

Отчёта по лабораторной работе №13

НКНбь-02-21

Акондзо Жордание Лади Гаэл

Содержание

1	Цель работы	4
2	Задание	5
3	Ход работы	6
4	Выводы	16
5	Контрольные вопросы	17

Список иллюстраций

3.1	Создание подкаталога и файлов	6
3.2	Написание программы 1: calculate.c	7
3.3	Написание программы 1: calculate.h	8
3.4	Написание программы 3: main.c	9
3.5	Компиляция программы 1(calculate.c), программы 2(main.c), программы 3(calculate.o main.o calcul)	10
3.6	Создание Makefile	11
3.7	выполнение отладки программы calcul с помощью gdb	12
3.8	Запуска программы (введение команды run	12
3.9	List 1	13
3.10	List 2	13
3.11	run, print Numeral, display Numeral, info breakpoints и delete 1	14
3.12	Вывод команды утилита splint: calculate.c	15
3.13	Вывод команды утилита splint: main.c	15

1 Цель работы

Цель данной 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. При необходимости исправьте синтаксические ошибки.
4. Создайте `Makefile`.
5. С помощью `gdb` выполните отладку программы `calcul` (перед использованием `gdb` исправьте `Makefile`)
6. С помощью утилиты `splint` попробуйте проанализировать коды файлов `calculate.c` и `main.c`.

3 Ход работы

1. В домашнем каталоге создал подкаталог ~/work/os/lab_prog.
2. Создал в нём файлы: calculate.h, calculate.c, main.c. (рис. 3.1)

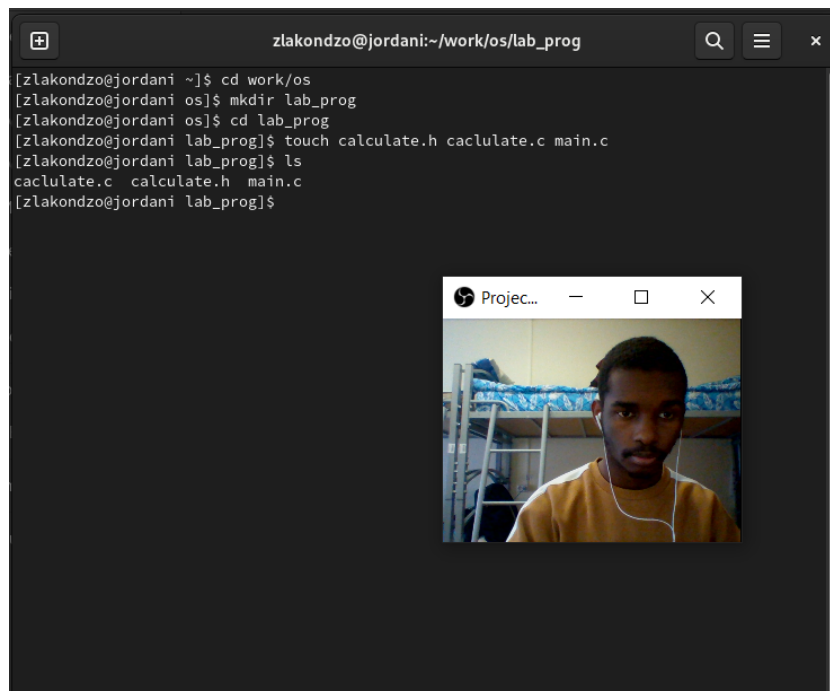


Рис. 3.1: Создание подкаталога и файлов

- Реализация функций калькулятора в файле calculate.c: (рис. 3.2)

