

Лабораторная работа №13

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

Козлов Всеволод Павлович

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
3	Ответы на контрольные вопросы	14
4	Выводы	19
	Список литературы	20

Список иллюстраций

2.1	calculate.c	6
2.2	calculate.h	7
2.3	main.c	7
2.4	Makefile	8
2.5	Создание объектных файлов	9
2.6	Запуск программы в отладчике	9
2.7	Команда list	10
2.8	Команда list	11
2.9	Установка точки останова	11
2.10	Запуск программы	12
2.11	Удаление точки останова	13

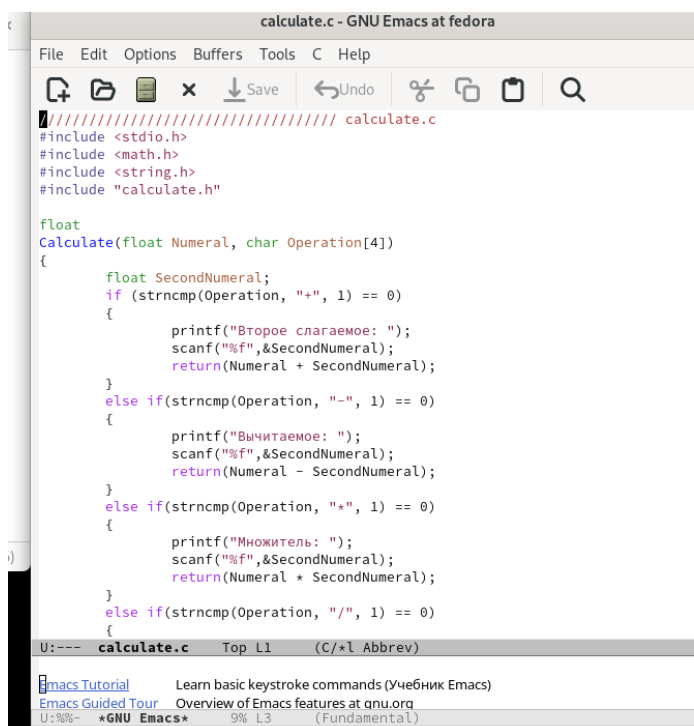
Список таблиц

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 Выполнение лабораторной работы

Написал программу calculate.c (рис. [2.1])



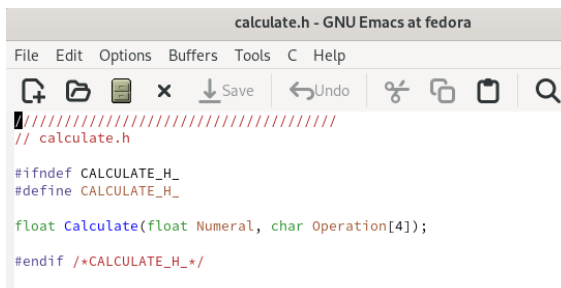
```
calculate.c - GNU Emacs at fedora
File Edit Options Buffers Tools C Help
[Icons: Open, Save, Undo, Cut, Copy, Paste, Find]
// calculate.c
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "calculate.h"

float
Calculate(float Numeral, char Operation[4])
{
    float SecondNumeral;
    if (strcmp(Operation, "+") == 0)
    {
        printf("Второе слагаемое: ");
        scanf("%f", &SecondNumeral);
        return(Numeral + SecondNumeral);
    }
    else if(strcmp(Operation, "-") == 0)
    {
        printf("Вычитаемое: ");
        scanf("%f", &SecondNumeral);
        return(Numeral - SecondNumeral);
    }
    else if(strcmp(Operation, "*") == 0)
    {
        printf("Множитель: ");
        scanf("%f", &SecondNumeral);
        return(Numeral * SecondNumeral);
    }
    else if(strcmp(Operation, "/" == 0)
    {
        printf("Делитель: ");
        scanf("%f", &SecondNumeral);
        return(Numeral / SecondNumeral);
    }
}

U:--- calculate.c Top L1 (C/*l Abbrev)
Emacs Tutorial Learn basic keystroke commands (Учебник Emacs)
Emacs Guided Tour Overview of Emacs features at gnu.org
U:%%- *GNU Emacs* 9% L3 (Fundamental)
```

