Отчет по лабораторной работе №14

Дисциплина Операционные системы

Колобова Елизавета, гр. НММбд-01-22

Содержание

1 Цель работы **5**

2 Выполнение лабораторной работы **6**

3 Контрольные вопросы **11**

Список литературы **14**

Список иллюстраций

2.1 Рис. 1. Заголовочный файл 7
2.2 Рис. 2. Реализация клиента 7
2.3 Рис. 3. Реализация клиента 8
2.4 Рис. 4. Реализация сервера 9
2.5 Рис. 5. Реализация сервера 10
figno Рис. 6. Пример вывода сервера 10

Список таблиц

1 Цель работы

Цель работы - приобретение практических навыков работы с именованными каналами.

2 Выполнение лабораторной работы

Изучив приведённые в тексте лабораторной работы программы server.c и client.c, напишем аналогичные программы, внеся следующие изменения:

1. Работает не 1 клиент, а несколько (например, два). Для этого (как объясняли на семинарских занятиях) достаточно просто запустить клиентскую программу в двух разных консолях

2. Клиенты передают текущее время с некоторой периодичностью (например, раз в пять секунд). Используйте функцию sleep() для приостановки работы клиента. Для этого в переменную, содержащую сообщение передаем переведенное в строку время, после чего приостанавливаем работу клиента на 5 секунд с помощью sleep, а в заголовочный файл подключаем time.h (рис. [2.2], [2.3], [2.1])

3. Сервер работает не бесконечно, а прекращает работу через некоторое время (напри- мер, 30 сек). Используйте функцию clock() для определения времени работы сервера. Указанной в задании функцией получаем время работы сервера, и с помощью цикла завершаем работу сервера, если оно превысит 30 секунд (иначе оставляем все как было). Что будет в случае, если сервер завершит работу, не закрыв канал? - Канал продолжит существовать (рис. [2.4], [2.5])

Открыть common.h ~/lab13

```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 <time.h>

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

#endif /* __COMMON_H__ */
```

Рис. 2.1: Рис. 1. Заголовочный файл

Открыть client.c ~/lab13

```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");

  /* 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);
  }

  /* передадим сообщение серверу */
  for(int i = 1; i >= 0; i++)
  {
    char MESSAGE[40];
    sprintf(MESSAGE, "%li", time(0), "\n");
    msglen = strlen(MESSAGE);
    sleep(5);
      if(write(writefd, MESSAGE, msglen) != msglen)
      {
        fprintf(stderr, "%s: Ошибка записи в FIFO (%s)\n",
                __FILE__, strerror(errno));
        exit(-2);
      }
```

