Лабораторная работа 14

Именованные каналы

Генералов Даниил, НПИ-01-21, 1032212280

Содержание

# 1 Цель работы

Целью данной работы является:

Приобретение практических навыков работы с именованными каналами.

# 2 Задание

Требуется проанализировать представленный код на C и добавить к нему функционал, описанный в задании.

# 3 Теоретическое введение

Именованные каналы – это способ предоставить каналам имя в файловой системе. Каналы – это способ коммуникации между процессами (IPC), основанные на принципе FIFO и выглядящие для программы как файловые дескрипторы.

Именованные каналы позволяют одному или нескольким процессам передавать данные серверному процессу. Например, именованные каналы используются для того, чтобы программа, которая хочет показать окно на X-сервере, представилась нему и передала информацию о желаемой операции с окнами.

# 4 Выполнение лабораторной работы

В задании лабораторной работы уже приведен код на C, который представляет собой пример использования именованных каналов. После прочтения кода видно, что для того, чтобы использовать именованный канал, нужно вызвать mknod, чтобы создать его по имени файла, а затем его можно открыть через open и читать и записывать данные через read и write.

В итоге я получил поведение, которое показано на рис. 1. Здесь же виден ответ на вопрос, что будет если сервер не закроет канал – когда мы создаем канал через mkfifo, мы не проверяем, существует ли уже такой файл, и если он действительно уже есть, то попытка его создать вызывает ошибку.

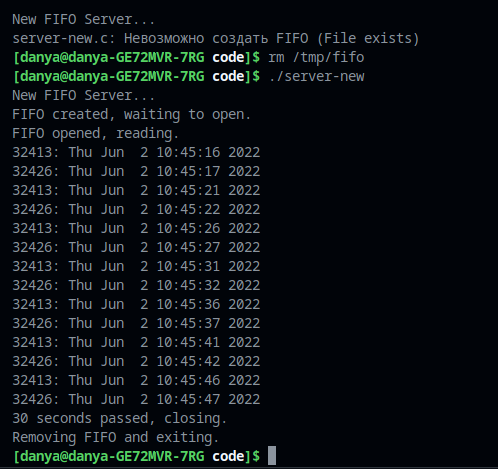


Рис. 1: Итоговое поведение

Чтобы сделать это, сначала я изменил файл client.c. Релевантная часть исходного кода показана на рис. 2.