Рис. 2.1: calculate.c

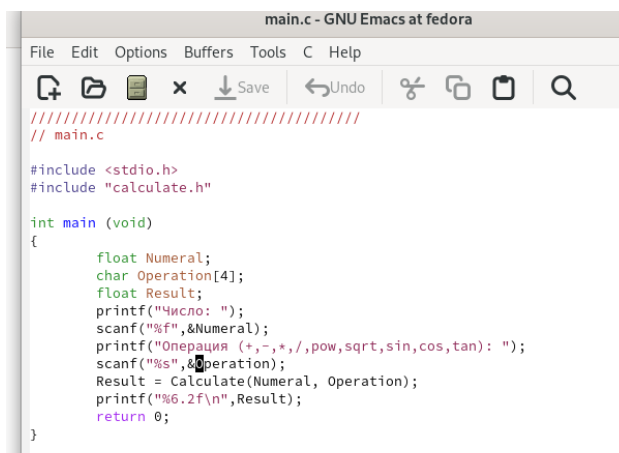
Написал программу calculate.h (рис. [2.2])



```
calculate.h - GNU Emacs at fedora
File Edit Options Buffers Tools C Help
[Icons: Open, Save, Undo, Redo, Search]
// calculate.h
#ifndef CALCULATE_H_
#define CALCULATE_H_
float Calculate(float Numeral, char Operation[4]);
#endif /*CALCULATE_H_*/
```

Рис. 2.2: calculate.h

Написал программу main.c (рис. [2.3])



```
main.c - GNU Emacs at fedora
File Edit Options Buffers Tools C Help
[Icons: Open, Save, Undo, Redo, Search]
// 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. 2.3: main.c

Написал программу Makefile (рис. [2.4])

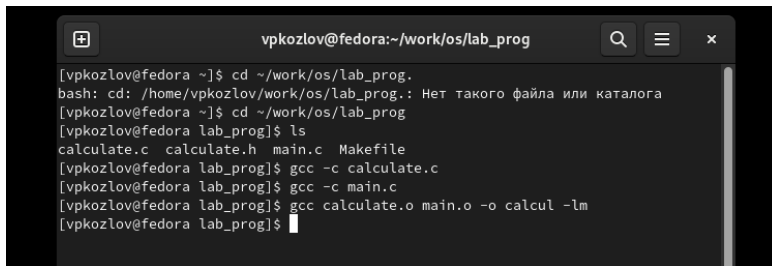


The screenshot shows a text editor window titled "Makefile" with the path "~/work/os/lab_prog". The editor contains the following text:

```
1 #
2 # Makefile
3 #
4
5 CC=gcc
6 CFLAGS=-g
7 LIBS=-lm
8
9 calcul: calculate.o main.o
10     $(CC) calculate.o main.o -o calcul $(LIBS)
11
12 calculate.o: calculate.c calculate.h
13     $(CC) -c calculate.c $(CFLAGS)
14
15 main.o: main.c calculate.h
16     $(CC) -c main.c $(CFLAGS)
17
18 clean:
19     -rm calcul *.o
20
21 # End Makefile
22
```

Рис. 2.4: Makefile

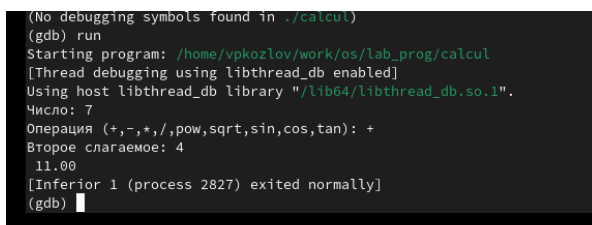
Создал каталог, файлы и создал объектные файлы (рис. [2.5])



