

Отчет по лабораторной работе №13

Дисциплина Операционные системы

Колобова Елизавета, гр. НММбд-01-22

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
3	Контрольные вопросы	12
	Список литературы	16

Список иллюстраций

2.1	Рис. 1. Отредактированный Makefile	7
2.2	Рис. 2. Компиляция программы и запуск отладчика	8
2.3	Рис. 3. Работа с отладчиком	9
2.4	Рис. 4. Работа с отладчиком	10
2.5	Рис. 5. Анализ calculate.c с помощью splint	11
2.6	Рис. 6. Анализ main.c с помощью splint	11

Список таблиц

1 Цель работы

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 создадим подкаталог `~/work/os/lab_prog`
2. Создадим в нём файлы: `calculate.h`, `calculate.c`, `main.c`.
3. Заполним созданные файлы согласно материалам лабораторной работы. Реализация функций калькулятора представлена в файле `calculate.c`, основной файл `main.c` реализует интерфейс пользователя к калькулятору. Синтаксические ошибки утилитой `ыздште` не выявляются, так что их там, скорее всего, нет
4. Создадим `Makefile` с содержанием, указанным в материалах лабораторной работы. Основное в содержании файла - это указанный в нем порядок компиляции исполняемых файлов. Сразу отредактируем файл, переместив блок с компиляцией программы `calcul` ниже всех остальных, сразу перед блоком `clean`, т.к. для компиляции итоговой программы требуются файлы, которые создаются в блоках выше. (рис. [2.1]))



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

Рис. 2.1: Рис. 1. Отредактированный Makefile

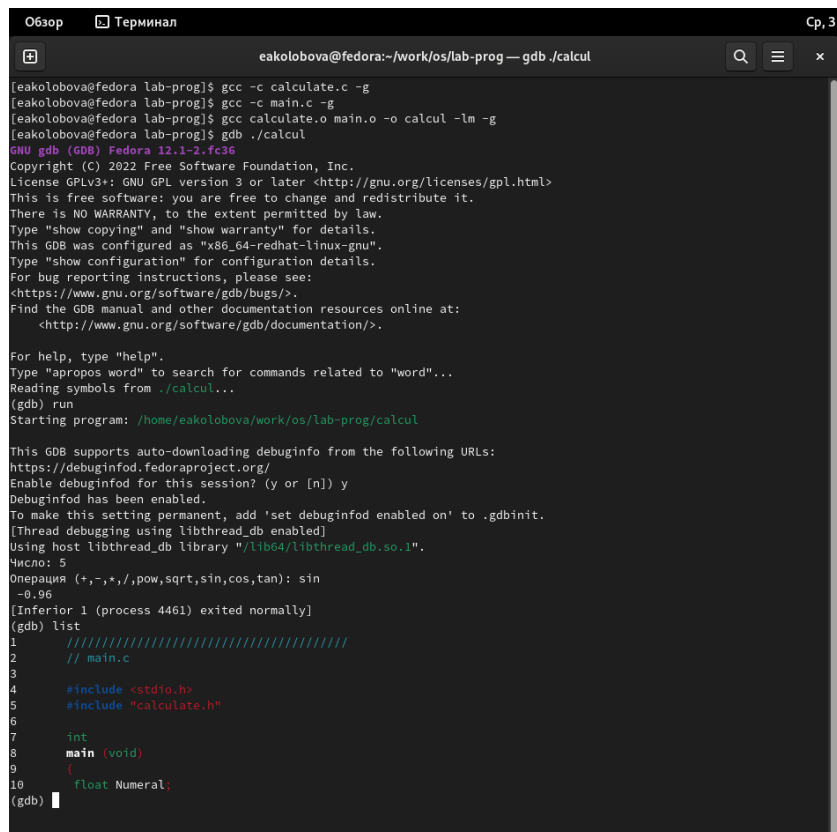
5. Выполним компиляцию программы посредством gcc, как указано в лабораторной работе, с добавлением ключа -g, чтобы далее программу можно было загрузить в отладчик без ошибок. (рис. [2.2])
6. С помощью gdb выполним отладку программы calcul и выполним следующие действия: (рис. [2.2], [2.3], [2.4]) – Запустим отладчик GDB, загрузив в него программу для отладки – Для запуска программы внутри отладчика введем команду run – Для постраничного (по 9 строк) просмотра исходного код используем команду list – Для просмотра строк с 12 по 15 основного файла используем list с параметрами – Для просмотра определённых строк не основного файла используем list с параметрами: calculate.c:20,29 – Установим точку останова в файле calculate.c на строке номер 21 – Выведем информацию об имеющихся в проекте точках останова – Запустим программу внутри отладчика и убедимся, что программа остановится в момент прохождения точки останова – Отладчик выдаст следующую информацию:

