Лабораторная работа №14

Именованные каналы

Мажитов Магомед Асхабович

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

Приобретение практических навыков работы с именованными каналами.

2 Задачи

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

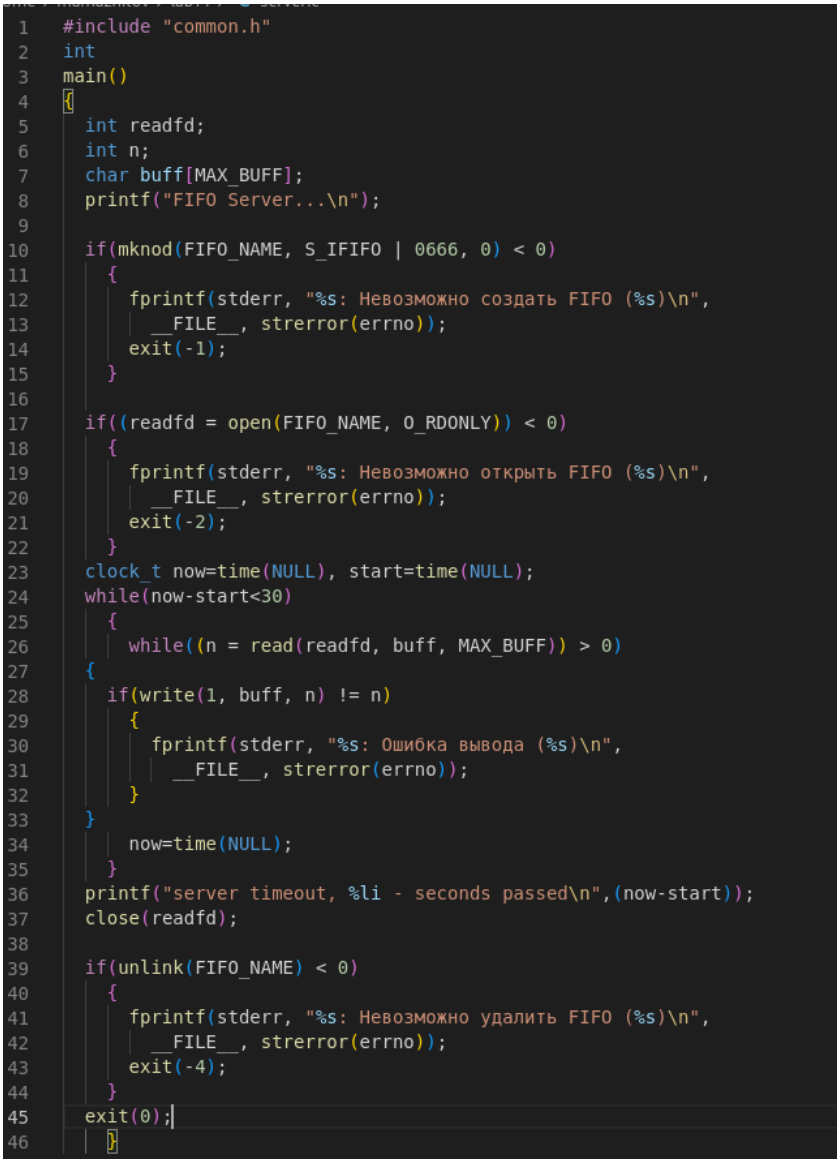
1. Работает не 1 клиент, а несколько (например, два).
2. Клиенты передают текущее время с некоторой периодичностью (например, раз в пять секунд). Используйте функцию sleep() для приостановки работы клиента.
3. Сервер работает не бесконечно, а прекращает работу через некоторое время (например, 30 сек). Используйте функцию clock() для определения времени работы сервера. Что будет в случае, если сервер завершит работу, не закрыв канал?

