchvanova@fedora lab_prog]$ ls
calculate.c  calculate.h  main.c
```

Рис. 4.2: Создание файлов

```
////////////////////////////////////
// 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: ");
```

Рис. 4.3: Файл calculate.c

```
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));
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. 4.4: Файл calculate.c

```
////////////////////////////////////////
// calculate.h

#ifndef CALCULATE_H_
#define CALCULATE_H_

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

#endif /*CALCULATE_H_*/
```

Рис. 4.5: Файл calculate.h

```
////////////////////////////////////////
 // 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. 4.6: Файл main.c

3. Выполните компиляцию программы посредством gcc.(рис. 4.7)

```
[adchvanova@fedora lab_prog]$ gcc -c calculate.c
[adchvanova@fedora lab_prog]$ gcc -c main.c
[adchvanova@fedora lab_prog]$ gcc calculate.o main.o -o calcul -lm
```

Рис. 4.7: Компиляция программы посредством gcc

4. При необходимости исправьте синтаксические ошибки.

Ошибок нет.

5. Создайте Makefile.(рис. 4.8,4.9)

```
[adchvanova@fedora lab_prog]$ touch Makefile
[adchvanova@fedora lab_prog]$ ls
calcul       calculate.c~  calculate.h~  main.c   main.o
calculate.c  calculate.h   calculate.o   main.c~  Makefile
```

Рис. 4.8: Создание Makefile

```
#
 # 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 Makefil
```

Рис. 4.9: Makefile

6. С помощью gdb выполните отладку программы calcul (перед использованием gdb исправьте Makefile). (рис. 4.10,4.11,4.12,4.13,4.14)

```
[adchvanova@fedora 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>
```

Рис. 4.10: Запуск отладчика GDB

```
(gdb) run
Starting program: /home/adchvanova/work/os/lab_prog/calcul
Downloading separate debug info for /home/adchvanova/work/os/lab_prog/system-sup
plied DSO at 0x7ffff7fc5000...
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: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): +
Второе слагаемое: 7
 12.00
[Inferior 1 (process 9256) exited normally]
(gdb) list
Downloading source file /usr/src/debug/glibc-2.34-33.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__[]
6         __attribute__ ((used, section (".eh_frame")))
7         = { 0 };
(gdb) list 12,15
```

Рис. 4.11: команда run, list(обычный и с параметром)

```
(gdb) list calculate.c:20,29
20          }
21        else if(strncmp(Operation, "*", 1)==0)
22          {
23            printf("Множитель: ");
24            scanf("%f",&SecondNumeral);
25            return(Numeral*SecondNumeral);
26          }
```

Рис. 4.12: list calculate.c:20,29

```
25            return(Numeral*SecondNumeral);
26          }
27        else if(strncmp(Operation,"/",1)==0)
(gdb) break 21
Breakpoint 1 at 0x555555400991: file calculate.c, line 21.
```

Рис. 4.13: Точка останова