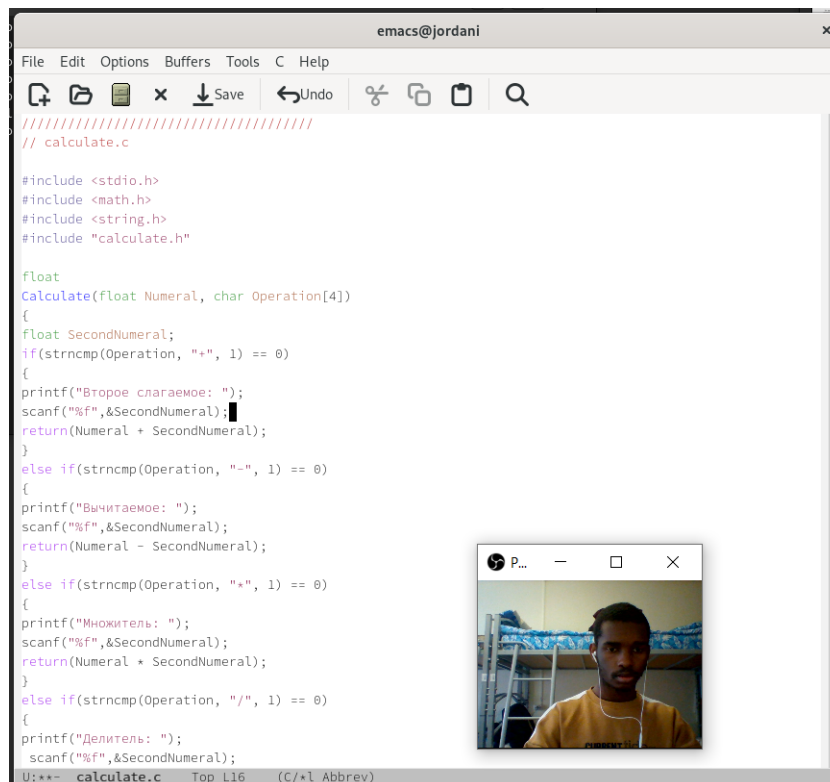


Рис. 3.2: Написание программы 1: calculate.c

- Интерфейсный файл calculate.h, описывающий формат вызова функции-калькулятора: (рис. 3.3)

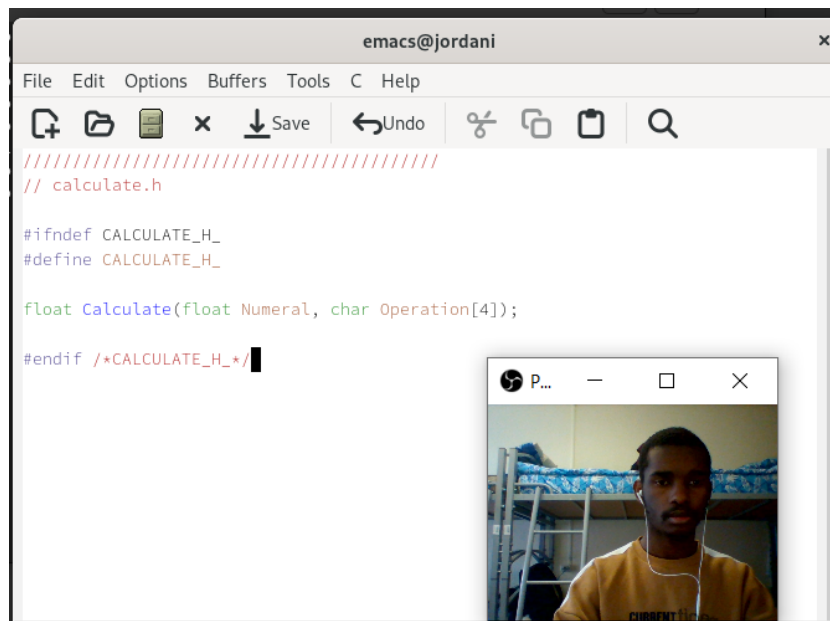


Рис. 3.3: Написание программы 1: calculate.h

- Основной файл main.c, реализующий интерфейс пользователя к калькулятору: (рис. 3.4)

