

Лабораторная работа №14

Ханина Людмила Константиновна

Table of Contents

Цель работы

Приобретение практических навыков работы с именованными каналами.

Задание

Изучите приведённые в тексте программы `server.c` и `client.c`. Взяв данные примеры за образец, напишите аналогичные программы, внося следующие изменения: * Работает не 1 клиент, а несколько (например, два). * Клиенты передают текущее время с некоторой периодичностью (например, раз в пять секунд). Используйте функцию `sleep()` для приостановки работы клиента. * 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

1. Создадим файлы с помощью команды
`touch common.h server.c client.c Makefile`
2. Вставим в них предложенные коды, но добавим некоторые вещи. Например, в файл `common.h` добавим заголовочные файлы `unistd.h` и `time.h`, которые пригодятся для других скриптов.

```
[lkkhanina@fedora ~]$ vi common.h
[lkkhanina@fedora ~]$ cat common.h
/*
 * common.h - заголовочный файл со стандартными определениями
 */

#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>
#include <time.h>
#include <unistd.h>

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

#endif /* __COMMON_H__ */
[lkkhanina@fedora ~]$
```

common.h

3. В `server.c` добавим цикл `while`, чтобы отслеживать время работы сервера. Разница между текущим временем `time(NULL)` и началом (`begin`) должна быть не больше 30 секунд.

```
clock_t begin = time(NULL);
while (time(NULL) - begin < 30)
{
    /* читаем данные из FIFO и выводим на экран */
    while((n = read(readfd, buff, MAX_BUFF)) > 0) {
        if(write(1, buff, n) != n)
        {
            fprintf(stderr, "%s: Ошибка вывода (%s)\n", __FILE__, strerror(errno));
            exit(-3);
        }
    }
}
close(readfd); /* закроем FIFO */

/* удалим FIFO из системы */
if(unlink(FIFO_NAME) < 0)
{
    fprintf(stderr, "%s: Невозможно удалить FIFO (%s)\n", __FILE__, strerror(errno));
}
-- INSERT --
```

server.c

4. В *client.c* добавим цикл, отвечающий за сообщения о текущем времени, и команду `sleep(5)`, которая будет приостанавливать работу клиента на 5 секунд.

```

/*
 * 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...\n");
    /* получим доступ к FIFO */

    for(int i = 0; i < 4; ++i) {
        if((writefd = open(FIFO_NAME, O_WRONLY)) < 0)
        {
            fprintf(stderr, "%s: Невозможно открыть FIFO (%s)\n",
                __FILE__, strerror(errno));
            exit(-1);
            break;
        }

        long int t = time(NULL);
        char* text = ctime(&t);

        /* передадим сообщение серверу */
        msglen = strlen(MESSAGE);
        if(write(writefd, MESSAGE, msglen) != msglen)
        {
            fprintf(stderr, "%s: Ошибка записи в FIFO (%s)\n",
                __FILE__, strerror(errno));
            exit(-2);
        }
    }
}
-- INSERT --

```

client.c

5. Makefile менять не будем.

```

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

```

Makefile

6. Компилируем коды с помощью команды `make all`.

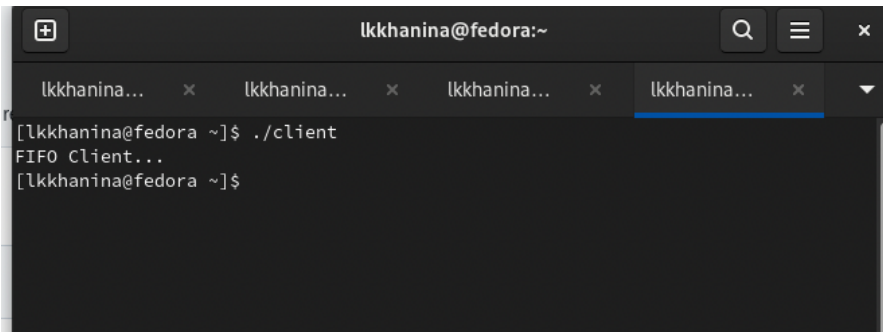
```
[lkkhanina@fedora ~]$ make all  
gcc client.c -o client
```

