Лабораторная работа по курсу "Операционные системы"

Межпроцессное взаимодействие в ОС Linux: каналы

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

I. Неименованные каналы (*unnamed pipes*)

В команде оболочки ***неименованный канал*** создается при помощи символа "|".

В программе канал создается при помощи системного вызова **pipe()**:

ret = pipe (fd);

где **ret** – код возврата целого типа, **fd** – массив **int fd[2]** из двух элементов, в который записываются значения в результате этого системного вызова. Системный вызов **pipe()** возвращает 0 в случае успеха и -1 в случае ошибки. Если системный вызов **pipe()** завершился успешно, то в первый элемент массива, **fd[0]**, записывается дескриптор выходного конца созданного канала (для чтения), и в **fd[1]** – дескриптор входного конца (для записи).

В общем случае читать из канала могут несколько процессов; писать в канал также могут несколько процессов. Для установления двухсторонней связи между процессами следует создать два канала, работающих в противоположных направлениях.

В этой части работы необходимо создать неименованный канал в начале работы родительского процесса, затем создать два дочерних процесса. Один из этих дочерних процессов будет выполнять какую-либо команду оболочки (например, **ls -l /home/student**) и записывать результаты работы во входной конец канала. Другой дочерний процесс (выполняющий, например, команду **sort**) будет получать данные из выходного конца того же канала. Таким образом, результат выполнения этих двух дочерних процессов будет идентичен выполнению последовательности двух команд оболочки: **ls -l /home/student | sort**

где "|" – символ операции канала в оболочке.

Задание 1.

1.1. На рис. 1 приведена структура программы **pipework.c** (текст программы имеется в папке данной лаб. работы).

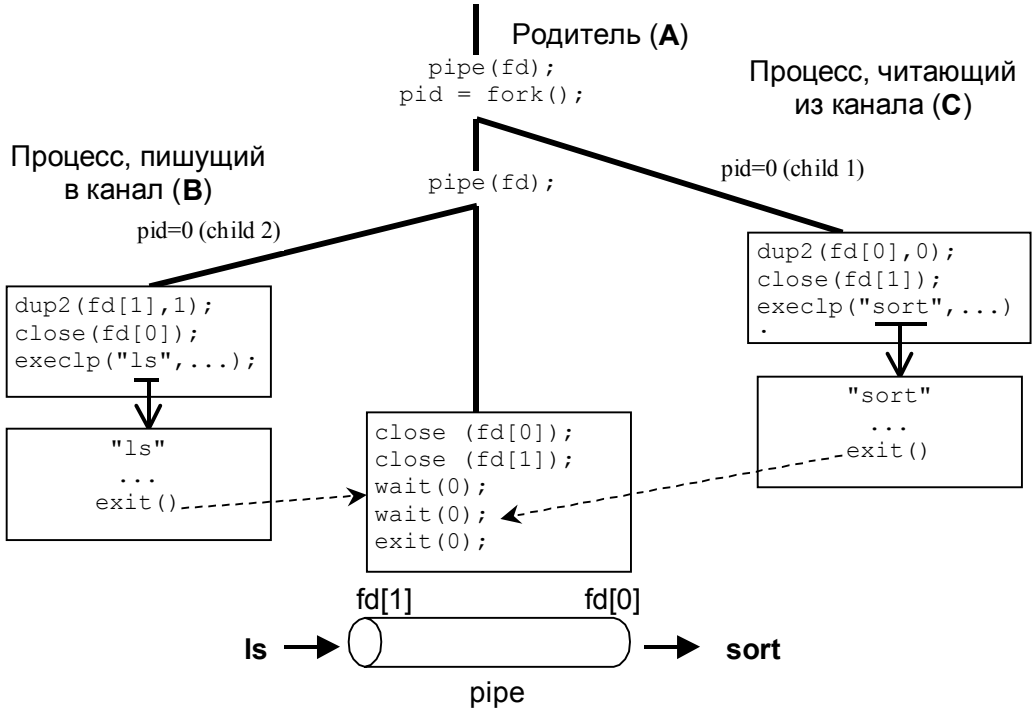

Рис. 1. Структура программы **pipework** для связи команд **ls** и **sort** при помощи канала

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**.

```
/* pipework.c */
#include <stdio.h>
#include <unistd.h>
main (){
int fd[2];  /*  Массив из двух дескрипторов для неименованного канала */
int pid;    /*  Идентификатор процесса */
if (pipe(fd) < 0){ /*  Канал должен быть создан до fork() */
    perror ("PIPE CREATION ERROR");
    exit (1);
    }
pid = fork ();  /*  Родитель:  создание первого дочернего процесса */
if (pid == 0) {  /*  Первый дочерний процесс начинается здесь */
   dup2 (fd[0],0); /* stdin назначен на выходной конец канала */
   close (fd[1]);  /* */
   execlp ("sort", "sort", 0); /* Запуск команды "sort", */
   }                           /* для которой stdin назначен на канал */
else   /* Родительский процесс прдолжает здесь */
pid = fork ();  /* Родитель:  создание второго дочернего процесса */
if (pid == 0) {   /* Второй дочерний процесс начинается здесь */
    dup2 (fd[1],1); /* stdout назначен на входной конец канала */
    close (fd[0]);    /* Закрыть для этого процесса входной конец канала */
    execlp ("ls", "ls", "-l", "/home/student", 0);
                        /* Запуск команды "ls", */
   }                    /* для которой stdout назначен на канал */
else { /* Родитель закрывает для себя оба конца канала и
            ожидает завершения дочерних процессов  */
   close (fd[0]);
   close (fd[1]);
   wait (0);
   wait (0);
   }
}
```

1.2. Запустите эту программу и сравните результаты с результатами следующих команд:
ls -l /home/student и **ls -l /home/student | sort**

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.

