

Операционные системы

Лабораторная работа №14

Матюшкин Денис Владимирович (НПИбд-02-21)

Содержание

1	Цель работы	3
2	Ход работы	4
3	Контрольные вопросы	9
4	Вывод	11

1 Цель работы

- Приобретение практических навыков работы с именованными каналами.

2 Ход работы

1. Изучим приведённые в тексте программы `server.c` и `client.c`. Взяв данные примеры за образец, напишем аналогичные программы, внося следующие изменения:

- Работает не 1 клиент, а несколько (например, два).
- Клиенты передают текущее время с некоторой периодичностью (раз в пять секунд). Используем функцию `sleep()` для приостановки работы клиента.
- Сервер работает не бесконечно, а прекращает работу через некоторое время (30 сек). Используем не функцию `clock()`, а взятие `unixTime` (`time(NULL)`). Дело в том, что `clock()` не засчитывает такты `sleep`. С методом `time()` работает все хорошо. Если сервер завершит свою работу, не закрыв канал, то при повторном запуске сервера будет выводить ошибку: “Невозможно создать FIFO”.

Измененный файл `server.c` (рис. 2.1 и рис. 2.2).

Измененный файл `client.c` (рис. 2.3 и рис. 2.4).

Файл `common.h` (рис. 2.5).

Файл `Makefile` (рис. 2.6).

```

#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...\n");
    /* 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);
    }
}

```

Рис. 2.1: Измененный server.c файл ч.1

```

/*Заведём переменные обозначающие время начала работы сервера и текущее время*/
clock_t now=time(NULL), start=time(NULL);
while(now-start<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);
        }
    }
    now=time(NULL);
}
printf("Времени прошло: %d с.\n", (now-start));
close(readfd); /* закроем FIFO */

/* удалим FIFO из системы */
if(unlink(FIFO_NAME) < 0)
{
    fprintf(stderr, "%s: Невозможно удалить FIFO (%s)\n",
            __FILE__, strerror(errno));
    exit(-4);
}
exit(0);
}

```

Рис. 2.2: Измененный server.c файл ч.2

```

#include "common.h"

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

int
main()
{
    int writefd; /* дескриптор для записи в FIFO */
    int msglen;
    int i;
    long int ttime;

    for(int i=0; i<5; i++)
    {
        sleep(5);
        ttime=time(NULL);
        /* баннер */
        printf("FIFO Client...\n");
    }
}

```

Рис. 2.3: Измененный client.c файл ч.1

```

/* 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);
}

/* 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);
}

close(writefd);
}
/* закроем доступ к FIFO */
exit(0);
}

```

Рис. 2.4: Измененный client.c файл ч.2

```

#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>

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

#endif /* __COMMON_H__ */

```

Рис. 2.5: Файл common.h

```

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. 2.6: Файл Makefile

2. Пропишем make в консоль, в итоге у нас должны создаваться исполняемые файлы (рис. 2.7).

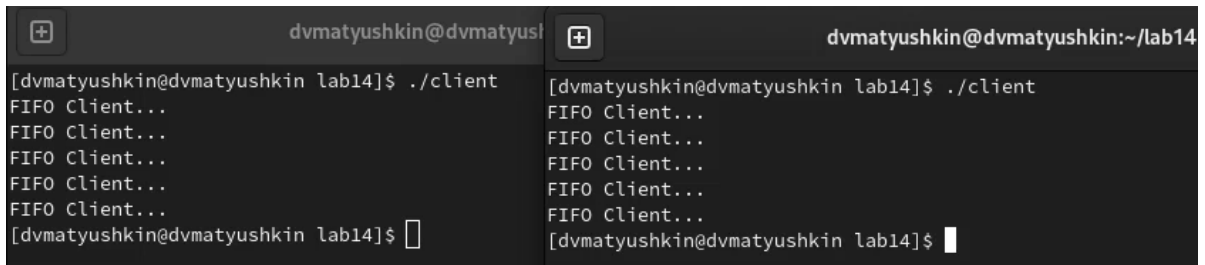
```

