

Лабораторная работа №15

Дисциплина: Операционные системы

Коновалова Татьяна Борисовна

Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	7
4	Библиография	16
5	Выводы	17

List of Tables

List of Figures

3.1	Создание файлов	7
3.2	Прграмма в файле common.h	8
3.3	Прграмма в файле server.c	8
3.4	Прграмма в файле server.c	9
3.5	Прграмма в файле client.c	9
3.6	Прграмма в файле client.c	10
3.7	Прграмма в Mikefile	10
3.8	Команда make all	11
3.9	Команда make all	11
3.10	Проверка длительности работы сервера	12

1 Цель работы

Цель данной лабораторной работы — Приобретение практических навыков работы с именованными каналами.

2 Задание

1. Сделать отчёт по лабораторной работе №15 в формате Markdown.
2. Приобрести практические навыки работы с именованными каналами.

3 Выполнение лабораторной работы

1). Для начала я создала необходимые файлы с помощью команды «touch common.h server.c client.c Makefile» и открыла редактор emacs для их редактирования (Рисунок 3.1). Вся необходимая информация про создания каталогов указана в следующем источнике: Программное обеспечение GNU/Linux. Лекция 10. Минимальный набор знаний (Г. Курячий, МГУ)

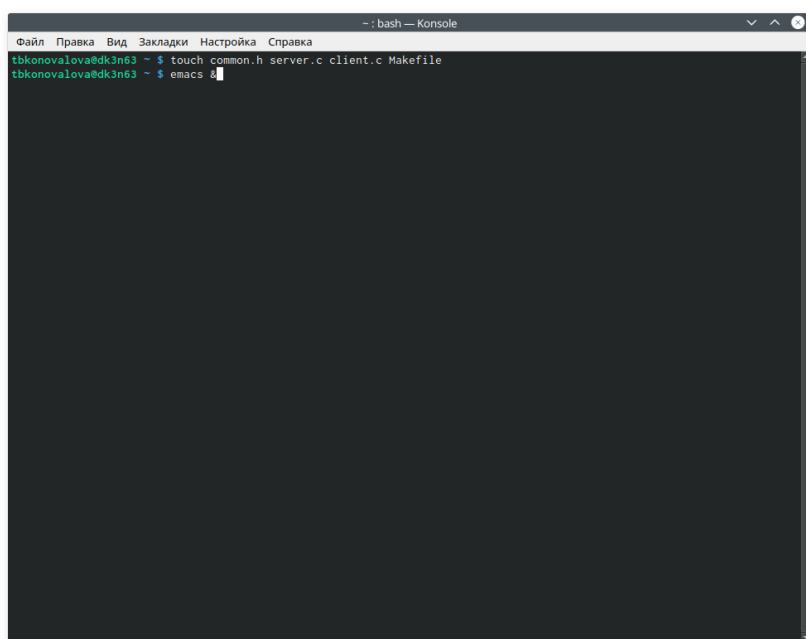
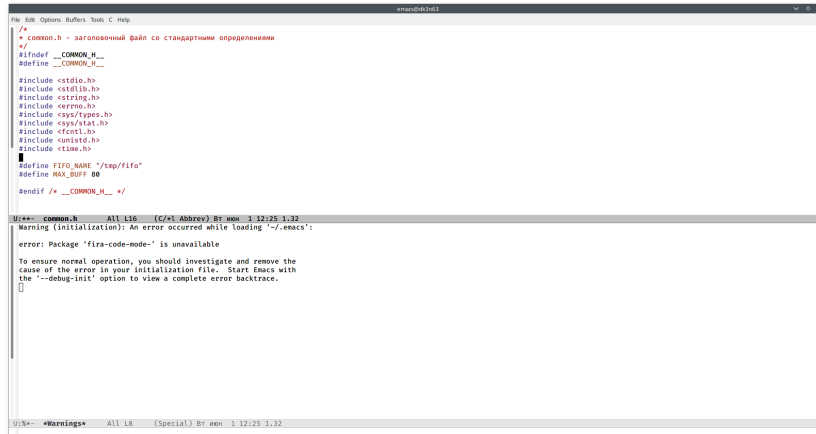


Figure 3.1: Создание файлов

2). Далее я изменила коды программ, представленных в тексте лабораторной работы. В файл common.h добавила стандартные заголовочные файлы unistd.h и time.h, необходимые для работы кодов других файлов. Common.h предназначен для заголовочных файлов, чтобы в остальных программах их не прописывать

