

Лабораторная работа №14

НКАбд-03-22

Шубнякова Дарья

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	8
5	Выводы	17
	Список литературы	19

Список иллюстраций

4.1	Готовый файл	8
4.2	Готовый файл	9
4.3	Готовый файл	10
4.4	Готовый файл	11
4.5	Готовый файл	12
4.6	Команда в терминале	13
4.7	Команда в терминале	14
4.8	Команда в терминала	15
4.9	Команда в терминале	16

Список таблиц

1 Цель работы

Приобретение практических навыков работы с именованными каналами.

2 Задание

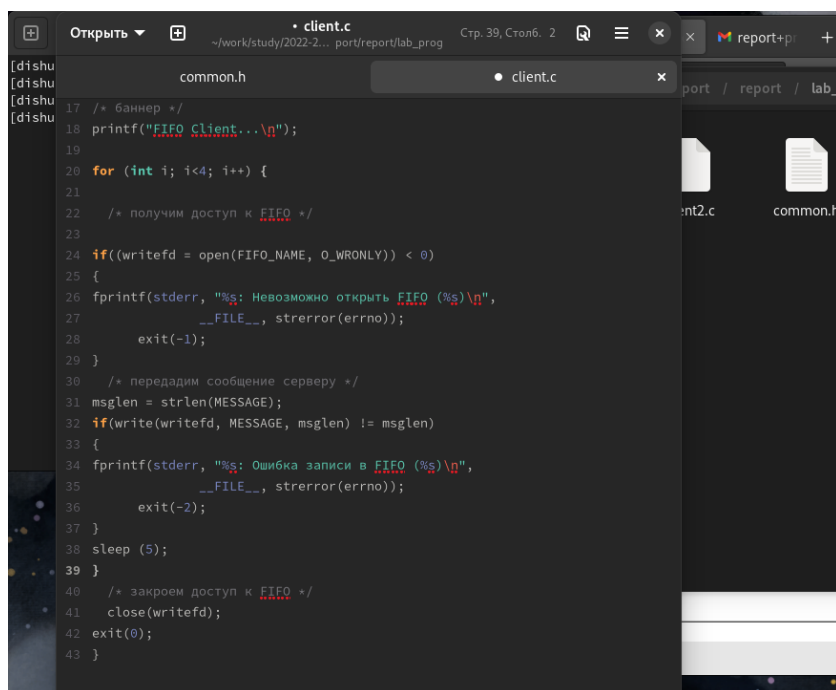
Изучите приведённые в тексте программы `server.c` и `client.c`. Взяв данные примеры за образец, напишите аналогичные программы, внося следующие изменения: 1. Работает не 1 клиент, а несколько (например, два). 2. Клиенты передают текущее время с некоторой периодичностью (например, раз в пять секунд). Используйте функцию `sleep()` для приостановки работы клиента. 3. Сервер работает не бесконечно, а прекращает работу через некоторое время (например, 30 сек). Используйте функцию `clock()` для определения времени работы сервера. Что будет в случае, если сервер завершит работу, не закрыв канал?

3 Теоретическое введение

Одним из видов взаимодействия между процессами в операционных системах является обмен сообщениями. Под сообщением понимается последовательность байтов, передаваемая от одного процесса другому. В операционных системах типа UNIX есть 3 вида межпроцессорных взаимодействий: общеоуниковые (именованные каналы, сигналы), System V Interface Definition (SVID — разделяемая память, очередь сообщений, семафоры) и BSD (сокеты). Для передачи данных между неродственными процессами можно использовать механизм именованных каналов (named pipes). Данные передаются по принципу FIFO (First In First Out) (первым записан — первым прочитан), поэтому они называются также FIFO pipes или просто FIFO. 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.