Рис. 2.2: Рис. 2. Реализация клиента

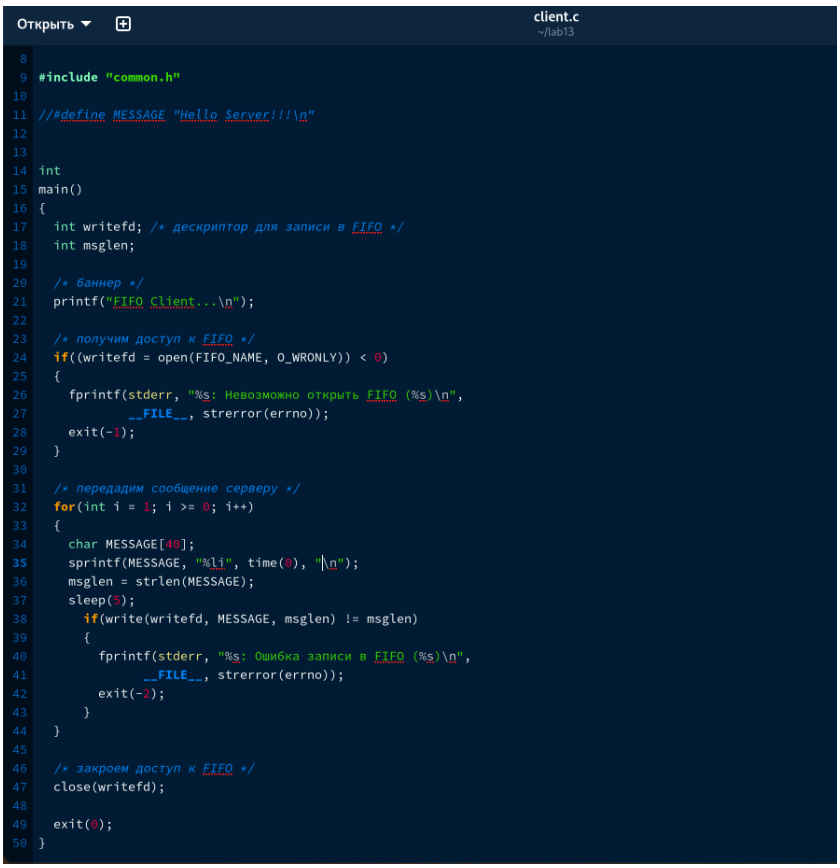

Рис. 2.3: Рис. 3. Реализация клиента

Открыть

server.c
~/lab13

```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");
  clock_t time_req;
  time_req = clock();
  if(time_req == 30)
    {
      exit(0);
    }
  else
    {
      /* 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);
```

Рис. 2.4: Рис. 4. Реализация сервера

Открыть

server.c
~/lab13

```
  else
    {
      /* 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);
        }
      /* 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));
          exit(-3);
          }
      }

      close(readfd); /* закроем FIFO */

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

        exit(0);
    }
}
```

Рис. 2.5: Рис. 5. Реализация сервера

```
[eakolobova@fedora lab13]$ ./server
FIFO Server...
168373024316837302481683730253[eakolobova@fedora lab13]$ ./server
FIFO Server...
^C
```

Рис. 6. Пример вывода сервера

3 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? Неименованные каналы подобны именованным, но они в файловой системе не существуют. Они не имеют путевых имен, ассоциированных с ними, и все они и их следы исчезают после того, как последний файловый дескриптор, ссылающийся на них, закрывается. Т.е. создать их из командной строки нельзя
3. Возможно ли создание именованного канала из командной строки? FIFO может быть создан и из командной строки shell:

```
$ mknod name p
```

4. Опишите функцию языка C, создающую неименованный канал. Создание неименованного канала выполняется по двум файловым дескрипторам, один из которых доступен только для чтения, а второй — только для записи.

```
#include <unistd.h>

int pipe(int fds[2]);
```

Единственный параметр-массив включает два файловых дескриптора — fd[0]

для чтения и fd[1] для записи. 5. Опишите функцию языка C, создающую именованный канал. Файлы именованных каналов создаются функцией mkfifo(3).

```
1 #include <sys/types.h>
2 #include <sys/stat.h>
3
4 int mkfifo(const char *pathname, mode_t mode)
```

6. Что будет в случае прочтения из fifo меньшего числа байтов, чем находится в канале? Большего числа байтов? При чтении числа байт, меньшего чем находится в канале, возвращается требуемое число байтов, остаток сохраняется для последующих чтений. При чтении числа байт, большего чем находится в канале, возвращается доступное число байт.
7. Аналогично, что будет в случае записи в fifo меньшего числа байтов, чем позволяет буфер? Большего числа байтов? 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=ERRPIPE) (если процесс не установил обработки сигнала SIGPIPE, производится обработка по умолчанию — процесс завершается). 8. Могут ли два и более процессов читать или записывать в канал? С точки зрения процессов, канал выглядит как пара открытых файловых дескрипторов – один на чтение и один на запись (можно больше, но неудобно). Мы можем писать в канал до тех пор пока есть место в буфере, если место в буфере кончится – процесс будет заблокирован на записи. 9. Опишите функцию write (тип возвращаемого значения, аргументы и логику

работы). Функция write() является частью UNIX-подобной системы ввода/вывода и не определена стандартом ANSI C. Функция write() переписывает count байт из буфера, на который указывает bufy в файл, соответствующий дескриптору файла handle. Указателю положения в файле дается приращение на количество записанных байт. 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. # Выводы

Результатом проделанной работы является приобретение практических навыков работы с именованными каналами.

Список литературы