Операционные системы и сети

Лабораторная работа №6 Джугели Дмитрий ПИН-31Д

Межпроцессное взаимодействие в ОС Linux: каналы

Цель работы: научиться использовать каналы для межпроцессного взаимодействия в ОС Linux.

Задание 1.

1.1. Наберите приведенный ниже текст программы **pipework.c** (текст программы имеется в папке данной лаб. работы) и создайте для нее исполняемый файл. На рис. 2 приведена структура программы.

Примечание. После вызова библиотечной функции **dup2(fd1, fd2)** файловый дескриптор **fd2** будет ссылаться на тот же файл, что и дескриптор **fd1**. Если **fd2** ссылается на уже открытый файл, то этот файл сначала будет закрыт. Т.о., **dup2(fd1, 0)** позволяет перенаправить *stdin* в файл **fd1**.

1.2. Запустите эту программу и сравните результаты с результатами следующих команд оболочки:

ls -l /home/student и **ls -l /home/student | sort**

```
[ant_daddy@Dmitriy 123 % gcc -g pipework.c -o pipework
[ant_daddy@Dmitriy 123 % ./pipework
-rw-r--r--@  1 ant_daddy  staff  1019 17 июн 10:19 pipework.c
-rwxr-xr-x   1 ant_daddy  staff  9112 17 июн 10:19 pipework
drwxr-xr-x   3 ant_daddy  staff    96 17 июн 07:33 pipework.dSYM
drwxr-xr-x  61 ant_daddy  staff  1952 15 июн 07:15 all
total 32
[ant_daddy@Dmitriy 123 % gcc -g pipework.c -o pipework
[ant_daddy@Dmitriy 123 % ./pipework
-rw-r--r--@  1 ant_daddy  staff   658 17 июн 10:21 pipework.c
-rwxr-xr-x   1 ant_daddy  staff  9112 17 июн 10:21 pipework
drwxr-xr-x   3 ant_daddy  staff    96 17 июн 07:33 pipework.dSYM
drwxr-xr-x  61 ant_daddy  staff  1952 15 июн 07:15 all
total 32
[ant_daddy@Dmitriy 123 % gcc -g pipework.c -o pipework
[ant_daddy@Dmitriy 123 % ./pipework
ls: /Users/ant_daddy/123 | sort: No such file or directory
[ant_daddy@Dmitriy 123 % gcc -g pipework.c -o pipework
[ant_daddy@Dmitriy 123 % ./pipework
-rw-r--r--@  1 ant_daddy  staff   658 17 июн 10:23 pipework.c
-rwxr-xr-x   1 ant_daddy  staff  9112 17 июн 10:23 pipework
drwxr-xr-x   3 ant_daddy  staff    96 17 июн 07:33 pipework.dSYM
drwxr-xr-x  61 ant_daddy  staff  1952 15 июн 07:15 all
total 32
[ant_daddy@Dmitriy 123 % gcc -g pipework.c -o pipework
[ant_daddy@Dmitriy 123 % ./pipework
ls: /home/student: No such file or directory
[ant_daddy@Dmitriy 123 % gcc -g pipework.c -o pipework
[ant_daddy@Dmitriy 123 % ./pipework
-rw-r--r--@  1 ant_daddy  staff   658 17 июн 11:04 pipework.c
-rwxr-xr-x   1 ant_daddy  staff  9112 17 июн 11:04 pipework
drwxr-xr-x   3 ant_daddy  staff    96 17 июн 07:33 pipework.dSYM
drwxr-xr-x  61 ant_daddy  staff  1952 15 июн 07:15 all
total 32
```

```
[ant_daddy@Dmitriy 123 % gcc -g pipework.c -o pipework
[ant_daddy@Dmitriy 123 % ./pipework
-rw-r--r--@  1 ant_daddy  staff  1271 17 июн 11:07 pipework.c
-rwxr-xr-x   1 ant_daddy  staff  9112 17 июн 11:07 pipework
drwxr-xr-x   3 ant_daddy  staff    96 17 июн 07:33 pipework.dSYM
drwxr-xr-x  61 ant_daddy  staff  1952 15 июн 07:15 all
total 32
ant_daddy@Dmitriy 123 %
```

1.3. Замените в исходном тексте программы оператор (дайте этой программе имя **pipework1.c**)

execlp ("ls", "ls", "-l", "/home/student", 0);

несколькими системными вызовами **write()**, которые запишут в канал последовательность слов (например, **“this\n”**, **“is\n”**, **“a\”**, **“message\n”**, **‘from\n”**, **“sending\n”**, **“process\”**). Для записи этих слов в канал следует использовать системный вызов **write()** так же, как и для обычного файла, указывая дескриптор **fd[1]**. После записи всех слов в канал закройте **fd[1]** и

завершите процесс (**exit()**). Выполните программу.