4 Выполнение лабораторной работы

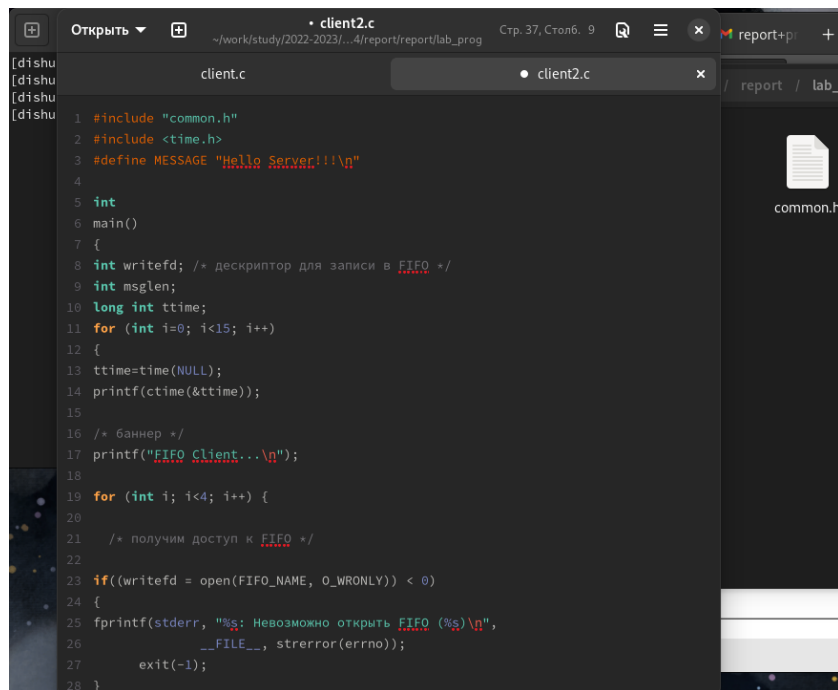
Переделываем файл client.c.



```
17 /* баннер */
18 printf("FIFO client...\n");
19
20 for (int i; i<4; i++) {
21
22     /* получим доступ к FIFO */
23
24     if((writefd = open(FIFO_NAME, O_WRONLY)) < 0)
25     {
26         fprintf(stderr, "%s: Невозможно открыть FIFO (%s)\n",
27             __FILE__, strerror(errno));
28         exit(-1);
29     }
30     /* передадим сообщение серверу */
31     msglen = strlen(MESSAGE);
32     if(write(writefd, MESSAGE, msglen) != msglen)
33     {
34         fprintf(stderr, "%s: Ошибка записи в FIFO (%s)\n",
35             __FILE__, strerror(errno));
36         exit(-2);
37     }
38     sleep (5);
39 }
40 /* закроем доступ к FIFO */
41 close(writefd);
42 exit(0);
43 }
```

Рис. 4.1: Готовый файл

Создаем файл client2.c.



```
1 #include "common.h"
2 #include <time.h>
3 #define MESSAGE "Hello Server!!!\n"
4
5 int
6 main()
7 {
8     int writefd; /* дескриптор для записи в FIFO */
9     int msglen;
10    long int ttime;
11    for (int i=0; i<15; i++)
12    {
13        ttime=time(NULL);
14        printf(ctime(&ttime));
15
16        /* баннер */
17        printf("FIFO Client...\n");
18
19        for (int i; i<4; i++) {
20
21            /* получим доступ к FIFO */
22
23            if((writefd = open(FIFO_NAME, O_WRONLY)) < 0)
24            {
25                fprintf(stderr, "%s: Невозможно открыть FIFO (%s)\n",
26                    __FILE__, strerror(errno));
27                exit(-1);
28            }
```

Рис. 4.2: Готовый файл

Редактируем файл server.c.

```
30 clock_t now=time(NULL), start=time(NULL);
31 while(now-start<30){
32     /* читаем данные из FIFO и выводим на экран */
33     while((n = read(readfd, buff, MAX_BUFF)) > 0)
34     {
35         if(write(1, buff, n) != n)
36         {
37             fprintf(stderr, "%s: Ошибка вывода (%s)\n",
38                 __FILE__, strerror(errno));
39             exit(-3);
40         }
41     }
42     now=time(NULL);
43 }
44 printf("Время работы сервера вышло, %li - сек. прошло\n", (now-start));
45 close(readfd); /* закроем FIFO */
46
47 /* удалим FIFO из системы */
48 if(unlink(FIFO_NAME) < 0)
49 {
50     fprintf(stderr, "%s: Невозможно удалить FIFO (%s)\n",
51         __FILE__, strerror(errno));
52     exit(-4);
53 }
54
55 exit(0);
56 }
```

Рис. 4.3: Готовый файл

Создаем Makefile на базе шаблона.