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.

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

II. Именованные каналы (*named pipe, FIFO*)

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

В оболочке ***именованный канал*** создается командой **mknod** или **mkfifo**.

Задание 2.

2.1. Выполните команду **$mkfifo fifo1** и выпишите в отчет информацию о созданном канале (команда **ls**). Затем выполните команду **$cat < fifo1**

Запустите второй экземпляр оболочки. 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.

2.2. Ниже приведены программы **server.c** и **client.c** (их тексты имеются в папке данной лаб. работы). 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.

```
/*  server.c SERVER PROGRAM */
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#define FIFONAME    "..." /* Уникальное имя канала  */
int main(void) {
    int n, fd;
    char buf[1024];  /* Буфер для чтения-записи */
    printf ("SERVER STARTS…\n");
    unlink(FIFONAME); /* Если существует файл с таким же именем, то удалить его */
/* Создать именованный канал с резрешением на запись и чтение для всех */
    if (mkfifo(FIFONAME, 0666) < 0)
       { perror("mkfifo problem in server"); exit(1);}
/* Проверить, что разрешение действительно равно 0666 */
    if (chmod(FIFONAME, 0666) < 0)
       {perror("chmod problem in server"); exit(1);}
/* Открыть созданный именованный канал для чтения     */
    if ((fd = open(FIFONAME, O_RDONLY)) < 0)
       {perror("open problem in server"); exit(1);}
/* Читать информацию из канала до конца файла, выводить ее в 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    "..."  /* Такое же имя, как в сервере */
int main(void) {
    int n, fd;
    char buf[1024];      /* Буфер для чтения-записи */
    printf ("CLIENT STARTS…\n");
/* Открыть существующий канал для записи. Канал уже создан сервером */
    if ((fd = open(FIFONAME, O_WRONLY)) < 0)
       {perror("open problem in client");exit(1);}
/* Чтение текста из stdin и запись его в канал. Текст должен завершиться Ctrl/d  */
    while ((n = read(0, buf, sizeof(buf))) > 0)
        write(fd, buf, n);
    printf ("CLIENT TERMINATED…\n");
    close(fd);
    exit(0);
}
```

2.1. 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**.

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

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

Задание 3.

Напишите программу в соответствии со своим вариантом.

Вариант 1, 5, 9, 13, 17, 21, 25. Эхо-сервер с использованием именованных каналов. Клиент читает текст из *stdin* и передает его серверу, сервер возвращает его клиенту. Клиент выводит на *stdout* ответ сервера и время, прошедшее между отправкой текста серверу и получением ответа от сервера.

Вариант 2, 6, 10, 14, 18, 22, 26. Эхо-сервер с использованием неименованных каналов (см. предыдущий вариант).

Вариант 3, 7, 11, 15, 19, 23, 27. Программа создает два дочерних процесса, которые записывают в неименованный канал по одному сообщению, в начале сообщения помещается PID процесса и длина сообщения. Родительский процесс читает из канала и распечатывает каждое сообщение и завершает процесс, пославший сообщение.

Вариант 4, 8, 12, 16, 20, 24, 28. Два процесса записывают в именованный канал по одному сообщению, в начале сообщения помещается длина сообщения и время его отправки. Третий процесс распечатывает эти сообщения и информацию о длительности передачи сообщения.

Порядок выполнения лабораторной работы

1. Выполните задания 1 – 3.
2. Занесите в отчет результаты выполнения заданий с ответами на все заданные в них вопросы.

Требования

1. При подготовке к лабораторной работе (дома) занесите в отчет прокомментированный текст программы из задания 3 и описания функций для работы с каналами.
2. Студент должен знать ответы на следующие вопросы:

Вопросы

1. Для чего предназначены неименованные и именованные каналы?
2. Какими синхронизирующими свойствами обладают неименованные и именованные каналы?
3. Объясните параметры системного вызова **pipe()**. Что такое входной и выходной конец канала?
4. Как в программе соединить концы созданного неименованного канала со стандартным входным и выходным устройствами?
5. Можно ли использовать неименованные и именованные каналы для обмена сообщениями между несвязанными процессами?
6. Как установить двухстороннюю связь при помощи каналов?
7. В чем отличие именованных каналов от неименованных?
8. Как создать в программе именованный канал?
9. Каково действие режима **O_NONBLOCK**, указываемого при открытии именованного канала?
10. Пусть некоторый процесс пытается писать в именованный канал, в то время как нет процесса, читающего из него. Что произойдет?
11. Какую информацию должен знать процесс для общения при помощи именованного канала?
12. Есть ли разница между именованным каналом и обычным файлом?
13. Можно ли при помощи именованного канала установить связь между процессами, выполняющимися на разных компьютерах? Почему?

Источники информации

1. Стивенс У. UNIX: взаимодействие процессов. – СПб: Питер, 2003. (Глава 4).
2. Конспект лекций.
