SoftPrayog (/)

Interprocess communication using FIFOs in Linux

1.0 FIFO

FIFOs are pipes with a name and are also commonly referred to as “named pipes”. Pipes are common on Linux command lines but do not have a system-wide name. So, any two processes that wish to communicate using a pipe need to be related, either parent and child or, sharing a common parent, who sets up the pipe and passes its file descriptors to individual processes. A FIFO is different because it has a name like a file. Also, since it exists in the filesystem, it is not necessary for the processes to be related to communicate using the FIFO.

We can create a FIFO from the shell using the `mknod` command. We can remove a FIFO just like a file using the `rm` command.

```
$ mknod myfifo p
$ ls -ls
total 0
0 prw-rw-r-- 1 user1 user1 0 May 30 15:08 myfifo
$ rm myfifo
$ ls -ls
total 0
```

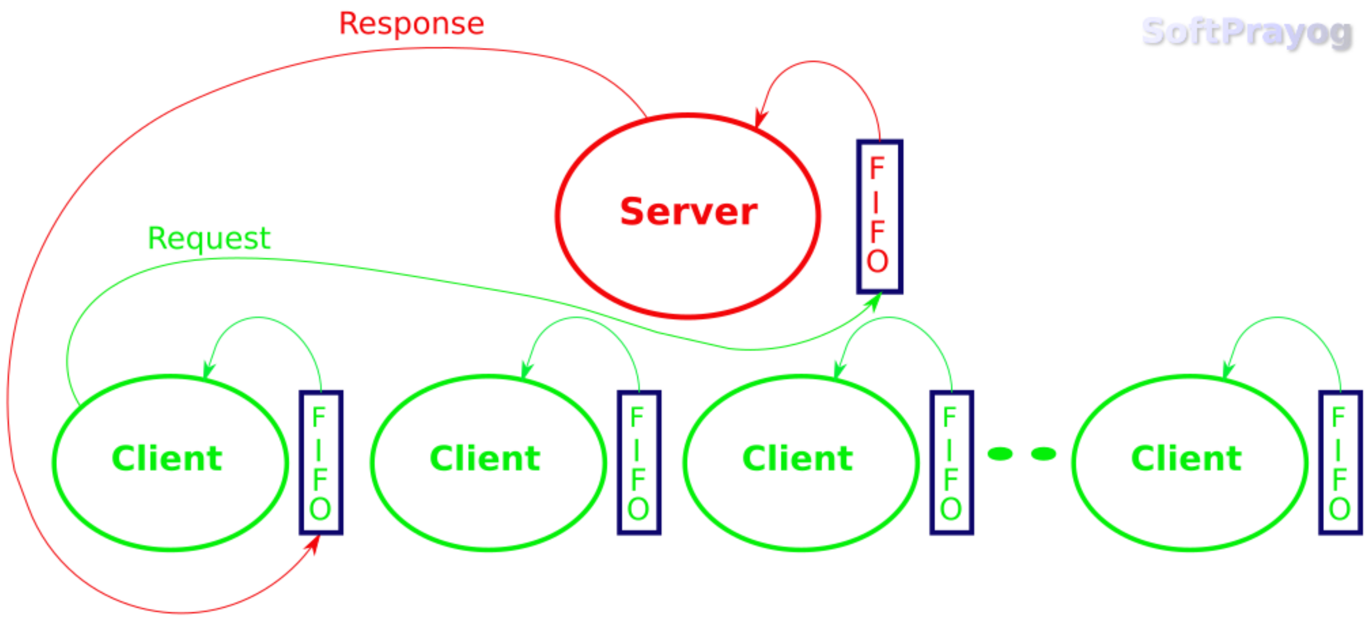
There is a specialized `mkfifo` command which could have been used in place of the `mknod` command, above.

```
$ mkfifo mynewfifo
$ file mynewfifo
mynewfifo: fifo (named pipe)
```

2.0 The Client-Server Paradigm

The client-server paradigm comprises of a single server process, which works all the time, receives requests from clients and gives them responses. A client is the process that manages the inputs and outputs for a live user. Clients come and go but the server works all the time. The clients communicate with the server using an interprocess communication mechanism. Each process in the paradigm has a system-wide mechanism for receiving messages.

In the example in a later section, we will use the FIFO as the mechanism for receiving messages. That is, the server will have a FIFO, where clients can put messages for the server. Similarly, each client will have a FIFO, where the server can put in messages for that client.