Примечание. Для этой модификации оператор **dup2(fd[1],1);** НЕ НУЖЕН, его следует убрать.

```
[ant_daddy@Dmitriy 123 % gcc -g pipework1.c -o pipework1
[ant_daddy@Dmitriy 123 % ./pipework1
process
sending
a
from
is
message
this
ant_daddy@Dmitriy 123 %
```

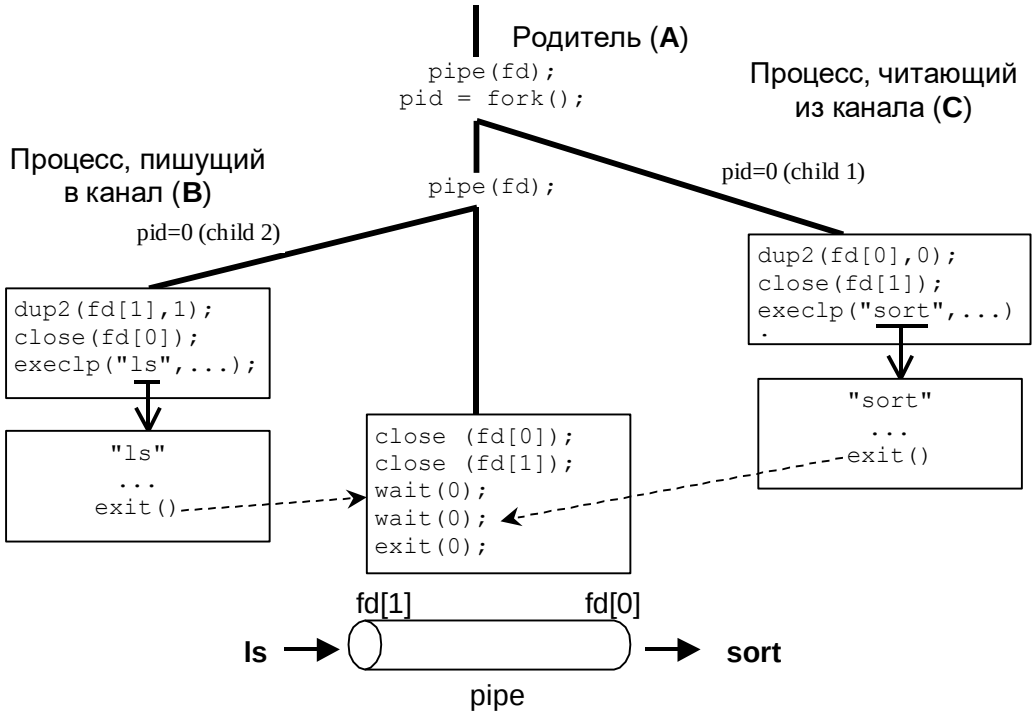


Рис. 2. Структура программы **pipework** для связи команд **ls** и **sort** при помощи канала

```
/* pipework.c A program to experiment with an unnamed pipe */
#include <stdio.h>
#include <unistd.h>

main ()
{
int fd[2];  /*  Массив для двух дескрипторов канала */
int pid;     /*  Идентификатор процесса */
```

```
/*  Канал должен быть создан до fork() Почему? */
if (pipe(fd) < 0)
   {perror ("PIPE CREATION ERROR");
    exit (1);
   }
pid = fork ();  /* Родитель: создать дочерний процесс 1 */
if (pid == 0)   /*  Доч.процесс 1 стартует здесь */
   {
   dup2 (fd[0],0); /* Стандартный ввод - из выходного конца
канала*/
   close (fd[1]);  /* Закрыть входной конец канала (не
используется)*/
   execlp ("sort", "sort", 0); /* Запуск команды “sort” которая
будет брать входную информацию из выходного конца канала */
   }
else   /* Продолжение работы родит. процесса */
pid = fork ();/* Родитель:создание дочернего процесса 2 */
if (pid == 0)   /*  Доч.процесс 2 стартует здесь */
   {dup2 (fd[1],1); /* stdout будет направлен во входной конец
канала */
    close (fd[0]);    /* Закрыть выходной конец канала (не
используется) */
    execlp ("ls", "ls", "-l", "/home/student", 0);
                    /* Запуск команды “who” чей вывод будет
               направлен в канал*/
   }

else /* Родитель закрывает оба конца канала
       и ожидает завершения дочерних процессов  */
   {
   close (fd[0]);
   close (fd[1]);
   wait (0);
   wait (0);
   }
}
```

