

Лабораторная работа №13

Именованные каналы

Чемоданова Ангелина Александровна

Содержание

1	Цель работы	4
2	Задание	5
3	Теоретическое введение	6
4	Выполнение лабораторной работы	7
5	Выводы	11
6	Контрольные вопросы	12

Список иллюстраций

4.1	client.c	7
4.2	client2.c	8
4.3	common.h	9
4.4	makefile	9
4.5	main.c	10
4.6	Запуск	10

1 Цель работы

Приобретение практических навыков работы с именованными каналами.

2 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?

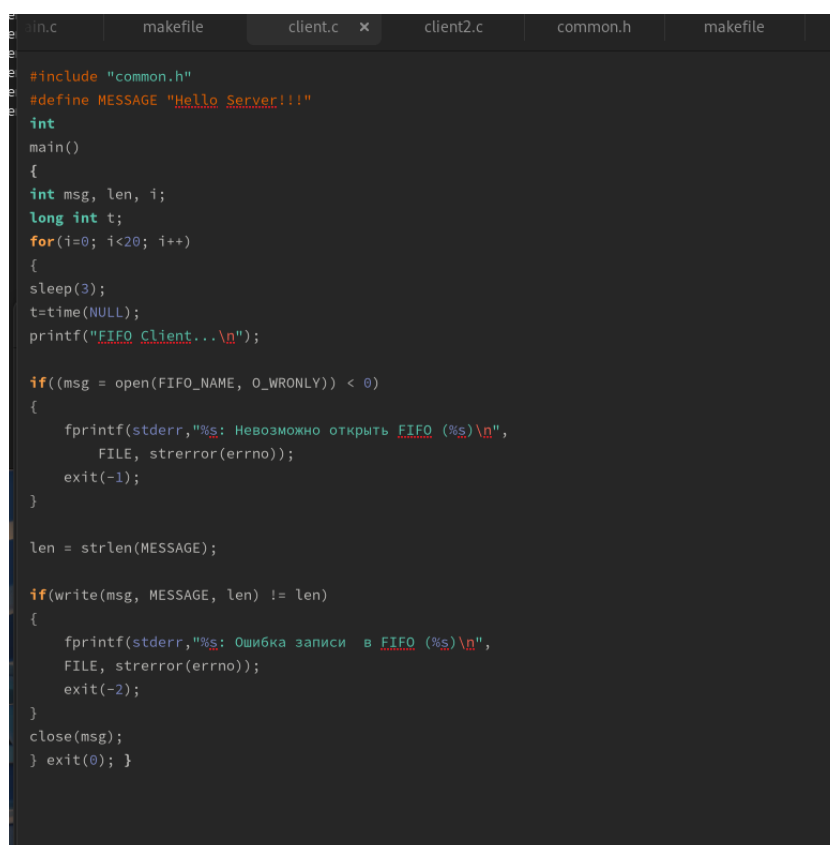
3 Теоретическое введение

Командный процессор (командная оболочка, интерпретатор команд shell) — это программа, позволяющая пользователю взаимодействовать с операционной системой компьютера. В операционных системах типа UNIX/Linux наиболее часто используются следующие реализации командных оболочек: – оболочка Борна (Bourne shell или sh) — стандартная командная оболочка UNIX/Linux, содержащая базовый, но при этом полный набор функций; – C-оболочка (или csh) — надстройка на оболочкой Борна, использующая C-подобный синтаксис команд с возможностью сохранения истории выполнения команд; – оболочка Корна (или ksh) — напоминает оболочку C, но операторы управления программой совместимы с операторами оболочки Борна; – BASH — сокращение от Bourne Again Shell (опять оболочка Борна), в основе своей совмещает свойства оболочек C и Корна (разработка компании Free Software Foundation). POSIX (Portable Operating System Interface for Computer Environments) — набор стандартов описания интерфейсов взаимодействия операционной системы и прикладных программ.