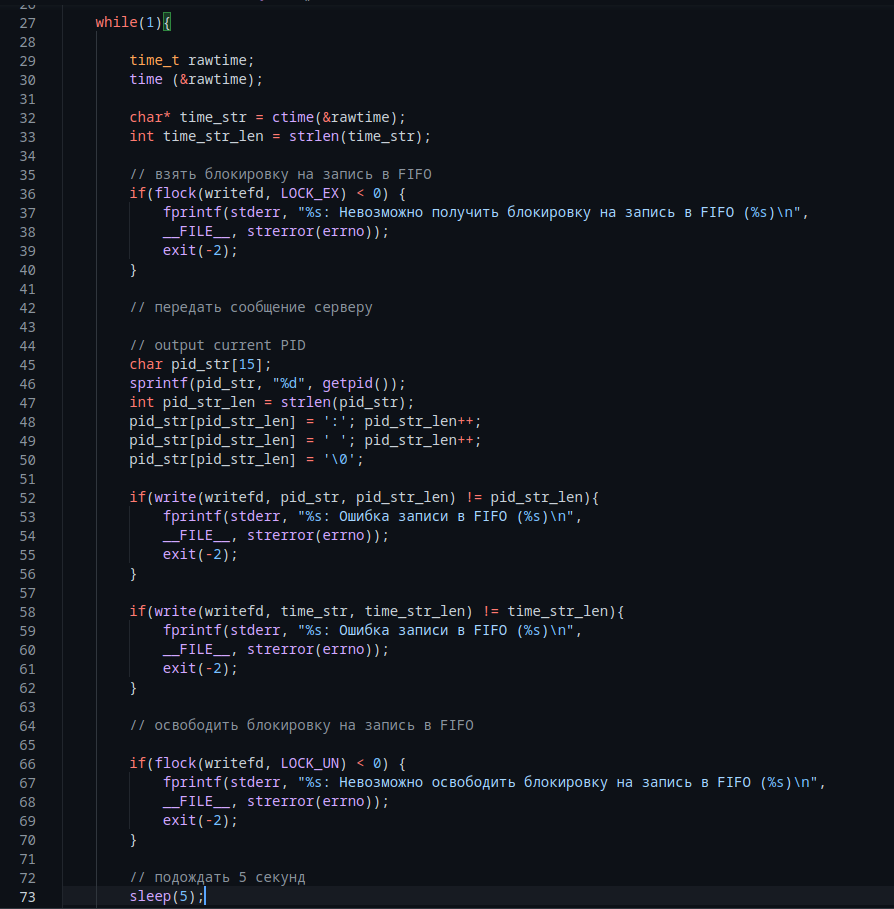


Рис. 2: Исходный код клиента

Теперь клиент, после того как открыл канал, входит в бесконечный цикл. В этом цикле мы сначала берем эксклюзивный доступ к каналу с помощью flock – это для того, чтобы другая копия клиента не начала писать в то же самое время. После этого мы создаем небольшой буфер, чтобы хранить строку, содержащую мой номер процесса. В этот буфер мы записываем номер процесса через sprintf, а затем в конец дописываем двоеточие и пробел. После этого мы выводим в канал сначала содержимое этого буфера, а затем вывод функции ctime – строкового представления текущего времени. Таким образом мы вывели в канал номер процесса и текущее время, и, закончив с этим, мы отпускаем блокировку канала и ждем несколько секунд.

Сервер тоже потребовалось изменить, и в server-new.c есть код, который показан на рис. 3.

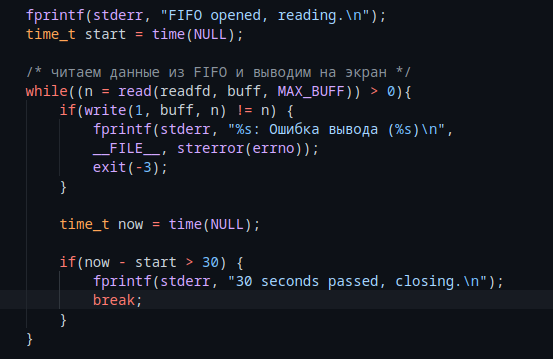


Рис. 3: Исходный код сервера

Прежде чем мы начали цикл считывания, мы измеряем текущее время (настоящее время, через time – в задании есть предложение использовать clock, но это фейк, потому что clock считает время процессора, которое не включает то время, которое мы ждем ввода в канал). После того, как измерено время начала, мы входим в цикл чтения из канала. Каждый раз, как мы получаем данные, мы выводим их в stdout, а после этого проверяем, сколько прошло времени – если это больше, чем 30 секунд, то мы выходим из цикла чтения. Это не полностью соответствует заданию, где написано, что сервер должен работать 30 секунд – на самом деле сервер будет работать 30 секунд, плюс сколько времени потребуется для получения следующего сообщения. Чтобы исправить это, потребовалось бы заменить вызов read на select, который является более общим способом работы с файлом и поддерживает timeout. Однако семантика вызова этого метода сильно отличается от вызова read, поэтому текущая реализация работает достаточно точно для наших задач.

# 5 Контрольные вопросы

1. В чем ключевое отличие именованных каналов от неименованных?

Именованный канал привязан к inode (и в итоге к файлу) в файловой системе, и подключение к нему осуществляется с помощью стандартного API работы с файлами. Неименованные каналы, напротив, существуют только в памяти процессов, и для создания необходимо вызвать функцию pipe из стандартной библиотеки unistd – в переданном массиве будут записаны два файловых дескриптора, которые этот процесс может использовать для коммуникации с другими – как правило, с теми, которые были созданы с помощью функции fork.

1. Возможно ли создание неименованного канала из командной строки?

Да, это происходит, когда программы соединяются с помощью оператора | – оболочка создает неименованный канал, затем запускает две программы и соединяет stdout первой с одной половиной канала, а stdin второй – с другой половиной.

1. Возможно ли создание именованного канала из командной строки?