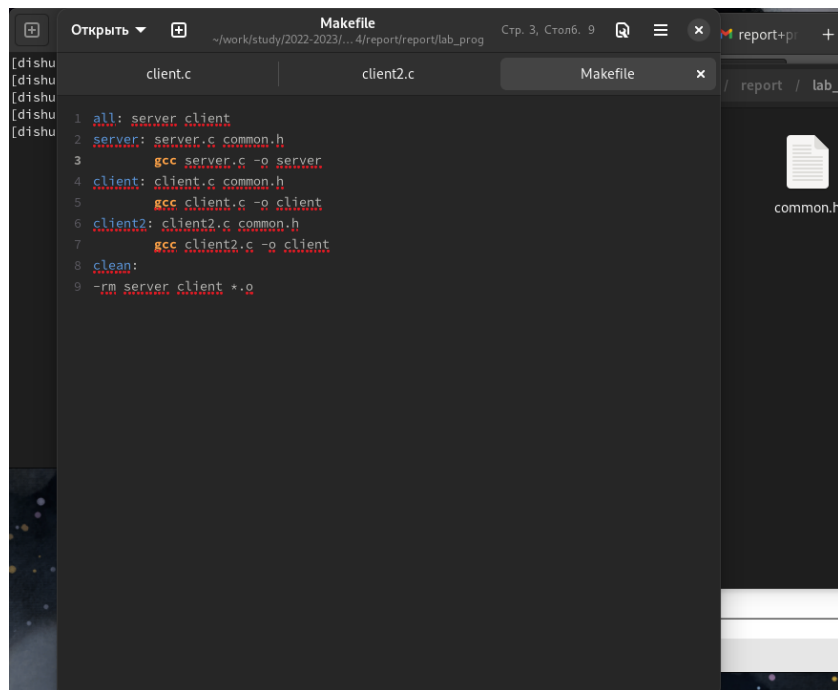


Рис. 4.4: Готовый файл

Копируем файл common.h

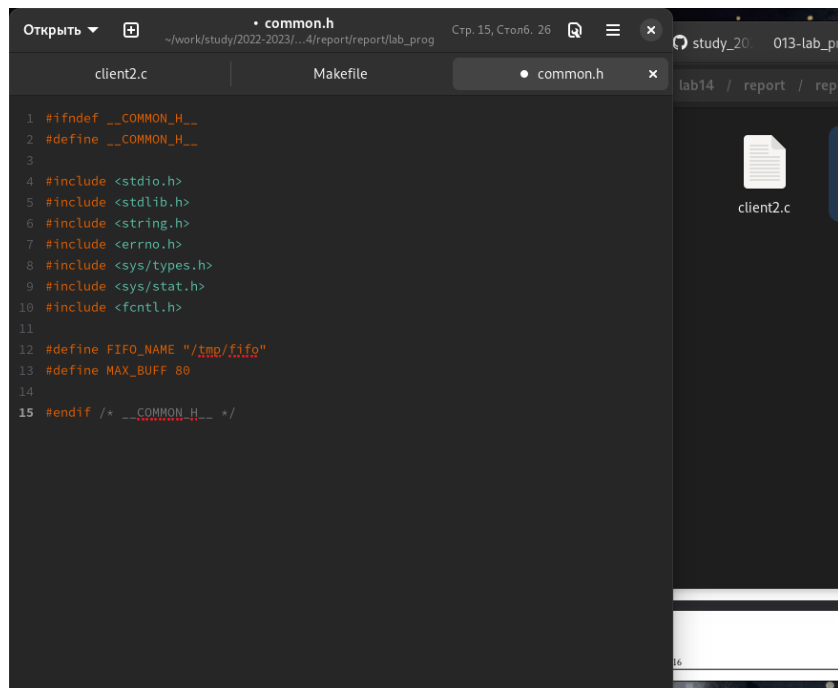
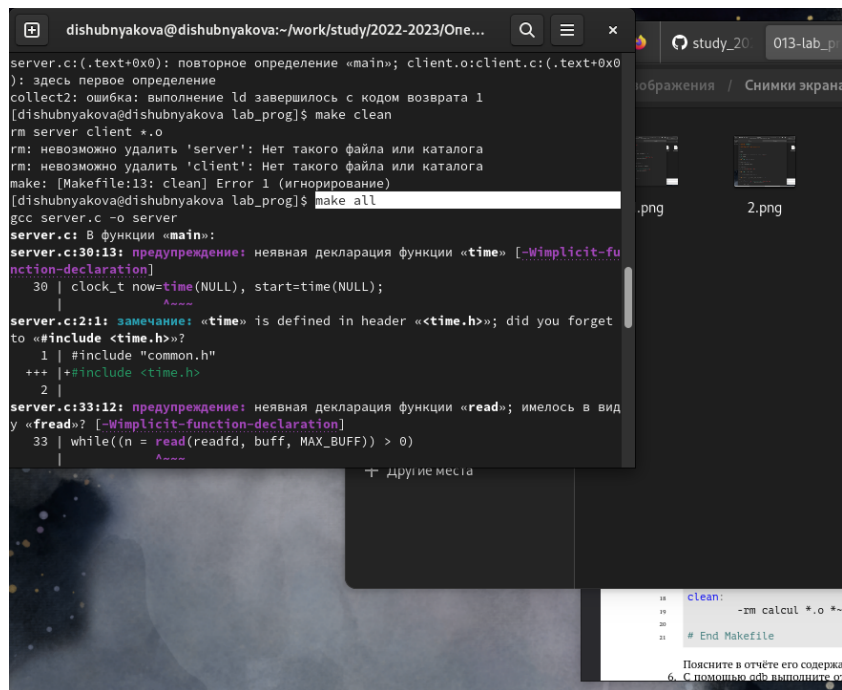


