

Отчёт по лабораторной работе № 14

Операционные системы

Ильина Любовь Александровна

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
3	Выводы	18

Список иллюстраций

2.1	Создание каталога, файлов <code>calculate.h</code> , <code>calculate.c</code> , <code>main.c</code>	6
2.2	Компиляция <code>calculate.h</code> , <code>calculate.c</code> , <code>main.c</code>	6
2.3	Создание <code>Makefile</code>	7
2.4	Исправлены синтаксические ошибки в <code>Makefile</code>	7
2.5	Запуск <code>Makefile</code>	8
2.6	Запуск GDB	8
2.7	Запуск программы	8
2.8	Просмотр кода командой <code>list</code>	9
2.9	Просмотр определенных строк кода	9
2.10	Просмотр определенных строк кода	9
2.11	Просмотр определенных строк кода и установка <code>break</code>	10
2.12	Просмотр точек останова	10
2.13	Запуск до останова	11
2.14	Стек вызываемых функций	11
2.15	Просмотр промежуточного значения переменной <code>Numeral</code>	11
2.16	Удаление точек останова	12

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

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. (рис. 2.1).

```
[lailjina@lailjina ~]$ cd ~/work/study/2022-2023/"Операции"
[lailjina@lailjina os-intro]$ mkdir labs/lab14/lab_prog
[lailjina@lailjina os-intro]$ cd labs/lab14/lab_prog
[lailjina@lailjina lab_prog]$ vim calculate.c
[lailjina@lailjina lab_prog]$ chmod u+x calculate.c
[lailjina@lailjina lab_prog]$ vim calculate.h
[lailjina@lailjina lab_prog]$ u+x calculate.h
bash: u+x: command not found...
[lailjina@lailjina lab_prog]$ chmod u+x calculate.h
[lailjina@lailjina lab_prog]$ vim main.c
[lailjina@lailjina lab_prog]$ chmod u+x main.c
[lailjina@lailjina lab_prog]$ ls
calculate.c calculate.h main.c
[lailjina@lailjina lab_prog]$
```

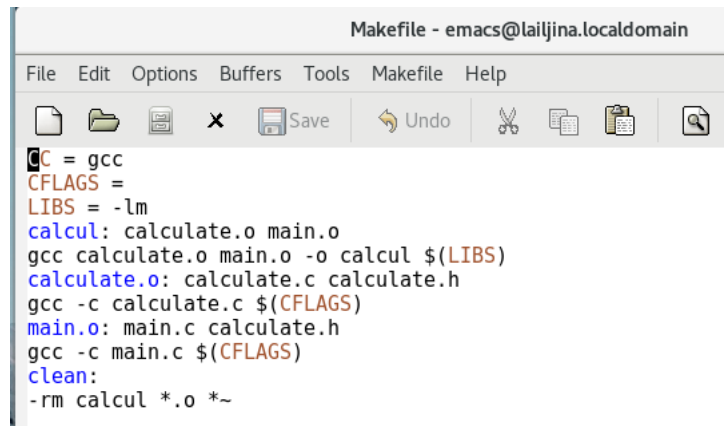
Рис. 2.1: Создание каталога, файлов `calculate.h`, `calculate.c`, `main.c`

3. Выполните компиляцию программы посредством `gcc` (рис. 2.2): `gcc -c calculate.c gcc -c main.c gcc calculate.o main.o -o calcul -lm`

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

Рис. 2.2: Компиляция `calculate.h`, `calculate.c`, `main.c`

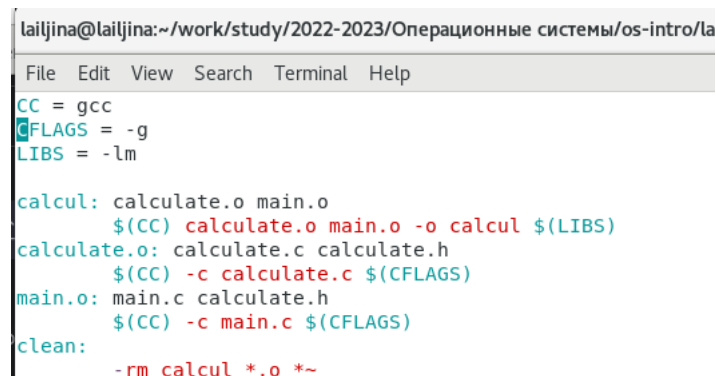
4. Создайте Makefile со следующим содержанием (рис. 2.3): CC = gcc CFLAGS = LIBS = -lm calcul: calculate.o main.o gcc calculate.o main.o -o calcul \$(LIBS) calculate.o: calculate.c calculate.h gcc -c calculate.c \$(CFLAGS) main.o: main.c calculate.h gcc -c main.c \$(CFLAGS) clean: -rm calcul .o ~



```
Makefile - emacs@lailjina.localdomain
File Edit Options Buffers Tools Makefile Help
[Icons] Save Undo [Icons]
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 *~
```

