

Цель работы

Приобретение практических навыков работы с именованными каналами.

Ход работы

Изучили приведенные в тексте программы `server.c`, `client.c`. Взяв данные примеры за образец, написали аналогичные программы, внося следующие изменения

Работает два клиента, а не один. Клиенты передают текущее время с некоторой периодичностью. Использовали функцию `sleep()` для приостановки работы клиента. Сервер работает не бесконечно, а прекращает работу через 30 секунд. Использовали функцию `clock()` для определения времени работы сервера.

Вывод

Приобрели практические навыки работы с именованными каналами.

Контрольные вопросы

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

2. Для создания неименованного канала используется системный вызов `pipe`. Массив из двух целых чисел является выходным параметром этого системного вызова.

3. Вы можете создавать именованные каналы из командной строки и внутри программы. С давних времен программой создания их в командной строке была команда: `mknod - $ mknod имя_файла`, однако команды `mknod` нет в списке команд `X/Open`, поэтому она включена не во все UNIX-подобные системы. Предпочтительнее применять в командной строке - `$ mkfifo имя_файла`.

4. `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. `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. При чтении меньшего числа байтов, чем находится в канале, возвращается требуемое число байтов, остаток сохраняется для последующих чтений. При чтении большего числа байтов, чем находится в канале или FIFO возвращается доступное число байтов.

7. При записи большого 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. 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.

8. В общем случае возможна многонаправленная работа процессов с каналом, т.е. возможна ситуация, когда с одним и тем же каналом взаимодействуют два и более процесса, и каждый из взаимодействующих каналов пишет и читает информацию в канал. Но традиционной

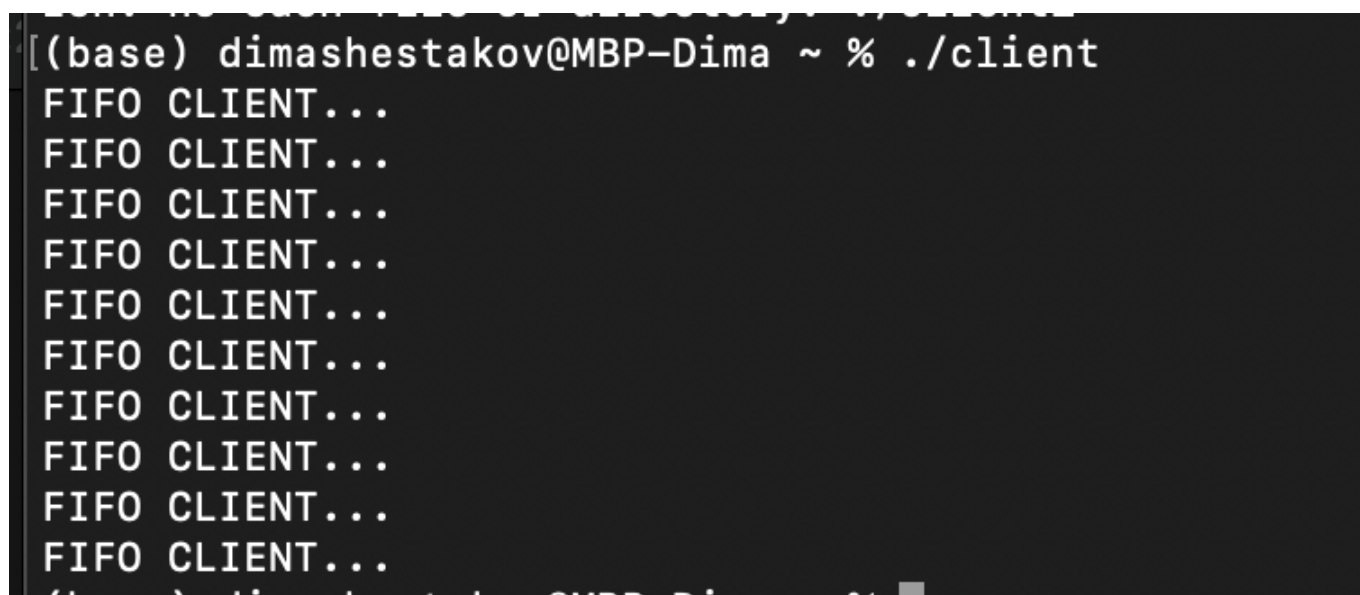
схемой организации работы с каналом является однонаправленная организация, когда канал связывает два, в большинстве случаев, или несколько взаимодействующих процесса, каждый из которых может либо читать, либо писать в канал.