каждый раз (Программа представлена на рис. 3.2). Вся необходимая информация про коды программ указана в следующем источнике: Программное обеспечение GNU/Linux. Лекция 12. Выбор дистрибутива (Г. Курячий, МГУ)



```
File Edit Options Buffers Tools C Menu
common.h - заголовочный файл со стандартными определениями
/*
 * common.h - заголовочный файл со стандартными определениями
 */
#ifndef COMMON_H_
#define COMMON_H_

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <time.h>

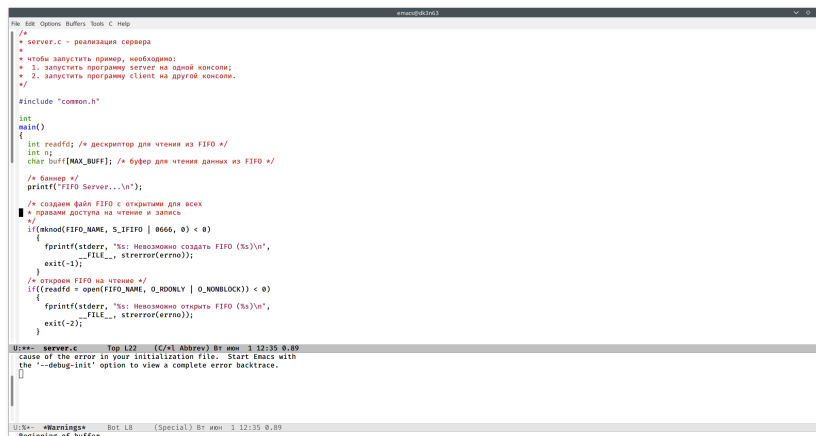
#define FIFO_NAME "/tmp/fifo"
#define MAX_BUF 80

#endif /* COMMON_H_ */

U: X= warnings All L8 (Special) Br: mem 1 12:29 1.32
Warning (initialization): An error occurred while loading '~/.emacs':
error: Package 'first-code-mode' is unavailable
To ensure normal operation, you should investigate and remove the
cause of the error in your initialization file. Start Emacs with
the '--debug-init' option to view a complete error backtrace.
[]
```

Figure 3.2: Программа в файле common.h

В файл server.c добавила цикл while для контроля за временем работы сервера. Разница между текущим временем time(NULL) и временем начала работы clock_t start=time(NULL) (инициализация до цикла) не должна превышать 30 секунд (алгоритм действий представлен на рис. 3.3 , 3.4).



```
File Edit Options Buffers Tools C Menu
server.c - реализация сервера
/*
 * server.c - реализация сервера
 * чтобы запустить пример, необходимо:
 * 1. запустить программу server на одной консоли;
 * 2. запустить программу client на другой консоли.
 */
#include "common.h"

int
main()
{
    int readfd; /* дескриптор для чтения из FIFO */
    int n;
    char buff[MAX_BUF]; /* буфер для чтения данных из FIFO */

    /* banner */
    printf("FIFO Server...\n");

    /* создаем файл FIFO с открытыми для всех
     * правами доступа на чтение и запись
     */
    if(mknod(FIFO_NAME, S_IFIFO | 0666, 0) < 0)
    {
        fprintf(stderr, "fs: Невозможно создать FIFO (%s)\n",
                FIFO_NAME);
        exit(-1);
    }

    /* отворяем FIFO на чтение */
    if((readfd = open(FIFO_NAME, O_RDONLY | O_NONBLOCK)) < 0)
    {
        fprintf(stderr, "fs: Невозможно открыть FIFO (%s)\n",
                FIFO_NAME);
        exit(-2);
    }

    while(1)
    {
        n = read(readfd, buff, MAX_BUF);
        if(n < 0)
            continue;
        printf("FIFO Server: %s\n", buff);
    }
}

U: X= warnings Bot L8 (Special) Br: mem 1 12:30 0.89
cause of the error in your initialization file. Start Emacs with
the '--debug-init' option to view a complete error backtrace.
[]
Beginning of buffer
```