Рис. 2.3: Создание Makefile

5. При необходимости исправьте синтаксические ошибки. (рис. 2.4 - 2.5)



```
lailjina@lailjina:~/work/study/2022-2023/Операционные системы/os-intro/la
File Edit View Search Terminal Help
CC = gcc
CFLAGS = -g
LIBS = -lm

calcul: calculate.o main.o
$(CC) calculate.o main.o -o calcul $(LIBS)
calculate.o: calculate.c calculate.h
$(CC) -c calculate.c $(CFLAGS)
main.o: main.c calculate.h
$(CC) -c main.c $(CFLAGS)
clean:
-rm calcul *.o *~
```

Рис. 2.4: Исправлены синтаксические ошибки в Makefile

```
[lailjina@lailjina lab_prog]$ make
make: `calcul' is up to date.
[lailjina@lailjina lab_prog]$ make clean
rm calcul *.o *~
rm: cannot remove `*~': No such file or directory
make: [clean] Error 1 (ignored)
[lailjina@lailjina lab_prog]$ make
gcc -c calculate.c -g
gcc -c main.c -g
gcc calculate.o main.o -o calcul -lm
```

Рис. 2.5: Запуск Makefile

6. С помощью gdb выполните отладку программы calcul (перед использованием gdb исправьте Makefile): – Запустите отладчик GDB, загрузив в него программу для отладки: `gdb ./calcul` (рис. 2.6)

```
[lailjina@lailjina lab_prog]$ gdb ./calcul
GNU gdb (GDB) Red Hat Enterprise Linux 7.6.1-120.el7
Copyright (C) 2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/lailjina/work/study/2022-2023/Операционные системы/os-
intro/labs/lab14/lab_prog/calcul...done.
(gdb)
```

Рис. 2.6: Запуск GDB

- Для запуска программы внутри отладчика введите команду `run: run` (рис. 2.7)

```
(gdb) run
Starting program: /home/lailjina/work/study/2022-2023/Операционные системы/os-in
tro/labs/lab14/lab_prog/./calcul
Число: 8
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): -
Вычитаемое: 7
1.00
[Inferior 1 (process 4534) exited normally]
Missing separate debuginfos, use: debuginfo-install glibc-2.17-326.el7_9.x86_64
(gdb)
```

Рис. 2.7: Запуск программы

- Для постраничного (по 9 строк) просмотра исходного код используйте ко-манду `list: list` (рис. 2.8)


```

lailjina@lailjina:~/work/study/2022-2023/Операционные системы/os-intro/labs/lab14/lab.
File Edit View Search Terminal Help
Missing separate debuginfos, use: debuginfo-install glibc-2.17-326.e
(gdb) list
1      #include <stdio.h>
2      #include "calculate.h"
3      int
4      main (void)
5      {
6          float Numeral;
7          char Operation[4];
8          float Result;
9          printf("Число: ");
10         scanf("%f",&Numeral);

```

Рис. 2.8: Просмотр кода командой list

– Для просмотра строк с 12 по 15 основного файла используйте list с параметрами: list 12,15 (рис. 2.9)

```

lailjina@lailjina:~/work/study/2022-2023/Операционные системы/os-
File Edit View Search Terminal Help
Line number 17 out of range; main.c has 16 lines.
(gdb) list 12, 15
12         scanf("%s",&Operation);
13         Result = Calculate(Numeral, Operation);
14         printf("%.2f\n",Result);
15         return 0;

```

Рис. 2.9: Просмотр определенных строк кода

– Для просмотра определённых строк не основного файла используйте list с параметрами: list calculate.c:20,29 (рис. 2.10)

```

lailjina@lailjina:~/work/study/2022-2023/Операционные системы/os-
File Edit View Search Terminal Help
(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     }
27     else if(strncmp(Operation, "/", 1) == 0)
28     {
29         printf("Делитель: ");

