

Лабораторная работа №13

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

Заболотная Кристина Александровна

Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	8
4	Контрольные вопросы	15
4.1	Ответы на контрольные вопросы	16
5	Выводы	20
	Список литературы	21

Список иллюстраций

3.1	создаем подкаталог	8
3.2	создаем файлы	8
3.3	скрипт 1	9
3.4	второй скрипт	10
3.5	main	10
3.6	компиляция	11
3.7	файлы	11
3.8	Создан Makefile	12
3.9	Создан Makefile	13
3.10	Создан Makefile	14

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

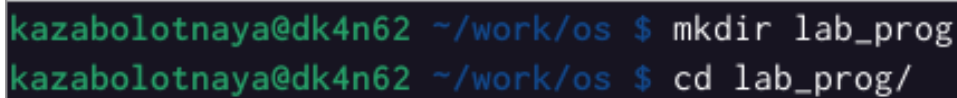
1. В домашнем каталоге создайте подкаталог `~/work/os/lab_prog`.
2. Создайте в нём файлы: `calculate.h`, `calculate.c`, `main.c`. Это будет примитивнейший калькулятор, способный складывать, вычитать, умножать и делить, возводить число в степень, брать квадратный корень, вычислять `sin`, `cos`, `tan`. 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`): – 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` – Для просмотра строк с 12 по 15 основного файла используйте `list` с параметрами – Для просмотра определённых строк не основного файла используйте `list` с параметрами – Установите точку останова в файле `calculate.c` на строке номер 21 – Выведите информацию об имеющихся в проекте точка останова – Запустите программу внутри отладчика и убедитесь, что программа остановится в момент прохождения точки останова – Отладчик выдаст следующую информацию а команда `backtrace` покажет весь стек вызываемых функций от начала программы до текущего места. – Посмотрите, чему равно на этом этапе значение пе-

ременной Numeral На экран должно быть выведено число 5. – Сравните с результатом вывода на экран после использования команды

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

3 Выполнение лабораторной работы

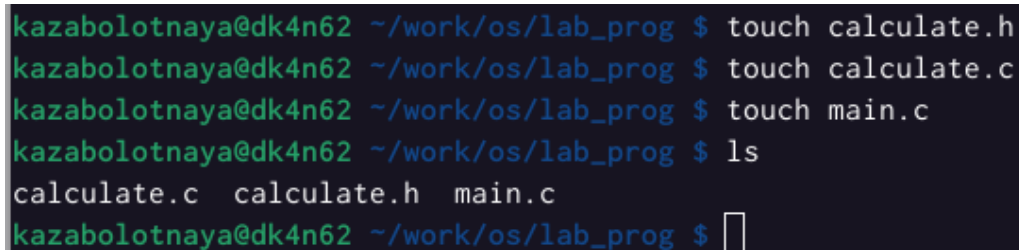
1. В домашнем каталоге создан подкаталог ~/work/os/lab_prog.



```
kazabolotnaya@dk4n62 ~/work/os $ mkdir lab_prog
kazabolotnaya@dk4n62 ~/work/os $ cd lab_prog/
```

Рис. 3.1: создаем подкаталог

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.



```
kazabolotnaya@dk4n62 ~/work/os/lab_prog $ touch calculate.h
kazabolotnaya@dk4n62 ~/work/os/lab_prog $ touch calculate.c
kazabolotnaya@dk4n62 ~/work/os/lab_prog $ touch main.c
kazabolotnaya@dk4n62 ~/work/os/lab_prog $ ls
calculate.c calculate.h main.c
kazabolotnaya@dk4n62 ~/work/os/lab_prog $
```

Рис. 3.2: создаем файлы

Реализация функций калькулятора в файле calculate.c.


```

1 #include <stdio.h>
2 #include <math.h>
3 #include <string.h>
4 #include "calculate.h"
5
6 float
7 Calculate(float Numeral, char Operation[4])
8 {
9     float SecondNumeral;
10    if(strncmp(Operation, "+", 1) == 0)
11    {
12        printf("Второе слагаемое: ");
13        scanf("%f",&SecondNumeral);
14        return(Numeral + SecondNumeral);
15    }
16    else if(strncmp(Operation, "-", 1) == 0)
17    {
18        printf("Вычитаемое: ");
19        scanf("%f",&SecondNumeral);
20        return(Numeral - SecondNumeral);
21    }
22    else if(strncmp(Operation, "*", 1) == 0)
23    {
24        printf("Множитель: ");
25        scanf("%f",&SecondNumeral);
26        return(Numeral * SecondNumeral);
27    }
28    else if(strncmp(Operation, "/", 1) == 0)
29    {
30        printf("Делитель: ");
31        scanf("%f",&SecondNumeral);
32        if(SecondNumeral == 0)
33        {
34            printf("Ошибка: деление на ноль! ");
35            return(HUGE_VAL);
36        }
37        else
38            return(Numeral / SecondNumeral);
39    }
40    else if(strncmp(Operation, "pow", 3) == 0)
41    {
42        printf("Степень: ");
43        scanf("%f",&SecondNumeral);
44        return(pow(Numeral, SecondNumeral));
45    }
46    else if(strncmp(Operation, "sqrt", 4) == 0)

```

Рис. 3.3: скрипт 1

Интерфейсный файл calculate.h, описывающий формат вызова функции калькулятора.

```
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: второй скрипт

Основной файл main.c, реализующий интерфейс пользователя и калькулятора.