1 #0 Calculate (Numeral=5, Operation=0x7fffffffdd280 "-")

2 at calculate.c:21

3 #1 0x0000000000400b2b in main () at main.c:17

а команда `backtrace` покажет весь стек вызываемых функций от начала программы до текущего места. – Посмотрим, чему равно на этом этапе значение переменной `Numeral` На экран должно быть выведено число 5, оно и выводится – Сравним с результатом вывода на экран после использования команды 1 `display Numeral` Результат один и тот же – Уберем точки останова



```
Обзор Терминал
eakolobova@fedora:~/work/os/lab-prog — gdb ./calcul

[eakolobova@fedora lab-prog]$ gcc -c calculate.c -g
[eakolobova@fedora lab-prog]$ gcc -c main.c -g
[eakolobova@fedora lab-prog]$ gcc calculate.o main.o -o calcul -lm -g
[eakolobova@fedora lab-prog]$ gdb ./calcul
GNU gdb (GDB) Fedora 12.1-2.fc36
Copyright (C) 2022 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".
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) run
Starting program: /home/eakolobova/work/os/lab-prog/calcul

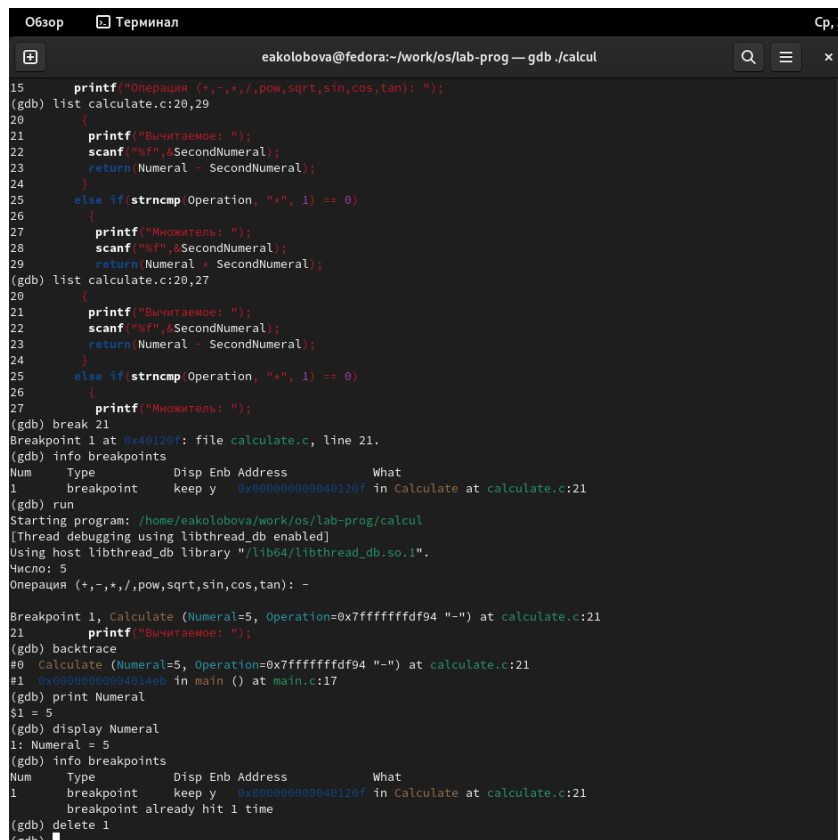
This GDB supports auto-downloading debuginfo from the following URLs:
https://debuginfod.fedoraproject.org/
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): sin
-0.96
[Inferior 1 (process 4461) exited normally]
(gdb) list
1  ///////////////////////////////////////////////////
2  // main.c
3
4  #include <stdio.h>
5  #include "calculate.h"
6
7  int
8  main (void)
9  {
10     float Numeral;
(gdb) █
```

Рис. 2.2: Рис. 2. Компиляция программы и запуск отладчика


```
Обзор Терминал Cp,3
eakolobova@fedora:~/work/os/lab-prog — gdb ./calcul

[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): sin
-0.96
[Inferior 1 (process 4461) exited normally]
(gdb) list
1  ///////////////////////////////////////////////////
2  // main.c
3
4  #include <stdio.h>
5  #include "calculate.h"
6
7  int
8  main (void)
9  {
10     float Numeral;
(gdb) list 12,15
12     float Result;
13     printf("Число: ");
14     scanf("%f",&Numeral);
15     printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): ");
(gdb) list calculate.c:20,29
20     {
21         printf("Вводимое: ");
22         scanf("%f",&SecondNumeral);
23         return(Numeral - SecondNumeral);
24     }
25     else if (strcmp(Operation, "+") == 0)
26     {
27         printf("Умножитель: ");
28         scanf("%f",&SecondNumeral);
29         return(Numeral * SecondNumeral);
(gdb) list calculate.c:20,27
20     {
21         printf("Вводимое: ");
22         scanf("%f",&SecondNumeral);
23         return(Numeral - SecondNumeral);
24     }
25     else if (strcmp(Operation, "+") == 0)
26     {
27         printf("Умножитель: ");
(gdb) break 21
Breakpoint 1 at 0x40120f: file calculate.c, line 21.
(gdb) info breakpoints
Num   Type             Disp Enb Address            What
1     breakpoint       keep y   0x000000000040120f in Calculate at calculate.c:21
(gdb)
```