Figure 3.3: Программа в файле server.c


```

File Edit Options Buffers Tools C Help
server.c
1  exit(-2);
2  }
3  /* начало отсчета времени */
4  clock_t start = time(NULL);
5  /* цикл работает, пока с момента начала отсчета времени прошло меньше 30 секунд */
6  while (time(NULL) - start < 30)
7  {
8      /* чтение данных из FIFO и вывод на экран */
9      while ((n = read(readfd, buff, MAX_BUF)) > 0)
10     {
11         if (write(1, buff, n) != n)
12         {
13             fprintf(stderr, "%s: Ошибка вывода (%s)\n",
14                     _FILE_, strerror(errno));
15             exit(-3);
16         }
17     }
18 }
19 close(readfd); /* закроем FIFO */
20 /* удалим FIFO из системы */
21 if (unlink(FIFO_NAME) < 0)
22 {
23     fprintf(stderr, "%s: Невозможно удалить FIFO (%s)\n",
24             _FILE_, strerror(errno));
25     exit(-4);
26 }
27 exit(0);
28 }

Usage: server.c [Bot LB] [C/r1 Abbrev] Br men 3 12:35 0.89
Cause of the error in your initialization file. Start Emacs with
the "--debug-init" option to view a complete error backtrace.
[]

U:No- *Warnings* Bot LB (Special) Br men 3 12:35 0.89

```

Figure 3.4: Программа в файле server.c

В файл client.c добавила цикл, который отвечает за количество сообщений о текущем времени (4 сообщения), которое получается в результате выполнения команд на Рисунке 7 (*/текущее время/*) и команду sleep(5) для приостановки работы клиента на 5 секунд (алгоритм действий представлен на рис. 3.5 , 3.6).

```

File Edit Options Buffers Tools C Help
client.c
1  /*
2   * client.c - реализация клиента
3   *
4   * чтобы запустить пример, необходимо:
5   * 1. запустить программу server на одной консоли;
6   * 2. запустить программу client на другой консоли.
7   */
8  #include "common.h"
9
10 int main ()
11 {
12     int writefd; /* дескриптор для записи в FIFO */
13     int n;
14     /* БANNER */
15     printf("FIFO Client...\n");
16     /* цикл, отвечающий за отправку сообщений о текущем времени */
17     for (int i=0; i<4; i++)
18     {
19         /* получим доступ к FIFO */
20         if ((writefd = open(FIFO_NAME, O_WRONLY)) < 0)
21         {
22             fprintf(stderr, "%s: Невозможно открыть FIFO (%s)\n",
23                     _FILE_, strerror(errno));
24             exit(-1);
25         }
26         break;
27     }
28 }

Usage: client.c [Top L29] [C/r1 Abbrev] Br men 3 12:49 0.94
Warning (initialization): An error occurred while loading './emacs':
error: Package 'fira-code-mode-' is unavailable
To ensure normal operation, you should investigate and remove the
cause of the error in your initialization file. Start Emacs with
the "--debug-init" option to view a complete error backtrace.
[]

U:No- *Warnings* All LB (Special) Br men 3 12:49 0.94
Beginning of buffer

```

Figure 3.5: Программа в файле client.c

```
exit(-1);
break;
}

/* текущее время */
long int time=time(NULL);
char* text=time(&time);

/* передаем сообщение серверу */
arglen = strlen(text);
if(write(sockfd, text, arglen) != arglen)
{
    fprintf(stderr, "No Quidam запись в FIFO (%s)\n",
            __FILE__, strerror(errno));
    exit(-2);
}

/* приостановка работы клиента на 5 секунд */
sleep(5);
}

/* закрываем доступ к FIFO */
close(sockfd);
exit(0);
}
```

U-- client.c Buf L10 (C/41.400x) By: mon 1 12:40 0.84
Warning (initialization): An error occurred while loading "/.emacs":
error: Package "fira-code-mode" is unavailable
To ensure normal operation, you should investigate and remove the
cause of the error in your initialization file. Start Emacs with
the "--debug-init" option to view a complete error backtrace.
[]

U-- *Warnings* All L8 (Special) By: mon 1 12:49 0.94
End of buffer

Figure 3.6: Программа в файле client.c

Makefile (файл для сборки) не изменяла (Рисунок 3.7).

```
all: server client

server: server.c common.h
    gcc server.c -o server

client: client.c common.h
    gcc client.c -o client

clean:
    rm server client *.o
```