```

Рис. 2.10: Просмотр определенных строк кода

– Установите точку останова в файле calculate.c на строке номер 21: list calculate.c:20,27 break 21 (рис. 2.11)

```
lailjina@lailjina:~/work/study/2022-2023/Операционные системы/os-intro/labs/lab14/lab_...
File Edit View Search Terminal Help
20 printf("Делитель: ");
(gdb) list calculate.c:20,27
20     }
21     else if(strncmp(operation, "*", 1) == 0)
22     {
23         printf("Множитель: ");
24         scanf("%f",&SecondNumeral);
25         return(Numeral * SecondNumeral);
26     }
27     else if(strncmp(operation, "/", 1) == 0)
(gdb) break 21
Breakpoint 1 at 0x400810: file calculate.c, line 21.
```

Рис. 2.11: Просмотр определенных строк кода и установка break

– Выведите информацию об имеющихся в проекте точка останова: info breakpoints (рис. 2.12)

```
lailjina@lailjina:~/work/study/2022-2023/Операционные системы/os-intro/labs/lab14/lab_...
File Edit View Search Terminal Help
(gdb) info breakpoints
Num      Type             Disp Enb Address                  What
1        breakpoint       keep y   0x0000000000400810      in Calculate
                                at calculate.c:21
```

Рис. 2.12: Просмотр точек останова

– Запустите программу внутри отладчика и убедитесь, что программа остановится в момент прохождения точки останова: run с числом 5(рис. 2.13)

```
lailjina@lailjina:~/work/study/2022-2023/Операционные системы/os-intro/labs/lab14/lab_...
File Edit View Search Terminal Help
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) y

Starting program: /home/lailjina/work/study/2022-2023/Операционные сист
tro/labs/lab14/lab_prog/./calcul
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): *
Breakpoint 1, Calculate (Numeral=5, Operation=0x7fffffffdbb0 "**")
at calculate.c:21
```

Рис. 2.13: Запуск до останова

- backtrace (рис. 2.14) – Отладчик выдаст следующую информацию: #0 Calculate (Numeral=5, Operation=0x7fffffffdbb0 "-") at calculate.c:21 #1 0x0000000000400b2b in main () at main.c:17

```
lailjina@lailjina:~/work/study/2022-2023/Операционные системы/os-intro/labs/lab14/lab_...
File Edit View Search Terminal Help
(gdb) backtrace
#0 Calculate (Numeral=5, Operation=0x7fffffffdbb0 "**") at calculate.c:21
#1 0x0000000000400a90 in main () at main.c:13
```

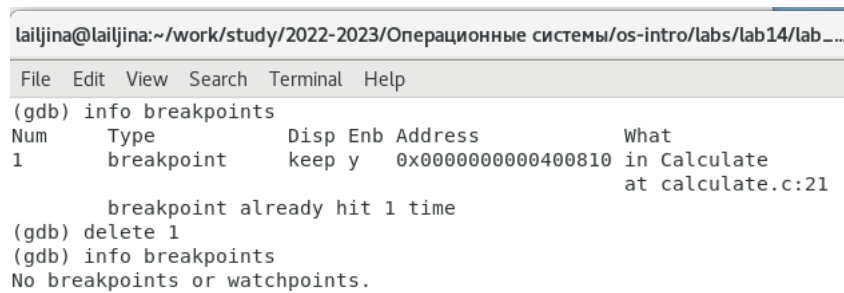
Рис. 2.14: Стек вызываемых функций

а команда backtrace покажет весь стек вызываемых функций от начала программы до текущего места. – Посмотрите, чему равно на этом этапе значение переменной Numeral, введя: print Numeral На экран должно быть выведено число 5. – Сравните с результатом вывода на экран после использования команды: display Numeral (рис. 2.15)

```
lailjina@lailjina:~/work/study/2022-2023/Операционные системы/os-intro/labs/lab14/lab_...
File Edit View Search Terminal Help
(gdb) print Numeral
$2 = 5
(gdb) display Numeral
1: Numeral = 5
```

Рис. 2.15: Просмотр промежуточного значения переменной Numeral

– Уберите точки останова: info breakpoints delete 1(рис. 2.16)



```
lailjina@lailjina:~/work/study/2022-2023/Операционные системы/os-intro/labs/lab14/lab_...
File Edit View Search Terminal Help
(gdb) info breakpoints
Num      Type      Disp Enb Address      What
1        breakpoint keep y  0x0000000000400810 in Calculate
                                     at calculate.c:21
        breakpoint already hit 1 time
(gdb) delete 1
(gdb) info breakpoints
No breakpoints or watchpoints.
```

Рис. 2.16: Удаление точек останова

1. Как получить информацию о возможностях программ gcc, make, gdb и др.?
Дополнительную информацию об 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 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`. Вторым способом 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 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. 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`?

Проверка корректности задания аргументов всех исполняемых функций , а также типов возвращаемых ими значений;

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

Общая оценка мобильности пользовательской программы.

3 Выводы

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