

Отчет по лабораторной работа №14

Именованные каналы

Лушин Артем Андреевич

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
3	Контрольные вопросы	10
4	Выводы	13
	Список литературы	14

Список иллюстраций

2.1	Код в файле client.c	6
2.2	Код в файле client2.c	7
2.3	Код в файле common.h	7
2.4	Код в файле server.c	8
2.5	Код в файле Makefile	8
2.6	Создание программ	9
2.7	Запуск программы	9

Список таблиц

1 Цель работы

Приобретение практических навыков работы с именованными каналами.

2 Выполнение лабораторной работы

- 1) Я создал каталог 14 и в нем создал файлы: client.c, client2.c, common.h, server.c. Взял пример с файла лабораторной работы и написал измененную программу, которая выполняет все нужные критерии.



```
client.c [----] 0 L: [ 1+ 0 1/ 35] *(0 / 621b) 0035 0x023 [*][X]
#include "common.h"

#define MESSAGE "Hello Server!!!\n"

int
main()
{
    int msg, len, i;
    long int t;

    for(i=0; i<20; i++)
    {
        sleep(3);
        t=time(NULL);
        printf("FIFO client...\n");

        if((msg = open(FIFO_NAME, O_WRONLY)) < 0)
        {
            fprintf(stderr, "%s: Невозможно открыть FIFO (%s)\n",
                __FILE__, strerror(errno));
            exit(-1);
        }
    }
}
```

Рис. 2.1: Код в файле client.c

```

client2.c      [----]  0 L: [ 1+ 0  1/ 35] *(0 / 701b) 0035 0x023 [*][X]
#include "common.h"

#define MESSAGE "Hello Server!!!\n"

int
main()
{
    int writefd, msglen, count;
    long long int t;
    char message[10];

    for(count=0; count<=5; ++count)
    {
        sleep(5);
        t=(long long int) time(0);
        sprintf(message, "%lli", t);
        if((writefd = open(FIFO_NAME, O_WRONLY)) < 0)
        {
            fprintf(stderr,"%s: Невозможно открыть FIFO (%s)\n",
                __FILE__, strerror(errno));
            exit(-1);
        }
    }
}

```

Рис. 2.2: Код в файле client2.c

```

common.h      [----]  0 L: [ 1+ 0  1/ 16] *(0 / 263b) 0035 0x023 [*][X]
#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>

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

#endif /* __COMMON_H__ */

```

Рис. 2.3: Код в файле common.h

```

server.c      [----]  0 L:[ 1+ 0  1/ 47] *(0  /1052b) 0035 0x023  [*][X]
#include "common.h"
int
main()
{
    int readfd;
    int n;
    char buff[MAX_BUFF];
    printf("FIFO Server...\n");

    if(mknod(FIFO_NAME, S_IFIFO | 0666, 0) < 0)
    {
        fprintf(stderr, "%s: Невозможно создать FIFO (%s)\n",
<-----> __FILE__, strerror(errno));
        exit(-1);
    }

    if((readfd = open(FIFO_NAME, O_RDONLY)) < 0)
    {
        fprintf(stderr, "%s: Невозможно открыть FIFO (%s)\n",
<-----> __FILE__, strerror(errno));
        exit(-2);
    }
}

```

Рис. 2.4: Код в файле server.c

- 2) Создал файл Makefile для запуска файлов. Код взял в файле для лабораторной и немного изменил его.

```

makefile      [----]  0 L:[ 1+ 0  1/ 10] *(0  / 154b) 0097 0x061  [*][X]
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