U-- Makefile All L10 (GNUmakefile) By: mon 1 12:50 0.88
Warning (initialization): An error occurred while loading "/.emacs":
error: Package "fira-code-mode" is unavailable
To ensure normal operation, you should investigate and remove the
cause of the error in your initialization file. Start Emacs with
the "--debug-init" option to view a complete error backtrace.
[]

U-- *Warnings* All L8 (Special) By: mon 1 12:50 0.88
(No changes need to be saved)

Figure 3.7: Программа в Makefile

3). После написания кодов, я, используя команду «make all», скомпилировала необходимые файлы (Скриншот 3.8). Подробную информацию по данной теме можно найти в следующих интернет-источниках: <https://habr.com/ru/post/122108/> , https://www.opennet.ru/docs/RUS/linux_parallel/node17.html

```

~: bash — Konsole
Файл  Правка  Вид  Закладки  Настройка  Справка
tbkonovalova@dk3n63 ~ % touch common.h server.c client.c Makefile
tbkonovalova@dk3n63 ~ % emacs &
[1] 15479
tbkonovalova@dk3n63 ~ % emacs &
[2] 16784
[1] Завершился          emacs
tbkonovalova@dk3n63 ~ % make all
gcc server.c -o server
gcc client.c -o client
client.c: В функции «main»:
client.c:34:17: ошибка: expected «=», «.», «;», «asm» or «__attribute__» before «-» token
   34 |         char* text=ctime(&time);
      |         ^
client.c:34:17: ошибка: неверный тип аргумента для унарного минуса
client.c:37:23: ошибка: «text» не описан (первое использование в этой функции)
   37 |         msglen = strlen(text);
      |                        ^~~~~~
client.c:37:23: замечание: сообщение о каждом неопisanном идентификаторе выдается один раз в каждой функции, где он вс
тречается
make: *** [Makefile:7: client] Ошибка 1
tbkonovalova@dk3n63 ~ % emacs &
[3] 17783
[2] Завершился          emacs
tbkonovalova@dk3n63 ~ % make all
gcc client.c -o client
tbkonovalova@dk3n63 ~ % make all
make: Цель «all» не требует выполнения команд.
tbkonovalova@dk3n63 ~ %

```

Figure 3.8: Команда make all

Далее я проверила работу написанного кода. Отрыла 3 консоли (терминала) и запустила: в первом терминале – «./server», в остальных двух – «./client». В результате каждый терминал-клиент вывел по 4 сообщения. Спустя 30 секунд работа сервера была прекращена (Рисунок 3.9). Программа работает корректно.

