

Лабораторная работа №14.

Именованные каналы.

Александр Андреевич Шуплецов

Содержание

1	Цель работы	5
2	Теоретическое введение	6
3	Выполнение работы	7
4	Выводы	11
5	Контрольные вопросы	12
	Список литературы	15

Список иллюстраций

3.1	файл client.c	7
3.2	файл client2.c	8
3.3	файл server.c	9
3.4	Makefile	9
3.5	файл common.h	10

Список таблиц

1 Цель работы

Приобретение практических навыков работы с именнованными каналами.

2 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: общепонимание (именованные каналы, сигналы), System V Interface Definition (SVID — разделяемая память, очередь сообщений, семафоры) и 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. 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.

3 Выполнение работы

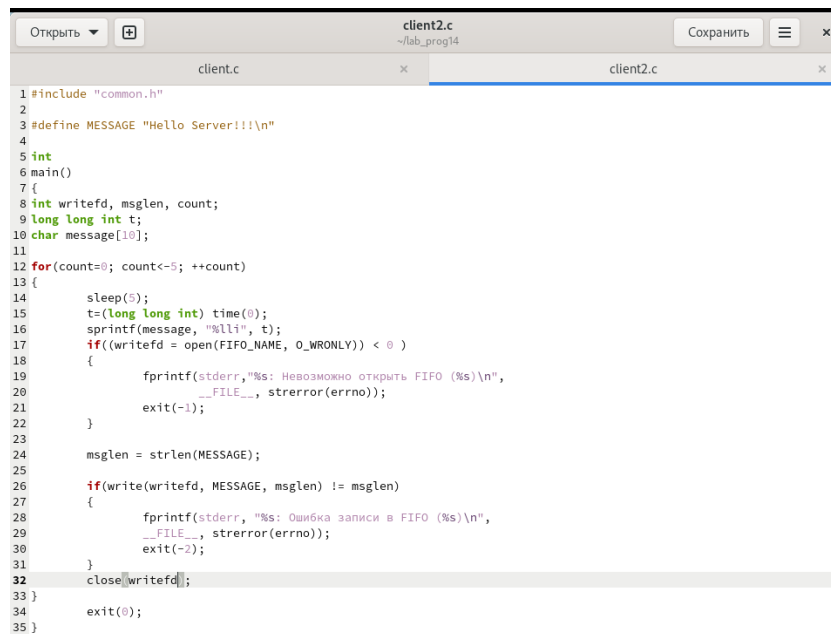
1. Создадим файл client.c, который реализует клиент.



```
1 #include "common.h"
2
3 #define MESSAGE "Hello Server!!!\n"
4
5 int
6 main()
7 {
8     int msg, len, i;
9     long int t;
10
11     for(i=0; i<20, i++)
12     {
13         sleep(3);
14         t=time(NULL);
15         printf("FIFO Client...\n");
16
17         if((msg = open(FIFO_NAME, O_WRONLY)) < 0)
18         {
19             fprintf(stderr, "%s: Невозможно открыть FIFO (%s)\n",
20                 _FILE_,strerror(errno));
21             exit(-1);
22         }
23
24         len = strlen(MESSAGE);
25
26         if(write(msg, MESSAGE, len) != len)
27         {
28             fprintf(stderr, "%s: Ошибка записи в FIFO (%s)\n",
29                 _FILE_, strerror(errno));
30             exit(-2);
31         }
32         close(msg);
33     }
34     exit(0);
35 }
```

Рис. 3.1: файл client.c

2. Создадим файл client2.c, который также реализует клиент.



```
1 #include "common.h"
2
3 #define MESSAGE "Hello Server!!!\n"
4
5 int
6 main()
7 {
8     int writefd, msglen, count;
9     long long int t;
10    char message[10];
11
12    for(count=0; count<=5; ++count)
13    {
14        sleep(5);
15        t=(long long int) time(0);
16        sprintf(message, "%lli", t);
17        if((writefd = open(FIFO_NAME, O_WRONLY)) < 0 )
18        {
19            fprintf(stderr, "%s: Невозможно открыть FIFO (%s)\n",
20                __FILE__, strerror(errno));
21            exit(-1);
22        }
23
24        msglen = strlen(MESSAGE);
25
26        if(write(writefd, MESSAGE, msglen) != msglen)
27        {
28            fprintf(stderr, "%s: Ошибка записи в FIFO (%s)\n",
29                __FILE__, strerror(errno));
30            exit(-2);
31        }
32        close(writefd);
33    }
34    exit(0);
35 }
```

Рис. 3.2: файл client2.c

3. Создадим файл server.c, который реализует сервер.


```
1 #include "common.h"
2 int
3 main()
4 {
5     int readfd;
6     int n;
7     char buff[MAX_BUFF];
8     printf("FIFO Server...\n");
9
10    if(mknod(FIFO_NAME, S_IFIFO | 0666, 0) < 0)
11    {
12        fprintf(stderr, "%s: Невозможно создать FIFO (%s)\n",
13            __FILE__, strerror(errno));
14        exit(-1);
15    }
16
17    if((readfd = open(FIFO_NAME, O_RDONLY)) < 0)
18    {
19        fprintf(stderr, "%s: Невозможно открыть FIFO (%s)\n",
20            __FILE__, strerror(errno));
21        exit(-2);
22    }
23
24    clock_t now=time(NULL), start=time(NULL);
25    while (now-start<30)
26    {
27        while((n = read(readfd, buff, MAX_BUFF)) > 0)
28        {
29            if(write(1, buff, n) != n)
30            {
31                fprintf(stderr, "%s: Ошибка вывода (%s)\n",
32                    __FILE__, strerror(errno));
33                exit(-3);
34            }
35        }
36        now=time(NULL);
37    }
38    printf("server timeout, %li - seconds passed\n", (now-start));
39    close(readfd);
40
41    if(unlink(FIFO_NAME) < 0)
42    {
43        fprintf(stderr, "%s: Невозможно удалить FIFO (%s)\n",
44            __FILE__, strerror(errno));
45        exit(-4);
46    }
```


Рис. 3.3: файл server.c

4. Создадим Makefile.

```
1 hll: server client
2
3 server: server.c common.h
4     gcc server.c -o server
5
6 client: client.c common.h
7     gcc client.c -o client
8
9 clean:
10     -rm server client *.o
```

Рис. 3.4: Makefile

5. Создадим файл common.h, являющийся заголовочным файлом со стандартными определениями.



The image shows a code editor window with the title bar "common.h" and the path "~/lab_prog14". The editor has a menu bar with "Открыть" (Open) and "Сохранить" (Save) buttons. The tab bar shows five open files: "client.c", "client2.c", "server.c", "Makefile", and "common.h". The "common.h" file is selected and its content is displayed in the editor area. The code is as follows:

```
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__ */
```

Рис. 3.5: файл common.h

4 Выводы

Я приобрел практические навыки работы с именованными каналами.

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).

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 неименованного канала из командной строки возможно с помощью команды `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()` создает файла 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? Большого 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. 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. 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.

Список литературы

Кулябов Д.С. “Материалы к лабораторным работам”