```

Рис. 2.5: Код в файле Makefile

- 3) Прописал команду make для создания программ. Создалось две нужные программы.


```

[aalushingfedora 14] $ ls
client2.c client.c common.h makefile server.c
[aalushingfedora 14] $ make
gcc server.c -o server
server.c:8: функции «main»:
server.c:13:15: предупреждение: неявная декларация функции «time» [-Wimplicit-function-declaration]
   23 |     clock_t now=time(NULL), start=time(NULL);
      |           ^~~~~
server.c:21:1: замечание: «time» is defined in header «time.h»; did you forget to «#include <time.h>»?
   21 |     #include "common.h"
      |     ^~~~~
server.c:26:18: предупреждение: неявная декларация функции «read»; имелось в виду «fread»? [-Wimplicit-function-declaration]
   26 |     while((n = read(readfd, buff, MAX_BUFF)) > 0)
      |                  ^~~~~
server.c:28:14: предупреждение: неявная декларация функции «write»; имелось в виду «fwrite»? [-Wimplicit-function-declaration]
   28 |         if(write(1, buff, n) != n)
      |            ^~~~~
server.c:37:13: предупреждение: неявная декларация функции «close»; имелось в виду «pclose»? [-Wimplicit-function-declaration]
   37 |         close(readfd);
      |         ^~~~~
server.c:39:6: предупреждение: неявная декларация функции «unlink» [-Wimplicit-function-declaration]
   39 |         if(unlink(FIFO_NAME) < 0)
      |         ^~~~~
gcc client.c -o client
client.c:8: функции «main»:
client.c:13:5: предупреждение: неявная декларация функции «sleep» [-Wimplicit-function-declaration]
   13 |     sleep(3);
      |     ^~~~~
client.c:14:5: предупреждение: неявная декларация функции «time» [-Wimplicit-function-declaration]
   14 |     t=time(NULL);
      |     ^~~~~
client.c:26:1: замечание: «time» is defined in header «time.h»; did you forget to «#include <time.h>»?
   26 |     #include "common.h"
      |     ^~~~~
client.c:26:15: предупреждение: неявная декларация функции «write»; имелось в виду «fwrite»? [-Wimplicit-function-declaration]
   26 |         if(write(msg, MESSAGE, len) != len)
      |            ^~~~~
client.c:32:5: предупреждение: неявная декларация функции «close»; имелось в виду «pclose»? [-Wimplicit-function-declaration]
   32 |         close(msg);
      |         ^~~~~
[aalushingfedora 14] $ ls
client client2.c client.c common.h makefile server server.c
[aalushingfedora 14] $

```

Рис. 2.6: Создание программ

- 4) Запустил с двух консолей программы client и server. Запуск производится с двух консолей, чтобы имитировать двух пользователей. Если хотя бы один из пользователей не будет доступен, то программа не будет работать.

```

client client2.c client.c common.h makefile server server.c
[aalushingfedora 14] $ ./server
FIFO Server...
Hello Server!!!
Hello Server!!!
[aalushingfedora 14] $ ./client
FIFO Client...
FIFO Client...

```

Рис. 2.7: Запуск программы

3 Контрольные вопросы

1. В чем ключевое отличие именованных каналов от неименованных?

Именованные каналы отличаются от неименованных наличием идентификатора канала, который представлен как специальный файл (соответственно имя именованного канала — это имя файла).

2. Возможно ли создание неименованного канала из командной строки?

Создание неименованного канала из командной строки возможно командой `pipe`.

3. Возможно ли создание именованного канала из командной строки?

Создание именованного канала из командной строки возможно с помощью `mkfifo`.

4. Опишите функцию языка C, создающую неименованный канал.

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. Опишите функцию языка C, создающую именованный канал.

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. Что будет в случае прочтения из `fifo` меньшего числа байтов, чем находится в канале? Большого числа байтов?

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 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=EP1PE`) (если процесс не установил обработки сигнала `SIGPIPE`, производится обработка по умолчанию – процесс завершается).

7. Аналогично, что будет в случае записи в `fifo` меньшего числа байтов, чем позволяет буфер? Большого числа байтов?

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=EPipe`) (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?

Два и более процессов могут читать и записывать в канал.

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)?

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. Опишите функцию `strerror`

Строковая функция `strerror` - функция языков C/C++, транслирующая код ошибки, который обычно хранится в глобальной переменной `errno`, в сообщение об ошибке, понятном человеку.

4 Выводы

Я приобрел практические навыки работы с именованными файлами.

Список литературы