Рис. 4.5: Готовый файл

Производим команду make all.

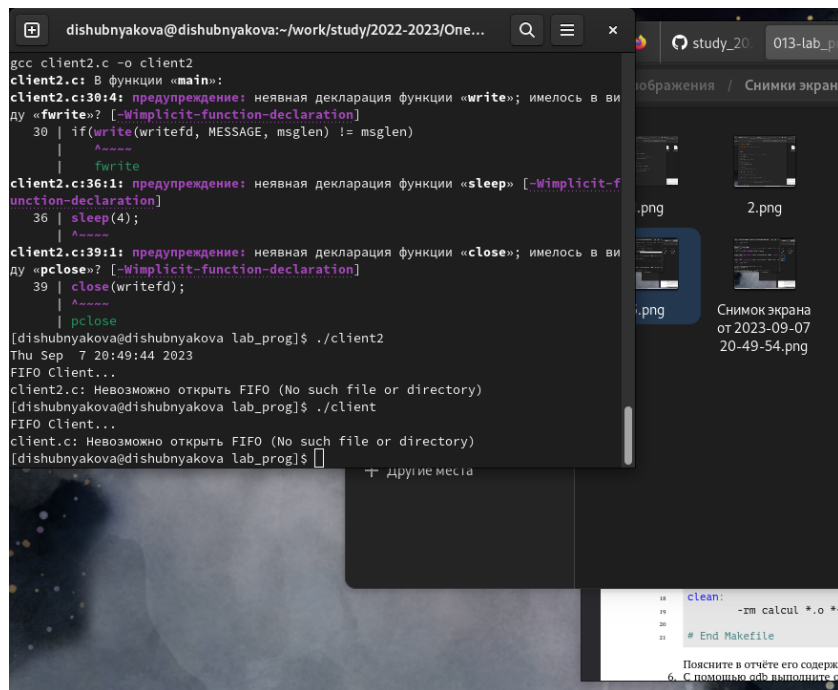


```
server.c:(.text+0x0): повторное определение «main»; client.o:client.c:(.text+0x0): здесь первое определение
collect2: ошибка: выполнение ld завершилось с кодом возврата 1
[dishubnyakova@dishubnyakova lab_prog]$ make clean
rm server client *.o
rm: невозможно удалить 'server': Нет такого файла или каталога
rm: невозможно удалить 'client': Нет такого файла или каталога
make: [Makefile:13: clean] Error 1 (игнорирование)
[dishubnyakova@dishubnyakova lab_prog]$ make all
gcc server.c -o server
server.c: В функции «main»:
server.c:30:13: предупреждение: неявная декларация функции «time» [-Wimplicit-function-declaration]
   30 | clock_t now=time(NULL), start=time(NULL);
      |               ^~~~~
server.c:12:1: замечание: «time» is defined in header «<time.h>»; did you forget to «#include <time.h>»?
   1 | #include "common.h"
+++ |+#include <time.h>
   2 |
server.c:33:12: предупреждение: неявная декларация функции «read»; имелось в виду «fread»? [-Wimplicit-function-declaration]
   33 | while((n = read(readfd, buff, MAX_BUFF)) > 0)
      |            ^~~~~
      |            |
      |            + другие места

clean:
rm calcul *.o *~
# End Makefile
```