1.4. Замените в программе из задания 1.3 оператор (дайте этой программе имя **pipework2.c**)

execlp ("sort", "sort", 0);

фрагментом, в котором читаются слова из канала и выводятся на стандартное устройство вывода (т.е. на экран). Для чтения данных из канала следует использовать системный вызов **read()** как и для обычного файла, указывая дескриптор **fd[0]**. После чтения и печати всех слов закройте **fd[0]** и завершите процесс (**exit()**).

Примечание. Для этой модификации оператор **dup2(fd[0],0);** НЕ НУЖЕН, его

следует убрать.

```
[ant_daddy@Dmitriy 123 % gcc -g pipework2.c -o pipework2
[ant_daddy@Dmitriy 123 % ./pipework2
this
is
a
message
from
sending
process
```

1.4. В общем случае читать из канала могут несколько процессов; писать в канал также могут несколько процессов. Напишите программу **pipework3.c**, которая создает три дочерних процесса. Два из них записывают в канал по одному сообщению каждый, а третий дочерний процесс читает эти сообщения из канала и распечатывает их.

```
DzhugeliDDmitriy^C
[ant_daddy@Dmitriy 123 % gcc -g pipework3.c -o pipework3
[ant_daddy@Dmitriy 123 % ./pipework3
DzhugeliDmitriy
```

В программе 1.4 создает три дочерних процесса. Первые два процесса записывают сообщения "Dzhugeli" и "Dmitriy" в канал соответственно, а третий процесс читает эти сообщения из канала и выводит их на экран.

1.5. Для установления двухсторонней связи между процессами следует создать два канала, работающих в противоположных направлениях. Напишите программу **pipework4.c** для двухстороннего общения между двумя процессами с использованием двух каналов.

```
[ant_daddy@Dmitriy 123 % ./pipework4
1 received: from 0 to 1
0 received: from 0 to 1
ant_daddy@Dmitriy 123 %
```

Программа 1.5 создает два канала: fd1 для обмена сообщениями от родительского процесса к дочернему и fd2 для обмена сообщениями от дочернего процесса к родительскому. Родительский процесс отправляет сообщение дочернему, а затем принимает сообщение от него. Дочерний процесс принимает сообщение от родительского и отправляет сообщение ему в ответ.

1.6. Запишите результаты работы всех программ и объясните их работу.

<u>Задание 2.</u> Выполните команду

$mkfifo fifo1

Выпишите в отчет параметры созданного именованного канала (**ls**). Затем выполните команду

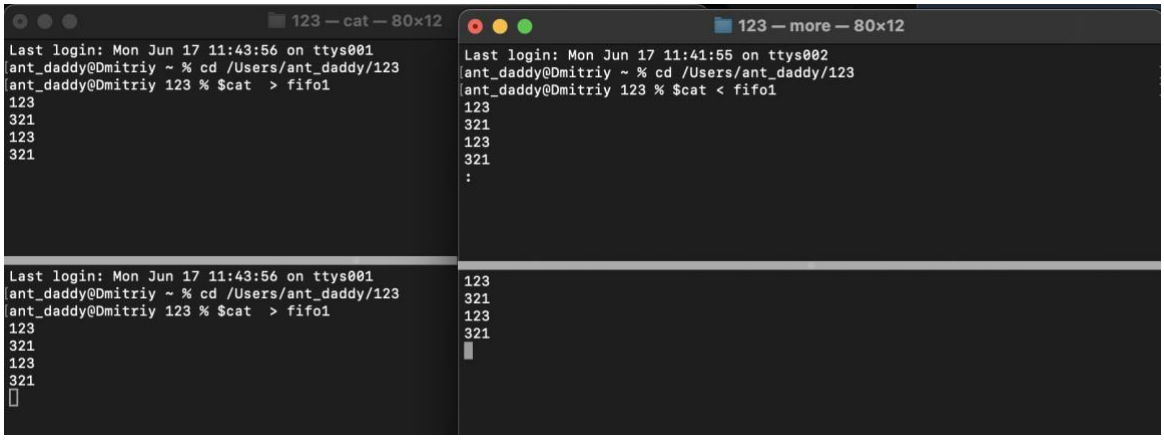
```
ant_daddy@Dmitriy 123 % ls -l fifo1
prw-r--r--  1 ant_daddy  staff  0 17 июн 11:33 fifo1
ant_daddy@Dmitriy 123 %
```

