

# **ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №14**

**Именованные каналы**

Анастасия Павловна Баранова, НБИбд-01-21

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>4</b>
<b>2</b>	<b>Задание</b>	<b>5</b>
<b>3</b>	<b>Выполнение лабораторной работы</b>	<b>6</b>
<b>4</b>	<b>Вывод</b>	<b>10</b>
<b>5</b>	<b>Ответы на контрольные вопросы</b>	<b>11</b>

## Список иллюстраций

3.1	Создам необходимые файлы с помощью команды touch. . . . .	6
3.2	Изменяю файл common.h. . . . .	6
3.3	Добавляю в файл server.c цикл while для контроля за временем работы сервера. . . . .	7
3.4	В файл client.c добавляю цикл, который отвечает за количество сообщений о текущем времени. . . . .	8
3.5	Makefile (файл для сборки) оставляю без изменений. . . . .	8
3.6	Скомпилирую необходимые файлы. . . . .	8
3.7	Проверю работу написанного кода. . . . .	9

# 1 Цель работы

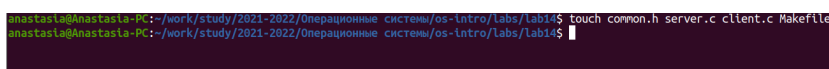
Целью данной лабораторной работы является приобретение практических навыков работы с именованными каналами.

## 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 Выполнение лабораторной работы

Создам необходимые файлы с помощью команды touch. (рис. 3.1)



```
anastasia@Anastasia-PC:~/work/study/2021-2022/Операционные системы/os-intro/labs/lab14$ touch common.h server.c client.c Makefile
anastasia@Anastasia-PC:~/work/study/2021-2022/Операционные системы/os-intro/labs/lab14$
```

Рис. 3.1: Создам необходимые файлы с помощью команды touch.

Изменяю коды программ, представленных в тексте лабораторной работы. В файл common.h добавляю 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. (рис. 3.2)



```
common.h
~/work/study/2021-2022/Операционные системы/os-intro/labs/lab14

1 #ifndef __COMMON_H__
2 #define __COMMON_H__
3
4 #include <stdio.h>
5 #include <stdlib.h>
6 #include <string.h>
7 #include <errno.h>
8 #include <sys/types.h>
9 #include <sys/stat.h>
10 #include <fcntl.h>
11 #include <unistd.h>
12 #include <time.h>
13
14 #define FIFO_NAME "/tmp/fifo"
15 #define MAX_BUFF 80
16
17 #endif
```

Рис. 3.2: Изменяю файл common.h.

В файл server.c добавляю 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. (рис. 3.3)



```

server.c
~/work/study/2021-2022/Операционные системы/os-intro/labs/lab14

1 #include "common.h"
2
3 int main() {
4     int readfd;
5     int n;
6     char buff[MAX_BUFF];
7
8     printf("FIFO Server...\n");
9
10    if(mknod(FIFO_NAME, S_IFIFO | 0666, 0) < 0)
11    {
12        fprintf(stderr, "%s: Невозможно создать FIFO (%s)\n", __FILE__, strerror(errno));
13        exit(-1);
14    }
15
16    if((readfd = open(FIFO_NAME, O_RDONLY)) < 0)
17    {
18        fprintf(stderr, "%s: Невозможно открыть FIFO (%s)\n", __FILE__, strerror(errno));
19        exit(-2);
20    }
21
22    clock_t start = time(NULL);
23
24    while(time(NULL) - start < 30)
25    {
26        while((n = read(readfd, buff, MAX_BUFF)) > 0)
27        {
28            if(write(1, buff, n) != n)
29            {
30                fprintf(stderr, "%s: Ошибка вывода (%s)\n", __FILE__, strerror(errno));
31                exit(-3);
32            }
33        }
34    }
35
36    close(readfd);
37
38    if(unlink(FIFO_NAME) < 0)
39    {
40        fprintf(stderr, "%s: Невозможно удалить FIFO (%s)\n", __FILE__, strerror(errno));
41        exit(-4);
42    }
43
44    exit(0);
45 }

```

Рис. 3.3: Добавлю в файл server.c цикл while для контроля за временем работы сервера.

В файл client.c добавлю цикл, который отвечает за количество сообщений о текущем времени (4 сообщения), которое получается в результате выполнения команд, и команду sleep(5) для приостановки работы клиента на 5 секунд. (рис. 3.4)

```

1 #include "common.h"
2
3 int main() {
4     int writefd;
5     int msglen;
6
7     printf("FIFO Client...\n");
8
9     for(int i=0; i<4; i++)
10    {
11
12        if((writefd = open(FIFO_NAME, O_WRONLY)) < 0)
13        {
14            fprintf(stderr, "%s: Невозможно открыть FIFO (%s)\n", __FILE__, strerror(errno));
15            exit(-1);
16        }
17
18        long int ttime = time(NULL);
19        char* text = ctime(&ttime);
20
21        msglen = strlen(text);
22        if(write(writefd, text, msglen) != msglen)
23        {
24            fprintf(stderr, "%s: Ошибка записи в FIFO (%s)\n", __FILE__, strerror(errno));
25            exit(-2);
26        }
27
28        sleep (5);
29    }
30
31    close(writefd);
32
33    exit(0);
34 }

```

Рис. 3.4: В файл client.c добавлю цикл, который отвечает за количество сообщений о текущем времени.

