

Именованные каналы

Лабораторная работа №14

Азимов М.

13 мая 2023

Российский университет дружбы народов, Москва, Россия

Информация

- Азимов Миразим
- студент 1 курса, группа НММбд-01-22
- Российский университет дружбы народов



Вводная часть

- Приобретение практических навыков работы с именованными каналами.

Изучить приведённые в тексте программы `server.c` и `client.c`. Взяв данные примеры за образец, написать 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?

Выполнение лабораторной работы №14

Внесение изменений в программы

```
[mazimov@fedora lab14]$ touch common.h  
[mazimov@fedora lab14]$ touch server.c  
[mazimov@fedora lab14]$ touch client.c  
[mazimov@fedora lab14]$ touch Makefile
```

Открыть ▼



common.h

~/work/os/lab14



```
/*  
 * common.h - заголовочный файл со стандартными определениями */  
#ifndef __COMMON_  
#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_BUFF 80  
#endif /* __COMMON_H__ */
```


Внесение изменений в программы

```
Открыть ▼ + server.c ~\work\os\lab14
/*
 * 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_BUFF]; /* буфер для чтения данных из FIFO */
    /* баннер */
    printf("FIFO Server-An");
    /* создаем файл FIFO с открытыми для всех
     * правами доступа на чтение и запись */
    if(mknod(FIFO_NAME, S_IFIFO | 0666, 0) < 0)
    {
        fprintf(stderr, "%s: Невозможно создать FIFO (%s)\n",
            __FILE__, strerror(errno));
        exit(-1);
    }
    /* откроем FIFO на чтение */
    if((readfd = open(FIFO_NAME, O_RDONLY)) < 0)
```

```
Открыть ▼ + client.c ~\work\os\lab14
/*
 * client.c - реализация клиента
 * чтобы запустить пример, необходимо:
 * 1. запустить программу server на одной консоли;
 * 2. запустить программу client на другой консоли.
 */
#include "common.h"
#define MESSAGE "Hello Server!!!\n"
int main()
{
    int writefd;
    /* дескриптор для записи в FIFO */
    int msglen;
    /* баннер */
    printf("FIFO Client-An");
    /*цикл, отвечающий за отправку сообщения о текущем времени */
    for(int i=0; i<4; i++)
    {
        /*получим доступ к FIFO*/
        if((writefd = open(FIFO_NAME, O_WRONLY)) < 0)
        {
            fprintf(stderr, "%s: Невозможно открыть FIFO (%s)\n", __FILE__,
                strerror(errno)); exit(-1);
```

Внесение изменений в программы

Открыть ▼



• Makefile

~/work/os/lab14

```
all: server client
```

```
server: server.c common.h
```

```
    gcc server.c -o server
```

```
client: client.c common.h
```

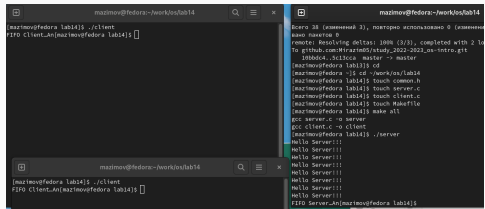
```
    gcc client.c -o client
```

```
clean:
```

```
    -rm server client *.o
```

```
[mazimov@fedora lab14]$ make all  
gcc server.c -o server  
gcc client.c -o client
```

Внесение изменений в программы



```
mazimov@fedora:~/work/os/lab14$ make all
gcc server.c -o server
gcc client.c -o client
[mazimov@fedora lab14]$ ./server
Hello Server!!!
Hello Server!!!
Hello Server!!!
Hello Server!!!
Hello Server!!!
Hello Server!!!
Hello Server!!!
Hello Server!!!
FIFO Server..An[mazimov@fedora lab14]$
[mazimov@fedora lab14]$ ./client
FIFO Client..An[mazimov@fedora lab14]$
```

```
[mazimov@fedora lab14]$ ./server
server.c: Невозможно создать FIFO (File exists)
FIFO Server..An[mazimov@fedora lab14]$
```

процесса A 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 записи указатель на файл (если он есть) увеличивается на количество действительно записанных байтов. Функция `write` возвращает число действительно записанных байтов. Возвращаемое значение должно быть

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.

Результаты

В ходе выполнения были приобретены навыки работы с именованными каналами.