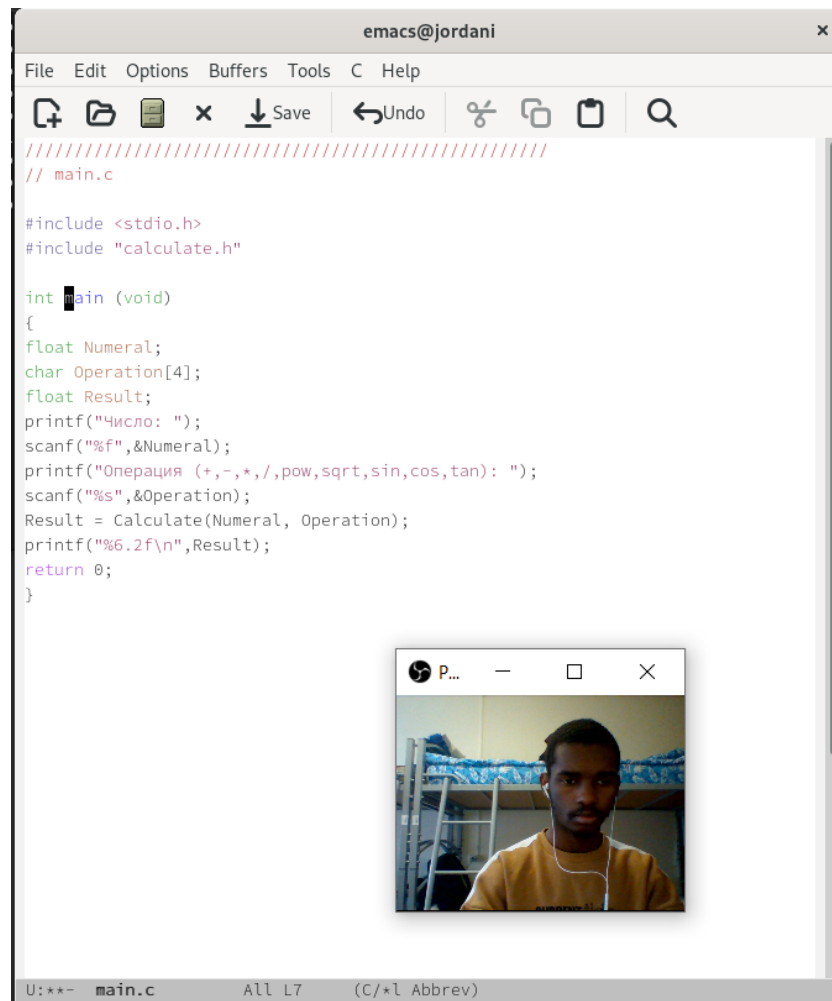


Рис. 3.4: Написание программы 3: main.c

3. Выполнил компиляцию программы посредством gcc: (рис. 3.5)

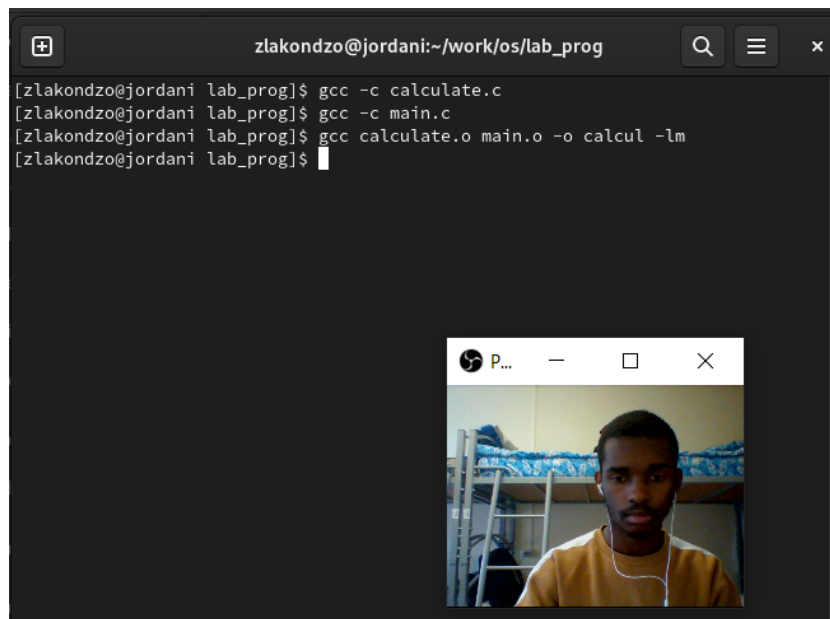


Рис. 3.5: Компиляция программы 1(calculate.c), программы 2(main.c), программы 3(calculate.o main.o calcul)

4. При необходимости исправьте синтаксические ошибки.

Не было ошибки.