$cat < fifo1

Запустите второй экземпляр оболочки. Перейдите во втором окне в каталог данной лаб. работы. Расположите оба окна так, чтобы они были видны одновременно. Во втором окне введите команду

$cat > fifo1

Затем набирайте произвольные строки, завершите ввод символом конца файла **CTRL-d**. Что будет выведено в первом окне? Удалите именованный канал.



В программе именованный канал создается при помощи системного вызова **mknod()** или библиотечной функции **mkfifo()**, которая, в свою очередь, использует **mknod()**.

В этой части лабораторной работы необходимо создать и проверить несколько вариантов клиент/серверной системы, основанной на именованном канале. Предполагается, что сервер создает именованный канал и пытается читать из него данные. Клиент запускается после старта сервера и записывает в канал некоторые данные для сервера, затем клиент завершается. После чтения данных из канала сервер распечатывает их и также завершается.

Задание 3.

3.1. Выполните программы **server.c** и **client.c** (их тексты имеются в папке данной лаб. работы):

```
/*  server.c SERVER PROGRAM */

#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

#define FIFONAME    "..." /* Some unique name of pipe must be
specified here  */

int main(void)
{
    int n, fd;
    char buf[1024];  /*A buffer for reading/writing the pipe*/

    printf ("SERVER STARTS…\n");
    unlink(FIFONAME); /* Remove any previously created pipe with
this name if any. */
    /* Create a named pipe with the permissions to write and
read for all.  */
    if (mkfifo(FIFONAME, 0666) < 0)
      { perror("mkfifo problem in server"); exit(1);}

  /* Make sure that the permission flag is  really 0666 */
    if (chmod(FIFONAME, 0666) < 0)
      {perror("chmod problem in server"); exit(1);}

    /* Open the created named pipe for reading.     */
    if ((fd = open(FIFONAME, O_RDONLY)) < 0)
      {perror("open problem in server"); exit(1);}
```

```
    /* Read from the named pipe until eof, and print what we get
on the stdout  */
    while ((n = read(fd, buf, sizeof(buf))) > 0)
        write(1, buf, n);

    close(fd);
    printf ("SERVER TERMINATED…\n");
    exit(0);
}

/*  client.c CLIENT  PROGRAM*/

#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

#define FIFONAME    "..."  /* The same name as in server must be
specified here */

int main(void)
{
    int n, fd;
    char buf[1024];      /* Buffer for reading */

    printf ("CLIENT STARTS…\n");

    /* Open existing named pipe for writing. It was created by
the server already */
    if ((fd = open(FIFONAME, O_WRONLY)) < 0)
      {perror("open problem in client");exit(1);}

    /* Read any text from standard input, and copy
     * this text data to the named pipe. The text must be
finished by Ctrl/d  */
    while ((n = read(0, buf, sizeof(buf))) > 0)
        write(fd, buf, n);

    printf ("CLIENT TERMINATED…\n");
    close(fd);
    exit(0);
}
```

3.2. Запустите серверный процесс (без параметров в командной строке) **в фоновом режиме**. Вы увидите соответствующий PID процесса. Проверьте Ваш каталог, чтобы убедиться, что именованный канал создан.

```
[ant_daddy@Dmitriy client-server % ./server
SERVER STARTS...
```

```
[ant_daddy@Dmitriy client-server % ./server &
[1] 54851
```

```
[ant_daddy@Dmitriy client-server % ls -l server
-rwxr-xr-x  1 ant_daddy  staff  9080 17 июн 13:48 server
```

3.3. Запустите клиентский процесс без параметров в командной строке. Введите с клавиатуры несколько строк какого-либо сообщения и завершите его символом конца файла, **Ctrl-d**. Запишите в отчет вид экрана.