Рис. 4.6: Команда в терминале

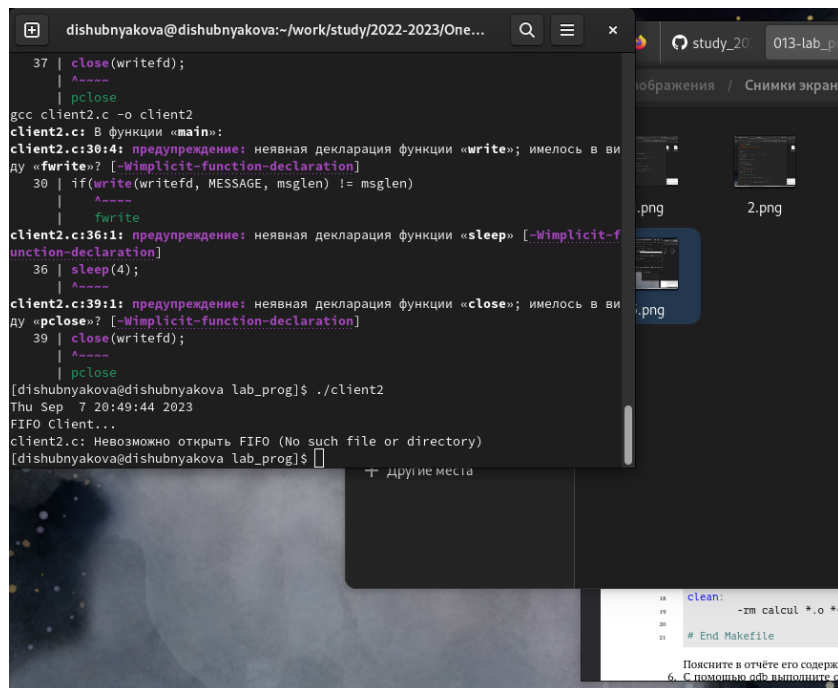
Смотрим, как работает client.c.



```
gcc client2.c -o client2
client2.c: В функции «main»:
client2.c:30:4: предупреждение: неявная декларация функции «write»; имелось в ви
ду «fwrite»? [-Wimplicit-function-declaration]
   30 | if(write(writefd, MESSAGE, msglen) != msglen)
      |      ^~~~~
      |      fwrite
client2.c:36:11: предупреждение: неявная декларация функции «sleep» [-Wimplicit-f
unction-declaration]
   36 |     sleep(4);
      |     ^~~~~
client2.c:39:11: предупреждение: неявная декларация функции «close»; имелось в ви
ду «pclose»? [-Wimplicit-function-declaration]
   39 |     close(writefd);
      |     ^~~~~
      |     pclose
[dishubnyakova@dishubnyakova lab_prog]$ ./client2
Thu Sep  7 20:49:44 2023
FIFO Client...
client2.c: Невозможно открыть FIFO (No such file or directory)
[dishubnyakova@dishubnyakova lab_prog]$ ./client
FIFO Client...
client.c: Невозможно открыть FIFO (No such file or directory)
[dishubnyakova@dishubnyakova lab_prog]$
```

Рис. 4.7: Команда в терминале

Смотрим, как работает client2.c.



```
dishubnyakova@dishubnyakova:~/work/study/2022-2023/One...
37 | close(writefd);
    | ^~~~~
    | pclose
gcc client2.c -o client2
client2.c: В функции «main»:
client2.c:30:4: предупреждение: неявная декларация функции «write»; имелось в ви
ду «fwrite»? [-Wimplicit-function-declaration]
30 | if(write(writefd, MESSAGE, msglen) != msglen)
    |     ^~~~~
    |     fwrite
client2.c:36:11: предупреждение: неявная декларация функции «sleep» [-Wimplicit-f
unction-declaration]
36 | sleep(4);
    |     ^~~~~
client2.c:39:11: предупреждение: неявная декларация функции «close»; имелось в ви
ду «pclose»? [-Wimplicit-function-declaration]
39 | close(writefd);
    |     ^~~~~
    | pclose
[dishubnyakova@dishubnyakova lab_prog]$ ./client2
Thu Sep  7 20:49:44 2023
FIFO Client...
client2.c: Невозможно открыть FIFO (No such file or directory)
[dishubnyakova@dishubnyakova lab_prog]$
```