Interprocess Communication between client and server using FIFOs

3.0 `mkfifo` system call

We can create a FIFO from a program using the `mkfifo` system call.

```
#include <sys/stat.h>

int mkfifo (const char *path, mode_t perms);
// Returns -1 on error.
```

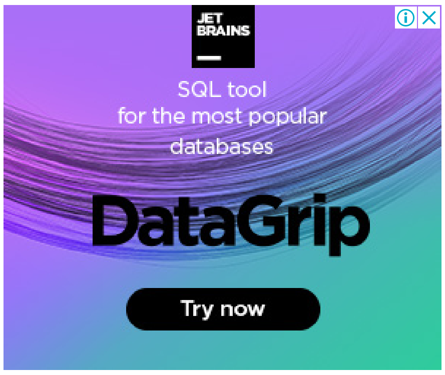
A FIFO can be opened using the `open` system call. After open, we can use the `read` and `write` system calls for reading from and writing to the FIFO, using the file descriptor returned by `open`. Of course, as per our design, we will either read or write to a FIFO but not do both. A process, be it a client or the server, reads from its own FIFO for receiving data and writes on other process's FIFO for sending data to that process.

4.0 The server

In this example, the server accepts numbers from clients in a message in its FIFO. It adds the numbers and sends the result in a message to the requesting client. The server needs the client FIFO name to get back to the client. So the server mandates that the clients first put their FIFO name and follow it up by the numbers to be added, all in a single line, in their requests. The server program is as follows.

```
/*
     add_server.c: A server to add numbers received in message.

 */
#include <stdio.h>
#include <stdlib.h>
#include <error.h>
#include <errno.h>
#include <sys/stat.h>
#include <unistd.h>
#include <fcntl.h>
#include <string.h>

#define SERVER_FIFO "/tmp/addition_fifo_server"
#define MAX_NUMBERS 500

int main (int argc, char **argv)
{
    int fd, fd_client, bytes_read, i;
    char buf [4096];
    char *return_fifo;
```

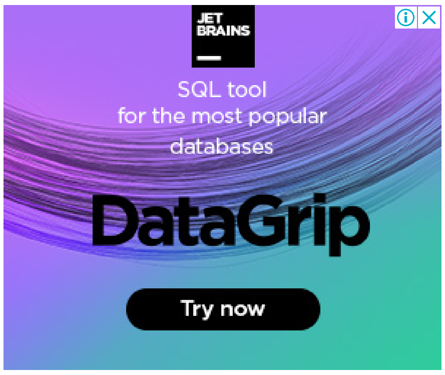
```
        char *numbers [MAX_NUMBERS];


        if ((mkfifo (SERVER_FIFO, 0664) == -1) && (errno != EEXIST)) {
            perror ("mkfifo");
            exit (1);
        }
        if ((fd = open (SERVER_FIFO, O_RDONLY)) == -1)
            perror ("open");


         while (1) {
             // get a message
             memset (buf, '\0', sizeof (buf));
             if ((bytes_read = read (fd, buf, sizeof (buf))) == -1)
                 perror ("read");
             if (bytes_read == 0)
                 continue;

             if (bytes_read > 0) {
                 return_fifo = strtok (buf, ", \n");

                 i = 0;
                 numbers [i] = strtok (NULL, ", \n");

                 while (numbers [i] != NULL && i < MAX_NUMBERS)
                     numbers [++i] = strtok (NULL, ", \n");

                 int total_numbers = i;
                 double sum = 0;
                 unsigned int error = 0;
                 char *ptr;

                 for (i = 0; i < total_numbers; i++) {
                     double f = strtod (numbers [i], &ptr);

                     if (*ptr) {
                         error = 1;
                         break;
                     }
                     sum += f;
                 }

                 /* Send the result */
                 if ((fd_client = open (return_fifo, O_WRONLY)) == -1) {
                     perror ("open: client fifo");
                     continue;
                 }

                 if (error)
                     sprintf (buf, "Error in input.\n");
                 else
                     sprintf (buf, "Sum = %.8g\n", sum);

                 if (write (fd_client, buf, strlen (buf)) != strlen (buf))
                     perror ("write");

                 if (close (fd_client) == -1)
                     perror ("close");
             }

         }
    }
```



5.0 The client

The client program takes user input comprising of numbers to be added. It makes a message for the server by writing its FIFO name followed by the numbers input by the user. It puts the message in the server's FIFO and then waits for the response to come in its own FIFO. After getting the server's response, it prints the answer on the screen. The client program is as follows.

```
/*
     add_client.c: Client program to take user input comprising
                   of numbers, send request message to add_server
                   and receive server's response

 */
#include <stdio.h>
#include <stdlib.h>
#include <error.h>
#include <errno.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
#include <fcntl.h>
#include <string.h>
#include <memory.h>

#define SERVER_FIFO "/tmp/addition_fifo_server"

char my_fifo_name [128];
char buf1 [512], buf2 [1024];

int main (int argc, char **argv)
{
    int fd, fd_server, bytes_read;

    // make client fifo
    sprintf (my_fifo_name, "/tmp/add_client_fifo%ld", (long) getpid ());

    if (mkfifo (my_fifo_name, 0664) == -1)
        perror ("mkfifo");

    while (1) {
        // get user input
        printf ("Type numbers to be added: ");
        if (fgets(buf1, sizeof (buf1), stdin) == NULL)
            break;

        strcpy (buf2, my_fifo_name);
        strcat (buf2, " ");
        strcat (buf2, buf1);

        // send message to server

        if ((fd_server = open (SERVER_FIFO, O_WRONLY)) == -1) {
            perror ("open: server fifo");
            break;
        }

        if (write (fd_server, buf2, strlen (buf2)) != strlen (buf2)) {
            perror ("write");
             break;
        }

        if (close (fd_server) == -1) {
            perror ("close");
            break;
        }

        // read the answer
        if ((fd = open (my_fifo_name, O_RDONLY)) == -1)
           perror ("open");
        memset (buf2, '\0', sizeof (buf2));
        if ((bytes_read = read (fd, buf2, sizeof (buf2))) == -1)
            perror ("read");

        if (bytes_read > 0) {
            printf ("Answer: %s\n", buf2);
        }

        if (close (fd) == -1) {
            perror ("close");
            break;
        }
    }
}
```