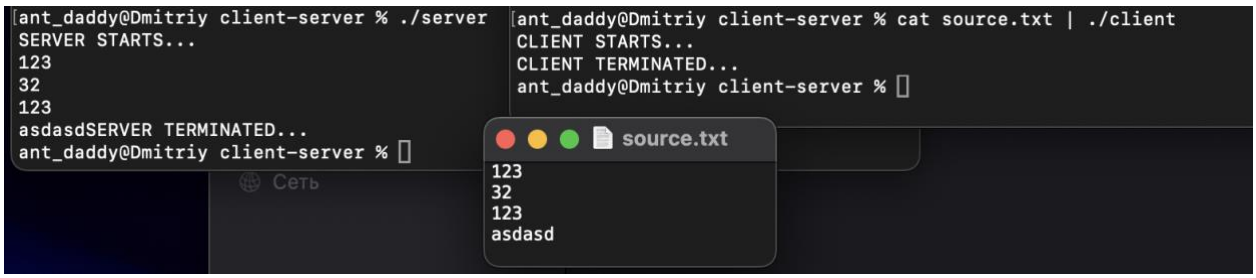
```
[ant_daddy@Dmitriy client-server % gcc -g client.c -o client
[ant_daddy@Dmitriy client-server % ./client
CLIENT STARTS...
123
123
123
CLIENT TERMINATED...
ant_daddy@Dmitriy client-server %
```

3.4. Повторите задание 3.2. Но теперь запустите сервер, как обычно, на переднем плане.

```
ant_daddy@Dmitriy client-server % ./server
SERVER STARTS...
```

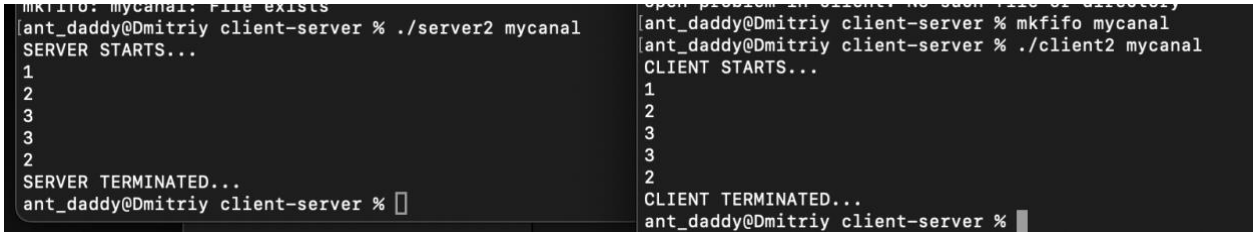
3.5. Запустите вторую копию оболочки. Теперь Вы будете иметь два псевдотерминала: один для сервера и другой для клиента. Расположите оба окна так, чтобы они одновременно были видны на экране. Со второго терминала запустите клиентский процесс и введите несколько строк с клавиатуры (завершив их символом **Ctrl-d**). Запишите в отчет вид обоих экранов.

3.6. Повторите задания 3.4 и 3.5. Но теперь запустите клиентский процесс таким образом, чтобы он считывал информацию из какого-либо текстового файла (например, из файла с исходным текстом клиентской программы). Что Вы увидите на экране серверного терминала?



На экране сервера выводится содержимое текстового файла

3.7. Модифицируйте обе программы (для сервера и клиента) так, чтобы имя канала можно было задавать в командной строке (вместо директивы **#define FIFONAME**). Дайте программам имена **server2.c** и **client2.c**. Создайте исполняемый файл для модифицированной программы и повторите задания 3.4, 3.5 и 3.6.



3.8. Проверьте действие режима **O_NONBLOCK** (или **O_NDELAY**), указывая его при открытии канала. Запишите в отчет Ваши выводы.

3.9. Проверьте Ваш каталог. Удалите все оставшиеся там именованные каналы.

Задание 4.[*]

Создайте клиент-серверную систему на основе именованных каналов. Клиент читает из *stdin* слово и пересылает его серверу по каналу, который создан сервером и имеет имя, известное всем клиентам. Вместе со словом клиент также сообщает серверу имя созданного клиентом канала для приема ответа от сервера. Этот канал имеет уникальное имя (используйте для него PID клиента). Сервер отправляет по этому каналу ответ клиенту - перевернутое слово, полученное от клиента. Клиент выводит ответ сервера в *stdout* и удаляет свой канал.

Выясните, может ли клиент заблокировать процесс сервера и почему.

[*] Необязательное задание. Оценивается дополнительно, если студент полностью выполнил обязательные задания.