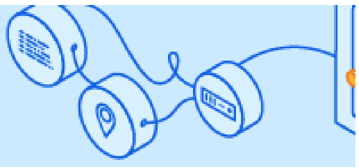

Proper FIFO client-server connection

Ask Question

I'm trying to write simple client and server C programs, communicating with each other in separate terminals.

The server has to create a public fifo and wait for the client. Meanwhile the client is creating his own fifo through which the server's response will come. The task of the client is sending the server a name created by the queue and get in return the result of the `ls` command.

I did search for an answer, for example: fifo-server-program, example-of-using-named-pipes-in-linux-bash, how-to-send-a-simple-string-between-two-programs-using-pipes. I started with the code from the third link and slowly modified it.

What I've got now, is a client taking input from the user, sending it to the server and receiving it back. But it only works once. I have no idea why. The body of main function is below. I will be grateful for any help.

EDIT: I got it working! :D The codes are below, maybe it will help someone.

The server.c code:

```
#include <unistd.h>
#include <stdio.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>

int main(int argc, char* argv[])
{
    int fds[2];
    char tab[BUFSIZ];
    int fd, n;

    char *myfifo = "/tmp/serwer";
    char *myfifo2 = "/tmp/client";
```

```
            fds[0]=open(myfifo2,O_RDONLY);
            fds[1]=open(myfifo,O_WRONLY);

            read(fds[0],tab,BUFSIZ);

            if (strcmp("klient",tab)==0) {
                printf("Od klienta: %s\n",tab);
                fd=open(tab,O_WRONLY);

                if(fork()==0)
                {
                    dup2(fds[1],1);
                    close(fds[1]);
                    execlp("ls","ls","-l",NULL);
                    close(fds[0]);
                    close(fds[1]);
                }
                else
                {
                    dup2(fds[0],0);
                    n = read(fds[0],tab,BUFSIZ);
                    write(fd,tab,n);
                    close(fds[0]);
                    close(fds[1]);
                }
            }
            memset(tab, 0, sizeof(tab));
            close(fd);
            close(fds[0]);
            close(fds[1]);
        }

        unlink(myfifo);
        return 0;
    }
```

The client.c code:

```
#include <unistd.h>
#include <stdio.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>

int main(int argc, char* argv[])
{
    int fds[2];
    char *myfifo = "/tmp/serwer";
    char *myfifo2 = "/tmp/client";

    mkfifo(myfifo2,0666);
    fds[0]=open(myfifo,O_RDONLY);
    fds[1]=open(myfifo2,O_WRONLY);

    char tab[BUFSIZ];
    memset(tab, 0, sizeof(tab));

    write(fds[1],"klient",6);
    perror("Write:"); //Very crude error check
    read(fds[0],tab,sizeof(tab));
    perror("Read:"); // Very crude error check

    printf("Odebrano od serwera: %s\n",tab);

    close(fds[0]);
    close(fds[1]);
    unlink(myfifo2);
    return 0;
}
```

1 1

asked Dec 23 '11 at 0:00

uluroki
91 2 2 11

1 It's not apparent from this code. Have you gotten the ls output working or are you just sending small msgs? The usual pitfall in this scenario is that you deadlock i.e. server is waiting for input while client is waiting for input. They wait forever because no one is sending anything. – Duck Dec 23 '11 at 1:17

No, the ls command is not working yet. These programs work once - the server waits for a message, returns it to the client, the client can close. When I want to send another message, both client and server become unresponsive. – uluroki Dec 23 '11 at 19:18

2 Answers

Why don't you just manage both fifo's in the server? Simply changing your code to do this makes it work correctly.

If you actually want to have a client-server relationship, with a server serving many different clients, sockets would probably be a better choice.

client.cpp

```
#include <stdio.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
   int client_to_server;
   char *myfifo = "/tmp/client_to_s

   int server_to_client;
   char *myfifo2 = "/tmp/server_to_

   char str[BUFSIZ];
   printf("Input message to serwer
   scanf("%s", str);


   /* write str to the FIFO */
   client_to_server = open(myfifo,
   server_to_client = open(myfifo2
   perror("Read:"); // Very crude

   printf("...received from the se
   close(client_to_server);
   close(server_to_client);

   /* remove the FIFO */

   return 0;
}
```

server.cpp

```
     #include <fcntl.h>
     #include <stdio.h>
     #include <sys/stat.h>
     #include <unistd.h>
     #include <string.h>

     int main()
     {
        int client_to_server;
        char *myfifo = "/tmp/client_to_

        int server_to_client;
        char *myfifo2 = "/tmp/server_to_

        char buf[BUFSIZ];

        /* create the FIFO (named pipe)
        mkfifo(myfifo, 0666);
        mkfifo(myfifo2, 0666);

        /* open, read, and display the
        client_to_server = open(myfifo,
        server_to_client = open(myfifo2,

        printf("Server ON.\n");

        while (1)
        {
           read(client_to_server, buf,

           if (strcmp("exit",buf)==0)
           {
              printf("Server OFF.\n");
              break;
           }

           else if (strcmp("",buf)!=0)
           {
              printf("Received: %s\n",
              printf("Sending back...\n
              write(server_to_client,bu
           }

           /* clean buf from any data *
           memset(buf, 0, sizeof(buf));
        }

        close(client_to_server);
        close(server_to_client);

        unlink(myfifo);
        unlink(myfifo2);
        return 0;
     }
```

answered Dec 23 '11 at 19:37

Lolaland

It only works once because of how named pipes work. Each time you `open` a named pipe for read you block until another process opens it for write. Then you are paired up and the file descriptor connects your processes. Once either end closes that connection that's the end of that pipe. In order for your server to "accept another connection" it needs to move the `open` and `close` of the pipes into its main loop so it can be paired up over and over.

answered Dec 23 '11 at 19:44
Ben Jackson
57.8k 6 74 128