

Отчёт по лабораторной работе №15

Именованные каналы

Владимир Александрович Пушкарев НПМбд-02-20

Содержание

1	Цель работы	4
2	Выполнение лабораторной работы	5
3	Вывод	10
4	Контрольные вопросы	11

List of Figures

2.1 Запуск клиента и сервера 9

1 Цель работы

Приобретение практических навыков работы с именованными каналами.

2 Выполнение лабораторной работы

Пишем и редактируем программы на С, так чтобы на одном сервере можно было запускать сначала один клиент. Далее напишем и отредактируем программы, так чтобы на одном сервере можно было запускать больше серверов, чем один, интервал между клиентами будет 5 секунд, сервер завершится через 30 секунд. Мы имеем 4 файла (программы) это заголовочный файл (common.h) клиент (client.c), сервер (server.c), и Makefile.

```
////////////////////////////////////
// common.h
#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>
#define FIFO_NAME "/tmp/fifo"
#define MAX_BUFF 80
#endif /* __COMMON_H__ */

////////////////////////////////////
```

```

// client.c
#include "common.h"
#define MESSAGE "Hello Server!!! \n"
int main ()
{
    int writefd;
    int msglen;
    printf("FIFO Client...\n");
    if ((writefd = open(FIFO_NAME, O_WRONLY)) < 0)
    {
        fprintf(stderr, "%s: Невозможно открыть FIFO (%s)\n",
            __FILE__, strerror(errno));
        exit(-1);
    }
    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);
    exit(0);
}

////////////////////////////////////
// server.c
#include "common.h"
void display() {
    printf("/n Server timeout...%u seconds passed!\n Total elapsedesed time

```

```

}
int main()
{
    clock_t start, now;
    start = time(NULL);
    int readfd;
    int n;
    char buff[MAX_BUFF];
    printf("Hell server...\n\n");
    if(mknod(FIFO_NAME, S_IFIFO | 0666, 0) < 0)
    {
        fprintf(stderr, "%s: Unable to create FiFO(%s)\n", __FILE__, strerror(errno));
        exit(-1);
    }
    if((readfd = open(FIFO_NAME, O_RDONLY)) < 0)
    {
        fprintf(stderr, "%s: Unable to open FIFO (%s)\n", __FILE__, strerror(errno));
        exit(-2);
    }
    for(;;)
    {
        while((n = read(readfd, buff, MAX_BUFF)) > 0)
        {
            if(write(1, buff, n) != n)
            {
                fprintf(stderr, "%s: Input error (%s)\n", __FILE__, strerror(errno));
            }
            sleep(5);
        }
    }
}

```

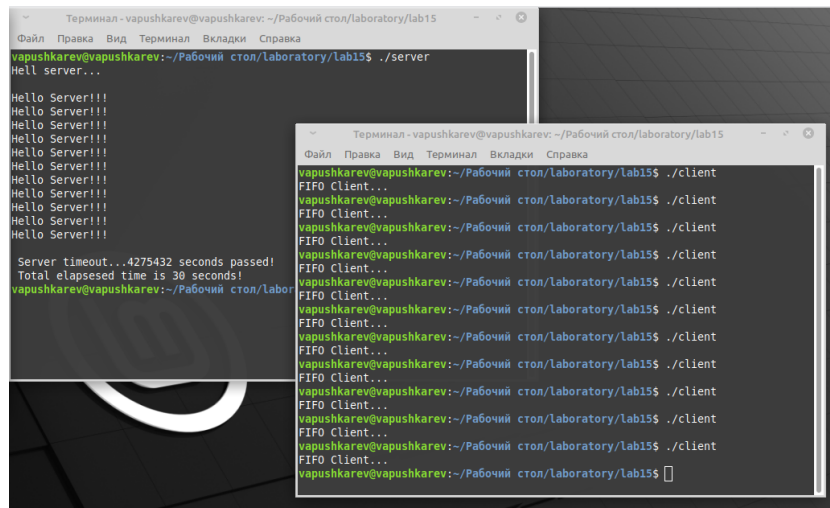
```

now=time(NULL);
if (now-start>30)
{
display();
return 0;
}
}
close(readfd);
if(unlink(FIFO_NAME) < 0)
{
fprintf(stderr, "%s: unable to delete FIFO (%s)\n", __FILE__, strerror(errno));
exit(-4);
}
exit(0);
}

////////////////////////////////////
// makefile
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