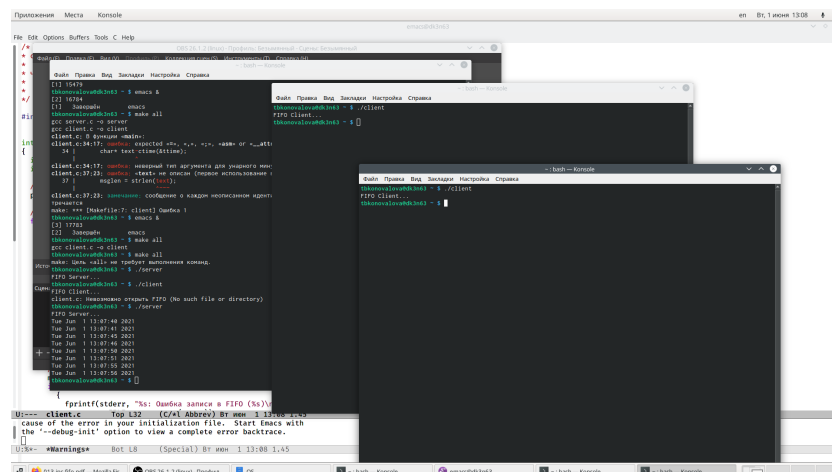


Figure 3.9: Команда make all

Также я отдельно проверила длительность работы сервера, введя команду

«./server» в одном терминале. Он завершил свою работу через 30 секунд (алгоритм действий представлен на рис. 3.10). Если сервер завершит свою работу, не закрыв канал, то, когда мы будем запускать этот сервер снова, появится ошибка «Невозможно создать FIFO», так как у нас уже есть один канал.

```

[2] 16784
[1] Завершён      emacs
tbkonovalova@dk3n63 ~ $ make all
gcc server.c -o server
gcc client.c -o client
client.c: В функции «main»:
client.c:34:17: ошибка: expected '=', ',', ';', 'asm' or '__attribute__' before «-» token
    34 |         char* text = ctime(&time);
        |                 ^
client.c:34:17: ошибка: неверный тип аргумента для унарного минуса
client.c:37:23: ошибка: «text» не описан (первое использование в этой функции)
    37 |         msglen = strlen(text);
        |                       ^
client.c:37:23: замечание: сообщение о каждом неопisanном идентификаторе выдается один раз в каждой функции, где он вс
тречается
make: *** [Makefile:7: client] Ошибка 1
tbkonovalova@dk3n63 ~ $ emacs &
[3] 17783
[2] Завершён      emacs
tbkonovalova@dk3n63 ~ $ make all
gcc client.c -o client
tbkonovalova@dk3n63 ~ $ make all
make: Цель «all» не требует выполнения команд.
tbkonovalova@dk3n63 ~ $ ./server
FIFO Server...
tbkonovalova@dk3n63 ~ $ ./client
FIFO Client...
client.c: Невозможно открыть FIFO (No such file or directory)
tbkonovalova@dk3n63 ~ $ ./server
FIFO Server...
Tue Jun 1 13:07:40 2021
Tue Jun 1 13:07:41 2021
Tue Jun 1 13:07:45 2021
Tue Jun 1 13:07:46 2021
Tue Jun 1 13:07:50 2021
Tue Jun 1 13:07:51 2021
Tue Jun 1 13:07:55 2021
Tue Jun 1 13:07:56 2021
tbkonovalova@dk3n63 ~ $ ./server
FIFO Server...
tbkonovalova@dk3n63 ~ $

```

Figure 3.10: Проверка длительности работы сервера

Контрольные вопросы:

- 1). Именованные каналы отличаются от неименованных наличием идентификатора канала, который представлен как специальный файл (соответственно имя именованного канала – это имя файла). Поскольку файл находится на локальной файловой системе, данное IPC используется внутри одной системы.
- 2). Чтобы создать неименованный канал из командной строки нужно использовать символ |, служащий для объединения двух и более процессов: процесс_1 | процесс_2 | процесс_3...
- 3). Чтобы создать именованный канал из командной строки нужно использовать либо команду «mknod», либо команду «mkfifo».
- 4). Неименованный канал является средством взаимодействия между связанными процессами – родительским и дочерним. Родительский процесс создает

канал при помощи системного вызова: `«int pipe(int fd[2]);»`. Массив из двух целых чисел является выходным параметром этого системного вызова. Если вызов выполнен нормально, то этот массив содержит два файловых дескриптора. `fd[0]` является дескриптором для чтения из канала, `fd[1]` – дескриптором для записи в канал. Когда процесс порождает другой процесс, дескрипторы родительского процесса наследуются дочерним процессом, и, таким образом, прокладывается трубопровод между двумя процессами. Естественно, что один из процессов использует канал только для чтения, а другой – только для записи. Поэтому, если, например, через канал должны передаваться данные из родительского процесса в дочерний, родительский процесс сразу после запуска дочернего процесса закрывает дескриптор канала для чтения, а дочерний процесс закрывает дескриптор для записи. Если нужен двунаправленный обмен данными между процессами, то родительский процесс создает два канала, один из которых используется для передачи данных в одну сторону, а другой – в другую.

5). Файлы именованных каналов создаются функцией `mkfifo()` или функцией `mknod`: 1. `«int mkfifo(const char *pathname, mode_t mode);»`, где первый параметр – путь, где будет располагаться FIFO (имя файла, идентифицирующего канал), второй параметр определяет режим работы с FIFO (маска прав доступа к файлу), 2. `«mknod (namefile, IFIFO | 0666, 0)»`, где `namefile` – имя канала, `0666` – к каналу разрешен доступ на запись и на чтение любому запросившему процессу), 3. `«int mknod(const char *pathname, mode_t mode, dev_t dev);»`. Функция `mkfifo()` создает канал и файл соответствующего типа. Если указанный файл канала уже существует, `mkfifo()` возвращает -1. После создания файла канала процессы, участвующие в обмене данными, должны открыть этот файл либо для записи, либо для чтения.

