Лабораторнаяработа14

Именованные каналы

Ерёмин Даниил

Содержание

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,внеся следующие изменения:

1. Работает не 1 клиент, а несколько (например,два).
2. Клиенты передаюттекущее время с некоторой периодичностью (например, раз в пять секунд). Используйте функцию sleep() для приостановки работы клиента.
3. Сервер работаетне бесконечно,а прекращаетработу 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

Одним из видов взаимодействия между процессами в операционных системах является обмен сообщениями. Под сообщением понимается последовательность байтов, передаваемая от одного процесса другому.

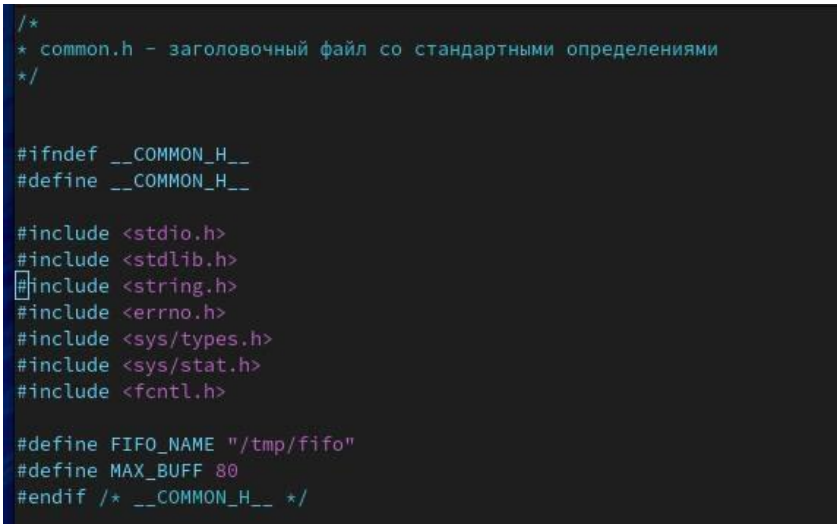
В операционных системах типа UNIX есть 3 вида межпроцессорных взаимодействий: общеюниксные (именованные каналы, сигналы), System V Interface Definition (SVID — разделяемая память, очередь сообщений, семафоры) и BSD (сокеты).

Дляпередачиданныхмеждунеродственнымипроцессамиможноиспользовать механизм именованных каналов (named pipes). Данные передаются по принципу FIFO (First In First Out) (первым записан — первым прочитан), поэтому они называются также FIFO pipes или просто FIFO. Именованные каналы отличаются от неименованных наличием идентификатора канала, который представлен как специальный файл (соответственно имя именованного канала — это имя файла). Поскольку файл находится на локальной файловой системе,данное IPC используется внутри одной системы.

4 Выполнение лабораторной работы

Создаю поддиректорию ~/work/os/lab14, в ней создаю и заполняю файлы common.h, service.c, client.c, client2.c:

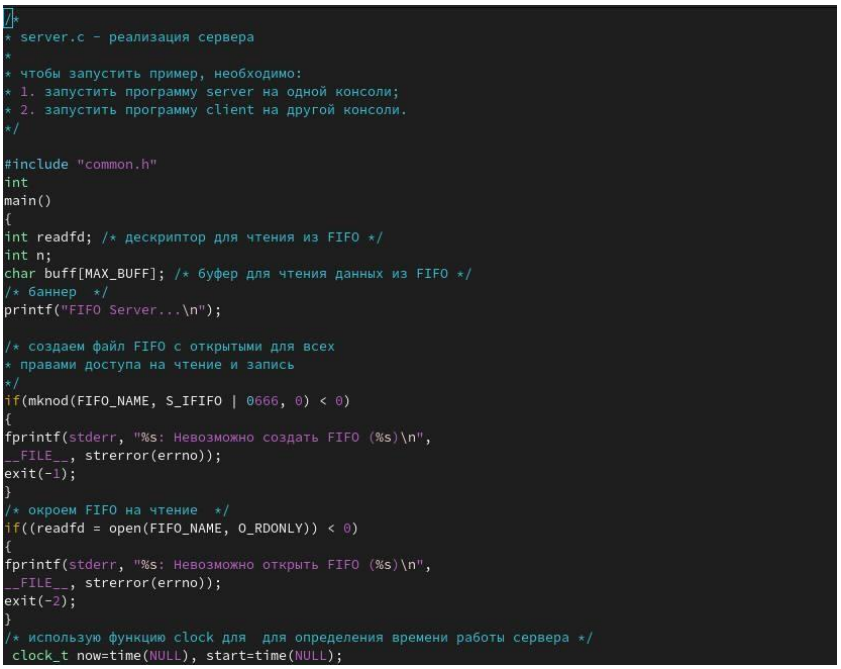
Содержимое заголовочного файла common.h заполняю согласно описанию лабораторной работы:



```c
/*
* common.h - заголовочный файл со стандартными определениями
*/


#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. 4.1: Содержимое заголовочного файла.

Заполняю файл server.c, реализующий сервер. Использую функцию clock для определения времени работы сервера. Сервер работает не бесконечно, а прекращает работу через некоторое время, в моем случае, через 30 секунд:



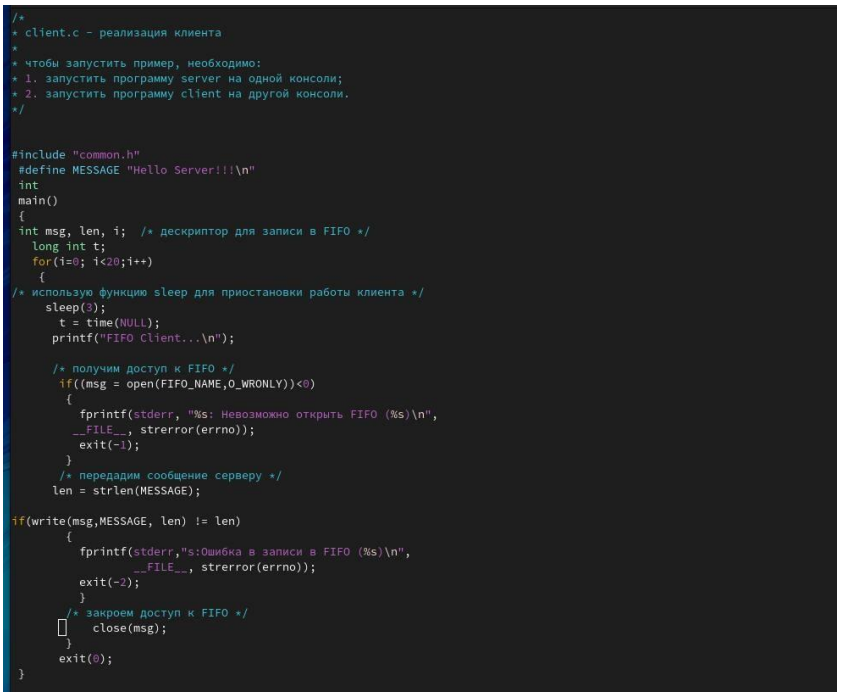
```
/*
* server.c - реализация сервера
*
* чтобы запустить пример, необходимо:
* 1. запустить программу server на одной консоли;
* 2. запустить программу client на другой консоли.
*/

