## Programming Exercise I - Client/Server FIFOs

### Assignment Summary

This assignment is a variation of the software described in Section 6.4 in the Robbins slides. In this assignment, you shall develop a C program that provides the client side of a simple client/server system using two FIFOs for inter-process communication. The server program, **server-program.c**, has already been written for you and is available on Blackboard. It acts as a simple time server by supplying the current time as the response to a request from a client. It also shuts down when requested by a client.

The client/server communication protocol for this program is illustrated in the **client-sample-run.txt** file and the **server-sample-run.txt** file. It is also summarized below in the next section. The client program shall display each of the "CLIENT:" messages in the same format and similar content as shown in the **client-sample-run.txt** file.

Submit the C source code file for your client-side program on Blackboard. Do not submit any other files. Your finished program should compile, link, and run in a POSIX standard environment with no warnings nor error messages.

### Client/Server Protocol

The server program and client program communicate using a specific protocol as described in the steps below. Your program should follow this protocol and produce the same messages along with the data values specific to each run (e.g., the current date and time). Note that the exact output messages produced by the server program and client program are only shown below in the first two bullets. The rest of the messages for both the client and server programs can be seen in the sample run files.

1. (SERVER) In the current working directory, the server program creates a FIFO named **server.fifo** with user read/write privileges. If an error occurs, the program prints "SERVER: Error creating the server FIFO" and terminates. If a FIFO by that name already exists (i.e., errno = EEXIST), the program prints "SERVER: An existing server FIFO has been detected"; otherwise, the program prints "SERVER: The server FIFO has been created". In either of the last two cases, the program proceeds to perform its server duties
2. (CLIENT) The client program parses the command line and either prints a usage message and terminates or sets a boolean flag indicating that server shutdown has been requested by way of a "-s" command line option
3. (CLIENT) In the current working directory, the client program creates a FIFO named **client-99999.fifo** with user read/write privileges, where 99999 is replaced with the actual process ID of the client. If an error occurs, the program prints "CLIENT: Error creating the client FIFO" and terminates. If a FIFO by that name already exists (i.e, errno == EEXIST), the program prints "CLIENT: An existing client FIFO has been detected" and terminates; otherwise, the program prints "CLIENT: The client FIFO named client-99999.fifo has been created", where 99999 is the actual process ID. In the last case, the program proceeds to perform its client duties
4. (SERVER) The server opens up it's FIFO in read mode, checks for a client request in its FIFO and implicitly blocks. (Note that this occurs at the top of a loop every time the server awaits a request from a new client)
5. (CLIENT) The client opens the server FIFO in write mode and sends either a time request or a shutdown request to the server. The time request is in the form of a stream of characters consisting of the process ID of the client. The shutdown request is a constant shutdown phrase
6. (CLIENT) If a shutdown request was sent, the client closes its connections to both the client and server FIFOs and unlinks its client FIFO so that the FIFO will be deleted from the current working directory. It then terminates.
7. (SERVER) The server reads the client request from its FIFO. If a shutdown request was received, the server closes it connections to the client and server FIFOs, exits the client processing loop, and terminates normally. If a time request was received, the server opens up the client FIFO in write mode and writes the current time (i.e., ctime() ) to it as a stream of characters. The server knows the exact name of the client's FIFO because the name follows the standard "client-99999.fifo" format described above
8. (SERVER) The server closes it connections to the client and server FIFOs and returns to the top of the client processing loop to await a request from another client
9. (CLIENT) The client opens a connection to its FIFO in read mode. The client then reads a stream of characters from its FIFO and displays it. The display should show the current date/time string. When this occurs, then the inter-process communication was successful
10. (CLIENT) The client closes its connections to both the client and server FIFOs and unlinks its client FIFO so that the FIFO will be deleted from the current working directory. It then terminates.

### Design and Implementation Constraints

* Download and use the **client-skeleton.c** file as the start of your client program. Rename the file to **client-program.c**
* The program shall have a modular design consisting of user-defined functions to perform the client steps
* The program shall not declare any global variables
* The program shall not prompt the user for any information
* The program shall use the fprintf() function and the stderr file descriptor to print each of the "CLIENT:" messages that involve completion of an action or waiting for an action to occur
* The program shall use the sprintf() function to construct the contents of any character strings
* The program shall use the perror() function to print any "CLIENT:" error messages
* The program shall utilize the built-in functions listed below, among others if needed, to perform the client/server communication and other processing. The program shall respond to any error conditions detected (i.e., returned) by these built-in functions

|  |  |  |  |
| --- | --- | --- | --- |
| getpid() | mkfifo() | perror() | open() |
| read() | write() | close() | unlink() |

### Running and Testing your Client/Server System

When testing out either program, be sure to first start the server program. That way it is running and waiting for a message from a client before you start a client program. Also, both the client and the server programs should each be located and started in the same working directory.