```
vpkozlov@fedora:~/work/os/lab_prog
[vpkozlov@fedora ~]$ cd ~/work/os/lab_prog.
bash: cd: /home/vpkozlov/work/os/lab_prog.: Нет такого файла или каталога
[vpkozlov@fedora ~]$ cd ~/work/os/lab_prog
[vpkozlov@fedora lab_prog]$ ls
calculate.c calculate.h main.c Makefile
[vpkozlov@fedora lab_prog]$ gcc -c calculate.c
[vpkozlov@fedora lab_prog]$ gcc -c main.c
[vpkozlov@fedora lab_prog]$ gcc calculate.o main.o -o calcul -lm
[vpkozlov@fedora lab_prog]$
```

Рис. 2.5: Создание объектных файлов

Открыл дебаггер и запустил программу (рис. [2.6])



```
(No debugging symbols found in ./calcul)
(gdb) run
Starting program: /home/vpkozlov/work/os/lab_prog/calcul
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".
Число: 7
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): +
Второе слагаемое: 4
11.00
[Inferior 1 (process 2827) exited normally]
(gdb)
```

Рис. 2.6: Запуск программы в отладчике

Просмотрел строки кода с помощью команды list (рис. [2.7])

```
[Inferior 1 (process 3301) exited normally]
(gdb) list
1  ///////////////////////////////////////////////////
2  // main.c
3
4  #include <stdio.h>
5  #include "calculate.h"
6
7  int main (void)
8  {
9      float Numeral;
10     char Operation[4];
(gdb) list 12, 15
12     printf("Число: ");
13     scanf("%f",&Numeral);
14     printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): ");
15     scanf("%s",&Operation);
(gdb) █
```

Рис. 2.7: Команда list

Просмотрел строки кода с помощью команды list (рис. [2.8])

```

29         scanf("%d", &operation);
(gdb) list calculate.c:20,29
20         scanf("%f", &SecondNumeral);
21         return (Numeral - SecondNumeral);
22     }
23     else if (strcmp (Operation, "+") == 0)
24     {
25         printf("Сложение: ");
26         scanf("%f", &SecondNumeral);
27         return (Numeral + SecondNumeral);
28     }
29     else if (strcmp (Operation, "/") == 0)
(gdb)

```

Рис. 2.8: Команда list

Установил точку останова (рис. [2.9])

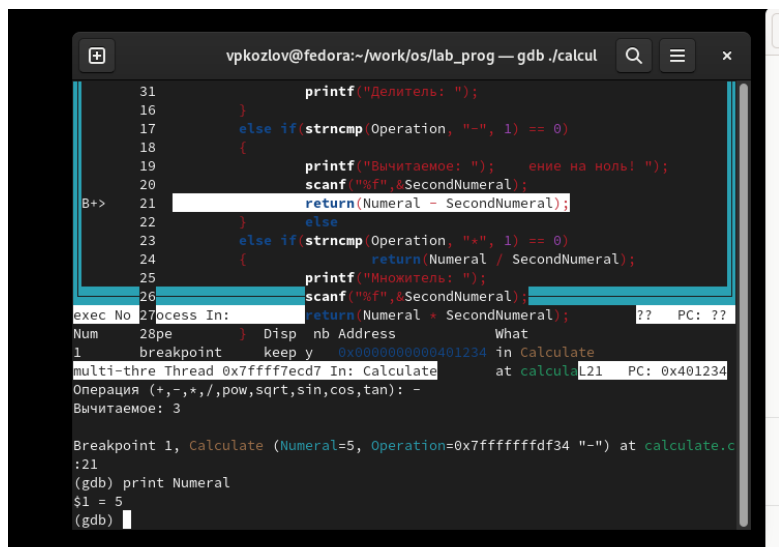
```