3 Ход работы

1. Создал файлы *common.h*, *server.c*, *client.c*, *client2.c*. Скопировал основной код из теоретической части лабораторной работы и немного подкорректировал его.(рис. 3.1, 3.2, 3.3, 3.4)

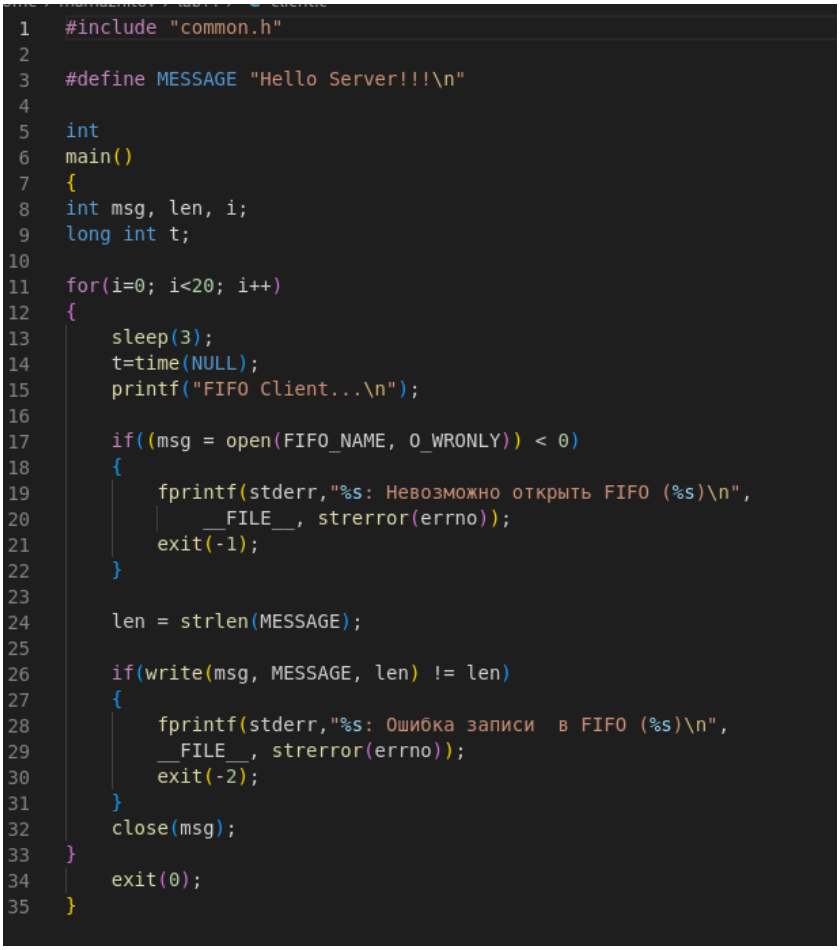
```
1   #ifndef __COMMON_H__
2   #define __COMMON_H__
3
4   #include <stdio.h>
5   #include <stdlib.h>
6   #include <string.h>
7   #include <errno.h>
8   #include <sys/types.h>
9   #include <sys/stat.h>
10  #include <fcntl.h>
11
12  #define FIFO_NAME "/tmp/fifo"
13  #define MAX_BUFF 80
14
15  #endif /* __COMMON_H__ */
16
```