Рис. 4.8: Команда в терминала

Спустя некоторое время сервер завершил работу.

```
dishubnyakova@dishubnyakova:~/work/study/2022-2023/One...
client2.c:30:4: предупреждение: неявная декларация функции «write»; имелось в ви
ду «fwrite»? [-Wimplicit-function-declaration]
  30 | if(write(writefd, MESSAGE, msglen) != msglen)
      |      ^~~~~
      |      fwrite
client2.c:36:11: предупреждение: неявная декларация функции «sleep» [-Wimplicit-f
unction-declaration]
  36 |     sleep(4);
      |     ^~~~~
client2.c:39:11: предупреждение: неявная декларация функции «close»; имелось в ви
ду «pclose»? [-Wimplicit-function-declaration]
  39 |     close(writefd);
      |     ^~~~~
      |     pclose
[dishubnyakova@dishubnyakova lab_prog]$ ./client2
Thu Sep  7 20:49:44 2023
FIFO Client...
client2.c: Невозможно открыть FIFO (No such file or directory)
[dishubnyakova@dishubnyakova lab_prog]$ ./client
FIFO Client...
client.c: Невозможно открыть FIFO (No such file or directory)
[dishubnyakova@dishubnyakova lab_prog]$ ./server
FIFO Server...

+ другие места
```

study_20 013-lab_pi

bs / lab14 / report / rep

lient client.c

akefile server

```
18 clean: -rm calcul *.o *~
19
20 # End Makefile
```

Поясните в отчёте его содержа
6. С помощью edf выполните от

Рис. 4.9: Команда в терминале

5 Выводы

Приобрели практические навыки по работе с именованными каналами. 1. Именованные каналы отличаются от неименованных наличием идентификатора канала, который представлен как специальный файл (соответственно имя именованного канала — это имя файла). 2. Создание неименованного канала из командной строки возможно командой `pipe`. 3. Создание именованного канала из командной строки возможно с помощью `mkfifo`. 4. Функция языка C, создающая неименованный канал: `int read(int pipe_fd, void area, int cnt); int write(int pipe_fd, void area, int cnt);` Первый аргумент этих вызовов - дескриптор канала, второй - указатель на область памяти, с которой происходит обмен, третий - количество байт. Оба вызова возвращают число переданных байт (или -1 - при ошибке). 5. Функция языка C, создающая именованный канал: `int mkfifo(const char *pathname, mode_t mode);` Первый параметр — имя файла, идентифицирующего канал, второй параметр маска прав доступа к файлу. Вызов функции `mkfifo()` создаёт файл канала (с именем, заданным макросом `FIFO_NAME`): `mkfifo(FIFO_NAME, 0600)`. 6. При чтении меньшего числа байтов, возвращается требуемое число байтов, остаток сохраняется для следующих чтений. При чтении большего числа байтов, возвращается доступное число байтов. 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 гарантиру-

ется. 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=EP1PE) (если процесс не установил обработки сигнала SIGPIPE, производится обработка по умолчанию – процесс завершается). 8. Два и более процессов могут читать и записывать в канал. 9. Функция write записывает length байтов из буфера buffer в файл, определенный дескриптором файла fd. Эта операция чисто ‘двоичная’ и без буферизации. При единице возвращает действительное число байтов. Функция write возвращает число действительно записанных в файл байтов или -1 при ошибке, устанавливая при этом errno. 10. Строковая функция strerror - функция языков C/C++, транслирующая код ошибки, который обычно хранится в глобальной переменной errno, в сообщение об ошибке, понятном человеку.

Список литературы

1. Dash P. Getting started with oracle vm virtualbox. Packt Publishing Ltd, 2013. 86 p.
2. Colvin H. Virtualbox: An ultimate guide book on virtualization with virtualbox. CreateSpace Independent Publishing Platform, 2015. 70 p.
3. van Vugt S. Red hat rhcsa/rhce 7 cert guide : Red hat enterprise linux 7 (ex200 and ex300). Pearson IT Certification, 2016. 1008 p.
4. Робачевский А., Немнюгин С., Стесик О. Операционная система unix. 2-е изд. Санкт-Петербург: БХВ-Петербург, 2010. 656 p.
5. Немет Э. et al. Unix и Linux: руководство системного администратора. 4-е изд. Вильямс, 2014. 1312 p.
6. Колисниченко Д.Н. Самоучитель системного администратора Linux. СПб.: БХВ-Петербург, 2011. 544 p.
7. Robbins A. Bash pocket reference. O'Reilly Media, 2016. 156 p.