```
1 #include <stdio.h>
2 #include "calculate.h"
3
4 int
5 main (void)
6 {
7     float Numeral;
8     char Operation[4];
9     float Result;
10    printf("Число: ");
11    scanf("%f",&Numeral);
12    printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): ");
13    scanf("%s",&Operation);
14    Result = Calculate(Numeral, Operation);
15    printf("%6.2f\n",Result);
16    return 0;
17 }
```

Рис. 3.5: main

3. Выполнена компиляция программы посредством gcc.

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

Рис. 3.6: компиляция



Рис. 3.7: файлы

4. Исправлены синтаксические ошибки.
5. Создан Makefile.

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

Рис. 3.8: Создан Makefile

6. С помощью gdb выполнили отладку программы calcul: – Запустили отладчик GDB, загрузив в него программу для отладки – Для запуска программы внутри отладчика ввели команду run – Для постраничного (по 9 строк) просмотра исходного код использовали команду list – Для просмотра строк с 12 по 15 основного файла использовали list с параметрами – Для просмотра определённых строк не основного файла использовали list с параметрами – Установили точку останова в файле calculate.c на строке номер 21 – Вывели информацию об имеющихся в проекте точка останова – Запустили программу внутри отладчика и убедились, что программа остановится в момент прохождения точки останова – Отладчик выдаёт информацию, а команда backtrace показывает весь стек вызываемых функций от начала программы до текущего места.

```

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...
(gdb) list
1      #include <stdio.h>
2      #include "calculate.h"
3
4      int
5      main (void)
6      {
7          float Numeral;
8          char Operation[4];
9          float Result;
10         printf("Число: ");
(gdb) list 12,15
12         printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): ");
13         scanf("%s",Operation);
14         Result = Calculate(Numeral,Operation);
15         printf("%6.2f\n",Result);
(gdb) list calculate.c:20,29
20         return(Numeral-SecondNumeral);
21     }
22     else if(strncmp(Operation,"*",1) == 0)
23     {
24         printf("Множитель: ");
25         scanf("%f",&SecondNumeral);
26         return(Numeral*SecondNumeral);
27     }
28     else if(strncmp(Operation,"/",1) == 0)
29     {
(gdb)

```

Рис. 3.9: Создан Makefile

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

```
kazaboltnaya@dk4n62:~/work/os/lab_prog  kazaboltnaya@dk4n62:~/work/os
<https://bugs.gentoo.org/>.
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"...
./calcul: Нет такого файла или каталога.
(gdb) run
Starting program:
No executable file specified.
Use the "file" or "exec-file" command.
(gdb) list
No symbol table is loaded.  Use the "file" command.
(gdb) run
Starting program:
No executable file specified.
Use the "file" or "exec-file" command.
(gdb) run
Starting program:
No executable file specified.
Use the "file" or "exec-file" command.
(gdb) list
No symbol table is loaded.  Use the "file" command.
(gdb) file
No executable file now.
No symbol file now.
(gdb) list
No symbol table is loaded.  Use the "file" command.
(gdb) 
```

Рис. 3.10: Создан Makefile

4 Контрольные вопросы

1. Как получить информацию о возможностях программ gcc, make, gdb и др.?
2. Назовите и дайте краткую характеристику основным этапам разработки приложений в UNIX.
3. Что такое суффикс в контексте языка программирования? Приведите примеры использования.
4. Каково основное назначение компилятора языка C в UNIX?
5. Для чего предназначена утилита make?
6. Приведите пример структуры Makefile. Дайте характеристику основным элементам этого файла.
7. Назовите основное свойство, присущее всем программам отладки. Что необходимо сделать, чтобы его можно было использовать?
8. Назовите и дайте основную характеристику основным командам отладчика gdb.
9. Опишите по шагам схему отладки программы, которую Вы использовали при выполнении лабораторной работы.
10. Прокомментируйте реакцию компилятора на синтаксические ошибки в программе при его первом запуске.
11. Назовите основные средства, повышающие понимание исходного кода программы.
12. Каковы основные задачи, решаемые программой splint?

4.1 Ответы на контрольные вопросы

1. Информацию об 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. 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.

7. Пошаговая отладка программ заключается в том, что выполняется один оператор программы и, затем контролируются те переменные, на которые должен был воздействовать данный оператор. Если в программе имеются уже отлаженные подпрограммы, то подпрограмму можно рассматривать, как один оператор программы и воспользоваться вторым способом отладки программ. Если в программе существует достаточно большой участок программы, уже отлаженный ранее, то его можно выполнить, не контролируя переменные, на которые он воздействует. Использование точек останова позволяет пропускать уже отлаженную часть программы. Точка останова устанавливается в местах, где необходимо проверить содержимое переменных или просто проконтролировать, передаётся ли управление данному оператору. Практически во всех отладчиках поддерживается это свойство (а также выполнение программы до курсора и выход из подпрограммы). Затем отладка программы продолжается в пошаговом режиме с контролем локальных и глобальных переменных, а также внутренних регистров микроконтроллера и напряжений на выводах этой микросхемы.
8. `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

9. 1) Выполнила компиляцию программы 2) Увидела ошибки в программе 3) Открыла редактор и исправила программу 4) Загрузила программу в отладчик `gdb` 5) `run` — отладчик выполнил программу, ввела требуемые значения. 6) Использовала другие команды отладчика и проверила работу программы
10. 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 использованных в программе 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.

5 Выводы

В ходе выполнения данной лабораторной работы мы приобрели простейшие навыки разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linux на примере создания на языке программирования с калькулятора с простейшими функциями.

Список литературы