Makefile (файл для сборки) оставлю без изменений. (рис. 3.5)

```

1 all: server client
2
3 server: server.c common.h
4     gcc server.c -o server
5
6 client: client.c common.h
7     gcc client.c -o client
8
9 clean:
10    -rm server client *.o

```

Рис. 3.5: Makefile (файл для сборки) оставлю без изменений.

После написания кодов, скомпилирую необходимые файлы, используя команду make all. (рис. 3.6)

```

anastasia@Anastasia-PC:~/work/study/2021-2022/Операционные системы/os-intro/labs/lab14$ make all
gcc server.c -o server
gcc client.c -o client
anastasia@Anastasia-PC:~/work/study/2021-2022/Операционные системы/os-intro/labs/lab14$

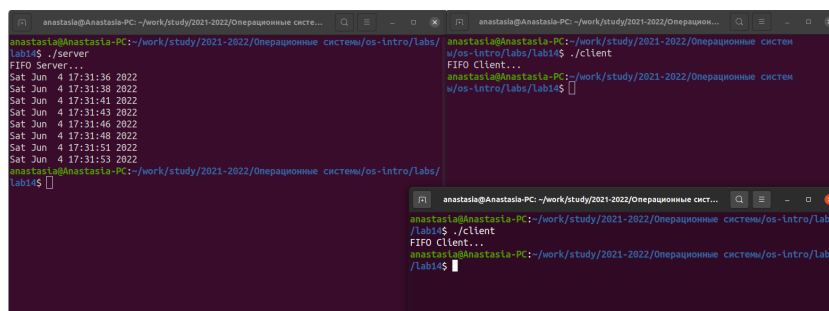
```

Рис. 3.6: Скомпилирую необходимые файлы.

Далее проверю работу написанного кода. Открою 3 консоли (терминала) и запущу: в первом терминале – «./server», в остальных двух – «./client». В результате



каждый терминал-клиент вывел по 4 сообщения. Спустя 30 секунд работа сервера была прекращена. Программа работает корректно. (рис. 3.7)



```
anastasia@Anastasia-PC: ~/work/study/2021-2022/Операционные системы/os-intro/Labs
lab1$ ./server
FIFO Server...
Sat Jun 4 17:31:36 2022
Sat Jun 4 17:31:38 2022
Sat Jun 4 17:31:41 2022
Sat Jun 4 17:31:43 2022
Sat Jun 4 17:31:46 2022
Sat Jun 4 17:31:48 2022
Sat Jun 4 17:31:51 2022
Sat Jun 4 17:31:53 2022
anastasia@Anastasia-PC: ~/work/study/2021-2022/Операционные системы/os-intro/Labs/
lab1$

anastasia@Anastasia-PC: ~/work/study/2021-2022/Операционные системы/os-intro/Labs/lab1$ ./client
FIFO Client...
anastasia@Anastasia-PC: ~/work/study/2021-2022/Операционные системы/os-intro/Labs/lab1$

anastasia@Anastasia-PC: ~/work/study/2021-2022/Операционные системы/os-intro/Labs/lab1$ ./client
FIFO Client...
anastasia@Anastasia-PC: ~/work/study/2021-2022/Операционные системы/os-intro/Labs/lab1$
```

Рис. 3.7: Проверю работу написанного кода.

Отдельно проверю длительность работы сервера, введя команду «./server» в одном терминале. Он завершил свою работу через 30 секунд. 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.

## **4 Вывод**

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

## 5 Ответы на контрольные вопросы

1. В чем ключевое отличие именованных каналов от неименованных? Ответ: 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. Возможно ли создание неименованного канала из командной строки? Ответ: Чтобы создать неименованный канал из командной строки нужно использовать символ |, служащий для объединения двух и более процессов: процесс\_1 | процесс\_2 | процесс\_3...
3. Возможно ли создание именованного канала из командной строки? Ответ: Чтобы создать именованный канал из командной строки нужно использовать либо команду «mknod », либо команду «mkfifo ».
4. Опишите функцию языка C, создающую неименованный канал. Ответ: 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. Опишите функцию языка C, создающую именованный канал. Ответ: Файлы именованных каналов создаются функцией `mkfifo()` или функцией `mknod`:
- `«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),
  - `«mknod (namefile, IFIFO | 0666, 0)»`, где `namefile` – имя канала, `0666` – к каналу разрешен доступ на запись и на чтение любому запросившему процессу),
  - `«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. Что будет в случае прочтения из `fifo` меньшего числа байтов, чем находится в канале? Большого числа байтов? Ответ: 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. Аналогично, что будет в случае записи в `fifo` меньшего числа байтов, чем

позволяет буфер? Большого числа байтов? Ответ: 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. Могут ли два и более процессов читать или записывать в канал? Ответ: 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 пытаются прочитать данные из канала, может случиться так, что каждый из них прочитает только часть необходимых данных.
9. Опишите функцию `write` (тип возвращаемого значения, аргументы и логику работы). Что означает 1 (единица) в вызове этой функции в программе

server.c (строка 42)? Ответ: 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 записи 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. Опишите функцию strerror. Ответ: 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.