```

Запуск проекта



The image shows two terminal windows from a Linux environment. The top window is titled 'Терминал - vapushkarev@vapushkarev: ~/Рабочий стол/Laboratory/lab15'. It shows the execution of a server program with the command `./server`. The output includes 'Hell server...', followed by ten 'Hello Server!!!' messages, and then a timeout message: 'Server timeout...4275432 seconds passed! Total elapsed time is 30 seconds!'. The bottom window is also titled 'Терминал - vapushkarev@vapushkarev: ~/Рабочий стол/Laboratory/lab15'. It shows the execution of a client program with the command `./client`. The output consists of ten 'FIFO Client...' messages. The terminal windows are overlaid on a dark background with a faint grid pattern.

```
Терминал - vapushkarev@vapushkarev: ~/Рабочий стол/Laboratory/lab15
vapushkarev@vapushkarev:~/Рабочий стол/Laboratory/lab15$ ./server
Hell server...
Hello Server!!!
Hello Server!!!
Hello Server!!!
Hello Server!!!
Hello Server!!!
Hello Server!!!
Hello Server!!!
Hello Server!!!
Hello Server!!!
Hello Server!!!
Server timeout...4275432 seconds passed!
Total elapsed time is 30 seconds!
vapushkarev@vapushkarev:~/Рабочий стол/Laboratory/lab15$

Терминал - vapushkarev@vapushkarev: ~/Рабочий стол/Laboratory/lab15
vapushkarev@vapushkarev:~/Рабочий стол/Laboratory/lab15$ ./client
FIFO Client...
vapushkarev@vapushkarev:~/Рабочий стол/Laboratory/lab15$ ./client
FIFO Client...
vapushkarev@vapushkarev:~/Рабочий стол/Laboratory/lab15$ ./client
FIFO Client...
vapushkarev@vapushkarev:~/Рабочий стол/Laboratory/lab15$ ./client
FIFO Client...
vapushkarev@vapushkarev:~/Рабочий стол/Laboratory/lab15$ ./client
FIFO Client...
vapushkarev@vapushkarev:~/Рабочий стол/Laboratory/lab15$ ./client
FIFO Client...
vapushkarev@vapushkarev:~/Рабочий стол/Laboratory/lab15$ ./client
FIFO Client...
vapushkarev@vapushkarev:~/Рабочий стол/Laboratory/lab15$ ./client
FIFO Client...
vapushkarev@vapushkarev:~/Рабочий стол/Laboratory/lab15$ ./client
FIFO Client...
vapushkarev@vapushkarev:~/Рабочий стол/Laboratory/lab15$
```

Figure 2.1: Запуск клиента и сервера

3 Вывод

В данной работе мы приобрели практические навыки работы с именованными каналами по типу клиент-сервер.

4 Контрольные вопросы

1. В чем ключевое отличие именованных каналов от неименованных? Ответ:
У именованных каналов есть идентификатор канала, а у неименованных его нет.

2. Возможно ли создание неименованного канала из командной строки? Ответ: Возможно создание неименованного канала из командной строки, но только с созданием временного канала с индикатором.

3. Возможно ли создание именованного канала из командной строки? Ответ: Да. При помощи `mknod`.

4. Опишите функцию языка C, создающую неименованный канал. Ответ:

```
#include int fd[2];
```

```
pipe(fd);
```

```
/* возвращает 0 в случае успешного завершения, -1 - в случае ошибки;*/
```

Это значит, что функция возвращает два файловых дескриптора: `fd[0]` и `fd[1]`

5. Опишите функцию языка C, создающую именованный канал. Ответ:

```
#include <sys/types.h>
```

```
#include <sys/stat.h>
```

```
int mkfifo(const char *pathname, mode_t mode);
```

6. . Что будет в случае прочтения из `fifo` меньшего числа байтов, чем находится в канале? Большого числа байтов? Ответ: При чтении меньшего числа байтов, чем находится в канале или FIFO, возвращается требуемое число байтов,

остаток сохраняется для последующих чтений. При чтении большего числа байтов, чем находится в канале или FIFO, возвращается доступное число байтов. Процесс, читающий из канала, должен соответствующим образом обработать ситуацию, когда прочитано меньше, чем требуется в программе.

7. Аналогично, что будет в случае записи в `fifo` меньшего числа байтов, чем позволяет буфер? Большого числа байтов? Ответ: Запись числа байтов, меньшего числа битов у канала или FIFO, в случае, когда несколько процессов одновременно записывают в канал, порции данных от этих процессов не перемешиваются. При записи большего числа байтов, чем это позволяет канал или FIFO, вызов `write(2)` блокируется до освобождения занятой нами до этого памяти.
8. Могут ли два и более процессов читать или записывать в канал? Ответ: Да. Если у `buff` достаточное количество памяти.
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)? Ответ: Функция записывает `length` памяти из буфера `buffer` в файл, определенный дескриптором файла `fd`. Эта операция чисто 'двоичная' и без буферизации. При единице возвращает действительное число байтов. При -1 сообщение об ошибке.
10. Опишите функцию `strerror` Ответ: 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`. 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.