4 Выполнение лабораторной работы

Изучите приведённые в тексте программы server.c и client.c. Взяв данные примеры за образец, напишите аналогичные программы, внося следующие изменения:

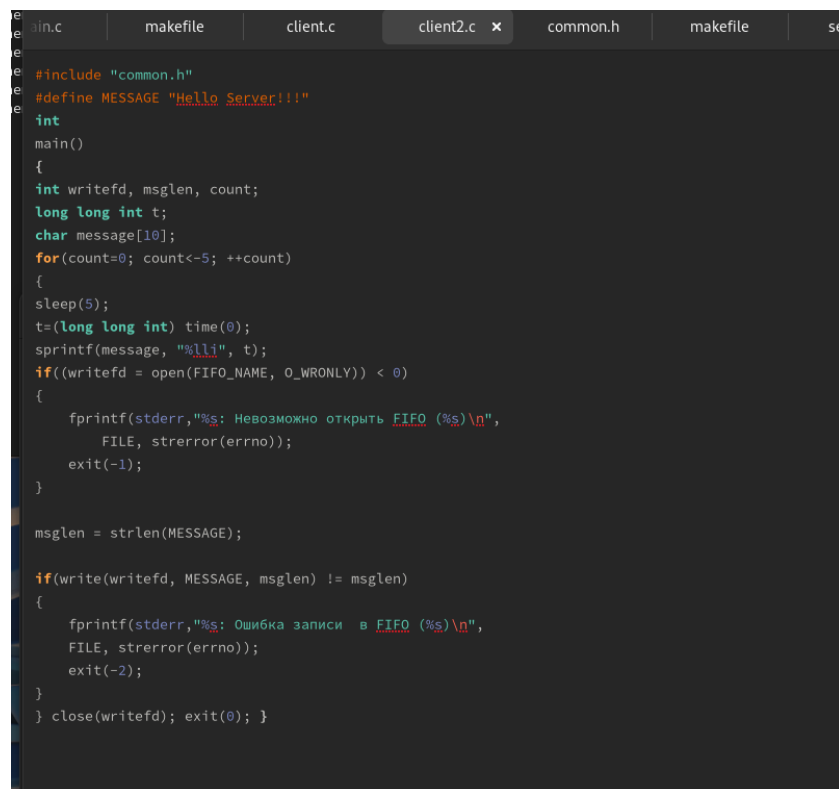
Работает не 1 клиент, а несколько (например, два). Клиенты передают текущее время с некоторой периодичностью (например, раз в пять секунд).(рис. 4.1).



```
1  #include "common.h"
2  #define MESSAGE "Hello Server!!!"
3
4  int
5  main()
6  {
7      int msg, len, i;
8      long int t;
9      for(i=0; i<20; i++)
10     {
11         sleep(5);
12         t=time(NULL);
13         printf("FIFO client...\n");
14
15         if((msg = open(FIFO_NAME, O_WRONLY)) < 0)
16         {
17             fprintf(stderr,"%s: Невозможно открыть FIFO (%s)\n",
18                     FILE, strerror(errno));
19             exit(-1);
20         }
21
22         len = strlen(MESSAGE);
23
24         if(write(msg, MESSAGE, len) != len)
25         {
26             fprintf(stderr,"%s: Ошибка записи в FIFO (%s)\n",
27                     FILE, strerror(errno));
28             exit(-2);
29         }
30         close(msg);
31     } exit(0); }
```

Рис. 4.1: client.c

Используйте функцию `sleep()` для приостановки работы клиента. (рис. 4.2).

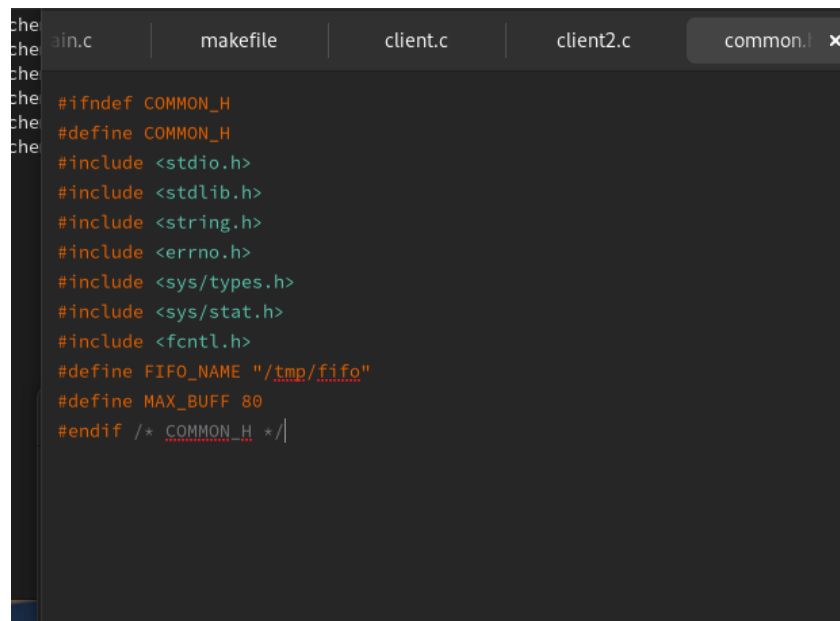


```
client2.c
#include "common.h"
#define MESSAGE "Hello Server!!!"
int
main()
{
    int writefd, msglen, count;
    long long int t;
    char message[10];
    for(count=0; count<-5; ++count)
    {
        sleep(5);
        t=(long long int) time(0);
        sprintf(message, "%lli", t);
        if((writefd = 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); }
```