Рис. 3.1: Файл common.h



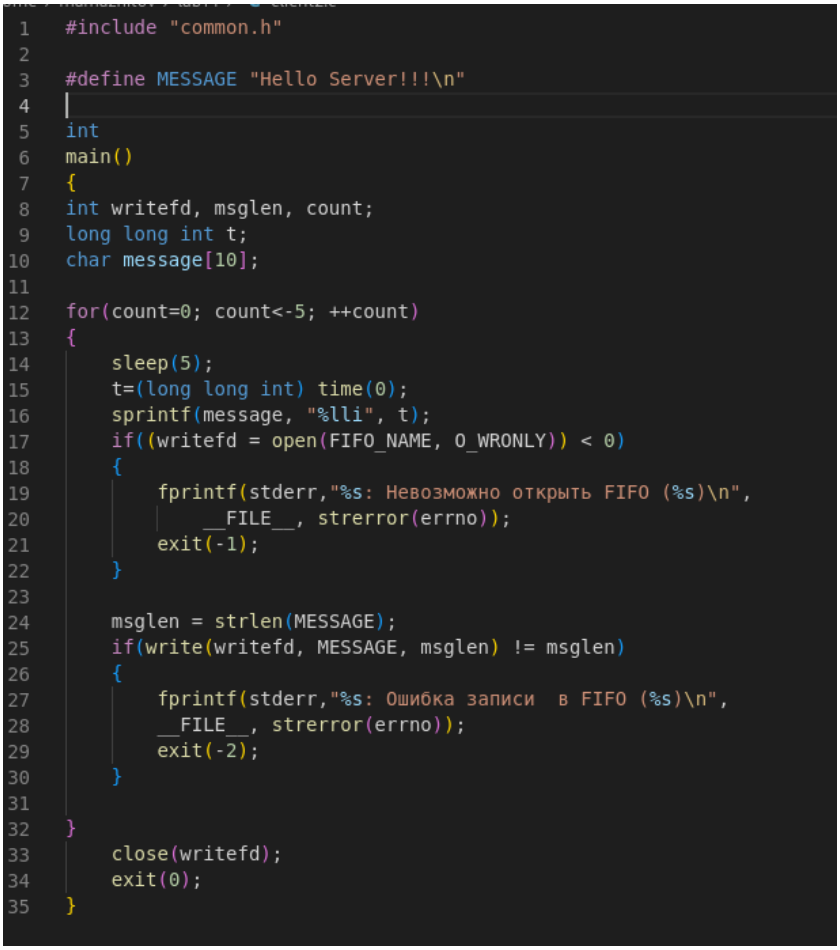
```c
#include "common.h"
int
main()
{
  int readfd;
  int n;
  char buff[MAX_BUFF];
  printf("FIFO Server...\n");

  if(mknod(FIFO_NAME, S_IFIFO | 0666, 0) < 0)
    {
      fprintf(stderr, "%s: Невозможно создать FIFO (%s)\n",
        __FILE__, strerror(errno));
      exit(-1);
    }

  if((readfd = open(FIFO_NAME, O_RDONLY)) < 0)
    {
      fprintf(stderr, "%s: Невозможно открыть FIFO (%s)\n",
        __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)\n",
          __FILE__, strerror(errno));
      }
  }
      now=time(NULL);
    }
  printf("server timeout, %li - seconds passed\n",(now-start));
  close(readfd);

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

Рис. 3.2: Файл server.c



```c
#include "common.h"

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

int
main()
{
int msg, len, i;
long int t;

for(i=0; i<20; i++)
{
    sleep(3);
    t=time(NULL);
    printf("FIFO Client...\n");

    if((msg = open(FIFO_NAME, O_WRONLY)) < 0)
    {
        fprintf(stderr,"%s: Невозможно открыть FIFO (%s)\n",
            __FILE__, strerror(errno));
        exit(-1);
    }

    len = strlen(MESSAGE);

    if(write(msg, MESSAGE, len) != len)
    {
        fprintf(stderr,"%s: Ошибка записи  в FIFO (%s)\n",
        __FILE__, strerror(errno));
        exit(-2);
    }
    close(msg);
}
    exit(0);
}
```

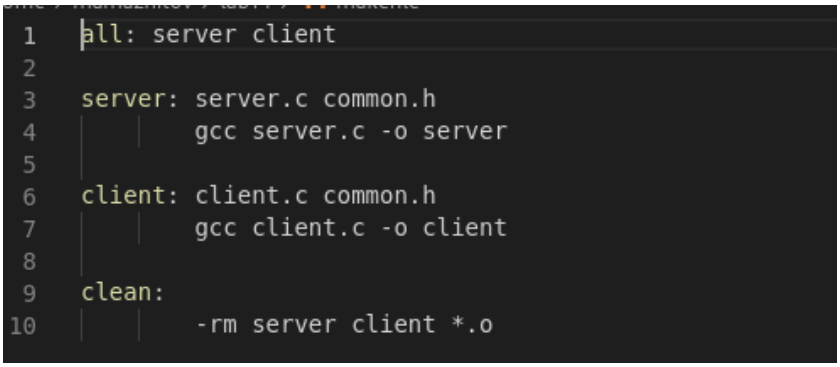
Рис. 3.3: Файл client.c