5. Создал Makefile со следующим содержанием: (рис. 3.6)

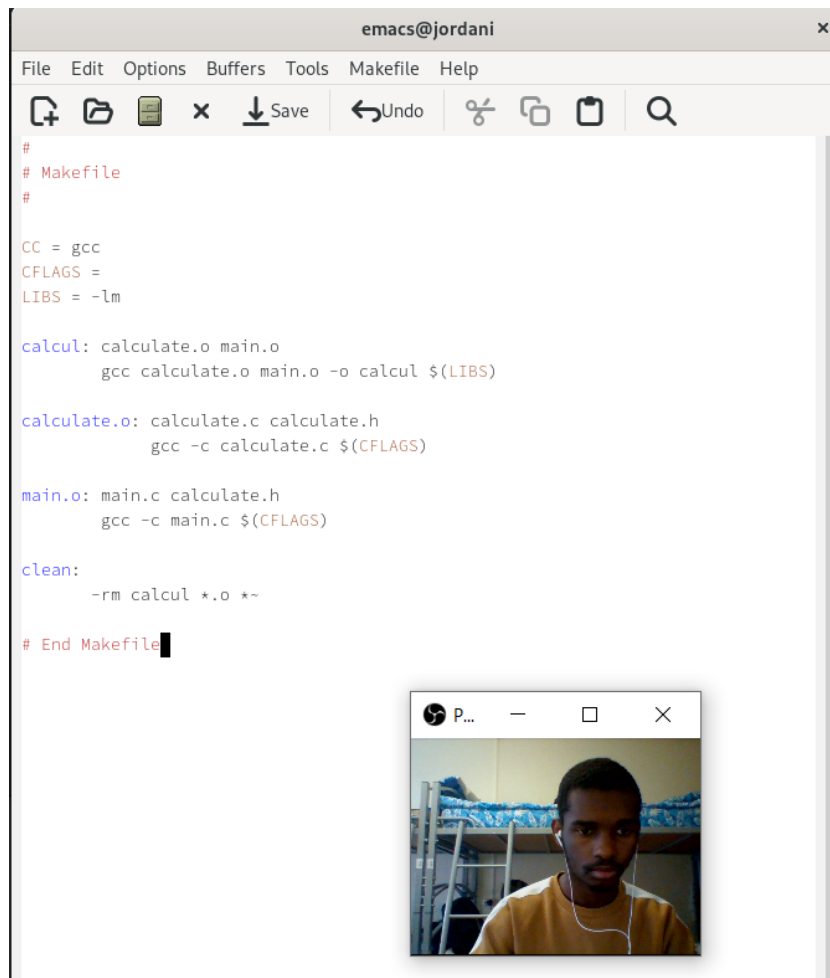


Рис. 3.6: Создание Makefile

6. С помощью gdb выполнил отладку программы calcul (перед использованием gdb исправьте Makefile):
- Запустите отладчик GDB, загрузив в него программу для отладки: `gdb ./calcul`. (рис. 3.7)