Рис. 4.2: client2.c

Сервер работает не бесконечно, а прекращает работу через некоторое время (например, 30 сек). (рис. 4.3).



```

#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 */

```

Рис. 4.3: common.h

Используйте функцию `clock()` для определения времени работы сервера. (рис. 4.4).



```

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. 4.4: makefile

main.c (рис. 4.5).

```
main.c | makefile | client.c | client2.c | common.h | makefile |
#include "common.h"
int
main()
{
    int readfd;
    int n;
    char buff[MAX_BUFF];
    printf("FIFO Server...");
    if(mknod(FIFO_NAME, S_IFIFO | 0666, 0) < 0)
    {
        fprintf(stderr, "%s: Невозможно создать FIFO (%s)",
            FILE, strerror(errno));
        exit(-1);
    }
    if((readfd = open(FIFO_NAME, O_RDONLY)) < 0)
    {
        fprintf(stderr, "%s: Невозможно открыть FIFO (%s)",
            FILE, strerror(errno));
        exit(-2);
    }
    clock_t now=time(NULL), start=time(NULL);
    while(now-start<30)
    {
        while((n = read(readfd, buff, MAX_BUFF)) > 0)
        {
            if(write(1, buff, n) != n)
            {
                fprintf(stderr, "%s: Ошибка вывода (%s)",
                    FILE, strerror(errno));
            }
        }
        now=time(NULL);
    }
    printf("server timeout, %li - seconds passed",
        (now-start));
    close(readfd);
    if(unlink(FIFO_NAME) < 0)
```

Рис. 4.5: main.c

Что будет в случае, если сервер завершит работу, не закрыв канал? Запуск. (рис. 4.6).

```
./ - Oct 10 18:05:
[aachemodanova@fedora vvv]$ make
gcc server.c -o server
server.c: В функции «main»:
```

Рис. 4.6: Запуск

5 Выводы

Мы приобрели практические навыки работы с именованными каналами.

6 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? Создание неименованного канала из командной строки возможно командой `pipe`.
3. Возможно ли создание именованного канала из командной строки? Создание именованного канала из командной строки возможно с помощью `mkfifo`.

Опишите функцию языка C, создающую неименованный канал. Функция языка C, создающая 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).

4. Опишите функцию языка C, создающую именованный канал. Функция языка C, создающая 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);

5. Что будет в случае прочтения из fifo меньшего числа байтов, чем находится в канале? Большого числа байтов?
6. При чтении меньшего числа байтов, возвращается требуемое число байтов, остаток сохраняется для следующих чтений. При чтении большего числа байтов, возвращается доступное число байтов 7. 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. При записи большего 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 SIGPIPE, а вызов write(2) возвращает 0 с установкой ошибки (errno=EPipe) (если процесс не установил обработки сигнала SIGPIPE, производится обработка по умолчанию — процесс завершается).
7. Могут ли два и более процессов читать или записывать в канал? Два и более процессов могут читать и записывать в канал.
8. 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 write записывает length байтов из буфера buffer в файл, определенный дескриптором файла fd. Эта операция чисто 'двоичная' и без буферизации. При единице возвращает действительное число байтов. Функция write возвращает число действительно записанных в файл байтов или -1 при ошибке, устанавливая при этом errno.
9. 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.