

Именованные каналы

Лабораторная работа №14

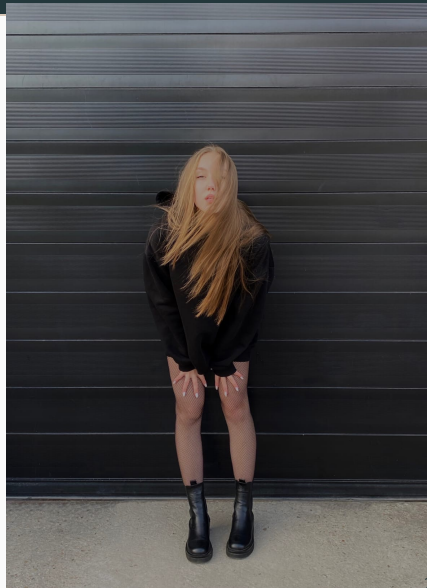
Миронова М. В.

13 мая 2023

Российский университет дружбы народов, Москва, Россия

Информация

- Миронова Мария Вадимовна
- студент 1 курса, группа НММбд-03-22
- 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

```
[mvmironova@fedora lab14]$ touch common.h
[mvmironova@fedora lab14]$ touch server.c
[mvmironova@fedora lab14]$ touch client.c
[mvmironova@fedora lab14]$ touch Mikefile
```

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)
    {
        fprintf(stderr, "%s: Невозможно открыть FIFO (%s)\n", __FILE__, strerror(errno));
        exit(-2);
    }
    /*начало отсчёта времени*/
    clock_t start = time(NULL);
    /*цикл работы пока с момента начала отсчёта времени прошло меньше 30 секунд*/
    while(time(NULL)-start < 30)
    /* читаем данные из FIFO и выводим на экран */
    {
        while((n = read(readfd, buff, MAX_BUFF)) > 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);
            break;
        }
        /*текущее время*/
        long int ttime=time(NULL);
        char* text=ctime(&ttime);
        /* передадим сообщение серверу */
        msglen = strlen(MESSAGE);
        if(write(writefd, MESSAGE, msglen) != msglen)
        {
            fprintf(stderr, "%s: Ошибка записи в FIFO (%s)\n",
                __FILE__, strerror(errno));
```

Внесение изменений в программы

```
Открыть ▼  • Mikefile  
~/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
```

```
[mvmironova@fedora lab14]$ make all
```

Внесение изменений в программы

```
mvmironova@fedora:~/work/os/lab14
[mvmironova@fedora lab14]$ ./client
FIFO Client_An[mvmironova@fedora lab14]$

mvmironova@fedora:~/work/os/lab14
Finished checking --- 4 code warnings
[mvmironova@fedora lab_prog]$ cd ../
[mvmironova@fedora os]$ mkdir lab14
[mvmironova@fedora os]$ cd lab14
[mvmironova@fedora lab14]$ touch common.h
[mvmironova@fedora lab14]$ touch server.c
[mvmironova@fedora lab14]$ touch client.c
[mvmironova@fedora lab14]$ touch Mikefile
[mvmironova@fedora lab14]$ make all
make: *** нет правила для сборки цели «all». Останов.
[mvmironova@fedora lab14]$ make all
make: *** нет правила для сборки цели «all». Останов.
[mvmironova@fedora lab14]$ gcc server.c -o server
[mvmironova@fedora lab14]$ gcc client.c -o client
[mvmironova@fedora lab14]$ ./server
Hello Server!!!
Hello Server!!!
Hello Server!!!
Hello Server!!!
Hello Server!!!
Hello Server!!!
Hello Server!!!
Hello Server!!!
FIFO Server_An[mvmironova@fedora lab14]$

mvmironova@fedora:~/work/os/lab14
[mvmironova@fedora lab14]$ ./client
FIFO Client_An[mvmironova@fedora lab14]$
```

```
[mvmironova@fedora lab14]$ ./server
server.c: Невозможно создать FIFO (File exists)
FIFO Server_An[mvmironova@fedora lab14]$
```

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

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

Результаты

В ходе выполнения были приобретены навыки работы с именованными каналами.