6). При чтении меньшего числа байтов, чем находится в канале или FIFO, возвращается требуемое число байтов, остаток сохраняется для последующих чтений. При чтении большего числа байтов, чем находится в канале или FIFO, возвращается доступное число байтов. Процесс, читающий из канала, должен соответствующим образом обработать ситуацию, когда прочитано меньше, чем

заказано.

7). Запись числа байтов, меньшего емкости канала или FIFO, гарантированно атомарно. Это означает, что в случае, когда несколько процессов одновременно записывают в канал, порции данных от этих процессов не перемешиваются. При записи большего числа байтов, чем это позволяет канал или FIFO, вызов `write(2)` блокируется до освобождения требуемого места. При этом атомарность операции не гарантируется. Если процесс пытается записать данные в канал, не открытый ни одним процессом на чтение, процессу генерируется сигнал `SIGPIPE`, а вызов `write(2)` возвращает 0 с установкой ошибки (`errno=ERRPIPE`) (если процесс не установил обработки сигнала `SIGPIPE`, производится обработка по умолчанию – процесс завершается).

8). Количество процессов, которые могут параллельно присоединяться к любому концу канала, не ограничено. Однако если два или более процесса записывают в канал данные одновременно, каждый процесс за один раз может записать максимум `PIPE BUF` байтов данных. Предположим, процесс (назовем его А) пытается записать X байтов данных в канал, в котором имеется место для Y байтов данных. Если X больше, чем Y, только первые Y байтов данных записываются в канал, и процесс блокируется. Запускается другой процесс (например. В); в это время в канале появляется свободное пространство (благодаря третьему процессу, считывающему данные из канала). Процесс В записывает данные в канал. Затем, когда выполнение процесса А возобновляется, он записывает оставшиеся X-Y байтов данных в канал. В результате данные в канал записываются поочередно двумя процессами. Аналогичным образом, если два (или более) процесса одновременно попытаются прочитать данные из канала, может случиться так, что каждый из них прочитает только часть необходимых данных.

9). Функция `write` записывает байты `count` из буфера `buffer` в файл, связанный с `handle`. Операции `write` начинаются с текущей позиции указателя на файл (указатель ассоциирован с заданным файлом). Если файл открыт для добавления, операции выполняются в конец файла. После осуществления операций запи-

си указатель на файл (если он есть) увеличивается на количество действительно записанных байтов. Функция `write` возвращает число действительно записанных байтов. Возвращаемое значение должно быть положительным, но меньше числа `count` (например, когда размер для записи `count` байтов выходит за пределы пространства на диске). Возвращаемое значение `-1` указывает на ошибку; `errno` устанавливается в одно из следующих значений: `EACCES` – файл открыт для чтения или закрыт для записи, `EBADF` – неверный `handle`-р файла, `ENOSPC` – на устройстве нет свободного места. Единица в вызове функции `write` в программе `server.c` означает идентификатор (дескриптор потока) стандартного потока вывода.

10). Прототип функции `strerror`: «`char * strerror(int errornum);`». Функция `strerror` интерпретирует номер ошибки, передаваемый в функцию в качестве аргумента `errornum`, в понятное для человека текстовое сообщение (строку). Откуда берутся эти ошибки? Ошибки эти возникают при вызове функций стандартных Си-библиотек. То есть хорошим тоном программирования будет – использование этой функции в паре с другой, и если возникнет ошибка, то пользователь или программист поймет, как исправить ошибку, прочитав сообщение функции `strerror`. Возвращенный указатель ссылается на статическую строку с ошибкой, которая не должна быть изменена программой. Дальнейшие вызовы функции `strerror` перезапишут содержание этой строки. Интерпретированные сообщения об ошибках могут различаться, это зависит от платформы и компилятора.

4 Библиография

1. Программное обеспечение GNU/Linux. Лекция 10. Минимальный набор знаний (Г. Курячий, МГУ)
2. Программное обеспечение GNU/Linux. Лекция 12. Выбор дистрибутива (Г. Курячий, МГУ)
3. Электронный ресурс: <https://habr.com/ru/post/122108/>
4. Электронный ресурс: https://www.opennet.ru/docs/RUS/linux_parallel/node17.html

5 Выводы

В ходе выполнения данной лабораторной работы я приобрела практические навыки работы с именованными каналами.