Рис. 2.3: Рис. 3. Работа с отладчиком



```
Обзор Терминал Cp. 3
eakolobova@fedora:~/work/os/lab-prog — gdb ./calcul

15     printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): ");
(gdb) list calculate.c:20,29
20     {
21         printf("Вычитаемое: ");
22         scanf("%f",&SecondNumeral);
23         return Numeral - SecondNumeral;
24     }
25     else if (strcmp(Operation, "+", 1) == 0)
26     {
27         printf("Умножитель: ");
28         scanf("%f",&SecondNumeral);
29         return Numeral * SecondNumeral;
(gdb) list calculate.c:20,27
20     {
21         printf("Вычитаемое: ");
22         scanf("%f",&SecondNumeral);
23         return Numeral - SecondNumeral;
24     }
25     else if (strcmp(Operation, "+", 1) == 0)
26     {
27         printf("Умножитель: ");
(gdb) break 21
Breakpoint 1 at 0x40120f: file calculate.c, line 21.
(gdb) info breakpoints
Num Type Disp Enb Address What
1 breakpoint keep y 0x0000000040120f in Calculate at calculate.c:21
(gdb) run
Starting program: /home/eakolobova/work/os/lab-prog/calcul
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): -

Breakpoint 1, Calculate (Numeral=5, Operation=0x7fffffffdf94 "-") at calculate.c:21
21     printf("Вычитаемое: ");
(gdb) backtrace
#0 Calculate (Numeral=5, Operation=0x7fffffffdf94 "-") at calculate.c:21
#1 0x0000000040140b in main () at main.c:17
(gdb) print Numeral
$1 = 5
(gdb) display Numeral
1: Numeral = 5
(gdb) info breakpoints
Num Type Disp Enb Address What
1 breakpoint keep y 0x0000000040120f in Calculate at calculate.c:21
breakpoint already hit 1 time
(gdb) delete 1
(gdb)
```

Рис. 2.4: Рис. 4. Работа с отладчиком

7. С помощью утилиты splint попробуем проанализировать коды файлов calculate.c и main.c. В основном выводятся предупреждения о несоответствии типов переменных и возвращаемых значений, игнорировании определенных функций, использовании параметров в кач-ве указателей и прочих сомнительных, но не влияющих на работу программы вещах ([2.5], [2.6])

```
eakolobova@fedora:~/work/os/lab-prog
(gdb) q
A debugging session is active.

    Inferior 1 [process 4554] will be killed.

Quit anyway? (y or n) y
[eakolobova@fedora lab-prog]$ splint calculate.c
Splint 3.1.2 --- 22 Jan 2022

calculate.h:7:37: 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)
calculate.c:10:31: Function parameter Operation declared as manifest array
    (size constant is meaningless)
calculate.c: (in function Calculate)
calculate.c:16:4: Return value (type int) ignored: scanf("%f", &Sec...
    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)
calculate.c:22:4: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:28:5: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:34:5: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:35:7: Dangerous equality comparison involving float types:
    SecondNumeral == 0
    Two real (float, double, or long double) values are compared directly using
    == or != primitive. This may produce unexpected results since floating point
    representations are inexact. Instead, compare the difference to FLT_EPSILON
    or DBL_EPSILON. (Use -realcompare to inhibit warning)
calculate.c:38:12: Return value type double does not match declared type float:
    (HUGE_VAL)
    To allow all numeric types to match, use -relaxtypes.
calculate.c:46:6: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:47:12: Return value type double does not match declared type float:
    (pow(Numeral, SecondNumeral))
calculate.c:50:8: Return value type double does not match declared type float:
    (sqrt(Numeral))
calculate.c:52:8: Return value type double does not match declared type float:
    (sin(Numeral))
calculate.c:54:8: Return value type double does not match declared type float:
    (cos(Numeral))
calculate.c:56:8: Return value type double does not match declared type float:
    (tan(Numeral))
calculate.c:60:8: Return value type double does not match declared type float:
    (HUGE_VAL)
Finished checking --- 15 code warnings
[eakolobova@fedora lab-prog]$
```