```c
#include "common.h"

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

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. 3.4: Файл client2.c

2. Создал *makefile*.(рис. 3.5)



```
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. 3.5: makefile

3. Запустил *makefile*. Затем запустил *server*. (рис. 3.6)

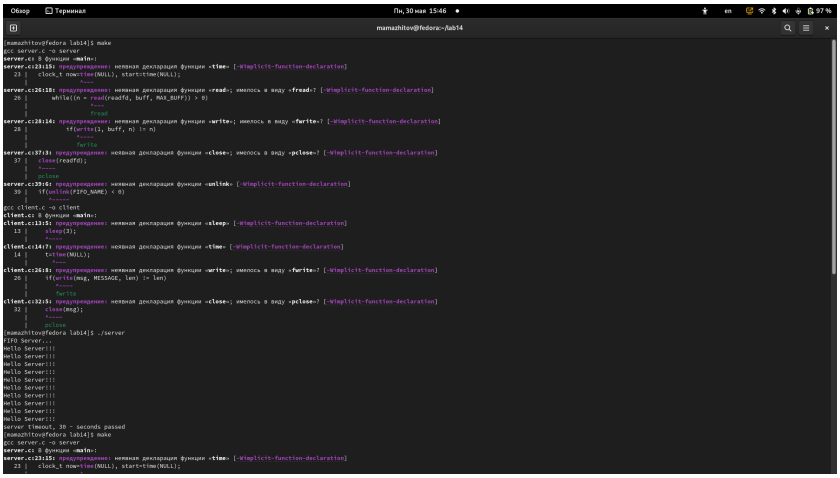


Рис. 3.6: Запуск makefile и server

4. Запустил client в отдельном окне терминала (рис. 3.7)

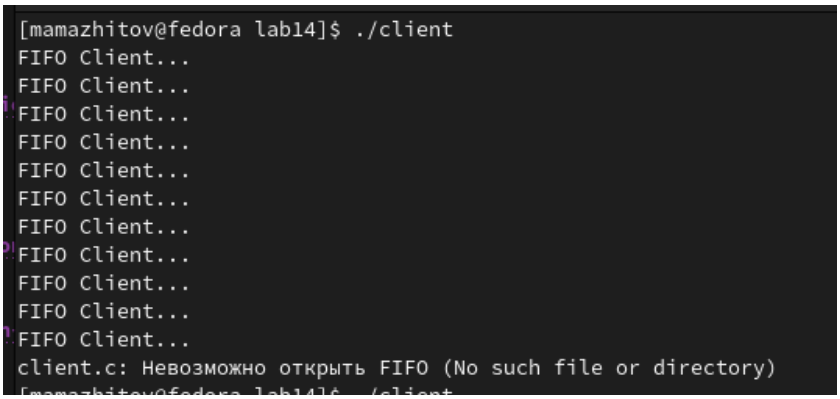


Рис. 3.7: Запуск client

4 Вывод

Мы научились пользоваться именованными каналами.

5 Контрольные вопросы.

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

2. Создание неименованного канала из командной строки возможно командой pipe.

3. Создание именованного канала из командной строки возможно с помощью mkfifo.

4. 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).

5. 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);

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=EP1PE) (если процесс не установил обработки сигнала SIGPIPE, производится обработка по умолчанию – процесс завершается).

7. Два и более процессов могут читать и записывать в канал.

8. 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. Строковая функция strerror - функция языков C/C++, транслирующая код ошибки, который обычно хранится в глобальной переменной errno, в сообщение об ошибке, понятном человеку.