#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);
}
/* окроем FIFO на чтение  */
if((readfd = open(FIFO_NAME, O_RDONLY)) < 0)
{
fprintf(stderr, "%s: Невозможно открыть FIFO (%s)\n",
__FILE__, strerror(errno));
exit(-2);
}
/* использую функцию clock для  для определения времени работы сервера */
 clock_t now=time(NULL), start=time(NULL);
```

Рис. 4.2: Содержимое файла реализации сервера, начало.

```
 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));
           }
       }
         now=time(NULL);
   }
printf("server timeout, %li - second passed\n",(now-start));
close(readfd); /* закроем FIFO */
/* удалим FIFO из системы */
if(unlink(FIFO_NAME) < 0)
{
fprintf(stderr, "%s: РќРµРІРѕР·РјРѕР¶РЅРѕ СѓРґР°Р»РёС‚СЊ FIFO (%s)\n",
__FILE__, strerror(errno));
exit(-4);
}
exit(0);
}
```

Рис. 4.3: Содержимое файла реализации сервера, конец.

Заполняю файлы client.c и client2.c. Использую функцию sleep для приостановки работы клиента. Клиенты передаюттекущее время с некоторой периодичностью, в моем случае раз в пять секунд:



```c
/*
* client.c - реализация клиента
*
* чтобы запустить пример, необходимо:
* 1. запустить программу server на одной консоли;
* 2. запустить программу client на другой консоли.
*/


#include "common.h"
 #define MESSAGE "Hello Server!!!\n"
 int
 main()
 {
 int msg, len, i;  /* дескриптор для записи в FIFO */
   long int t;
   for(i=0; i<20;i++)
    {
/* использую функцию sleep для приостановки работы клиента */
     sleep(3);
       t = time(NULL);
     printf("FIFO Client...\n");

     /* получим доступ к FIFO */
      if((msg = open(FIFO_NAME,O_WRONLY))<0)
       {
         fprintf(stderr, "%s: Невозможно открыть FIFO (%s)\n",
        __FILE__, strerror(errno));
         exit(-1);
       }
      /* передадим сообщение серверу */
     len = strlen(MESSAGE);

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

Рис. 4.4: Содержимое файла реализации клиента 1.

```
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. 4.5: Содержимое файла реализации клиента 2.

Создаю и заполняю Makefile:

Создание и заполнение Makefile

Рис. 4.6: Создание и заполнение Makefile

Содержание Makefile.

Рис. 4.7: Содержание Makefile.

Затем выполняю программу. На одной консоли запускаю программу server, а на другой консоли запускаю программу client.

5 Вывод

В ходе выполнения лабораторной работы я приобрела практические навыки в работы с именованными каналами.

6 Ответы на 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.

4. 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).

5. 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).

6. Что будет в случае прочтения из fifo меньшего числа байтов, чем находится в канале? Большего 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? Большего числа байтов?

Запись числа байтов, меньшего ёмкости 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 блокируется до освобождения требуемого места.

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

Два и более процессов могут читать и записывать в канал.

9. Опишите функцию write (тип возвращаемого значения,аргументы илогику работы).

Функцияwriteзаписываетlengthбайтовизбуферabufferвфайл,определенный дескриптором файла fd. Эта операция чисто ‘двоичная’ и без буферизации. При единице возвращает действительное число байтов. Функция write возвращает числодействительно записанных в файл байтов или -1 при ошибке,устанавливая при этом errno.

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.