make all

7. Проверим работу скриптов. Запустим в одном терминале `./server`, а в нескольких других `./client`. Спустя 30 секунд сервер прекратить работу. Скрипт работает корректно.

```
[lkkhanina@fedora ~]$ ./server  
FIFO Server...  
Hello Server!!!  
Hello Server!!!  
Hello Server!!!  
Hello Server!!!  
Hello Server!!!  
Hello Server!!!  
Hello Server!!!  
Hello Server!!!  
Hello Server!!!  
Hello Server!!!  
Hello Server!!!  
Hello Server!!!  
Hello Server!!!  
[lkkhanina@fedora ~]$
```

Работа server.c



```
lkkhanina@fedora:~  
[lkkhanina@fedora ~]$ ./client  
FIFO Client...  
[lkkhanina@fedora ~]$
```

Другие терминалы

8. Если сервер завершит работу, не закрыв канал, то при повторном запуске сервера появится ошибка:

```
[lkkhanina@fedora ~]$ ./server  
FIFO Server...  
server.c: Невозможно создать FIFO (File exists)  
[lkkhanina@fedora ~]$
```

Ошибка

Контрольные вопросы

1. Именованные каналы, в отличие от неименованных, могут использоваться неродственными процессами. Они дают вам, по сути, те же возможности, что и неименованные каналы, но с некоторыми преимуществами, присущими обычным файлам. Именованные каналы используют специальную запись в директории для управления правами доступа.
2. Чтобы создать неименованный канал из командной строки, нужно использовать символ |.
3. Чтобы создать именованный канал, используется команда `mkfifo` либо команда `mknod` с типом файла `p`
4. Неименованный канал создается вызовом `pipe`, который заносит в массив `int [2]` два дескриптора открытых файлов. `fd[0]` – открыт на чтение, `fd[1]` – на запись (вспомните `STDIN == 0, STDOUT == 1`). Канал уничтожается, когда будут закрыты все файловые дескрипторы ссылающиеся на него.
5. Именованный канал FIFO доступен как объект в файловой системе. При этом, до открытия объекта FIFO на чтение, собственно коммуникационного объекта не создаётся. После открытия объекта FIFO в одном процессе на чтение, а в другом на запись, возникает ситуация полностью эквивалентная использованию неименованного канала.

Объект FIFO в файловой системе создаётся вызовом функции `int mkfifo(const char *pathname, mode_t mode);`

6. Если прочитать меньшее число байтов, то вернется требуемое число байтов, остаток сохраняется для последующих чтений. Если прочитать большее число байтов, чем находится в канале или FIFO, то вернется доступное число байтов.
7. Запись числа байтов, меньшего емкости канала или FIFO, гарантированно атомарно. Если записать число байтов большее, чем это позволяет канал или FIFO, вызов `write(2)` блокируется до освобождения требуемого места. При этом атомарность операции не гарантируется.
8. Да, могут. Количество процессов, которые могут параллельно присоединяться к любому концу канала, не ограничено.
9. Функция `write()` переписывает `count` байт из буфера, на который указывает `bufy` в файл, соответствующий дескриптору файла `handle`. Указателю положения в файле дается приращение на количество записанных байт. Если файл открыт в текстовом режиме, то символы перевода строки автоматически дополняются символами возврата каретки. Тем не менее `_rtl_write()` не производит таких дополнений. Возвращаемым значением является количество действительно записанных байт. Если встретится ошибка, это количество может быть меньше, чем `count`. В случае ошибки возвращается — 1. Единица в вызове функции `write` в `server.c` означает идентификатор стандартного потока вывода.

10. Интерпретирует номер ошибки, передаваемый в функцию в качестве аргумента — `errno`, 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`.

Выводы

Я научилась работать с именованными каналами и изменять скрипты так, чтобы они выполняли определенные вещи.