Рис. 2.5: Рис. 5. Анализ calculate.c с помощью splint

```
eakolobova@fedora:~/work/os/lab-prog
[eakolobova@fedora lab-prog]$ splint main.c
Splint 3.1.2 --- 22 Jan 2022

calculate.h:7:37: 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:14:2: 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:16:13: Format argument 1 to scanf (%s) expects char * gets char [4] *:
    &Operation
    Type of parameter is not consistent with corresponding code in format string.
    (Use -formattype to inhibit warning)
    main.c:16:10: Corresponding format code
main.c:16:2: Return value (type int) ignored: scanf("%s", &ope...
Finished checking --- 4 code warnings
[eakolobova@fedora lab-prog]$
```

Рис. 2.6: Рис. 6. Анализ main.c с помощью splint

3 Контрольные вопросы

1. Как получить информацию о возможностях программ gcc, make, gdb и др.? запустив их с ключом `-help` или `-h` или просмотрев мануал
2. Назовите и дайте краткую характеристику основным этапам разработки приложений в UNIX. – 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. Что такое суффикс в контексте языка программирования? Приведите примеры использования. Это расширение, указывающее, на каком языке написан файл. 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. Каково основное назначение компилятора языка C в UNIX? gcc по расширению (суффиксу) `.c` распознает тип файла для компиляции и формирует объектный модуль — файл с расширением `.o`.
5. Для чего предназначена утилита make? Она позволяет автоматизировать процесс преобразования файлов программы из одной формы в другую,

отслеживает взаимосвязи между файлами.

6. Приведите пример структуры Makefile. Дайте характеристику основным элементам этого файла. Общий синтаксис Makefile имеет вид:
- ```
1 target1 [target2...]:[:] [dependment1...] 2 [(tab)commands]
[#commentary] 3 [(tab)commands] [#commentary]
```
- Здесь знак # определяет начало комментария (содержимое от знака # и до конца строки не будет обрабатываться. 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:

```
1 #
2 # Makefile for abcd.c
3 #
4
5 CC = gcc
6 CFLAGS =
7
8 # Compile abcd.c normaly
9 abcd: abcd.c
10 $(CC) -o abcd $(CFLAGS) abcd.c
11
12 clean:
13 -rm abcd *.o *~
14
15 # End Makefile for abcd.c
```

В этом примере в начале файла заданы три переменные: CC и CFLAGS. Затем указаны цели, их зависимости и соответствующие команды. В командах происхо-

дит обращение к значениям переменных. Цель с именем `clean` производит очистку каталога от файлов, полученных в результате компиляции. Для её описания использованы регулярные выражения.

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`.

- `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
- `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. Опишите по шагам схему отладки программы, которую Вы использовали при выполнении лабораторной работы.

- Запустим отладчик `GDB`, загрузив в него программу для отладки
- Для запуска программы внутри отладчика введем команду `run`
- Для постраничного (по 9 строк) просмотра исходного кода используем команду `list`
- Для просмотра строк с 12 по 15 основного файла используем `list` с параметрами
- Для просмотра определённых строк не основного файла используем `list` с параметрами: `calculate.c:20,29`
- Установим точку останова в файле `calculate.c` на строке номер 21
- Выведем информацию об имеющихся в проекте точках останова
- Запустим

программу внутри отладчика и убедимся, что программа остановится в момент прохождения точки останова – Отладчик выдаст следующую информацию:

```
1 #0 Calculate (Numeral=5, Operation=0x7fffffff280 "-")
2 at calculate.c:21
3 #1 0x0000000000400b2b in main () at main.c:17
```

а команда `backtrace` покажет весь стек вызываемых функций от начала программы до текущего места. – Посмотрим, чему равно на этом этапе значение переменной `Numeral` На экран должно быть выведено число 5, оно и выводится – Сравним с результатом вывода на экран после использования команды `1 display Numeral` Результат один и тот же – Уберем точки останова

10. Прокомментируйте реакцию компилятора на синтаксические ошибки в программе при его первом запуске. из-за синтаксических ошибок программа не компелится, т.к. компилятор не может распознать содержащие ошибки части кода
11. Назовите основные средства, повышающие понимание исходного кода программы. Для облегчения понимания исходного кода используются комментарии
12. Каковы основные задачи, решаемые программой `splint` Эта утилита анализирует программный код, проверяет корректность задания аргументов использованных в программе функций и типов возвращаемых значений, обнаруживает синтаксические и семантические ошибки. # Выводы

Результатом проделанной работы является Приобретение простейших навыков разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linux на примере создания на языке программирования C калькулятора с простейшими функциями.

## **Список литературы**