```
Breakpoint 1 at 0x555555400991: file calculate.c, line 21.
(gdb) info breakpoints
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x0000555555400991 in Calculate at
(gdb)
```

Рис. 4.14: ИНформация о е точке останова

5 Выводы

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

6 Контрольные вопросы

1). Чтобы получить информацию о возможностях программ gcc, make, gdbи др.нужно воспользоваться командой manили опцией -help(-h)для каждой команды.

2). Процесс разработки программного обеспечения обычно разделяется на следующие этапы:

1. планирование, включающее сбор и анализ требований к функционалу и другим характеристикам разрабатываемого приложения;
2. проектирование, включающее в себя разработку базовых алгоритмов и спецификаций, определение языка программирования;
3. непосредственная разработка приложения: окодирование –по сути создание исходного текста программы (возможно в нескольких вариантах); –анализ разработанного кода; осборка, компиляция и разработка исполняемого модуля; отестирование и отладка, сохранение произведённых изменений;
4. документирование. Для создания исходного текста программы разработчик может воспользоваться любым удобным для него редактором текста: vi, vim, mceditor, emacs, geanyи 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) .своспринимаются gccкак программы на языке C, файлы с расширением .ссили

.C–как файлы на языке C++, а файлы срасширением .осчитаются объектными.Например, в команде «gcc-cmain.c»:gccпо расширению (суффиксу) .сраспознает 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 -oи в качестве параметра задать имя создаваемого файла: «gcc-ohellomaiВ ходе выполнения данной лабораторной работы я приобрелапростейшие навыки разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linuxна примере создания на языке программирования С калькулятора с простейшими функциями.n.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 makeнеобходимо в корне рабочего каталога с Вашим проектом создать файл с названием makefileили 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 Makefileможет 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 Makefileимеет вид: target1 [target2…]:[:] [dependment1…][(tab)commands] [#commentary][(tab)commands] [#commentary]. Здесь знак # определяет начало комментария (содержимое от знака # и до конца строки не будет обрабаты-

ваться. 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.Пример более сложного синтаксиса Makefile:## Makefile for abcd.c#CC = gccCFLAGS =# Compile abcd.c normalyabcd: abcd.c$(CC) -o abcd $(CFLAGS) abcd.cclean:-rm abcd .*o* ~# EndMakefileforabcd.c. В этом примере в начале файла заданы три переменные: CC и CFLAGS. Затем указаны цели, их зависимости и соответствующие команды. В командах происходит обращение к значениям переменных. Цель с именем cleanпроизводит очистку каталога от файлов, полученных в результате компиляции. Для её описания использованы регулярные выражения.

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 GNUдля ОС типа UNIXвходит отладчик GDB(GNUDebugger). Для использования GDB необходимо скомпилировать анализируемый код программы таким образом, чтобы отладочная информация содержалась в результирующем бинарном файле. Для этого следует воспользоваться опцией -gкомпилятора gcc: gcc-cfile.c-g. После этого для начала работы с gdbнеобходимо в командной строке ввести одноимённую команду, указав в качестве аргумента анализируемый бинарный файл: gdbfile.o

8). Основные команды отладчика gdb: 1. backtrace – вывод на экран пути к текущей точке останова (по сутивывод – названий всех функций); 2. break – установить точку останова (в качестве параметра можетбыть указан номер строки или название функции); 3. clear – удалить все точки останова в функции; 4. continue – продолжить выполнение программы; 5. delete – удалить точку останова; 6. display – добавить выражение в список выражений, значения которых отображаются при достижении точки останова программы; 7. finish – выполнить

программу до момента выхода из функции; 8. info breakpoints –вывести на экран список используемых точек останова; 9. info watchpoints –вывести на экран список используемых контрольных выражений; 10. list – вывести на экран исходный код (вВ ходе выполнения данной лабораторной работы я приобрелапростейшие навыки разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linuxна примере создания на языке программирования С калькулятора с простейшими функциями. качестве параметра может быть указано название файла и через двоеточие номера начальнойи конечной строк); 11. next – выполнить программу пошагово, но без выполнения вызываемых в программе функций; 12. print – вывести значение указываемого в качестве параметра выражения; 13. run – запуск программы на выполнение; 14. set – установить новое значение переменной; 15. step – пошаговое выполнение программы; 16. watch – установить контрольное выражение, при изменении значения которого программа будет остановлена. Для выхода из gdbможно 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-hи mangdb.

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 массивасимволов уже является указателемна первый элементэтого 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 Санализатор splintгенерирует 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 работт программы, переменные с некорректно заданными значениямии типами и многое другое.

7 Библиография

1. Программное обеспечение GNU/Linux. Лекция 9. Хранилище и дистрибутив (Г. Курячий, МГУ)
2. Программное обеспечение GNU/Linux. Лекция 10. Минимальный набор знаний (Г. Курячий, МГУ)
3. Программное обеспечение GNU/Linux. Лекция 11. udev, DBus, PolicyKit (Г. Курячий, МГУ)
4. Электронный ресурс: https://vunivere.ru/work23597
5. Электронный ресурс: https://it.wikireading.ru/34160