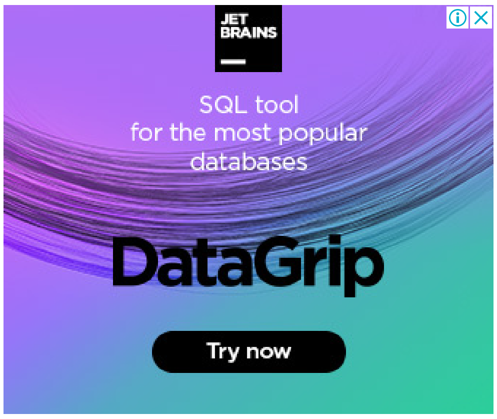
6.0 Running the server and client

We can run the server and test it with a client from the shell.

```
$ nohup ./add_server &
$ ./add_client
Type numbers to be added: 23 45.99 3.01
Answer: Sum = 72

Type numbers to be added: 0.009 4 3.33
Answer: Sum = 7.339
```

7.0 See also

1. Interprocess communication using pipes in Linux (interprocess-communication-using-pipes-in-linux)


JET BRAINS
SQL tool for the most popular databases
DataGrip
Try now

Recent Posts (/all-posts)

- Failed to fetch webpage, temporary failure resolving domain (/troubleshooting/failed-to-fetch-webpage-temporary-failure-resolving-domain)
- Network Socket Programming using UDP in C (/programming/network-socket-programming-using-udp-in-c)
- Composer Manager: Composer's install command must be run (/troubleshooting/composer-manager-composers-install-command-must-be-run)
- Drupal: Notice: Trying to get property of non-object in block_block_view() (/troubleshooting/drupal-notice-trying-to-get-property-of-non-object-in-block_block_view)
- Dovecot: Fatal: Error in configuration file /etc/dovecot/dovecot.conf: protocols: Unknown protocol: sieve (/troubleshooting/dovecot-fatal-error-in-configuration-file-unknown-protocol-sieve)

(/rss.xml)

App

- Random Quotes (https://randomquotesapp.com)

GET IT ON Google Play (https://play.google.com/store/apps/details?id=in.softprayog.randomquote)

Related Posts

- Interprocess communication using POSIX message queues in Linux (/programming/interprocess-communication-using-posix-message-queues-in-linux)
- Interprocess communication using POSIX Shared Memory in Linux (/programming/interprocess-communication-using-posix-shared-memory-in-linux)
- D-Bus Tutorial (/programming/d-bus-tutorial)
- Interprocess communication using pipes in Linux (/programming/interprocess-communication-using-pipes-in-linux)
- Interprocess communication using System V Shared Memory in Linux (/programming/interprocess-communication-using-system-v-shared-memory-in-linux)

Most Popular (/most-popular-posts)

- Interprocess communication using POSIX message queues in Linux (/programming/interprocess-communication-using-posix-message-queues-in-linux)
- Interprocess communication using POSIX Shared Memory in Linux (/programming/interprocess-communication-using-posix-shared-memory-in-linux)
- D-Bus Tutorial (/programming/d-bus-tutorial)
- Creating processes with fork and exec in Linux (/programming/creating-processes-with-fork-and-exec-in-linux)
- Interprocess communication using pipes in Linux (/programming/interprocess-communication-using-pipes-in-linux)
- Interprocess communication using FIFOs in Linux (/programming/interprocess-communication-using-fifos-in-linux)
- Interprocess communication using System V Shared Memory in Linux (/programming/interprocess-communication-using-system-v-shared-memory-in-linux)
- Interprocess communication using System V message queues in Linux (/programming/interprocess-communication-using-system-v-message-queues-in-linux)
- ps command usage examples in Linux (/tutorials/ps-command-usage-examples-in-linux)
- htop command in Linux (/tutorials/htop-command-in-linux)

Search

About SoftPrayog (/about) Privacy Policy (/privacy-policy) Contact Us (/contact) Photographs (https://goo.gl/R7quJL)