(gdb) break 21
Breakpoint 1 at 0x401234: file calculate.c, line 21.
(gdb) info breakpoints
Num   Type             Disp Enb Address            What
1     breakpoint      keep y   0x0000000000401234 in calculate
                                           at calculate.c:21
(gdb)

```

Рис. 2.9: Установка точки останова

Запуск программы с точкой останова (рис. [2.10])



```
vpkozlov@fedora:~/work/os/lab_prog — gdb ./calcul
31         printf("Делитель: ");
16     }
17     else if(strcmp(Operation, "-") == 0)
18     {
19         printf("Вычитаемое: ");   ение на ноль! ");
20         scanf("%f",&SecondNumeral);
B+> 21         return(Numeral - SecondNumeral);
22     }
23     else if(strcmp(Operation, "*") == 0)
24     {
25         printf("Множитель: ");
26         scanf("%f",&SecondNumeral);
27         return(Numeral * SecondNumeral);
?? PC: ??
Num 28pe  ) Disp nb Address What
1 breakpoint keep y 0x0000000000401234 in Calculate
multi-thre Thread 0x7ffff7ecd7 In: Calculate at calculate.L21 PC: 0x401234
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): -
Вычитаемое: 3

Breakpoint 1, Calculate (Numeral=5, Operation=0x7ffff7f34 "-") at calculate.c
:21
(gdb) print Numeral
$1 = 5
(gdb)
```

Рис. 2.10: Запуск программы

Удаление точки останова (рис. [2.11])

```
multi-tnre inread 0x/TTTT/ecd/ in: Calculate at calculate.c:21 PC: 0x401234
1: Numeral = 5
(gdb) info breakpoints
Num Type Disp Enb Address What
1 breakpoint keep y 0x0000000000401234 in Calculate
at calculate.c:21
breakpoint already hit 1 time
(gdb) delete 1
(gdb)
```

Рис. 2.11: Удаление точки останова

3 Ответы на контрольные вопросы

1. Как получить информацию о возможностях программ gcc, make, gdb и др.?
Дополнительную информацию о этих программах можно получить с помощью функций info и man.
2. Назовите и дайте краткую характеристику основным этапам разработки приложений в UNIX. 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. Что такое суффикс в контексте языка программирования? Приведите примеры использования. 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. Каково основное назначение компилятора языка C в UNIX? Основное назначение компилятора с языка Си заключается в компиляции всей программы в целом и получении исполняемого модуля.
5. Для чего предназначена утилита `make`? 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. Приведите пример структуры `Makefile`. Дайте характеристику основным элементам этого файла. `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
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 ко-

мандами 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. Назовите основное свойство, присущее всем программам отладки. Что необходимо сделать, чтобы его можно было использовать? 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 вы-

полнение 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. Назовите и дайте основную характеристику основным командам отладчика gdb. • `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. Опишите по шагам схему отладки программы, которую Вы использовали при выполнении лабораторной работы. Выполнили компиляцию программы 2) Увидели ошибки в программе Открыли редактор и исправили

программу Загрузили программу в отладчик `gdb run` — отладчик выполнил программу, мы ввели требуемые значения. программа завершена, `gdb` не видит ошибок.

10. Прокомментируйте реакцию компилятора на синтаксические ошибки в программе при его первом запуске. Отладчику не понравился формат `%s` для `&Operation`, т.к `%s` — символьный формат, а значит необходим только `Operation`.
11. Назовите основные средства, повышающие понимание исходного кода программы. 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. Каковы основные задачи, решаемые программой `splint`?
13. Проверка корректности задания аргументов всех исполняемых функций , а также типов возвращаемых ими значений;
14. Поиск фрагментов исходного текста, корректных с точки зрения синтаксиса языка Си, но малоэффективных с точки зрения их реализации или содержащих в себе семантические ошибки;
15. Общая оценка мобильности пользовательской программы.

4 Выводы

Приобрел 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