Да, для этого есть команда mkfifo.

1. Опишите функцию языка С, создающую неименованный канал.

#include <unistd.h>  
int pipe(int fildes[2]);

Принимает указатель на массив из двух элементов типа int. Пытается создать именованный канал. Если успешно, возвращает 0. Тогда filedes[0] – дескриптор для чтения, а filedes[1] – для записи в новый канал. Если неуспешно, возвращает -1, задает errno, не создает никаких файловых дескрипторов и не изменяет переданный массив.

1. Опишите функцию языка С, создающую именованный канал.

#include <unistd.h>  
int mkfifo(const char \*pathname, mode\_t mode);

Принимает строку – путь к файлу, и настройку прав доступа. Создает файл. Если файл создан успешно, возвращает 0, иначе возвращает -1 и устанавливает errno.

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

Если прочитать меньше байтов, чем находится в канале, то оставшиеся байты останутся в канале и будут прочитаны при следующем чтении из этого канала.

Если попробовать прочитать больше байтов, чем находится в канале, то операция чтения будет блокировать, пока в канале не появится достаточное количество байтов. Из-за этого свойства рекомендуется читать по одному байту за раз и собирать их в строку вручную – тогда любое блокирование значит конец сообщения, и это можно обнаружить с использованием select вместо read.

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

Каналы имеют буфер, размер которого на моей системе равен 65536 байтов. Если из канала не читают, то все записи в канал идут в этот буфер.

Если пространство, которое остается в буфере, больше, чем количество байтов, которое процесс пытается записать, то запись осуществляется моментально и эти данные идут в буфер канала.

Если процесс пытается одномоментно записать в канал больше чем 65536 байтов, или он пытается записать больше байтов, чем осталось места в буфере, то операция записи блокируется. Вызов записи завершится, как только вся исходная строка будет записана в буфер канала.

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

Да, могут. Если два процесса записывают в один и тот же канал, их записи будут прочитаны в порядке, в котором они были отправлены – если один процесс запишет abcd, а затем другой процесс запишет 1234, то из канала будет прочитано abcd1234.

Одновременное чтение имеет гораздо более сложную структуру. Пусть два процесса оба хотят получить 5 байтов из канала, и первым запросил чтение процесс 1. В канал записывается сначала строка 1234, затем 5, затем 6789, затем 10, затем 1 и 2. Каждый положенный в канал байт оказывается только у одного из читающих процессов. Сначала первый процесс получит 1234, затем второй процесс получит 5, затем первый процесс получит 6789, затем второй процесс получит 10, затем первый процесс получит 10, 1 и 2. В итоге первый процесс прочитает строку 12346, и у него в буфере окажется 789, доступные для дальнейшего прочтения, а второй процесс получит строку 51012 и пустой буфер.

1. Опишите функцию write (тип возвращаемого значения, аргументы и логику работы). Что означает 1 (единица) в вызове этой функции в программе server.c (строка 42)?

#include <unistd.h>  
ssize\_t write(int fildes, const void \*buf, size\_t nbyte);

Принимает номер файлового дескриптора (где 1 – переданный аргумент – указывает на stdout, а 2 – на stderr), указатель на буфер, который нужно записать в файловый дескриптор, и количество байтов, которые нужно записать. Если сколько-то байтов было успешно записано, то возвращается количество байтов, которые были записаны, и оно никогда не будет больше nbyte. Если произошла ошибка, то возвращается -1 и задается errno.

1. Опишите функцию strerror.

#include <string.h>  
char \*strerror(int errnum);

Принимает номер ошибки (соответствующий тем, которые задаются в errno), и возвращает указатель на строку, которая содержит текстовое описание ошибки. Если это поддерживается системой, то строка переведена в текущую системную локаль согласно LC\_MESSAGES. Если переданный параметр не является кодом ошибки, то возвращается NULL.

# 6 Выводы

В этой работе мы смогли опробовать методы разработки для Linux, а также рассмотреть системный API для именованных каналов. Используя именованные каналы можно осуществлять взаимодействие между разными процессами в системе. Это – главный способ IPC в Unix-подобных системах, и он до сих пор часто используется, хотя постепенно заменяется на TCP/IP через localhost.