Отчёт по лабораторной работе №15

дисциплина: Операционные системы

Быстров Глеб Андреевич

Содержание

1 Цель работы

В данной лабораторной работе мне будет необходимо приобрести практические навыки работы с именованными каналами.

2 Теория

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),
- BSD (сокеты).

Для передачи данных между неродственными процессами можно использовать 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. Файлы именованных каналов создаются функцией mkfifo(3). int mkfifo(const char *pathname, mode_t mode); *Вызов функции mkfifo() создаёт файл канала (с именем, заданным макросом FIFO_NAME): mkfifo(FIFO_NAME, 0600); В качестве маски доступа используется восьмеричное значение 0600, разрешающее процессу с аналогичными реквизитами пользователя чтение и запись. Можно также установить права доступа 0666. Открываем созданный файл для чтения: f = fopen(FIFO_NAME, O_RDONLY); Ждём сообщение от клиента. Сообщение читаем с помощью функции read() и печатаем на экран. После этого удаляется файл FIFO_NAME и сервер прекращает работу. Клиент*

открывает FIFO для записи как обычный файл: f = fopen(FIFO_NAME, O_WRONLY); Посылаем сообщение серверу с помощью функции write() . Для создания файла FIFO можно использовать более общую функцию mknod(2) , предназначенную для создания специальных файлов различных типов (FIFO, сокеты, файлы устройств и обычные файлы для хранения данных). int mknod(const char pathname, mode_t mode, dev_t dev); Тогда, вместо mkfifo(FIFO_NAME, 0600); пишем mknod(FIFO_NAME, S_IFIFO | 0600, 0); Каналы представляют собой простое и удобное средство передачи данных, которое, однако, подходит не во всех ситуациях. Например, с помощью каналов довольно трудно организовать обмен асинхронными сообщениями между процессами.

3 Задание

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

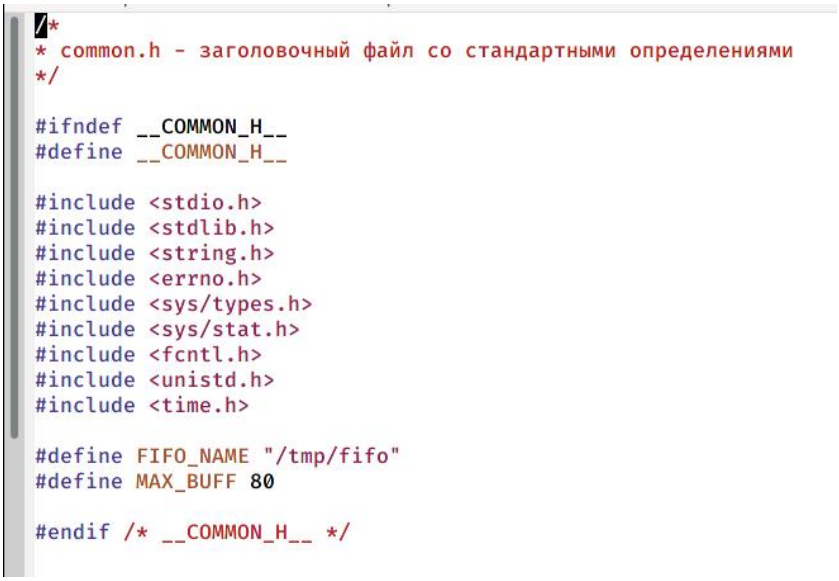
1. Работает не 1 клиент, а несколько (например, два).
2. Клиенты передают текущее время с некоторой периодичностью (например, раз в пять секунд). Используйте функцию sleep() для приостановки работы клиента.
3. Сервер работает не бесконечно, а прекращает работу через некоторое время (например, 30 сек).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

4 Выполнение лабораторной работы

1. В домашнем каталоге создал необходимые файлы для работы. Использовал команду emacs. (рис. 4.1)

Figure 4.1: Создание файлов

2. Отредактировал файл common.h - заголовочный файл со стандартными определениями. Добавил заголовочные файлы unistd.h и time.h. (рис. 4.2)



```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>
#include <unistd.h>
#include <time.h>

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

#endif /* __COMMON_H__ */
```

Figure 4.2: Файл common.h

3. Добавил в файл server.c (реализация сервера) контроль за временем работы. (рис. 4.3) (рис. 4.4)

```c
/*
* 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);
  }
  /* 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);
  }

  clock_t start = time(NULL);
  while (time(NULL)-start<5)
  {
  /* 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));
```