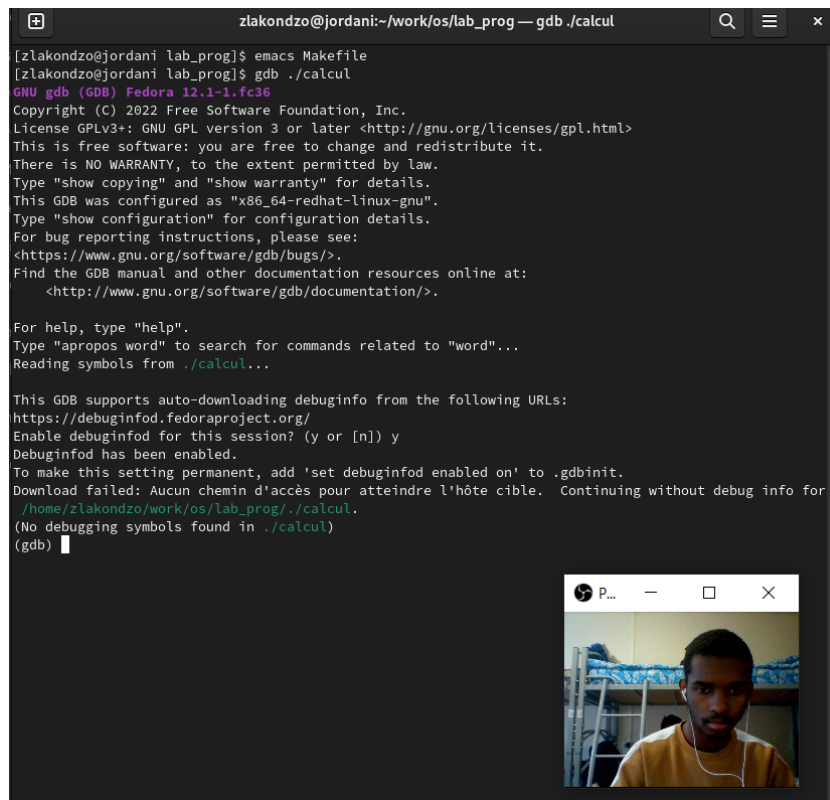


Рис. 3.7: выполнение отладки программы calcul с помощью gdb

- Для запуска программы внутри отладчика введите команду run: (рис. 3.8)

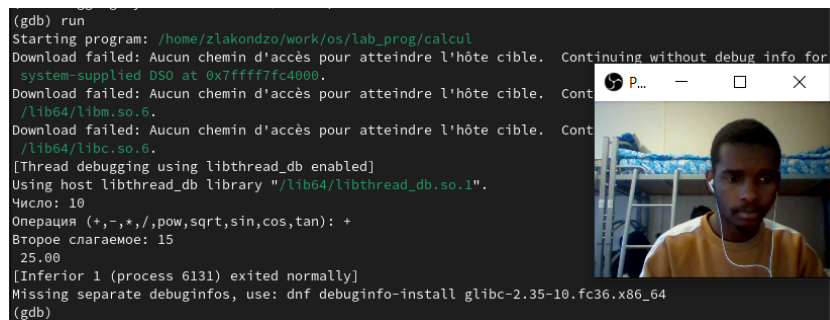


Рис. 3.8: Запуска программы (введение команды run)

- Для постраничного (по 9 строк) просмотра исходного код используйте команду list: list; list 12,15; list calculate.c:20,29; list calculate.c 20,27 break 21; info breakpoints. (рис. 3.9, 3.10)

```

(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
No source file named calculate.c.
(gdb) list main.c:20,29
20 }
(gdb) list main.c:10,20
10 float Numeral;
11 char Operation[4];
12 float Result;
13 printf("Число: ");
14 scanf("%f",&Numeral);
15 printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): ");
16 scanf("%s",&Operation);
17 Result = Calculate(Numeral, Operation);
18 printf("%6.2f\n",Result);
19 return 0;
20 }
(gdb) break 18
Breakpoint 1 at 0x6c: file main.c, line 18.
(gdb) info breakpoints
Num Type Disp Enb Address What
1 breakpoint keep y 0x000000000000006c in main at main.c:18
(gdb)

```

Рис. 3.9: List 1

```

(gdb) list
1 //////////////////////////////////////////////////
2 // calculate.c
3
4 #include <stdio.h>
5 #include <math.h>
6 #include <string.h>
7 #include "calculate.h"
8
9 float
10 Calculate(float Numeral, char Operation[4])
(gdb) list 12,15
12 float SecondNumeral;
13 if (strcmp(Operation, "+", 1) == 0)
14 {
15 printf("Введите второе число: ");
(gdb) list calculate.c:20,27
20
21 printf("Введите второе число: ");
22 scanf("%f",&SecondNumeral);
23 return Numeral + SecondNumeral;
24 }
25 else if (strcmp(Operation, "+", 1) == 0)
26 {
27 printf("Введите второе число: ");
(gdb) break 21
Breakpoint 1 at 0x70: file calculate.c, line 21.
(gdb) info breakpoints
Num Type Disp Enb Address What
1 breakpoint keep y 0x0000000000000070 in Calculate at calculate.c:21
(gdb) run
Starting program: /home/zlakondzo/work/os/lab_prog/calculate.o
/bin/bash: ligne 1: /home/zlakondzo/work/os/lab_prog/calculate.o : impossible d'exécuter le fichier binaire : Erreur de format pour exec()
/bin/bash: ligne 1: /home/zlakondzo/work/os/lab_prog/calculate.o: Succès
During startup program exited with code 126.
(gdb) print Numeral
No symbol "Numeral" in current context.
(gdb) display Numeral
No symbol "Numeral" in current context.
(gdb) quit
[zlakondzo@jordani lab_prog]$ splint calculate.c
bash: splint: commande inconnue...
Voulez-vous installer le paquet « splint » qui fournit la commande « splint » ? [N/y]

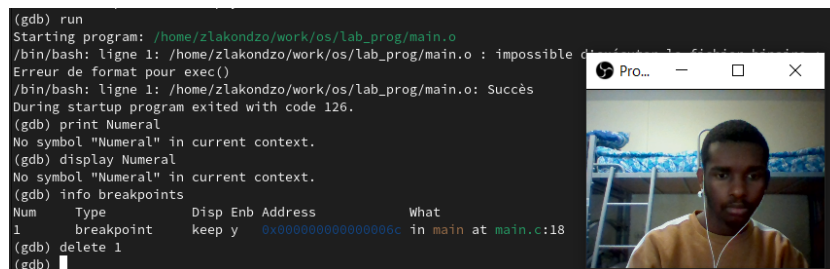
```