9. `Write` - Функция записывает `length` байтов из буфера `buffer` в файл, определенный дескриптором файла `fd`. Эта операция чисто 'двоичная' и без буферизации. Реализуется как непосредственный вызов `DOS`. С помощью функции `write` мы посылаем сообщение клиенту или серверу.

10. Строковая функция `strerror` - функция языков C/C++, транслирующая код ошибки, который обычно хранится в глобальной переменной `errno`, в сообщение об ошибке, понятном человеку. Ошибки эти возникают при вызове функций стандартных Си-библиотек.

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.

Приложение



```
(base) dimashestakov@MBP-Dima ~ % ./client
FIFO CLIENT...
FIFO CLIENT...
FIFO CLIENT...
FIFO CLIENT...
FIFO CLIENT...
FIFO CLIENT...
FIFO CLIENT...
FIFO CLIENT...
FIFO CLIENT...
```

Рис.1

```

Last login: Sat Jun 12 19:29:48 on ttys000
(base) dimashestakov@MBP-Dima ~ % rm fifo
(base) dimashestakov@MBP-Dima ~ % ./server
FIFO SERVER...

Server timeout
30 seconds passed%
(base) dimashestakov@MBP-Dima ~ %

```

Рис.2

```

#include "common.h"

#define MESSAGE "client1!!\n"

int main(){
    for(int i = 0; i < 10; ++i){
        FILE* writefd;
        int msglen;
        long int ttime;
        ttime = time(NULL);

        printf("FIFO CLIENT...\n");

        if((writefd = fopen(FIFO_NAME, "w+")) == NULL){
            fprintf(stderr, "%s: It's impossible to open FIFO (%s)\n", __FILE__, strerror(errno));
            exit(-1);
        }

        msglen = strlen(MESSAGE);

        if(fwrite(MESSAGE, msglen, 1, writefd) != 1){
            fprintf(stderr, "%s: FIFO write error (%s)\n", __FILE__, strerror(errno));
            exit(-2);
        }

        sleep(5);
        fclose(writefd);
    }
    exit(0);
}

```

Рис.3

```
#include "common.h"

#define MESSAGE "client2!!!\n"

int main(){
    for(int i = 0; i < 10; ++i){
        FILE* writefd;
        int msglen;
        long int ttime;
        ttime = time(NULL);
        (char [16])"FIFO CLIENT...\n"
        printf("FIFO CLIENT...\n");

        if((writefd = fopen(FIFO_NAME, "w+")) == NULL){
            fprintf(stderr, "%s; It's impossible to open FIFO (%s)\n", __FILE__, strerror(errno));
            exit(-1);
        }

        msglen = strlen(MESSAGE);

        if(fwrite(MESSAGE, msglen, 1, writefd) != 1){
            fprintf(stderr, "%s: FIFO write error (%s)\n", __FILE__, strerror(errno));
            exit(-2);
        }

        sleep(4);
        fclose(writefd);
    }
    exit(0);
}
```

Рис.4

```
#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 <zconf.h>

#define FIFO_NAME "/Users/dimashestakov/fifo"
#define MAX_BUFF 1024

#endif /* __COMMON_H__ */
```

Рис.5

```
#include "common.h"

int main() {
    FILE* readfd = NULL;
    int n;
    char buff[MAX_BUFF];

    printf("FIFO SERVER...\n");
    if(mknod(FIFO_NAME, S_IFIFO | 0600, 0) < 0){
        fprintf(stderr, "%s: It's impossible to create FIFO (%s)\n", __FILE__, strerror(errno));
        exit(-1);
    }

    if((readfd = fopen(FIFO_NAME, "r")) == NULL){
        fprintf(stderr, "%s: It's impossible to open FIFO (%s)\n", __FILE__, strerror(errno));
        exit(-2);
    }

    clock_t now = time(NULL), start = time(NULL);

    while(now - start < 30){
        if((n = fread(buff, MAX_BUFF, 1, readfd)) > 0){
            if(fwrite(buff, MAX_BUFF, n, readfd) != n)
            {
                fprintf(stderr, "%s: Input error (%s)\n", __FILE__, strerror(errno));
                exit(-3);
            }
        }
        now = time(NULL);
    }

    printf("\nServer timeout\n%lu seconds passed", (now - start));
    fclose(readfd);

    if(remove(FIFO_NAME) != 0)
    {
        fprintf(stderr, "%s: It's impossible to delete FIFO (%s)\n", __FILE__, strerror(errno));
        exit(-4);
    }

    exit(0);
}
```

Рис.6

```
all: server client1 client2

server: server.c common.h
    gcc server.c -o server

client1: client1.c common.h
    gcc client1.c -o client

client2: client2.c common.h
    gcc client2.c -o client2

clean:
    -rm server client1 client2 *.o
```

Рис.7