Figure 4.3: Файл server.c

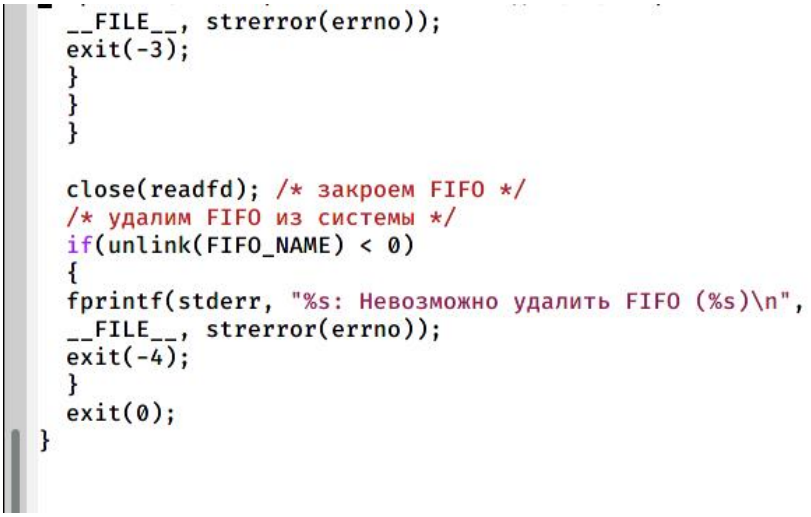


Figure 4.4: Файл server.c

4. Добафил в файл client.c (реализация клиента) цикл для вывода времени. (рис. 4.5)

```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 writefd; /* дескриптор для записи в FIFO */
  int msglen;
  /* баннер */
  printf("FIFO Client...\n");
  for (int i=0; i<4; i++)
  {
  /* 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);
  break;
  }
  long int ttime=time(NULL);
  char* text=ctime(&ttime)
  /* 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);
  }
  sleep(5);
  }
  /* закроем доступ к FIFO */
  close(writefd);
  exit(0);
}
```

Figure 4.5: Файл client.c

5. Файл Makefile не изменял. (рис. 4.6)

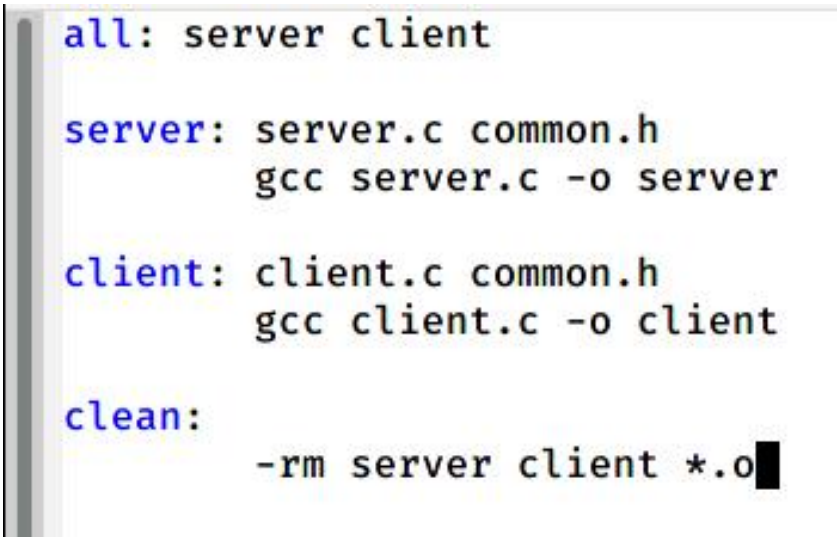


Figure 4.6: Файл Makefile

6. С помощью команды “make all” скомпилировал необходимые файлы. (рис. 4.7)

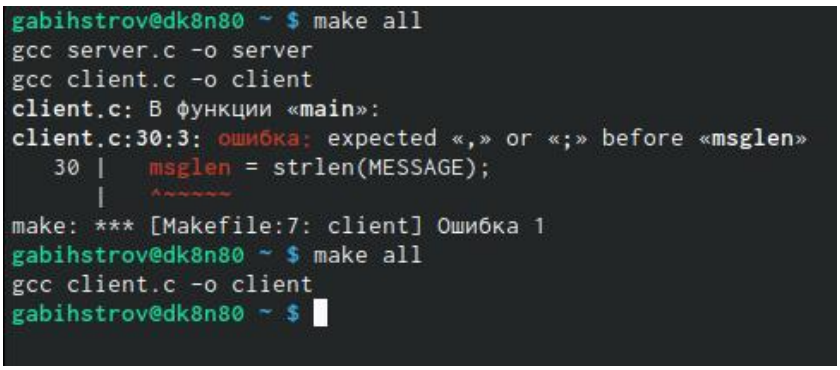


Figure 4.7: Команда make all

7. Запустил код в трёх консолях. В двух ./client и в одной ./server. Через 30 секунд работа программы была прекращена. (рис. 4.8)

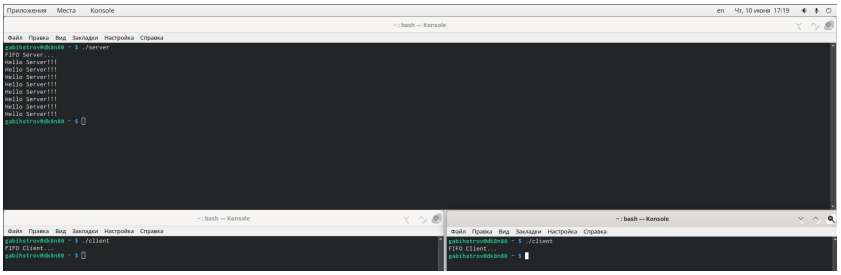


Figure 4.8: Демонстрация работы

8. Если только в одном терминале запускать ./server, то появится ожидаемая ошибка. (рис. 4.9)