Рис. 3.10: List 2

- Запустите программу внутри отладчика и убедитесь, что программа останавливается в момент прохождения точки останова:

- Посмотрите, чему равно на этом этапе значение переменной Numeral, введя:
- Сравните с результатом вывода на экран после использования команды:
- Уберите точки останова:

Все из этих: run, print Numeral, display Numeral, info breakpoints и delete 1. (рис. 3.11)



```
(gdb) run
Starting program: /home/zlakondzo/work/os/lab_prog/main.o
/bin/bash: ligne 1: /home/zlakondzo/work/os/lab_prog/main.o : impossible d'executer le fichier /home/zlakondzo/work/os/lab_prog/main.o : Erreur de format pour exec()
/bin/bash: ligne 1: /home/zlakondzo/work/os/lab_prog/main.o: Succès
During startup program exited with code 126.
(gdb) print Numeral
No symbol "Numeral" in current context.
(gdb) display Numeral
No symbol "Numeral" in current context.
(gdb) info breakpoints
Num   Type      Disp Enb Address            What
1     breakpoint keep y   0x000000000000006c in main at main.c:18
(gdb) delete 1
(gdb)
```

Рис. 3.11: run, print Numeral, display Numeral, info breakpoints и delete 1

7. С помощью утилиты splint попробуйте проанализировать коды файлов calculate.c и main.c. (рис. 3.12, 3.13)

```
zlakondzo@jordani:~/work/os/lab_prog
version (information on compilation, maintainer)

[zzlakondzo@jordani 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:16:1: 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:22:1: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:28:1: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:34:2: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:35:4: 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:38:7: Return value type double does not match declared type float:
(HUGE_VAL)
To allow all numeric types to match, use +relaxtypes.
calculate.c:46:1: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:47:7: Return value type double does not match declared type float:
(pow(Numeral, SecondNumeral))
calculate.c:50:7: Return value type double does not match declared type float:
(sqrt(Numeral))
calculate.c:52:7: Return value type double does not match declared type float:
(sin(Numeral))
calculate.c:54:7: Return value type double does not match declared type float:
(cos(Numeral))
calculate.c:56:7: Return value type double does not match declared type float:
(tan(Numeral))
calculate.c:60:7: Return value type double does not match declared type float:
(HUGE_VAL)

Finished checking --- 15 code warnings
[zzlakondzo@jordani lab_prog]$
```

Рис. 3.12: Вывод команды утилита splint: calculate.c

```
[zzlakondzo@jordani 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:14:1: 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:16:12: 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:16:9: Corresponding format code
main.c:16:1: Return value (type int) ignored: scanf("%s", &Ope...

Finished checking --- 4 code warnings
[zzlakondzo@jordani lab_prog]$
```

Рис. 3.13: Вывод команды утилита splint: main.c

4 Выводы

Во время выполнения 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.

5 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 “.с” 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 .с компилятор распознает, что файл abcd.c должен компилироваться, а по суффиксу .о, 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 `-p` 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. 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 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.

- 1) Выполнили компиляцию программы
- 2) Увидели ошибки в программе
- 3) Открыли редактор и исправили программу
- 4) Загрузили программу в отладчик `gdb`
- 5) `run` — отладчик выполнил программу, мы ввели требуемые значения.
- 6) 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`?