[dvmatyushkin@dvmatyushkin lab14]$ ls
client client.c common.h Makefile server server.c
[dvmatyushkin@dvmatyushkin lab14]$

```

Рис. 2.7: Использование Makefile

3. Проверим работу файлов (рис. 2.8 и рис. 2.9).

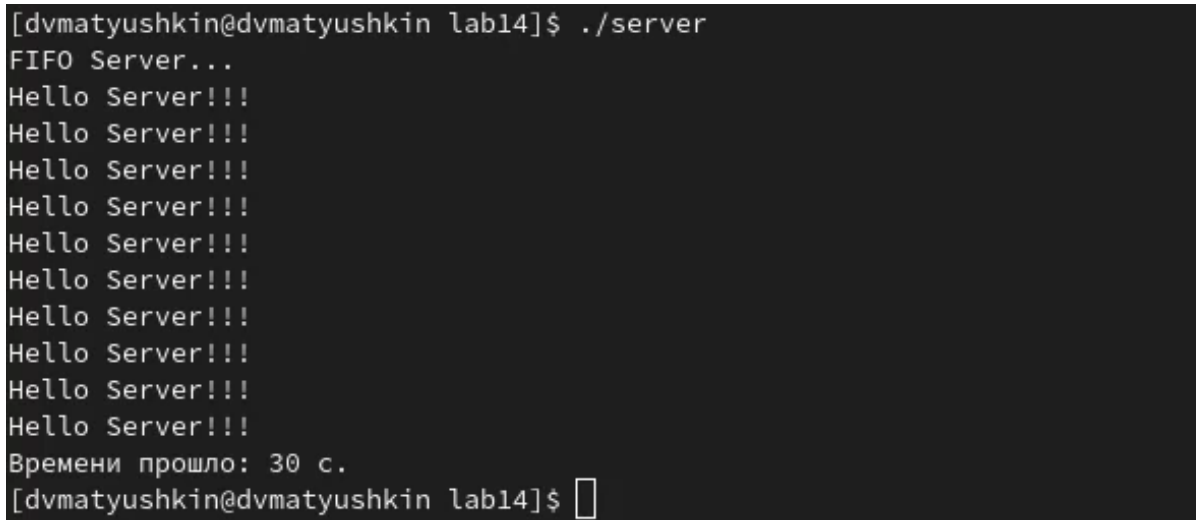


The image shows two terminal windows. The left window has a title bar 'dvmatyushkin@dvmatyushkin' and shows the command './client' being executed, resulting in five 'FIFO Client...' messages. The right window has a title bar 'dvmatyushkin@dvmatyushkin:~/lab14' and shows the same command and output.

```
dvmatyushkin@dvmatyushkin lab14]$ ./client
FIFO Client...
FIFO Client...
FIFO Client...
FIFO Client...
FIFO Client...
[dvmatyushkin@dvmatyushkin lab14]$
```

```
dvmatyushkin@dvmatyushkin:~/lab14$ ./client
FIFO Client...
FIFO Client...
FIFO Client...
FIFO Client...
FIFO Client...
[dvmatyushkin@dvmatyushkin lab14]$
```

Рис. 2.8: Проверка client.c файла



The image shows a terminal window with a title bar 'dvmatyushkin@dvmatyushkin lab14'. It shows the command './server' being executed, resulting in ten 'Hello Server!!!' messages and a message 'Времени прошло: 30 с.'.

```
[dvmatyushkin@dvmatyushkin lab14]$ ./server
FIFO Server...
Hello Server!!!
Hello Server!!!
Hello Server!!!
Hello Server!!!
Hello Server!!!
Hello Server!!!
Hello Server!!!
Hello Server!!!
Hello Server!!!
Hello Server!!!
Времени прошло: 30 с.
[dvmatyushkin@dvmatyushkin lab14]$
```

Рис. 2.9: Проверка server.c файла

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?

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

3. Возможно ли создание именованного канала из командной строки?

Да

4. Опишите функцию языка C, создающую 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.

```
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. Аналогично, что будет в случае записи в fifo меньшего числа байтов, чем позволяет буфер? Большого числа байтов? При записи большего числа байтов, 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, гарантированно атомарно. Это означает, что в случае, когда несколько процессов одновременно записывают в канал, данные будут записывать отдельно и не будут “повреждать” структуру файла.

8. Могут ли два и более процессов читать или записывать в канал? Могут

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 записывает length байтов из буфера buffer в файл, определенный дескриптором файла fd. Эта операция двоичная без буферизации. Реализуется как непосредственный вызов DOS. С помощью функции write мы посылаем сигнал клиенту или серверу

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 Вывод

- В ходе этой лабораторной работы мы приобрели практические навыки работы с именованными каналами.