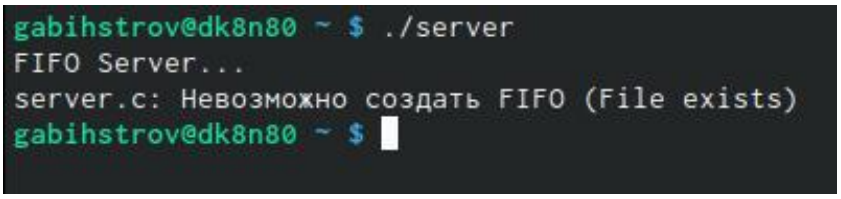


Figure 4.9: Демонстрация работы

5 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). Поскольку файл находится на локальной файловой системе, данное IPC используется внутри одной системы.

2. Возможно ли создание неименованного канала из командной строки?

Для этого необходимо использовать команду pipe. Для передачи данных между неродственными процессами можно использовать 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.

3. Возможно ли создание именованного канала из командной строки?

Вызов функции mkfifo() создаёт файл канала (с именем, заданным макросом FIFO_NAME): mkfifo(FIFO_NAME, 0600);

4. Опишите функцию языка C, создающую неименованный канал.

Библиографический список ссылка №1

Неименованный канал создается вызовом pipe, который заносит в массив int [2] два дескриптора открытых файлов. fd[0] – открыт на чтение, fd[1] – на запись

(вспомните STDIN == 0, STDOUT == 1). Канал уничтожается, когда будут закрыты все файловые дескрипторы ссылающиеся на него.

В рамках одного процесса pipe смысла не имеет, передать информацию о нем в произвольный процесс нельзя (имени нет, а номера файловых дескрипторов в каждом процессе свои). Единственный способ использовать pipe – унаследовать дескрипторы при вызове fork (и последующем exec). После вызова fork канал окажется открытым на чтение и запись в родительском и дочернем процессе. Т.е. теперь на него будут ссылаться 4 дескриптора. Теперь надо определиться с направлением передачи данных – если надо передавать данные от родителя к потомку, то родитель закрывает дескриптор на чтение, а потомок - дескриптор на запись.

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.

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

В случае прочтения меньшего числа байтов, чем находится в канале, возвращается требуемое число байтов, остаток сохраняется для последующих чтений. В случае прочтения большего числа байтов, чем находится в канале или FIFO возвращается доступное число байтов.

7. Аналогично, что будет в случае записи в fifo меньшего числа байтов, чем позволяет буфер? Большего числа байтов?

В случае записи большего числа байтов, чем это позволяет канал или FIFO, вызов write(2) блокируется до освобождения требуемого места. Если процесс пытается записать данные в канал, не открытый ни одним процессом на чтение, процессу генерируется сигнал. 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.

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

Библиографический список ссылка №2

Теперь рассмотрим особенности организации записи в канал. Если процесс пытается записать в канал порцию данных, превосходящую доступное в канале свободное пространство, то часть этой порции данных, равная размеру свободного пространства канала, помещается в канал, и процесс блокируется до появления в канале необходимого свободного пространства. Можно избежать блокировки, используя системный вызов fcntl().

Если процесс пытается записать информацию в канал, с которым в данный момент не связан ни один открытый дескриптор чтения, то процесс получает сигнал SIGPIPE. Таким образом система уведомляет процесс, что произвести операцию записи в канал в настоящий момент нельзя, поскольку нет читающей стороны (а в случае неименованных каналов восстановить ее невозможно).

В общем случае возможна многонаправленная работа процессов с каналом, т.е. возможна ситуация, когда с одним и тем же каналом взаимодействуют два и более процесса, и каждый из взаимодействующих каналов пишет и читает информацию в канал. Но традиционной схемой организации работы с каналом является однонаправленная организация, когда канал связывает два, в большинстве случаев, или несколько взаимодействующих процесса, каждый из которых может либо читать, либо писать в канал.

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)?

Посылаем сообщение серверу с помощью функции write(). Функция записывает

байты из буфера в файл, определённый дескриптором файла. Данная операция работает без буферизации и реализуется как вызов DOS.

10. Опишите функцию strerror.

Библиографический список ссылка №3

Интерпретирует номер ошибки, передаваемый в функцию в качестве аргумента — errornum, в понятное для человека текстовое сообщение (строку).Ошибки эти возникают при вызове функций стандартных Си-библиотек. 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.

6 Выводы

В данной лабораторной работе мне успешно удалось приобрести практические навыки работы с именованными каналами.

7 Библиографический список

1. Каналы (pipe,fifo) (https://parallel.uran.ru/book/export/html/464)
2. Базовые средства реализации взаимодействия процессов в ОС Unix (https://poisk-ru.ru/s81816t1.html)
3. Функция strerror (http://cppstudio.com/post/669/)